C in Hindi

BccFaIna.com
097994-55505

Kuldeep Chand

With this eBook you can Learn Programming Fundamentals with Deep Details in easy to understand Hindi Language.

I have Included so many Example Programs and Code Fragements in this ebook to easily understand various kinds of Programming Concept with Detaild Program Flow Discussion to understand the working of the Program Step by Step.

Without learning “C” Language, you can’t learn any Modern Programming Language which is used mostly for Professional Application Software development like C++, Java, C#, JavaScript, PHP, Python, Perl, etc…

So, learn “C” and start moving in the way of Professional Development for full of Joy and Healthy Programming Career.

C

In Hindi

Kuldeep Chand

BetaLab Computer Center
Falna

Programming Language "C" in HINDI

Lead Editors: **Kuldeep Chand**

Distributed to the book trade worldwide by Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

e-mail bccfalna@gmail.com,

or

visit http://www.bccfalna.com

For information on translations, please contact BetaLab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

Phone **097994-55505**

This book is dedicated to those

who really wants to be

a

PROFESSIONAL DEVELOPER

INDEX OF CONTENTS

Table of Contents

PROGRAMMING INTRODUCTION

Introduction

सभ्यता की शुरूआत से ही मानव को Information की जरूरत रही है। इसीलिए वह समय–समय पर सूचनाओं को एकत्रित करने व उन सूचनाओं के आधार पर सही व उचित निर्णय लेने के नए व विकसित तरीके खोजता रहा है। सूचना की आवश्यकता व महत्व के कारण सबसे पहला आविष्कार कागज व कलम हुआ।

जैसे–जैसे मानव का विकास होता गया वैसे–वैसे उसने नए शहर, राज्य व देश बनाए और उन देशों के बीच व्यापार व वाणिज्य के कारण विभिन्न सम्बंध बने और आज केवल व्यापार व वाणिज्य ही नहीं बल्कि जीवन की लगभग हर सूचना का Internet के माध्यम से इन देशों के बीच आदान प्रदान हो रहा है। कृषि क्रांति व औधोगिक क्रांति के बाद आज हम सूचना क्रांति के युग में जी रहे हैं।

पहले सूचनाओं को मिट्टी के बर्तनों पर चित्रात्मक रूप में व शब्दों के रूप में लिखा जाता था। फिर कागज व कलम के विकास से इन पर विभिन्न सूचनाओं को Store करके रखा जाने लगा और आज हम इन्हीं सूचनाओं को Computer पर Manage करते हैं।

विभिन्न प्रकार के आंकडों (**Data**) का संकलन (Collection) करना और फिर उन आंकडों को विभिन्न प्रकार से वर्गीकृत (Classify) करके उनका विश्लेषण (Analyze) करना तथा उचित समय पर उचिन निर्णय लेने की क्षमता प्राप्त करना, इस पूरी प्रक्रिया को Computer की भाषा में **Data Processing** करना कहा जाता है।

Data – Value or a Set of Values

असिद्ध तथ्य (Facts) अंक (Figures) व सांख्यिकी (Statics) का वह समूह, जिस पर प्रक्रिया (Processing) करने पर, एक अर्थपूर्ण (Meaningful) सूचना (Information) प्राप्त (Generate) हो, **Data** कहलाता है। Data, मान या मानों का एक समूह (***Value or a Set of Values***) होता है, जिसके आधार पर (After Processing) हम निर्णय (Decision) लेते हैं।

इसे एक उदाहरण द्वारा समझने की कोशिश करते हैं। संख्याएं (0 से 9 तक) कुल दस ही होती हैं। लेकिन यदि इन्हें एक व्यवस्थित क्रम में रख दिया जाए, तो एक सूचना Generate होती है। इसलिए ये संख्याएं Data हैं।

अंग्रेजी भाषा में Small व Capital Letters के कुल 52 Characters ही होते हैं, लेकिन यदि इन्हें एक सुव्यवस्थित क्रम में रखा जाए, तो हजारों पुस्तकें बन सकती हैं। इसलिए ये Characters Data हैं।

Computer में हम इन्हीं दो रूपों में वास्तविक जीवन की विभिन्न बातों को Store करते हैं और उन पर Processing करके आवश्यकतानुसार Information Generate करते हैं। जैसे किसी School

के विभिन्न Students की ये जानकारी Manage करनी हो कि किसी Class में कौन–कौन से Students हैं, उनका Serial Number क्या है और वे किस Address पर रहते हैं, तो ये सभी तथ्य असिद्ध रूप में Computer के लिए **Data** हैं क्योंकि किसी Student के Serial number को 0 से 9 के कुछ अंकों के समूह रूप में Express किया जाता है और Student का नाम व पता Characters के एक सुव्यवस्थित समूह के रूप में Express किया जाता है।

जब 0 से 9 तक के कुछ अंकों को एक समूह में व्यवस्थित किया जाता है तब किसी एक Student का एक Serial Number बन जाता है और जब विभिन्न Characters को एक समूह में व्यवस्थित किया जाता है, तब किसी Student का नाम व Address बन जाता है। ये नाम व Address ही किसी Student की कुछ **Information** प्रदान करते हैं।

Processing – Generating Results

Data जैसे कि अक्षर, अंक, सांख्यकी Statics या किसी चित्र को सुव्यवस्थित करना या उनकी Calculation करना, **Processing** कहलाता है। किसी भी Processing मे निम्न काम होते हैं:

Calculation	किसी मान को जोडना, घटाना, गुणा करना, भाग देना आदि।
Comparison	कोई मान बडा, छोटा, शून्य, Positive, Negative, बराबर है, आदि।
Decision Making	किसी Condition के आधार पर निर्णय लेना।
Logic	आवश्यक परिणाम को प्राप्त करने के लिए अपनाया जाने वाला Steps का क्रम।

केवल अंकों की गणना करना ही Processing नहीं कहलाता है। बल्कि किसी भी प्रकार के मान को जैसे कि किसी Document में से गलतियों को खोजने की प्रक्रिया या कुछ नामों के समूह को आरोही **(Ascending)** या अवरोही **(Descending)** क्रम में व्यवस्थित करने की प्रक्रिया को भी Processing की कहते हैं।

Computer में Keyboard से जो भी Data Input किया जाता है, उस Data का तब तक कोई अर्थ नहीं होता है, जब तक कि Computer द्वारा उस Data पर किसी प्रकार की कोई Processing ना की जाए। जैसे उदाहरण के लिए Computer में R, a, d, h, a ये पांच अक्षर अलग–अलग Input किए जाते हैं इसलिए ये सभी अक्षर ***Row Data*** के समान हैं। Computer इन पांचों अक्षरों पर ***Processing*** करके इन्हें एक क्रम में व्यवस्थित कर देता है और हमें **"Radha"** नाम प्रदान करता है जो कि एक अर्थपूर्ण सूचना (***Information***) है।

Information – Processed Data

जिस Data पर Processing हो चुकी होती है, उसे **Processed Data** या **Information** कहते हैं। दूसरे शब्दों में कहें तो किसी Data पर Processing होने के बाद जो अर्थपूर्ण परिणाम (Result)

प्राप्त होता है, उसे ही सूचना **(Information)** कहते हैं। एक Processing से Generate होने वाली किसी Information को हम किसी दूसरी Processing में फिर से Data के रूप में उपयोग में लेकर नई Information Generate कर सकते हैं और ये क्रम आगे भी जारी रखा जा सकता है।

उदाहरण के लिए R, a, m, K, i, l, l, e, d, R, a, v, a, n ये Characters हम अलग–अलग Input करते हैं। Computer पहले इन पर Processing करके Ram, Killed, व Ravan तीन शब्द बनाता है, जो कि हमारे लिए तीन अलग सूचनाओं को Represent करता है। क्योंकि **Ram, Ravan** व **Killed** तीनों ही शब्द अपने आप में परिपूर्ण हैं, इसलिए ये तीनों ही शब्द एक प्रकार की सूचना हैं जबकि यदि **"Ram Killed Ravan"** लिखा जाए तो इस वाक्य के लिए ये तीनों ही शब्द एक Data के समान हैं, जो Processing के कारण आपस में एक व्यवस्थित क्रम में Arrange होकर एक सूचना प्रदान करते हैं।

सारांश में कहें तो Computer में हम सभी प्रकार की सूचनाओं को Data के आधार पर Store करते हैं। इन Data पर Processing करते हैं जिससे सूचनाएं Generate होती हैं और इन सूचनाओं के आधार पर हम निर्णय लेते हैं। Data वास्तव में कोई अंक अक्षर या चित्र हो सकता है। Computer में इन्हीं मानों को Manage किया जाता है। यानी Data वास्तव में कोई मान या मानों का एक समूह होता है।

What is a Computer

Computer एक ऐसी Electronic Machine है, जो निर्देशों के समूह (जिसे **Program** कहते हैं) के नियंत्रण में Data या तथ्यों पर Processing करके **Information** Generate करता है।

Computer में Data को Accept करने और उस Data पर Required Processing करने के लिए किसी Program को Execute करने की क्षमता होती है। ये किसी Data पर Mathematical व Logical क्रियाएं करने में सक्षम होता है। Computer में Data को Accept करने के लिए Input Devices होती है, जबकि Processed Data यानी Information को प्रस्तुत करने के लिए Output Devices होती हैं। Data पर Processing का काम जिस Device द्वारा सम्पन्न होता है, उसे Central Processing Unit या CPU कहते हैं। ये एक Microprocessor होता है, जिसे Computer का दिमाग भी कहते हैं। किसी भी Computer की निम्नलिखित क्षमताएं होती हैं:

1. User द्वारा Supplied Data को Accept कर सकता है।
2. Input किए गए Data को Computer की Memory में Store करके Required परिणाम प्राप्त करने के लिए किसी Instructions के समूह यानी किसी Program को Execute कर सकता है, जो कि उस Input किए गए Data पर Processing कर सकता है।
3. Data पर Mathematical व Logical क्रियाओं (Operations) को क्रियान्वित (Perform) कर सकता है।
4. User की आवश्यकतानुसार Output प्रदान कर सकता है।

System – Group of Units to Solve a Problem

Computer एक System होता है। जब किसी एक या एक से अधिक समस्याओं को सुलझाने या किसी लक्ष्य को प्राप्त करने के लिए कई स्वतंत्र इकाईयां (Individual Units) मिलकर काम कर रहे होते हैं, तो उन इकाईयों के समूह को **System** कहा जाता है।

जैसे कोई Hospital एक System होता है जिसे **Hospital System** कहा जाता है। Doctors, Nurses, चिकित्सा से सम्बंधित विभिन्न उपकरण, Operation Theater, Patient आदि किसी Hospital System की विभिन्न इकाईयां हैं। यदि इन में से किसी की भी कमी हो तो Hospital अधूरा होता है। इसी तरह से Computer भी एक System है, जिसके विभिन्न अवयव जैसे कि Monitor, Mouse, Keyboard, CPU आदि होते हैं और ये सभी आपस में मिलकर किसी समस्या का एक उचित समाधान प्रदान करते हैं।

Program and Software

Computer Programming समझने से पहले हमें ये समझना होता है कि Computer क्या काम करता है और कैसे काम करता है। कम्प्युटर का मुख्य काम Data का Management करना होता है। हमारे आस–पास जो भी चीजें हमें दिखाई देती हैं, Computer के लिए वे सभी Data हैं और एक Programmer को इन सभी चीजों को Computer में Data के रूप में ही Represent करना होता है। Computer केवल Electrical Signals या मशीनी भाषा को समझता है। ये मशीनी भाषा बायनरी रूप में होती है, जहां किसी Signal के होने को 1 व ना होने को 0 से प्रदर्शित किया जाता है। यदि हम हमारी किसी बात को Binary Format में Computer में Feed कर सकें, तो Computer हमारी बात को समझ सकता है।

Computer भाषा वह भाषा होती है जिसे Computer समझ सकता है, क्योंकि हर Computer भाषा का एक Software होता है। ये Software हमारी बात को Computer के समझने योग्य मशीनी भाषा या Binary Format में Convert करता है। Computer को कोई बात समझाने के लिए उसे एक निश्चित क्रम में सूचनाएं देनी होती हैं, जिन्हें **Instructions** कहा जाता है।

जब किसी काम का एक सुव्यवस्थित परिणाम प्राप्त करने के लिए Computer को दिए जाने वाले विभिन्न प्रकार के Instructions को एक समूह के रूप में व्यवस्थित कर दिया जाता है, तो Instructions के इस समूह को **Program** कहा जाता है। Computer इन दी गई Instructions के अनुसार काम करता है और जिस तरह का परिणाम प्राप्त करने के लिए Program लिखा गया होता है, Computer हमें Program के आधार पर उसी प्रकार का परिणाम प्रदान कर देता है।

Computer में हर Electrical Signal या उसके समूह को Store करके रखने की सुविधा होती है। इन Electrical Signals के समूह को **File** कहते हैं। Computer में जो भी कुछ होता है वह File के

रूप में होता है। Computer में दो तरह की File होती है। पहली वह File होती है जिसमें हम हमारे महत्वपूर्ण Data Store करके रखते हैं। इसे **Data File** कहा जाता है। दूसरी File वह File होती है, जिसमें Computer के लिए वे Instructions होती हैं, जो Computer को बताती हैं कि उसे किसी Data पर किस प्रकार से Processing करके Result Generate करना है। इस दूसरी प्रकार की File को **Program File** कहा जाता है।

हम विभिन्न प्रकार की Computer Languages में Program Files ही Create करते हैं। जब बहुत सारी Program Files मिल कर किसी समस्या का समाधान प्राप्त करवाती हैं, तो उन Program Files के समूह को **Software** कहा जाता है। Computer Software मुख्यतया दो प्रकार के होते हैं:

System Software:

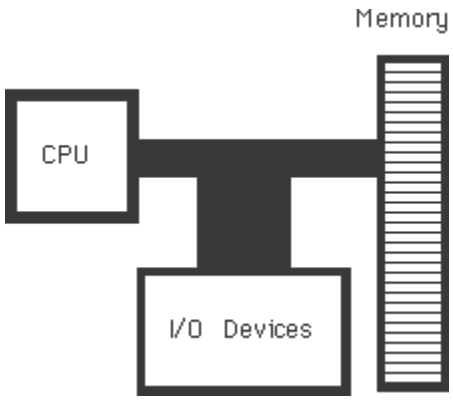
ये Software उन प्रोग्रामों का एक समुह होता हैं जो कम्प्यूटर की Performance को Control करता है। यानी Computer पर किस तरह से एक प्रोग्राम रन होगा और किस तरह से प्रोग्राम Output देगा। किस तरह Hard Disk पर Files Save होंगी, किस तरह पुनः प्राप्त होंगी, आदि। Windows, Unix, Linux, आदि System Software के उदाहरण हैं।

Application Software:

ये Software प्रोग्रामरों द्वारा लिखे जाते हैं व ये Software किसी खास प्रकार की समस्या का समाधान प्राप्त करने के लिए होते हैं। जैसे Tally, MS–Office आदि Application Software के उदाहरण हैं।

Computer Architecture

Computer से अपना मनचाहा काम करवाने के लिए, सबसे पहले हमें Computer के Architecture को समझना होगा। Computer के Architecture को समझे बिना, हम Computer Programming को ठीक से नहीं समझ सकते। Computer System के मुख्य–मुख्य तीन भाग होते हैं–



I/O Devices

वे Devices जिनसे Computer में Data Input किया जाता है और Computer से Data Output में प्राप्त किया जाता है, I/O Devices कहलाती हैं। Keyboard एक Standard Input Device है और Monitor एक Standard Output Device है।

Center Processing Unit (CPU)

यह एक Microprocessor Chip होता है। इसे Computer का दिमाग भी कहा जाता है क्योंकि Computer में जो भी काम होता है, उन सभी कामों को या तो CPU करता है या Computer के अन्य Devices से उन कामों को करवाता है। इसका मुख्य काम विभिन्न प्रकार के Programs को Execute करना होता है। इस CPU में भी निम्न विभाग होते हैं जो अलग–अलग काम करते हैं:

Control Unit

इस Unit का मुख्य काम सारे Computer को Control करना होता है। CPU का ये भाग Computer की आंतरिक प्रक्रियाओं का संचालन करता है। यह Input/Output क्रियाओं को Control करता है, साथ ही ALU व Memory के बीच Data के आदान–प्रदान को निर्देशित करता है।

यह Program को Execute करने के लिए Program के Instructions को Memory से प्राप्त करता है और इन Instructions को Electrical Signals में Convert करके उचित Devices तक पहुंचाता है, जिससे Data पर Processing हो सके। Control Unit ALU को बताता है कि Processing के लिए Data Memory में कहां पर स्थित हैं, Data पर क्या प्रक्रिया करनी है और Processing के बाद Data को वापस Memory में कहां पर Store करना है।

Arithmetic Logic Unit (ALU)

CPU के इस भाग में सभी प्रकार की अंकगणितीय व तार्किक प्रक्रियाएं होती हैं। इस भाग में ऐसा Electronic Circuit होता है जो Binary Arithmetic की गणनाएं करता है। ALU Control Unit से निर्देश या मार्गदर्शन लेता है, Memory से Data प्राप्त करता है और परिणाम को या Processed Data को वापस Memory में ही Store करता है।

Registers

Microprocessor में कुछ ऐसी Memory होती है जो थोडे समय के लिए Data को Store कर सकती है। इन्हें Registers कहा जाता है। Control Unit के निर्देशानुसार जो भी Program Instructions व Data Memory से आते हैं वे ALU में Calculation के लिए इन्हीं Registers में Store रहते हैं। ALU में Processing के बाद वापस ये Data Memory में Store हो जाते हैं।

Memory

Memory Computer की Working Storage या कार्यकारी मेमोरी होती है। यह Computer का सबसे महत्वपूर्ण भाग होता है। इसे RAM कहते हैं। इसी में Process होने वाले Data और Data पर Processing करने के Program Instructions होते हैं, जिन्हें Control Unit ALU में Processing के लिए Registers में भेजता है। Processing के बाद जो सूचनाएं या Processed Data Generate होते हैं, वे भी Memory में ही आकर Store होते हैं।

Memory में Data को संग्रह करने के लिए कई Storage Locations होती हैं। हर Storage Location एक Byte की होती है और हर Storage Location का एक पूर्णांक Number होता है जिसे उस Memory Location का Address कहते हैं।

हर Storage Location की पहचान उसके Address से होती है। 1 Byte की RAM में एक ही Character Store हो सकता है और इसमें सिर्फ एक ही Storage Location हो सकती है। इसी तरह 1 KB की RAM में 1024 Storage Locations हो सकती हैं और इसमें 1024 अक्षर Store हो सकते हैं। जो Memory जितने Byte की होती है उसमें उतने ही Characters Store हो सकते हैं और उसमें उतनी ही Storage Locations हो सकती हैं।

जिस तरह से किसी शहर में ढेर सारे घर होते हैं और हर घर का एक Number होता है। किसी भी घर की पहचान उसके घर के Number से भी हो सकती है। उसी तरह से Memory में भी विभिन्न Storage Cell होते हैं जिनका एक Unique Number होता है। हम किसी भी Storage Cell को उसके Number से पहचान सकते हैं और Access कर सकते हैं। हर Storage Cell के इस Unique Number को उस Storage Cell का Address कहते हैं।

जिस तरह से हम किसी घर में कई तरह के सामान रखते हैं और जरूरत होने पर उस घर से उस सामान को प्राप्त करके काम में ले लेते हैं, उसी तरह से Memory में भी अलग–अलग Storage Cells में हम अपनी जरूरत के अनुसार अलग–अलग मान Store कर सकते हैं और जरूरत पडने पर उस Data को प्राप्त कर के काम में ले सकते हैं।

Types of Programming

Computer एक Digital Machine है। Computer तभी कोई काम कर सकता है जब उसे किसी काम को करने के लिए Program किया गया हो। Programming दो तरह की होती हैः

एक Programming वह होती है जो किसी Computer को काम करने लायक अवस्था में लाने के लिए की जाती है। इस Programming को भी दो भागों में बांटा जा सकता है :

Hardware Programming

इस Programming के अन्तर्गत Computer के Hardware यानी Computer के Motherboard पर लगाए गए विभिन्न प्रकार के Chips व Computer से जुडे हुए अन्य विभिन्न प्रकार के Peripherals जैसे कि Keyboard, Mouse, Speaker, Monitor, Hard Disk, Floppy Disk, CD Drive आदि को Check करने व Control करने के लिए हर Mother Board पर एक **BIOS** Chip लगाई जाती है। इस BIOS Chip का मुख्य काम Computer को ON करते ही विभिन्न प्रकार के Devices को Check करना होता है। यदि Computer के साथ जुडी हुई कोई Device ढंग से काम नहीं कर रही है, तो BIOS User को विभिन्न प्रकार की Error Messages देता है।

BIOS Chip के अन्दर ही प्रोग्राम को लिखने का काम BIOS बनाने वाली Company करती है। इसे Hard Core Programming या Firmware कहा जाता है। Hardware Programming में Chip को बनाते समय ही उसमें Programming कर दी जाती है। किसी भी Computer के Motherboard पर लगी BIOS Chip यदि खराब हो जाए, तो Computer किसी भी हालत में काम करने लायक अवस्था में नहीं आ सकता यानी Computer कभी Boot नहीं होता।

Software Programming

Computer को काम करने लायक अवस्था में लाने के लिए जिस Software को बनाया जाता है, उसे Operating System Software कहा जाता है। BIOS Chip का काम पूरा होने के बाद Computer का पूरा Control Operating System Software के पास आ जाता है। Computer के पास BIOS से Controlling आने के बाद सबसे पहले Memory में Load होने वाला Software Operating System Software ही होता है। इसे **Master Software** भी कहते हैं।

आज विभिन्न प्रकार के Operating System Software बन चुके हैं जैसे DOS, Windows, OS/2, WRAP, Unix, Linux आदि। इन सभी Software का मुख्य काम Computer को Boot करके User के काम करने योग्य अवस्था में लाना होता है।

दूसरी Programming वह Programming होती है, जिससे Computer हमारी बात को समझता है और हमारी इच्छानुसार काम करके हमें परिणाम प्रदान करता है। इन्हें **Application Software** कहा जाता है।

हम किसी भी Operating System के लिए किसी भी भाषा में जब कोई Program लिखते हैं, तो वास्तव में हम Application Software ही लिख रहे होते हैं। Application Software का मुख्य काम किसी विशेष समस्या का समाधान प्रदान करना होता है। MS-Office, Corel-Draw, PageMaker, Photoshop आदि Application Software के उदाहरण हैं, जो हमें किसी विशेष समस्या का समाधान प्रदान करते हैं। जैसे यदि हमें Photo Editing से सम्बंधित कोई काम करना हो, तो हम Photoshop जैसे किसी Application Software को उपयोग में लेते हैं।

LANGUAGE INTRODUCTION

Language Introduction

भाषा, दो व्यक्तियों के बीच संवाद, भावनाओं या विचारों के आदान–प्रदान का माध्यम प्रदान करती है। हम लोगों तक अपने विचार पहुंचा सकें व अन्य लोगों के विचारों का लाभ प्राप्त कर सकें इसके लिए जरूरी है कि संवाद स्थापित करने वाले दोनों व्यक्तियों के बीच संवाद का माध्यम समान हो। यही संवाद का माध्यम भाषा कहलाती है। अलग–अलग स्थान, राज्य, देश, परिस्थितियों के अनुसार भाषा भी बदलती रहती हैं, लेकिन सभी भाषाओं का मकसद संदेशों या सूचनाओं का आदान प्रदान करना ही होता है।

ठीक इसी तरह कम्प्यूटर की भी अपनी कई भाषाएं है, जो जरूरत व उपयोग के अनुसार विकसित की गई हैं। हम जानते हैं, कि कम्प्यूटर एक इलेक्ट्रॉनिक मशीन मात्र है। ये हम सजीवों की तरह सोंच विचार नहीं कर सकता है और ना ही हमारी तरह इनकी अपनी कोई भाषा है, जिससे हम इनसे सम्बंध बना कर सूचनाओं का लेन–देन कर सकें। इसलिए कम्प्यूटर को उपयोग में लेने के लिए एक ऐसी भाषा की जरूरत होती है, जिससे हम हमारी भाषा में कम्प्यूटर को सूचनाएं दें व कम्प्यूटर उसे उसकी मशीनी भाषा में समझे और हमारी चाही गई सूचना या परिणाम को हमें हमारी भाषा में दे ताकि हम उसे हमारी भाषा में समझ सकें।

Level of Computer Languages

कम्प्यूटर मुख्यतः एक ही भाषा यानी मशीनी भाषा को ही समझता है। फिर भी मोटे तौर पर कम्प्यूटर भाषा को निम्नानुसार तीन भागों में बांटा गया है। ये High Level Languages हैं, जिनमें एक ऐसा Software या Program होता है जो इन High Level Languages के Program Codes को मशीनी भाषा के Low Level Codes में Convert करने का काम करता है, जिन्हें Computer समझता है।

Low Level Language or Machine Language

इसे मशीनी भाषा भी कहते हैं। यह भाषा केवल बाइनरी कोड के अनुसार लिखनी होती है, इसलिए ये भाषा केवल वे ही लोग उपयोग में ले सकते हैं जो कम्प्युटर की सारी आंतरिक संरचना को जानते हों साथ ही इस भाषा में लिखे प्रोग्राम केवल उसी कम्प्यूटर पर चलते हैं, जिस पर ये लिखे जाते हैं। यह एक बहुत ही कठिन भाषा होती है।

Middle Level or Assembly Language

इसे असेम्बली भाषा भी कहते हैं। इस भाषा में सामान्य अंग्रेजी के शब्दों को उपयोग में लेकर प्रोग्राम लिखा जाता है इसलिए ये भाषा उपयोग में मशीनी भाषा से सरल होती है, लेकिन फिर भी काफी जटिल होती है। इसमें एक असेम्बलर होता है, जो सामान्य अंग्रेजी के शब्दों को मशीनी भाषा में बदलने का काम करता है ताकि कम्प्यूटर उसे समझ सके। इस भाषा में भी प्रोग्राम बनाने वाले प्रोग्रामर

को कम्प्यूटर हार्डवेयर का सम्पूर्ण ज्ञान होना जरूरी होता है व ये प्रोग्राम भी उसी कम्प्यूटर पर Run हैं, जिस पर इन्हे लिखा गया हो।

High Level Language

ये हमारे आज के वातावरण में उपयोग में आने वाली भाषाएं हैं। ये भाषाए इतनी सरल हैं कि कोई भी सामान्य व्यक्ति इनमें प्रोग्राम बना सकता है। इसमें सारे के सारे कोड अंग्रेजी में लिखे जाते हैं व इसमें एक कम्पायलर होता है जो सीधे ही प्रोग्राम को मशीनी कोड में बदल देता है।

Assembler

Assembly Language में लिखे प्रोग्राम को मशीनी भाषा में बदलने का काम **Assembler** करता है। ये एक ऐसा Software होता है, जो किसी Text File में लिखे गए विभिन्न Assembly Codes को Computer की मशीनी भाषा में Convert करके Computer के CPU पर Process करता है। Computer का CPU उन Converted Codes को समझता है और हमें हमारा वांछित परिणाम उस भाषा में प्रदान करता है, जिस भाषा को हम समझ सकते हैं यानी CPU हमें सामान्य English भाषा में Processed Results प्रदान करता है।

Compiler and Interpreter

Compiler व Interpreter भी High Level Program Codes को मशीनी भाषा में बदलने का काम करते हैं लेकिन दोनों के काम करने के तरीके में कुछ अन्तर हैं। Compiler पूरे प्रोग्राम को एक ही बार में मशीनी भाषा में बदल देता है व सभी Errors को Debug करने के बाद एक Executable Program File Provide करता है, जो कि एक Machine Language Code File होती है। इस Machine Language Code File को फिर से Compile करने की जरूरत नहीं होती है। जबकि Interpreter प्रोग्राम की हर लाइन को हर बार मशीनी कोड में बदलता है, जिससे एक Interpreted Program को हर बार Run करने के लिए Interpret करना जरूरी होता है। HTML Code File Interpreted Program का एक उदाहरण है, जिसे हर बार Run होने के लिए Web browser Interpreter की जरूरत होती है।

Similarities between Real Word and Computer Program

प्रोग्राम को हम हर रोज के हमारे दैनिक जीवन के कामों से भी समझ सकते हैं। जिस तरह हमें कोई सामान्य सा काम के लिए भी एक निश्चित क्रम का पालन करना पडता है, उसी तरह कम्प्यूटर को भी एक निश्चित क्रम में सूचनाएं देनी होती हैं, कि किस काम के बाद क्या काम करना है। ताकि एक निश्चित समाधान या मनचाहा परिणाम प्राप्त किया जा सके। उदाहरण के लिए, माना हमें कुछ सामान खरीदने के लिए बाजार जाना हैं, तो हमें निम्न क्रम में अपना काम करना पडेगा :

- किस समय बाजार जाए ताकि अधिकतर दुकाने खुली हों और भीड कम हो ?
- किस दिन सस्ता सामान मिल सकेगा ?
- क्या–क्या खरीदना है ?
- कितने रूपयों की जरूरत होगी ?
- किस सवारी से जाना है ?
- किसके साथ बाजार जाना है ?
- खरीददारी के साथ और क्या काम किया जा सकता है ? आदि – आदि

ठीक इसी तरह से ''सी'' Language में भी प्रोग्राम बनाया जाता है। यानी कामों का एक सुव्यवस्थित समूह Create किया जाता है और उस समूह को Computer के समझने योग्य Programming Language में Coding के रूप में एक File में लिख दिया जाता है। इस File को Program की **Source File** कहते हैं।

जिस File में Computer के समझने योग्य Coding के रूप में विभिन्न Steps या Instructions को लिखे गए होते हैं, उस File को Compile किया जाता है। Source File को Compile करने पर एक नई File बनती है, जिसके Instructions को Computer का CPU समझ सकता है। इस Compiled File को **Executable File** या **Exe File** कहा जाता है, क्योंकि Compiling के बाद Create होने वाली इस नई File का Extension **.EXE** होता है।

अब हमें जब भी वह काम करना होता है, जिसके लिए हमने Program लिखा है, तो हमें Source File को वापस से Compile करने की जरूरत नहीं होती है। हमें केवल उस Create होने वाली नई Executable File को ही Run करना होता है। इस File में CPU को जो कुछ करना है उसकी Instructions होती हैं जिन्हे CPU समझ सकता है। इस प्रकार से Computer में एक Program Create होता है।

इस पूरे Discussion के आधार पर यदि हम किसी Computer Program की परिभाषा देना चाहें तो ये कह सकते हैं कि **Computer Instructions** का एक ऐसा सुव्यवस्थित क्रम, जिससे Computer द्वारा किसी समस्या का उचित समाधान प्राप्त हो सके, **Program** कहलाता है।

Steps of Program

1 **(Problem Definition)** प्राग्राम परिभाषण

इस चरण में उस समस्या को पूरी तरह से समझना होता है, जिसका प्रोग्राम बना कर कम्प्यूटर से समाधान प्राप्त करना हैं। यानी प्रोग्राम के द्वारा हमें क्या प्राप्त परिणाम करना है, यह निष्कर्ष निकालना होता है।

सारांश :– क्या परिणाम प्राप्त करना हैं ?

2 (Problem Design) प्रोग्राम डिजाइन

इस चरण में समस्या को कई भागों में बांट कर उसे बीजगणितीय एल्गोरिद्म के अनुसार लिख लिया जाता है। एल्गोरिद्म लिखने के लिए फ्लोचार्ट आदि को उपयोग में लिया जाता है।

सारांश :– कैसा परिणाम प्राप्त करना है ?

3 (Program Coding) कोडिंग

इस चरण में हाई लेवल भाषा के कोडों के अनुसार एल्गोरिद्म व फ्लोचार्ट की मदद से प्रोग्राम की कोडिंग की जाती है।

सारांश :– कब क्या होगा जब User इसे उपयोग में लेगा ?

4 (Program Execution) प्रोग्राम को **Execute** करना

इस चरण में बनाए गए प्रोग्राम को चलाया जाता है।

5 (Program Debugging) डीबगिंग

जब प्रोग्राम को बनाया जाता है, तब कई तरह की गलतियां रह जाती हैं। जिससे जब प्रोग्राम को चलाया जाता है तब या तो प्रोग्राम रन नहीं होता या फिर सही परिणाम प्राप्त नहीं होता है। जब प्रोग्राम को कम्पाइल किया जाता है तो कम्पायलर में एक डीबगर होता है, जो प्रोग्राम में जिस जगह पर गलती होती है, वहीं पर आकर रूक जाता है। हम वहां पर होने वाली बग को सही करके प्रोग्राम को पुनः रन करते हैं। प्रोग्राम मे होने वाली गलतियों को ढूंढना व उन्हे सही करना ही डीबगिंग कहलाता हैं।

सारांश :– प्रोग्राम की किसी भी तरह की व्याकरण सम्बंधी या तर्क सम्बंधी गलती को खोजना व उसे संसोधित करके प्रोग्राम को सही करना।

6 (Program Testing) प्रोग्राम टेस्टिंग

कई बार प्रोग्राम पूरी तरह सही रन होता है, लेकिन फिर भी उसमें गलती होती है। इसे तार्किक गलती कहते हैं। इस प्रकार की गलती से हमें वांछित सही परिणाम प्राप्त नहीं होता है। इसे सुधारने के लिए प्रोग्राम से ऐसी समस्याओं का हल मांगा जाता है, जिसका परिणाम हमें पहले से ही पता होता है। ऐसा करने से यदि प्रोग्राम में कहीं पर तार्किक कमी हो तो पता चल जाता है। इस प्रक्रिया को प्रोग्राम टेस्टिंग करना कहते हैं।

7 (Program Documentation) प्रोग्राम विवरण

कई बार प्रोग्राम इतने बडे व जटिल हो जाते हैं कि कब कहां और क्या होना है और कौनसा प्रोग्राम क्यों लिखा गया था इसका पता ही नहीं चल पाता है। इस तरह की समस्याओं से बचने के लिए प्रोग्राम में कई जगहों पर ऐसी टिप्पणीयां डाल दी जाती हैं, जिससे पता चल सके कि प्रोग्राम क्या है व वह प्रोग्राम किसलिए लिखा गया है।

Characteristics of a Good Program

प्रोग्राम लिखते समय हमें कई बिंदुओं को ध्यान में रखना होता है। इसमें से कुछ खास बिन्दु निम्नानुसार हैं:

1 (Reliability) विश्वसनीयता

यह जरूरी है कि प्रोग्राम बिना किसी व्यवधान के वही काम करे जिसके लिए उसे बनाया गया है। माना कि हमने एक ऐसा प्रोग्राम बनाया जिसमें किसी भिन्नात्मक संख्या का हर कोई वेरियेबल है, जो घटते–घटते अन्त में शून्य हो जाता है। ऐसी दशा में संख्या का भागफल अनन्त हो जाएगा क्योंकि किसी भी संख्या में शून्य का भाग देने पर भागफल अनन्त प्राप्त होता है, जिससे प्रोग्राम सही परिणाम नहीं देगा। इस प्रकार की गलतियों का ध्यान रखना चाहिये।

2 (Flexibility) लचीलापन

प्रोग्राम इस तरह का होना चाहिये कि जब भी भविष्य में कभी जरूरत पडे, तो उसमें नया कुछ जोडा जा सके या अनावश्यक चीजों को हटाया जा सके। इसे प्रोग्राम की **Maintainability** कहा जाता है। जैसे कि किसी प्रोग्राम में 20 वर्षों का ब्याज निकालने की व्यवस्था है, तो उसमें यह ऐसी सुविधा होनी चाहिये कि आवश्यकता होने पर कुछ फेर बदल करके 25 वर्षों का ब्याज भी निकाला जा सके।

3 (Portability)

प्रोग्राम इस तरह लिखा होना चाहिये कि एक Computer पर Develop किया गया Program बिना फिर से Compile किए हुए किसी दूसरे Computer पर भी आसानी से Execute हो सके।

4 (Readability) सुपाठ्यता

प्रोग्राम में जगह–जगह पर कई ऐसी टिप्पणीयां होनी चाहिये जिससे प्रोग्राम का **Flow** व प्रोग्राम का उद्देश्य पता चलता रहे।

5 (Performance)

प्रोग्राम द्वारा कम से कम समय में अच्छा से अच्छा परिणाम प्राप्त होना चाहिये।

Problem – Doing Something

Computer द्वारा हम किसी ना किसी प्रकार की समस्या का समाधान प्राप्त करने के लिए ही विभिन्न प्रकार के Programs लिखते हैं। इसलिए सबसे पहले हमें यही तय करना होगा कि आखिर हम Computer के संदर्भ में किस बात को एक समस्या के रूप में देख सकते हैं ?

यदि बिल्कुल ही सरल शब्दों में किसी समस्या को परिभाषित करें, तो Computer पर हम जिस किसी भी काम को Perform करके किसी प्रकार का कोई Result प्राप्त करना चाहते हैं, हम उस काम को समस्या के रूप में देख सकते हैं।

उदाहरण के लिए दो संख्याओं का योग **करना**, किसी परिणाम को Computer के Monitor पर Display **करना**, किसी भी प्रकार की कोई Calculation या Comparison **करना** आदि इन सभी कामों को हम समस्या के रूप में देख सकते हैं। यानी हम जो कुछ भी **करना** चाहते हैं, वह सबकुछ Computer के लिए एक समस्या ही है।

Algorithm – List of Sequential Steps to Solve a Problem

हम हमारे दैनिक जीवन में जिस किसी भी काम को भी करते हैं, उस काम को Problem कह सकते हैं। हर Problem को Solve करने का एक निश्चित क्रम होता है और इस निश्चित क्रम के अन्तर्गत हमें विभिन्न प्रकार के Steps Use करने होते हैं। उदाहरण के लिए मानलो कि हमें किसी को Phone **करना** है। ये भी एक तरह की समस्या ही है क्योंकि हमें कुछ **करना** है। अब Phone करने के लिए हमें निम्न काम करने होते हैं:

1. सबसे पहले हम Phone को इस बात के लिए Check करेंगे, कि Phone चालू है या नहीं। यानी Dial Tone आ रही है या नहीं।
2. यदि Dial Tone आ रही है, तो हमें उस व्यक्ति का Phone Number Dial करना होता है, जिससे हम बात करना चाहते हैं।
3. Phone Number Dial करने के बाद हमें Target व्यक्ति के Phone पर Bell जाने का इन्तजार करना होगा। यदि Bell जाती है, तो Target व्यक्ति Phone उठाएगा और बात हो जाएगी।

इन Steps के समूह से हम समझ सकते हैं कि हमें Phone करने जैसी मामूली सी समस्या को सुलझाने के लिए भी एक निश्चित क्रम का पालन करना जरूरी होता है, साथ ही सभी जरूरी Steps Follow करने भी जरूरी होते हैं। ना ही हम इन Steps के क्रम को Change कर सकते हैं और ना ही हम किसी Step को छोड सकते हैं। यदि हम इन दोनों में से किसी भी एक बात को Neglect करते हैं, तो हम Target व्यक्ति से बात नहीं कर सकते हैं, यानी समस्या का Solution प्राप्त नहीं कर सकते हैं।

इस उदाहरण का सारांश ये है कि किसी भी समस्या का एक निश्चित व उचित समाधान प्राप्त करने के लिए हमें उस समस्या को विभिन्न प्रकार के Steps के एक समूह के रूप में Define करना होता है, जो कि एक निश्चित क्रम में होते हैं। Steps के इस समूह को ही **Algorithm** कहा जाता है।

दूसरे शब्दों में कहें तो किसी भी समस्या के एक निश्चित समाधान को प्राप्त करने के लिए अनुक्रमिक व चरणबद्ध रूप में अपनाई जाने वाली लिखित प्रक्रिया को हम **एल्गोरिद्म** कहते हैं।

उदाहरण के लिए मानलो कि हम दो संख्याओं **A** व **B** को जोड कर उसका परिणाम **C** में प्राप्त **करना** चाहते हैं और फिर **C** के मान को Monitor पर Display करना चाहते हैं। यानी हमें **C = A + B**

करना है। इस काम को पूरा करने के लिए या इस समस्या को सुलझाने के लिए हमें निम्नानुसार क्रम का पालन करना होता है:

हल :–

चरण 1 प्रक्रिया का प्रारम्भ।

चरण 2 वेरिएबल A का मान पढना।

चरण 3 वेरिएबल B का मान पढना।

चरण 4 A व B के मान का योग निकालना ।

चरण 5 मान A व B के योगफल को Variable C के स्थान पर रखना।

चरण 6 C के मान को प्रिंट करना।

चरण 7 प्रक्रिया का अंत करना।

History of Programming Language "C"

इस भाषा का विकास होने से पहले जितने भी Program बनाए जाते थे, वे सभी Assembly Language में बनाए जाते थे। Assembly Language में बनाए गए Programs की Speed काफी ज्यादा होती है, लेकिन इसकी एक कमी भी है। Assembly Language में Develop किया गया Program उसी Computer पर Execute होता है, जिस पर उसे Develop किया गया होता है।

इसलिए एक ऐसी Programming Language की आवश्यकता हुई, जो कि Portable हो। इस जरूरत के आधार पर सन् 1960 में केम्ब्रिज यूनिवर्सिटी ने एक कम्प्यूटर प्रोग्रामिंग भाषा का विकास किया, जिसका नाम "BASIC COMBINED PROGRAMMING LANGUAGE" यानी **BCPL** रखा गया। सन् 1970 में केन थॉम्पसन ने इसमें कुछ परिवर्तन किये व सामान्य बोलचाल में इसे "B" भाषा कहा। "C" का विकास अमेरिका में सन् 1972 में हुआ। AT &T Laboratory के कम्प्यूटर वैज्ञानिक डेनिस रिची ने इस का विकास किया था।

''सी'' एक शक्तिशाली भाषा है जिसमें हम एप्लीकेशन सॉफ्टवेयर व सिस्टम सॉफ्टवेयर दोनों तरह के सॉफ्टवेयर बना सकते हैं। इसमें सामान्य अंग्रेजी शब्दों के माध्यम से प्रोग्राम बनाए जाते हैं, जो कि समझने व बनाने में आसान होते हैं। ''सी'' एक हाई लेवल Structured Programming Language भाषा हैं, यानी सूचनाओं के एक निश्चित क्रम में Program Run होता है।

Characteristics of "C"

''सी'' अन्य कई भाषाओं से काफी सरल है। अन्य हाई लेवल भाषाओं की तुलना में ''सी'' काफी लचीली भाषा है। ''सी'' ही एक ऐसी भाषा है, जिसमें कम्प्यूटर के हार्ड वेयर के साथ भी काम किया जा सकता है। इसके द्वारा मेमोरी मेनेजमेन्ट किया जा सकता है। सबसे बडी खासियत ''सी'' की पोर्टेबिलिटी है।

यानी ''सी'' भाषा में लिखे गए प्रोग्राम किसी भी अन्य कम्प्यूटर वातावरण में चल सकते हैं। ''सी'' एक फंक्शनल भाषा है यानी इसमें सभी काम विभिन्न प्रकार के फंक्शनस् को यूज करके किया जाता है। ''सी'' में कोई इनपुट आउटपुट ऑपरेशन नहीं है। ''सी'' कम्पाइलर सभी इनपुट आउटपुट का काम लाइब्रेरी फंक्शन के द्वारा करता है।

Block Structure of "C" Programs

Documentation Section
Link Section
Definition Section
Global Declaration Section

Main() Function Section
{
Declaration Part
Executable Part
}

Sub Program Section Function 1 Function 2 . . . Function n

Layout Structure of "C" Programs

```
1       /* Comment about the Program */
2       Including The Header Files
3       Global Variables Declaration
4       Main()
5       {
6               Local Variables Declaration
7               Necessary Statements
8       }
9       Sub Program Functions
    Function 1
    Function 2
        ;
    Function n
```

1 Documentation Section

प्रोग्राम के इस भाग में हम प्रोग्राम से सम्बन्धित कुछ बिन्दु टिप्पणी के रूप में लिखते हैं, ताकि प्रोग्राम किस कारण से बनाया गया है और प्रोग्राम की विशेषता क्या है, ये बताया जा सके।

2 Link Section

यहां पर हम ''सी'' प्रोग्राम की उन **हेडर फाइलों** को डिक्लेयर करते हैं, जिनकी हमारे प्रोग्राम में आवश्यकता है।

3 Definition Section

यहां उन वेरियेबल्स को डिफाईन किया जाता हैं जिनका प्रोग्राम में सीधे ही उपयोग हो सकता हो। ये एक तरह से स्थिरांक होता हैं। इसे ग्लोबल कोंस्टेंट भी कह सकते हैं।

4 Global Declaration Section

जिस किसी वेरियेबल को इस स्थान पर डिक्लेयर कर दिया जाता है, उस वेरियेबल को प्रोग्राम में कहीं भी उपयोग में लिया जा सकता हैं।

5 Main() Function Section

यह फंक्शन हर ''सी'' प्रोग्राम में होता है। कम्पाईल करते समय Program Control हमेशा main() Function को ही ढूंढता है। हर ''सी'' प्रोग्राम में सिर्फ एक ही main() Function हो सकता है व हर ''सी'' प्रोग्राम में main() Function का होना जरूरी होता है क्योंकि Program का Execution हमेंशा main() Function से ही शुरू होता है।

6 { Opening Parenthesis

main() Function मिलने के बाद प्रोग्राम का एक्जीक्यूशन इसी मंझले कोष्ठक से शुरू होता है।

7 Declaration Part

प्रोग्राम में काम आने वाले सभी वेरियेबल्स्, कोंस्टेंट, एरे आदि को यहीं पर डिक्लेयर करना होता है। यहां पर हम जिसे भी डिक्लेयर करते हैं, उसके लिए ''सी'' प्रोग्राम Execution के समय मेमोरी में जगह बना देता है, जिन्हें बाद में अपनी आवश्यकता के अनुसार उपयोग में लिया जाता है।

8 Executable Part

यहां पर प्रोग्राम के वे सभी स्टेटमेंट्स होते हैं जिनके द्वारा हम प्रोग्राम से कोई परिणाम प्राप्त करना चाहते हैं। यही वह भाग होता है जहां से User के लिए Interface का काम शुरू होता हैं।

9 } Closing Parenthesis

प्रोग्राम में दूसरे मंझले कोष्ठक का प्रयोग वहां करते हैं, जहां पर प्रोग्राम का अन्त करना होता है।

Sub Program Section

Function 1;
Function 2;
. . .
. . .
Function n;

प्रोग्राम के इस भाग में यूजर डिफाइन फंक्शन होते हैं। एक main() प्रोग्राम में main() Function तो एक ही होता है लेकिन User Defined Function आवश्यकता के अनुसार कई हो सकते हैं।

Coding Structure of "C" Programs

सबसे पहले किसी प्रोग्राम की कोडिंग की जाती है। फिर प्रोग्राम को कम्पाइल किया जाता है। कम्पाइल करने से प्रोग्राम की हाई लेवल के कोड मशीनी भाषा के बाइनरी डिजिटस् में बदल जाते हैं, जिन्हें हमारा Computer समझ सकता है। हम ''सी'' प्रोग्राम के एक्जीक्युशन को एक ब्लॉक डायग्राम या Flow Chart से समझाने की कोशिश कर रहे हैं।

सबसे पहले कम्प्यूटर चालू करेंगे और ''सी'' भाषा के कोडों को लिख कर प्रोग्राम बनाएंगे। इसे **Source Program** कहते हैं। प्रोग्राम बनाने के बाद इसकी किसी भी प्रकार की व्याकरण सम्बंधी गलती को Edit Source Program Block में Edit करके सही करते हैं।

अब ''सी'' कम्पाइलर द्वारा प्रोग्राम को कम्पाइल करते हैं, जिससे प्रोग्राम को कम्प्यूटर अपनी मशीनी भाषा में समझ सके। यदि इस प्रोग्राम में कोई अन्य वाक्य रचना सम्बंधी गलती हो, तो प्रोग्राम कंट्रोल पुनः सभी गलतियों के साथ Source Editing के लिए उसी Edit Source Program Block में चला जाता है।

जब प्रोग्राम में किसी भी प्रकार की कोई व्याकरण सम्बंधी गलती नहीं रह जाती है, तब Program Control उन **System Library Files** को प्रोग्राम में लिंक करता है, जिनके Function Program में Use हुए हैं।

जैसे Input/Output के सारे Functions **stdio.h** नाम की Header File में Store रहते हैं, इसलिए I/O की सुविधा प्राप्त करने के लिए इस Header File को हर C Program में Include किया जाता है।

जब Program Control सभी आवश्यक Header Files को Program से Link कर देता है। फिर अगली Stage में यूजर से Data Input करवाया जाता है व प्रोग्राम Execute होता रहता है। अब यदि किसी प्रकार की तार्किक गलती हो तो वह गलती अगले प्रोसेस बॉक्स में पकड में आती है।

यदि गलती है, तो प्रोग्राम Control पुनः Edit Source Program Block में पहुंच जाता है, और सारी की सारी प्रक्रिया पुनः प्रोग्राम को डिबग करने में अपनाई जाती है। लेकिन यदि प्रोग्राम में कोई Error नहीं हो तो प्रोग्राम Correct Output देता है और समाप्त हो जाता है। इस तरह पूरा प्रोग्राम Step-By-Step Execute होता है।

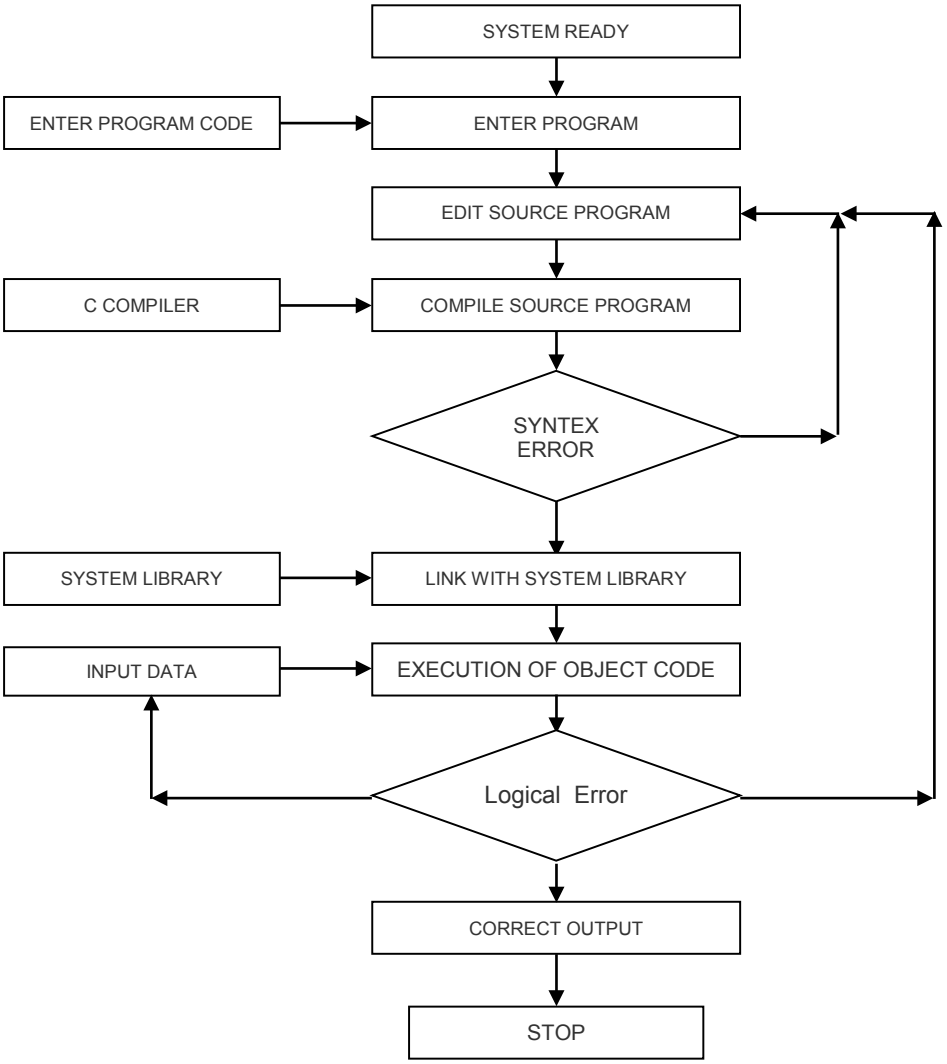
main() Function

```
{
        Function Body ;
}
```

यह किसी भी प्रोग्राम का एक अनिवार्य हिस्सा है। जब भी कोई प्रोग्राम कम्पाइल करते हैं तो कम्पाइलर सर्वप्रथम main() Function को ढूंढता है और इसके मंझले कोष्ठक से प्रोग्राम का Execution शुरू करता है। सभी Executables Code इन्ही मंझले कोष्ठकों के बीच लिखे जाते हैं।

किसी भी Function की शुरूआत व अन्त के Statements इन्हीं मंझले कोष्ठकों के बीच लिखे जाते हैं, फिर चाहे ये User Defined Functions हों या main() Function, Program के हर Statement का अन्त '' ; '' सेमीकॉलन के चिन्ह द्वारा ही होता है।

Program Flow



Functions – Pre-Defined and User-Defined

"C" भाषा एक **Functional Programming Language** है। जब हम इस भाषा का प्रयोग करके किसी समस्या का समाधान प्राप्त करना चाहते हैं, तब हमें उस समस्या को छोटे–छोटे हिस्सों में बांटना होता है और उन सभी हिस्सों को अलग–अलग Solve करके अन्त में सभी हिस्सों को जोडना होता है।

किसी समस्या से सम्बंधित इन विभिन्न प्रकार के छोटे–छोटे हिस्सों को **Function** कहा जाता है। ये Function किसी एक काम को पूरी तरह से पूरा करते हैं और केवल एक ही काम को पूरा करते हैं। यानी हर Function अपने आप में केवल एक ही काम परिपूर्ण तरीके से पूरा करता है।

उदाहरण के लिए जो Function Keyboard से Input लेने का काम करता है, वह Function केवल Keyboard से Input लेने का ही काम करेगा और जो Function किसी Data को Monitor पर Display करने के लिए लिखा गया है, वह Function Data को केवल Screen पर Display करने का काम ही करेगा।

"C" Language में दो तरह के Functions होते हैः

1 जो Functions हमें Directly Use करने के लिए पहले से ही प्राप्त होते हैं, उन्हें **Pre-Defined** या **Built-In Functions** कहा जाता है। उदाहरण के लिए *printf(), clrscr(), getch()* आदि Functions हमें पहले से ही प्राप्त हैं। इन्हें Use करने के लिए हमें केवल उन Header Files को अपने Source Program में Include करना होता है, जिनमें इन Functions को Define किया गया होता है। जब हम किसी Predefined Function को अपने Source Program में Use करते हैं, तो इस प्रक्रिया को Function Call करना भी कहा जाता है।

2 दूसरे प्रकार के Functions वे Functions होते हैं, जिन्हें Programmer अपनी जरूरत के आधार पर Develop करता है। जिन Functions को एक Programmer स्वयं Create करके Use करता है, उन Functions को **User-Defined Functions** कहते हैं। User-Defined Functions बनाना एक Programmer की इच्छा पर निर्भर करता है।

   यदि Programmer चाहे, तो वह सभी प्रकार के कामों को बिना किसी प्रकार का User-Defined Function Create किए हुए भी पूरा कर सकता है। लेकिन Functions Create करने से Program की जटिलता में कमी आ जाती है और Program को Debug करना सरल होता है।

   चूंकि **main**() Function भी एक Programmer किसी समस्या का समाधान प्राप्त करने के लिए बनाता है, इसलिए main() Program को भी User-Defined Function ही कहा जाता है।

लेकिन ये एक ऐसा Function होता है, जिसे बनाना जरूरी होता है। यही वह Function होता है, जहां से Compiler Program को Execute करना शुरू करता है।

#include<Header File>

''सी'' भाषा में विभिन्न प्रकार के कामों को पूरा करने के लिए फंक्शनों की अपनी एक पूरी लाईब्रेरी है, जिसमें ढेर सारे **Built–In** Functions हैं। विभिन्न प्रकार के Functions को उनके काम करने की प्रकृति के आधार पर विभिन्न प्रकार की Files में Define या परिभाषित किया गया है। Functions की इन Files को **"C"** भाषा में **Header File** कहा जाता है।

हम जिस किसी भी Function को Use करना चाहते हैं, हमें उससे सम्बधिंत Header File को **#include** शब्द के साथ प्रोग्राम में जोडना पडता है। जैसे Input/Output से सम्बन्धित सारे Functions **stdio.h** नाम की Header File में होते हैं।

अतः हमें अपने हर सी प्रोग्राम में इस Header File को **#include<stdio.h>** Code द्वारा Link करना जरूरी होता है। यदि हम ऐसा नहीं करते हैं, तो हमें Input व Output की सुविधा प्राप्त नहीं होती है। यानी इस Header File को अपने Program में Include किए बिना हम हमारे Program में Keyboard से Input नहीं ले सकते हैं Monitor पर Output को Display नहीं कर सकते हैं।

इसी तरह से हमें आउटपुट स्क्रीन पर दिखाई दे रहे पिछले Program के विभिन्न Statements को साफ करके Screen को Clear करना है, तो clrscr() Function को Use करना होता है, जो कि **conio.h** नाम की Header File में Defined है, अतः हमें हमारे प्रोग्राम में इस Header File को **#include<conio.h>** Code द्वारा Link करना पडेगा।

Header Files को Header File इसलिए कहा जाता है, क्योंकि ये Files किसी भी Source File के Head में यानी सबसे Top पर व सबसे पहले Include की जाती हैं। किसी भी Header File को प्रोग्राम में जोडने के लिए **#** के साथ **include** Keyword लगाया जाता है। फिर < > के चिन्हों के बीच में उस Header File का नाम लिखा जाता है, जिसे प्रोग्राम में जोडना होता है। इनको Declare करने का Syntax निम्नानुसार होता है–

```
Syntax :    #include <header file name.h>
जैसे   :–   #include <stdio.h>
            #include <conio.h>
```

#define

ये एक **Macro** Define करने का काम करता है। इसका उपयोग **Constant Global Variables Define** करने में किया जाता है। लेकिन इसका उपयोग इतना ही नहीं है। आगे इसके कई उपयोग बताए जाएंगे जो प्रोग्राम Development में काफी मदद करते हैं व प्रोग्राम को अधिक विश्वसनीय व व्यावहारिक बनाने में मददगार होते हैं।

Syntax	:	#define Constant Name	Constant Value	
जैसे	:	**#define**	**pi**	**3.142857**

ध्यान दें कि स्थिरांक के नाम व उसके मान के बीच किसी प्रकार का कोई चिन्ह नहीं होता है।

जब हम Computer में कोई Program बना कर उस Program के आधार पर किसी समस्या का कोई समाधान प्राप्त करना चाहते हैं, तब हम देखते हैं कि हर Computer Program के हमेंशा तीन हिस्से होते हैं, जिन्हें **Input, Process** व **Output** कहा जाता है।

Input Section

Program के Input Section में Program को Use करने वाला User समस्या से सम्बंधित विभिन्न प्रकार के Row Data Input करता है। इन Row Data के आधार पर ही Program अपना आगे का काम सम्पन्न करके कोई Meaningful Result प्रदान करता है। इस Section में User द्वारा Input किए गए विभिन्न प्रकार के मानों को Computer की Memory में Store करने के लिए सभी Data को Memory Allot किया जाता है। User जो भी Data Input करता है, वे सभी Data उनसे सम्बंधित Memory Block में Store हो जाते हैं।

उदाहरण के लिए यदि दो संख्याओं को जोडने का Program हो, तो इस Section में कुल तीन Memory Block Allot किए जाते हैं। दो Memory Block दो संख्याओं को Store करने के लिए होते हैं और तीसरा Memory Block उन संख्याओं को जोडने से प्राप्त होने वाले परिणाम को Store करने के लिए होता है।

Process Section

इस Section में समस्या से सम्बंधित Input किए गए विभिन्न प्रकार के Data पर विभिन्न प्रकार के Operations Perform करके उचित Result Generate किया जाता है। उदाहरण लिए यदि दो संख्याओं को जोडने का Program हो, तो दोनों संख्याओं को जोडने का काम इस Section में ही किया जाता है।

Output Section

समस्या से सम्बंधित Input किए गए Data पर Required Operations Perform करने के बाद जो Results Generate होते हैं, उन Results को Monitor पर Display करने या Printer पर Print करने का काम इस Section में किया जाता है।

उदाहरण के लिए दो संख्याओं को जोडने पर जो परिणाम प्राप्त होता है, उस परिणाम को इसी Section में Output Devices पर भेजा जाता है। एक User को हमेंशा Input व Output Section

ही दिखाई देता है, इसलिए Input व Output Section को हमेंशा काफी सरल व अच्छे तरीके से Represent करना जरूरी होता है, ताकि User Program से अपनी समस्या का समाधान सरल तरीके से प्राप्त कर सके।

Output Function

"C" Language में जब हम किसी परिणाम को Computer की Screen यानी Output Device पर Display करना चाहते हैं, तब हमें "**stdio.h**" नाम की Header File में Define किए गए ***printf***() Function को Use करना होता है।

printf() Function

''सी'' भाषा में सभी I/O Functions **stdio.h** नाम की Header File में होते हैं। जब हमे कोई Message या किसी Variable में Stored मान को Screen पर Display करना होता है, तो हम **printf()** Function का प्रयोग करते हैं। इसका Syntax निम्नानुसार है–

```
printf( " Message CtrlStr1 CtrlStr2 CtrlStrN, Variable1, variable2,  variableN);
```

मानलो कि हम एक ऐसा Program बनाना चाहते हैं, जिसे Run करने पर Monitor पर एक String Display हो। चूंकि हम हमारे इस Program में किसी प्रकार का कोई भी Input व Processing नहीं कर रहे हैं, इसलिए इस Program में केवल Output Section ही होगा। यदि हम इस Program का Algorithm बनाना चाहें, तो ये Algorithm निम्नानुसार बनेगा :

```
Algorithm
   1   START                    [Algorithm Starts here.]
   2   PRINT "Brijvasi"         [Print the message.]
   3   END                      [Algorithm Ends here.]
```

यदि इस Algorithm के आधार पर हम यदि हम "C" Language में Program बनाना चाहें, तो उस Program का Source Code निम्नानुसार होगा :

```
/* Printing Only One Statement on the screen . */

#include<stdio.h>      /* To Get the Input and Output Services */
main()                 /* Main Function from where Compiler Executes Program */
{                      /* Starting of Main Function */
  printf(" Brijvasi ");  /* Prints the Message */
}                      /* Ends the Program */
```

इस Program को **Turbo C++** के IDE में एक New File में Type करें और File को ***FirstPro.c*** नाम से Save करें। इसके बाद File को Compile करके Run करें। File को Compile करने के लिए हम **Ctrl + F9** Key Combination का प्रयोग भी कर सकते हैं। इस Key Combination का प्रयोग करने पर File Compile होकर Run भी हो जाएगी और हमें Output में **Brijvasi** लिखा हुआ Print हो जाएगा।

जैसा कि पहले बताया कि सारे Input/Output Functions "C" की Library की एक Header File **stdio.h** में होते हैं, इसलिए Keyboard से Input लेने या Screen पर Output दर्शाने का काम इसी Header File में Stored Functions के प्रयोग द्वारा सम्पन्न होता है। इसलिए इस Program में "**stdio.h"** नाम की Header File को **#include** किया गया है।

1 हर प्रोग्राम में एक ***main***() Function होता है। main() Function एक Special Function होता है, क्योंकि जब हम "C" Language के किसी Program को Compile करते हैं, तो Compiler सबसे पहले Source Program में main() Function को ही खोजता है और Compiler को जहां पर main() Function मिलता है, Compiler वहीं से Program को Machine Language में Convert करना शुरू करता है।

2 {} (**Opening** व **Closing**) Curly Braces के बीच लिखे गए सभी Statements के समूह को **Statement Block** कहा जाता है और इन्हीं Statements का Execution होता है। चूंकि "C" Language में हर Function की शुरूआत एक Opening Curly Brace से व अन्त एक Closing Curly Brace पर होता है, इसलिए किसी भी Program के जितने भी Executable Instructions होते हैं, उन्हें main() Function के Statement Block में ही लिखा जाता है।

3 "C" Language में हर Statement का अन्त एक Semi Colon द्वारा होता है और "C" में Double Quote के बीच लिखे जाने वाले Statements को **String** कहा जाता है।

4 printf() Function के " " (**Opening** and **Closing**) Double Quotes के बीच लिखा गया Statement Screen पर ज्यों का त्यों Print हो जाता है, क्योंकि ये एक Output Statement है जो किसी Message या मान को Screen पर Display करने का काम करता है।

इस Program को Run करने पर हमें निम्नानुसार Output प्राप्त होता है:

Output

```
Brijvasi
```

इसी Program को यदि चार बार Run किया जाए, तो हमें निम्नानुसार Output प्राप्त होता है :

Output

```
BrijvasiBrijvasiBrijvasiBrijvasi
```

ऐसा इसलिए होता है, क्योंकि जब हम दूसरी बार इसी Program को Run करते हैं, तब पिछली बार Run किए गए Program का Output भी हमें फिर से दिखाई देता है। यदि हम चाहें कि हम जितनी बार भी Program को Run करें, हमें पिछली बार का Output Screen पर दिखाई ना दे, तो हमें "**conio.h**" नाम की Header File में Define किया गया ***clrscr***() Function Use करना होता है। जब हम इस Function को Use करते हैं, तो जिस स्थान पर इस Function को Use करते हैं, उस स्थान पर ये Function Screen पर स्थित Message को Clear कर देता है।

Program को Compile व Run करने के लिए हम **Ctrl+F9** Key Combination का प्रयोग करते हैं। लेकिन जब Program को Run किया जाता है, तो Program Result को Monitor पर Display करते ही तुरन्त Terminate हो जाता है और Output को देखने के लिए हमें **Ctrl+F5** Key Combination का प्रयोग करना पडता है। यदि हम चाहें कि Program Terminate होने से पहले हमें Program का Output Display करे उसके बाद Terminate हो, तो इस सुविधा को प्राप्त करने के लिए हम **getch**() Function का प्रयोग कर सकते हैं।

getch() Function भी "**conio.h**" नाम की Header File में ही Define किया गया है। ये Function Keyboard से एक Character को Input के रूप में प्राप्त करने का काम करता है। इसलिए जब हम इस Function को अपने Program में Use करते हैं, तो हमारा Program तब तक रूका रहता है, जब तक कि User Keyboard से कोई Key Press नहीं करता है।

इस स्थिति में यदि हम इस Statement को हमारे Program के अन्तिम Statement के रूप में Use करें, तो हमारा Program तब तक रूक कर Output Display करता रहेगा, जब तक कि User Keyboard से कोई Key Press नहीं कर देता। इन दोनों सुविधाओं को प्राप्त करते हुए यदि हम पिछले Program को Modify करें, तो हम इस Program को निम्नानुसार Modify कर सकते हैं:

```
#include<stdio.h>          //To get Input and Output Services
main()                     //Main Function from where
                           //Compiler Executes Program
{                          //Starting of Main Function
  clrscr();                // Clears the Screen
  printf("Gopal & Krishna"); // Prints the Name on Screen
  getch();                 // To Pause the output screen until we press a key
}
```

Output

```
Gopal & Krishna
```

Program Flow

जब इस Program को Run किया जाता है, तबः

1. यदि Program में किसी तरह की कोई Typing Mistake ना हो, तो "C" का Compiler सबसे पहले main() Function को खोजता है।

2. main() Function के मिल जाने के बाद Compiler main() Function के Statement Block में प्रवेश करता है और सबसे पहले ***clrscr***() Function को Execute करता है। ये Statement Output Screen को Clear कर देता है।

3. फिर Program का अगला Statement ***printf***() Function Execute होता है, जो Screen पर "**Gopal & Krishna**" Message को Display करता है।

4. अन्त में तीसरा Function ***getch***() Execute होता है। ये Function User से एक Key Press करने का इन्तजार करता है और जब तक User Key Press नहीं करता है, तब तक वह Output को Screen पर देख सकता है। जैसे ही User Keyboard से किसी Key को Press करता है, Program Terminate हो जाता है।

A Answer the following questions.

1 **Data**, **Processing** व **Information** को समझाईए तथा इनके बीच के आपसी सम्बंध की व्याख्या कीजिए।

2 Computer Program किसे कहते हैं? Program व Software में क्या सम्बंध होता है?

3 File किसे कहते हैं? Data File व Program File के बीच क्या अन्तर होता है।

4 Application Software व System Software के बीच के अन्तर को स्पष्ट करते हुए दोनों प्रकार के कुछ Software का उदाहरण दीजिए।

5 Programming के विभिन्न प्रकारों का वर्णन कीजिए।

6 भाषा से आप क्या समझते हैं? Computer किस भाषा को समझता है?

7 Computer Languages को कितने भागों में बांटा गया है? वर्णन कीजिए।

8 High Level Language व Low Level Languages में अन्तर बताते हुए Assembler, Compiler व Interpreter को समझाईए साथ ही ये भी बताईए कि Assembler, Compiler व Interpreter का मुख्य काम क्या होता है?

9 किसी भी Program को Develop करने के विभिन्न Steps को समझाते हुए एक अच्छे Program की विशेषताओं का वर्णन कीजिए।

10 Algorithm किसे कहते हैं? दो संख्याओं को गुणा करके तीसरी संख्या का भाग देने का Algorithm बनाईए।

11 "C" Language के विकास को बताते हुए "C" Language की विभिन्न Characteristics का वर्णन कीजिए।

12 "C" Language के Program का Block Structure बनाकर उसके हर Block को समझाईए।

13 "C" Language में Develop किया गया कोई भी Program किस प्रकार से Execute होकर Output प्रदान करता है? Program के Flow को Flow Chart का प्रयोग करते हुए समझाईए।

14 Function किसे कहते हैं? Pre-Defined Function व User-Defined Function में क्या अन्तर है?

15 Header File से आप क्या समझते हैं? किसी "C" Program में इसकी क्या जरूरत होती है? **stdio.h** Header File को Program में क्यों Include किया जाता है।

16 किसी भी Program के मुख्य तीन हिस्से कौन–कौन से होते हैं? यदि किसी Program के तीनों हिस्सों (Input, Process व Output) में से किसी एक हिस्से को क्रम से Program से हटाया जाए, तो हर हिस्से के हटने का Program पर क्या प्रभाव पडेगा।

17 निम्न Message को Screen पर Display करने का Algorithm बनाइए। इस Algorithm के आधार पर "C" Language में एक Program बनाइए व Program के Flow को समझाईए।

"This is my first program in C Language"

B True/False

1 **Value or a Set of Values** is **Data** for computer program.

2 विभिन्न प्रकार के आंकडों (**Data**) का संकलन (Collection) करना और फिर उन आंकडों को विभिन्न प्रकार से वर्गीकृत (Classify) करके उनका विश्लेषण (Analyze) करने प्रक्रिया को Computer की भाषा में **Data Processing** करना कहा जाता है।

3 Row Data व Processed Data में कोई अन्तर नहीं होता है।

4 जब किसी एक या एक से अधिक समस्याओं को सुलझाने या किसी लक्ष्य को प्राप्त करने के लिए कई स्वतंत्र इकाईयां (Individual Units) मिलकर काम कर रहे होते हैं, तो उन इकाईयों के समूह को **System** कहा जाता है।

5 Data File व Program File दोनों में किसी प्रकार का कोई अन्तर नहीं होता है।

6 Instructions के समूह को Software भी कहते हैं।

7 Compiler व Antivirus एक प्रकार के System Software के हिस्से होते हैं।

8 Computer Architecture व CPU दोनों को तीन–तीन हिस्सों में विभाजित किया जा सकता है।

9 Programming तीन तरह की होती है।

10 Compiler व Interpreter दोनों के किसी Program को Machine Language में Convert करने का तरीका अलग–अलग होता है।

11 Computer एक Electronic Machine है, जो केवल Electrical Signals की Binary language को ही समझता है।

12 किसी प्रोग्राम मे होने वाली गलतियों को खोजकर उन्हे सही करने की प्रक्रिया को Bugging कहते हैं।

13 किसी भी समस्या के एक निश्चित समाधान को प्राप्त करने के लिए अनुक्रमिक व चरणबद्ध रूप में अपनाई जाने वाली लिखित प्रक्रिया को हम **एल्गोरिद्म** कहते हैं।

14 High Level Languages में लिखे जाने वाले Programs को एक Computer Directly Execute करने में सक्षम होता है।

15 “C” Language एक Functional Language है।

16 किसी “C” Program में एक से ज्यादा main() Function हो सकते हैं।

17 Function दो तरह के होते हैं, Pre-Defined व User-Defined

18 किसी भी Computer Program को **Input, Process** व **Output** तीन हिस्सों में बांटा जा सकता है।

19 printf() Function का प्रयोग Keyboard से Data Read करने के लिए किया जाता है।

BASIC ELEMENTS OF C LANGUAGE

Basic Elements of "C"

''सी'' को शुरू करने से पहले इसके कुछ आधारभूत अवयवों को जान लेना बहुत जरूरी है। कुछ खास तरह की Statements को लिखने के लिए विभिन्न प्रकार के Operators व Expressions की जरूरत होती है। हर भाषा में कुछ खास Statements व उनको लिखने के कुछ खास तरीके होते हैं। ये ही बात ''सी'' भाषा पर भी लागू होती है। इस अध्याय में हम ''सी'' के आधारभूत अवयवों के बारे में जानेंगे।

"C" Characterset

प्रत्येक भाषा में चिन्हो, अंकों, अक्षरों का एक समूह होता है। इन चिन्हो, अंकों व अक्षरों को एक विशेष क्रम में रखने पर एक शब्द बनता है जिसका कि अपना एक खास अर्थ होता है। जैसे र् + अ + म् मिलकर राम शब्द बनाते हैं जिसका अपना एक अर्थ होता है।

इसी तरह ''सी'' में भी कुछ खास चिन्हो, अंको व अक्षरों को मान्यता दी गई है, जिनके मिलने से कुछ खास अर्थ निकलते हैं जिन्हे वास्तविक तौर पर सिर्फ कम्प्यूटर ही समझता है। इन चिन्हो, अंकों व अक्षरों के समूह को ''सी'' भाषा का **''सी'' केरेक्टर सेट** कहा जाता है, जो कि निम्नानुसार होता हैः

1 Uppercase (A-Z) and Lowercase (a-z) Alphabet
2 0 to 9 Digits
3 Whitespace Characters (Blank Space, H-Tab, V-Tab, Form Feed, New Line Character, Carriage Return)
4 Special Characters

,	Comma	;	Semi Colon
:	Colon	?	Question Mark
.	Dot	'	Single Quote
"	Double Quote	|	V-Bar
$	Dollar Sign	#	Pound Sign
&	Ampersand	*	Asterisk
(	Left Parentheses	)	Right parentheses
[	Left Bracket	]	Right Bracket
{	Left Curly Brace	}	Right Curly Brace
<	Less Then Sign	>	Greater Then Sign
	Blank	=	Equal to
\	Back Slash	/	Slash
_	Under Score	%	Percent
~	Tilde	^	Upper Carat
+	Plus	-	Minus

! Exclamation mark

इस सारणी में हमने जितने भी Characters को दर्शाया हैं, उन सभी Characters को हम एक "C" Program में समय–समय पर व जरूरत के आधार पर Use कर सकते हैं।

"C" Tokens

जिस प्रकार से शब्द, किसी भी पैराग्राफ की वह लघुत्तम इकाई होती है, जिसमें एक विशेष अर्थ विधमान रहता है, ठीक इसी तरह इस भाषा में भी ऐसे ही कुछ शब्द, चिन्ह आदि हैं, जो स्वतंत्र रूप से अपना कुछ अर्थ रखते हैं। ''सी'' भाषा की वह लघुत्तम इकाई जो स्वतंत्र रूप से अपना कोई अर्थ रखती है, **''सी'' टोकन्** कहलाती है। ''सी'' भाषा में पांच तरह के ''सी'' टोकनस् होते हैं, जिन्हे निम्नानुसार समझाया गया हैः

Keywords या Reserve Words

''सी'' भाषा के कुछ शब्दों को Reserve रखा गया है। इन शब्दों का C Compiler के लिए Special Meaning होता है, इसलिए इन्हें **Keyword** या **Reserve Words** कहते हैं। हर Reserve Word का अपना एक Special Meaning होता है और हर Reserve Word को किसी विशेष परिस्थिति में विशेष काम को पूरा करने के लिए ही Use किया जाता है। हम किसी Reserve Word को किसी सामान्य काम के लिए Use नहीं कर सकते हैं। C भाषा में निम्नानुसार 36 Keywords Define किए गए हैं। कुछ Compilers में इनकी संख्या 32 ही होती है तो कुछ Compilers में इनकी संख्या 36 से ज्यादा भी हो सकती है।

1	**auto**	2	**break**	3	**case**	4	**char**
5	**const**	6	**continue**	7	**default**	8	**do**
9	**double**	10	**else**	11	**enum**	12	**extern**
13	**float**	14	**for**	15	**goto**	16	**if**
17	**int**	18	**long**	19	**register**	20	**return**
21	**short**	22	**signed**	23	**static**	24	**struct**
25	**switch**	26	**typedef**	27	**union**	28	**unsigned**
29	**void**	30	**while**	31	**asm**	32	**fortran**
33	**pascal**	34	**huge**	35	**far**	36	**near**

Identifiers – Constant and Variable Name

जब हम Program Develop करते हैं, तब हमें विभिन्न प्रकार के Data को Computer की Memory में Input करके उस पर विभिन्न प्रकार की Processing करनी होती है। Computer में Data के साथ हम चाहे किसी भी प्रकार की प्रक्रिया करना चाहें, हमें हर Data को सबसे पहले

Computer की Memory में Store करना जरूरी होता है। Computer की Memory में किसी Data को Store किए बिना हम उस Data के साथ किसी प्रकार की कोई प्रक्रिया नहीं कर सकते हैं।

Computer में Memory के हर Location का एक **Unique Address** होता है। जब हम Computer में किसी Data को Process करने के लिए Input करते हैं, तब वह Data Memory के किसी ना किसी Location पर जाकर Store हो जाता है।

लेकिन हमें कभी भी सामान्य तरीके से ये पता नहीं चल सकता है कि हमारे द्वारा Input किया गया Data Computer की किस Memory Location पर Store हुआ है और ना ही हम स्वयं कभी ये तय कर सकते हैं कि हमारा Data किस Memory Location पर Store होगा, क्योंकि Data को Memory Allocate करने का काम अपनी सुविधानुसार हमारा Operating System स्वयं करता है।

जिस समय हमारे Data को Store करने के लिए Compiler Memory Reserve करता है, उसी समय हम उस Reserve होने वाली Memory Location का एक नाम Assign कर देते हैं। इस नाम के द्वारा ही हम हमारे Data को Computer की Memory में Identify कर सकते हैं। हमारे द्वारा किसी Data की Memory Location को दिए जाने वाले इस नाम को ही **Identifier** कहते हैं।

हम किसी Memory Location का जो नाम Assign करते हैं, उन नामों को कुछ नियमों को ध्यान में रख कर परिभाषित करना होता हैं, क्योंकि ''सी'' कम्पाइलर उन विशेष प्रकार के नियमों के आधार पर परिभाषित किये गए नामों के साथ ही विभिन्न प्रकार की प्रक्रियाएं करता है। किसी Identifier को नाम देने के लिए हमें निम्न नियमों को Follow करना होता है, जिन्हें ***Identifier Naming Convention*** कहा जाता हैः

- किसी भी Identifier के नाम में किसी भी **Upper Case** व **Lower Case Character** का प्रयोग किया जा सकता है।
- किसी भी Identifier के नाम में **Underscore** का भी प्रयोग किया जा सकता है।
- किसी भी Identifier के नाम में यदि हम अंकों का प्रयोग करना चाहें, तो अंकों का प्रयोग करने से पहले कम से कम एक Character या Underscore का होना जरूरी होता है।
- इसके अलावा Identifier के नाम में किसी भी प्रकार के Special Symbol जैसे कि Period, Comma, Blank Space आदि का प्रयोग नहीं किया जा सकता है। साथ ही हम Identifier के नाम में किसी Reserve Word या किसी Built-In Function के नाम का प्रयोग भी नहीं कर सकते हैं।
- किसी भी नाम की शुरूआत किसी अंक से नहीं हो सकती है।
- ''सी'' एक **Case Sensitive Language** है, इसलिए इस भाषा में Capital Letters व Small Letters के नाम अलग–अलग माने जाते हैं। जैसे int Sum व int sum दो अलग–अलग **Variable Name** या **Identifies** होंगे ना कि समान।

किसी Variable Identifier या Constant Identifier का हम निम्न तरीके का कोई भी नाम रख सकते हैं, जो कि "C" के Naming Rules का पूरी तरह से पालन करते हैं:

```
number
number2
amount_of_sale
_amount
salary
daysOfWeek
monthsOfYear
```

लेकिन आगे दिए जा रहे नाम गलत हैं और हम इन तरीकों के नाम किसी Variable या Constant Identifier को Assign नहीं कर सकते हैं, क्योंकि ये नाम "C" Language के Naming Rules का पूरी तरह से पालन नहीं करते हैं:

```
number#          /* illegal use of Special Symbol # */
number2*         /* illegal use of Special Symbol * */
1amount_of_sale  /* Name could not start with a Digit */
&$amount         /* illegal use of Special Symbol & and $ */
days Of Week     /* illegal use of Special Symbol Blank Space between name */
months OfYear10  /* illegal use of Special Symbol Blank Space between name */
```

Exercise

1 Specify invalid variable names and give proper reason why they are invalid?

TOTALPERCENT	_BASIC	basic-salary	1st value
$per#	daysIn1Year	LeAPyEAr	432
float	Integer	longInteger	hours.
daysInWeek	Book Name	population	day of week
minute.	father's Name	2910_India	_total_days_

2 Keyword किसे कहते हैं ?

3 Identifiers से आप क्या समझते हैं ? **Keywords** व **Identifiers** में क्या अन्तर है ?

4 *Identifier Naming Convention* से आप क्या समझते हैं ?

5 Identifiers Create करते समय हमें किन नियमों को ध्यान में रखना जरूरी होता है ?

Constants and Variables

सभी Programming Languages में यदि कोई चीज Common होती है, तो वह यही है कि सभी Programming Languages में Develop किए जाने वाले Programs में Data को Input किया जाता है और उन पर Required Processing Perform करके Output Generate किया जाता है।

चूंकि किसी भी Computer Program में सबसे Important चीज Data ही होती है, इसलिए हर Computer Program में इसी बात का ध्यान रखा जाता है कि Data को विभिन्न तरीकों से Store किया जाए, ताकि उन पर विभिन्न प्रकार की Processing को Apply करके विभिन्न प्रकार के Results Generate किए जा सकें। Data Memory में किस प्रकार से Store होंगे और किस प्रकार से उन पर Processing को Apply किया जाएगा, इस बात का Track रखने के लिए Programs में **Constants** व **Variables** का प्रयोग किया जाता है।

Constants

किसी भी Computer Program में हम विभिन्न प्रकार के मानों को Computer में Store करते हैं, उन्हें Manage करते हैं, उन पर Required Processing Apply करते हैं और उनके परिणाम को Output में प्राप्त करते हैं। यदि हम Real World में देखें तो दो तरह के मान होते हैं। एक मान वे होते हैं जिन्हें कभी Change नहीं किया जाता है।

जैसे कि साल में कुल 12 महीने होते हैं। इन महीनों की संख्या हमेंशा निश्चित होती है। कभी भी किसी भी साल में 11 या 13 महीने नहीं हो सकते। इसी तरह से हर महीने का एक निश्चित नाम होता है। हर Week में सात दिन होते हैं। हर दिन का एक निश्चित नाम होता है। इसी तरह से PI का मान 22/7 होता है।

हम समझ सकते हैं कि ऐसी ही हजारों चीजें हैं, जिनके मान हमेंशा निश्चित होते हैं। जो मान हमेंशा निश्चित होते हैं, उन मानों को Hold करने वाले Identifiers को **Constants** कहा जाता है। इसी तरह से किसी Computer Program में Declare किया गया वह Identifier जो ऐसे ही किसी Constant मान को Hold करता है और पूरे Program में अपने Data को Change नहीं करने देता है, Constant कहलाता है।

हम किसी भी Data को मान या मानों के एक समूह के रूप में मान सकते हैं। यानी किसी भी तथ्य को Computer Program में Represent करने के लिए हमें उस तथ्य को किसी ना किसी मान के रूप में परिभाषित करना होता है। Computer में मानों को या तो Texts के रूप में Represent किया जाता है या फिर किसी अंक के रूप में।

उदाहरण के लिए यदि हमें साल के कुल महीनों को Computer में Store करना हो तो हम अंक 12 को उपयोग में लेते हैं, जो कि एक संख्या है। जबकि यदि हमें किसी महीने के नाम माना "**January**" को Computer में Store करना हो तो हम Characters के समूह का प्रयोग करते हैं।

इस उदाहरण के आधार पर हम कह सकते हैं कि किसी भी Real World मान को Computer में या तो किसी अंक या अंकों के समूह के रूप में Define किया जाता है या किसी Character या Characters के समूह के रूप में।

विभिन्न अंक या अंकों के समूह को हम **Numeral Constants** कह सकते हैं और विभिन्न Characters व Characters के समूह को **Character** या **String Constants** कह सकते हैं। उदाहरण के लिए मान लो कि हमें 100 रूपये का 6.0 प्रतिशत की दर से ब्याज ज्ञात करना है। ये Calculation हम निम्नानुसार Perform कर सकते हैं:

```
Interest = 100 * 6.0 / 100
```

इस Statement में Numerical मान **100** व **6.0** स्थिर मान हैं, इसलिए इन्हें **Constant** कहा जाता है। मानलो कि हमें किसी Program में इस Calculation को कई बार Perform करना पडता है। इस स्थिति में हम इस Statement को पूरे Program में कई बार लिख सकते हैं।

लेकिन थोडे समय बाद यदि हमें 6.0 के बजाय 7.0 प्रतिशत की दर से ब्याज Calculate करने के लिए इसी Program को Modify करना पडे, तो हमने Program में जितनी बार इस Calculation को Perform किया है, उतनी ही बार अंक 6.0 के स्थान पर 7.0 को Replace करना पडेगा।

यदि हमने हमारे Program में 200 बार इस Statement को Use किया गया हो तो हमें हमारे Program में 200 स्थानों पर 6.0 के स्थान पर 7.0 करना पडेगा जो कि काफी असुविधाजनक काम होगा। क्योंकि Program को Modify करने में भी काफी समय लगेगा और गलतियां होने की भी काफी सम्भावना रहेगी, क्योंकि पूरे Program में किसी एक भी स्थान पर यदि हमने **6.0** को **7.0** से Replace नहीं किया, तो Program का Output गलत ही आएगा।

इस प्रकार की स्थितियों को Avoid करने के लिए Programmers हमेंशा कुछ Symbolic Constants का प्रयोग करते हैं, जो सामान्यतया वे शब्द होते हैं, जो Program में किसी मान को Represent करते हैं।

यदि हम हमारे इस पिछले Expression की ही बात करें, तो 6.0 को Represent करने के लिए हम **PERCENT** नाम का एक Symbolic Content Use कर सकते हैं, जो Current Percent को Represent करता है और Program की शुरूआत में इस Percent को वह दर प्रदान कर सकते हैं, जिसे पूरे Program में Calculate करना है।

"C" Language में किसी Constant को Represent करने के लिए जो Statement लिखा जाता है, उसे Constant Declare करना कहते हैं और इसे निम्नानुसार Declare करते हैं:

```
const float PERCENT = 6.0;
```

"C" में **const** Keyword का प्रयोग तब किया जाता है, जब हमें "C" Compiler को ये बताना होता है, कि हम जिस Identifier द्वारा किसी मान को Program में Represent कर रहे हैं, उस Identifier के मान में पूरे Program के दौरान किसी प्रकार का Change नहीं किया जा सकता है।

इसी तरह से **float** Keyword "C" Compiler को ये बताता है कि हम जिस Constant मान को Store करना चाहते हैं, वह मान एक Floating Point मान या दसमलव वाला मान है। PERCENT शब्द एक Symbolic Content है और इस Expression में **=** (***Equal To***) का चिन्ह बताता है कि **=** के Left Side में जो Word है वह Word उस मान के बराबर है जो **=** चिन्ह के Right Side में है जो कि हमारे इस Statement में 6.0 है।

यानी हम इस Calculation में 6.0 लिखें या PERCENT लिखें, दोनों से निकलने वाला परिणाम समान ही प्राप्त होगा, क्योंकि दोनों ही समान मान को Represent कर रहे हैं।

```
Interest = 100 * PERCENT / 100;
```

सामान्यतया Symbolic Constants को Program के अन्य Codes से अलग दिखाने के लिए UPPERCASE Letters में लिखा जाता है।

Variables

Program के वे मान जो पूरे Program में समय–समय पर आवश्यकतानुसार बदलते रहते हैं, Variables कहलाते हैं। Variables कभी भी किसी स्थिर मान को Represent करने के लिए Use नहीं किए जाते हैं। जब भी हमें किसी Constant को Program में Use करना होता है, तो उस Constant को Represent करने के लिए हमें Symbolic Constants की जरूरत होती है। इन Symbolic Constants को ही Literal भी कहा जाता है।

सवाल ये पैदा होता है कि Program में Variables की क्या जरूरत है ? इसे समझने के लिए पिछले Statement को ही लेते हैं, जो कि निम्नानुसार है:

```
Interest = 100 * PERCENT / 100;
```

इस Statement में Interest एक **Variable** है। यानी किसी Calculation के Result को Store करने के लिए हमें हमारे Program में हमेंशा एक ऐसी Memory की जरूरत होती है, जिसमें विभिन्न

प्रकार के बदलते हुए मान Store हो सकें। इस Statement द्वारा हम केवल 100 का ही PERCENT ज्ञात कर सकते हैं।

लेकिन सामान्यतया हमें अलग–अलग स्थानों पर अलग–अलग प्रकार के मानों का Percent ज्ञात करना होता है। ऐसे में हर संख्या का Percent ज्ञात करने के लिए यदि हमें अलग से Program बनाना पडे तो ये एक बहुत ही असुविधाजनक बात होगी।

Program ऐसा होना चाहिए कि किसी एक ही Program से एक प्रकार से Perform होने वाली विभिन्न प्रकार की Calculations को Perform किया जा सके। यानी हम यदि 100 की जगह 200 कर दें, तो हमें 200 का Interest प्राप्त हो जाए। यदि हम Program को Multipurpose बनाना चाहते हैं, तो हमें 100 को भी किसी Symbolic तरीके से Represent करना होगा। ये काम हम निम्नानुसार Statement द्वारा कर सकते हैं:

```
Principal = 100;
Interest = Principal * PERCENT / 100;
```

हम देख सकते हैं कि यदि Principal का मान 100 से 200 कर दिया जाए तो Interest नाम के Variable में हमें Principal 200 का Interest प्राप्त होगा। चूंकि मूलधन 100 के Symbolic Representative Principal का मान बदल कर 200, 300, 400 आदि किया जा सकता है, इसलिए **Principal** भी एक Variable है और Principal के Change होने से Calculate होने वाले Interest में भी परिवर्तन होता है, इसलिए **Interest** भी एक Variable है।

वास्तव में Constant Identifier व Variable Identifier के नाम में किसी प्रकार का कोई अन्तर नहीं होता है। अन्तर केवल इनके Declaration के तरीके में होता है। हम Variable Identifier को Declare करें या Constant Identifier को, दोनों ही स्थितियों में हमें Identifier Naming Convention के उपरोक्त सभी नियमों का पालन करना होता है।

Identifier Declaration

किसी भी प्रकार के Data को Process करने के लिए हमें सबसे पहले ये तय करना होता है, कि हम किस प्रकार के Data को Computer की Memory में Store करना चाहते हैं, क्योंकि जब तक हम Process किए जाने वाले Data को Computer की Memory में Store नहीं कर देते हैं, तब तक हम उस Data को Process नहीं कर सकते हैं।

चूंकि अलग–अलग प्रकार के Data Memory में अलग–अलग Size की Space Reserve करते हैं, इसलिए जब हमें Process किए जाने वाले Data के Type का पता चल जाता है, तब हम उस Data Type को Represent करने वाले Keyword के आधार पर Memory में कुछ Space Reserve करते हैं और उस Space का कोई नाम Assign करते हैं। ये नाम उस Reserved

Memory Location का एक Symbolic Identifier होता है, जो कि *Identifier Naming Convention* के नियमों के आधार पर तय किया जाता है।

Program की जरूरत के आधार पर किसी Data को Store करने के लिए Computer की Memory में Space Reserve करने व उस Space का कोई Symbolic नाम देने की प्रक्रिया को **Identifier Declaration** कहते हैं।

यदि Define किए जाने वाले Identifier का मान पूरे Program में स्थिर रहे, तो इस प्रक्रिया को **Constant Declaration** कहते हैं, जबकि यदि Define किए जाने वाले Identifier का मान पूरे Program में समय–समय पर Program की जरूरत के आधार पर बदलता रहे, तो इसे **Variable Declaration** कहते हैं।

Identifier Declaration के समय हमें हमेंशा दो बातें तय करनी होती हैं। पहली ये कि हमें किस प्रकार (***Data Type***)का Data Computer की Memory में Store करना है और दूसरी ये कि Reserve होने वाली Memory Location को क्या नाम (***Identifier Name***) देना है। Identifier Declare करने का General Syntax निम्नानुसार होता हैः

Syntax:

DataTypeModifier DataType *IdentifierName*;

DataTypeModifier

Syntax के इस शब्द के स्थान कुछ ऐसे Keywords का प्रयोग किया जाता हैं, जिनका प्रयोग करके DataType की किसी मान को Store करने की क्षमता को बढाया या घटाया जा सकता है। इस शब्द के स्थान पर जरूरत के आधार पर ***short, long, signed*** या ***unsigned*** Keywords का प्रयोग किया जा सकता है। यहां Use किया जाने वाला Keyword Optional होता है। यदि हमें जरूरत ना हो तो हम इसका प्रयोग किए बिना भी Memory Create कर सकते हैं।

DataType

Syntax के इस शब्द के स्थान पर कुछ ऐसे Keywords का प्रयोग किया जाता हैं, जो ये तय करते हैं कि हम Computer की Memory में किस प्रकार के Data को Store करना चाहते हैं। उदाहरण के लिए यदि हमें केवल पूर्णांक संख्याओं को Store करने के लिए Memory Reserve करना हो, तो हम इस शब्द के स्थान पर ***int*** Keyword का प्रयोग करते हैं, जबकि यदि हमें किसी दसमलव वाली संख्या के लिए Memory Reserve करना हो, तो हमें इस शब्द के स्थान पर ***float*** Keyword को Use करना होता है।

IdentifierName

Syntax के इस शब्द के स्थान पर हम “C” के Identifier Naming Convention के नियमों के आधार पर Reserve होने वाली Memory Location का एक Symbolic नाम Specify किया जाता

है। हम Reserve किए गए Memory Location पर स्थित जिस मान को पूरे Program में Access करना चाहते हैं, उस मान को हम इसी Symbolic नाम से Access करते हैं।

मानलो कि हम किसी Student की **Age** को Computer में Store करना चाहते हैं। चूंकि Age एक प्रकार का Numerical पूर्णांक मान होता है, इसलिए हमें इस Integer Data Type के मान को Store करने के लिए "C" Language के ***int*** Keyword का प्रयोग करना होता है।

"C" Language में सभी प्रकार के Numerical मान Positive Signed मान होते हैं, जिसमें Minus की संख्या को भी Store किया जा सकता है। लेकिन चूंकि Age कभी भी Minus में नहीं हो सकती है, इसलिए हमें ***int*** Data Type से पहले हमें ***unsigned*** Modifier का प्रयोग करना होगा।

चूंकि हम Student की Age को Computer में Store करने के लिए Unsigned Integer प्रकार की Memory Location को Reserve कर रहे हैं, इसलिए इस Memory Location को Identify करने के लिए हम Symbolic नाम के रूप में ***studentAge*** शब्द का प्रयोग कर सकते हैं, जो कि *Identifier Naming Convention* के नियमों के आधार पर पूरी तरह से सही है।

इस Discussion के आधार पर यदि हम Student की **Age** को Store करने के लिए एक Identifier Create करें, तो हमें "C" Language में निम्नानुसार Statement लिखना होगाः

Variable Declaration

```
unsigned int studentAge;
```

जहां **unsigned** Keyword Modifier है। **int** Data Type है और *studentAge* Reserve होने वाली Memory Location का Symbolic नाम है। अब यदि हम Student की Age को Reserve होने वाली Memory Location पर Store करना चाहें, तो हमें निम्नानुसार Statement लिखना होता हैः

```
studentAge = 21;
```

इस Statement द्वारा उस Memory Location पर Integer मान 21 Store हो जाता है, जिसका नाम studentAge है। यदि हम चाहें तो इस Memory Location पर 21 के स्थान पर अगले Statement में निम्नानुसार 23 भी कर सकते हैंः

```
studentAge = 32;
```

यानी हम जितनी बार चाहें, उतनी बार अपनी जरूरत के आधार पर *studentAge* Symbolic Name वाली Memory Location का मान Change कर सकते हैं। इसलिए इस Symbolic Identifier को हम **Variable** भी कह सकते हैं। लेकिन यदि हम चाहते हैं, कि studentAge में केवल एक ही बार किसी Integer मान को Store किया जाए और किसी भी स्थिति में पूरे Program में studentAge

के मान को Change ना किया जा सके, तो हमें इसी Declaration के समय निम्नानुसार ***const*** Keyword का प्रयोग करना होता हैः

Constant Declaration

```
const unsigned int studentAge = 21;
```

जब हम इस प्रकार से किसी Identifier को Declare करते हैं, तब इस प्रकार के Declaration को Constant Identifier Declaration कहते हैं। हम जब भी कभी किसी Identifier को Constant Declare करते हैं, तो उस Identifier को Declare करते समय ही हमें Symbolic Name की Memory Location पर Store होने वाले मान को भी Specify करना जरूरी होता है। क्योंकि Constant Identifier हम उसी स्थिति में Declare करते हैं, जब हमें पता होता है कि किसी Constant Identifier का मान पूरे Program में क्या होना चाहिए।

जब हम किसी Identifier को Constant Declare करते हैं, तब यदि हम Program में किसी Statement द्वारा उस Constant के मान को Change करने की कोशिश करते हैं, तो Compiler हमें ऐसा नहीं करने देता है। यानी यदि हम उपरोक्त Statement लिखने के बाद निम्नानुसार दूसरा Statement लिखें और Program को Compile करें, तो Compiler हमें निम्नानुसार Error Message प्रदान करता हैः

```
studentAge = 23;  // Error: Cannot modify a const object.
```

इसी तरह से यदि हम किसी const Identifier को Declare करते समय उसे कोई मान प्रदान ना करें, यानी Constant को निम्नानुसार Declare करेंः

Constant Declaration

```
const unsigned int studentAge;
```

तो इस स्थिति में हमें निम्नानुसार Error Message प्राप्त होता हैः

```
Error:  Constant variable 'studentAge' must be initialized
```

Initialization

हम किसी भी Identifier को उसके Declaration के समय ही किसी ना किसी प्रकार का मान भी प्रदान कर सकते हैं। Identifier को उसके Declaration के समय ही कोई मान प्रदान करने की प्रक्रिया को **Value Initialization** करना कहते हैं। जैसे :–

```
int digit1 = 12;
int digit2 = 33;
```

हम एक ही समय में एक से अधिक Identifiers को जो कि समान प्रकार के Data Type के हों, Declare कर सकते हैं व किसी ना किसी मान से Initialize भी कर सकते हैं। जैसेः

```
int digit1, digit2 ;                OR
int digit1 = 12, digit2 = 33 ;
```

Expressions

जब दो या दो से अधिक **Operands** पर **Operators** की सहायता से कोई प्रक्रिया करके कोई परिणाम प्राप्त करना होता है, तो उस स्थिति में हम जो **Statement** लिखते हैं, उसे **Expression** कहते हैं। उदाहरण के लिए दो संख्याओं को जोड कर प्राप्त मान को किसी तीसरे Identifier में Store करने के लिए हम निम्न Statement लिखते हैंः

```
sum = digit1 + digit2
```

इस Statement को Expression कहा जाता है। उपरोक्त Statement एक Arithmetical Expression का उदाहरण है। इसी तरह विभिन्न प्रकार के Logical, Relational आदि Operators को Use करके विभिन्न प्रकार के Expressions बनाए जा सकते हैं।

Exercise:

1 Variable व Constant के बीच के अन्तर को Explain कीजिए।

2 Identifier Declaration क्यों किया जाता है? Identifier Declare करने के लिए हमें किस Syntax का प्रयोग करना पडता है? इस Syntax को समझाईए।

3 Variable Identifier व Constant Identifier में अन्तर स्पष्ट करते हुए Integer प्रकार के दो Variable व Constant Declare कीजिए।

4 एक उदाहरण देते हुए Initialization व Expression के अन्तर को स्पष्ट कीजिए।

Data and Data Types

मान या मानों के समूह Computer के लिए Data होता है। Real World में भी Data (मान या मानों का समूह [**Value** or a **Set of Values**]) कई प्रकार के होते हैं। जैसे किसी व्यक्ति की उम्र को हम संख्या के रूप में दिखाते हैं, जबकि उस व्यक्ति के नाम को Characters के समूह के रूप में परिभाषित करते हैं। इसी Concept के आधार पर "C" Language में भी विभिन्न प्रकार के Data को Store करने के लिए विभिन्न प्रकार के Data Types के Keywords को Develop किया गया है।

यदि गौर किया जाए तो वास्तव में Data केवल दो तरह के ही होते हैं। या तो Data Numerical होता है, जिसमें केवल आंकिक मान होते हैं और इनके साथ किसी ना किसी प्रकार की Calculation को Perform किया जा सकता है या फिर Alphanumerical जो कि Characters का समूह होते हैं, जिनके साथ किसी प्रकार की किसी Calculation को Perform नहीं किया जा सकता।

"C" Language में भी Data को Store करने के लिए दो अलग तरह के Data Types में विभाजित किया गया है, जिन्हें क्रमशः Primary (Standard) Data Type व Secondary (Abstract or Derived) Data Type कहा जाता है। Primitive Data Type **Standard Data Type** होते हैं, जबकि Derived या Abstract Data Type Primitive Data Type पर आधारित होते हैं। फिर जरूरत के अनुसार इन दोनों Data Types को भी कई और भागों में बांटा गया है, जिन्हें हम निम्न चित्र द्वारा समझ सकते हैं:

''सी'' Data Type

- **Fundamental Data Type**
  - Character
  - Integer
  - Float
  - Double
  - Void
- **Derived Data Type**
  - **Built – in – Type**
    - Array
    - Functions
    - Pointer
  - **User Defined Type**
    - Structure
    - Unions

किसी भी Identifier, या Constant को Define करने से पहले यह निश्चित करना जरूरी होता है कि वह Identifier (Variable, Array, Constant आदि) किस तरह का मान Store करेगा।

यानी यदि हमें छोटी संख्या को Computer में Store करना हो तो हमें **int** प्रकार के Data Type के Identifier को Use करना होता है, जबकि यदि हमें बडी संख्या को Computer में Store करना हो तो हमे **long** प्रकार के Data Type का Identifier Use करना होता है।

इसी तरह से यदि हमें Computer में केवल एक Character को Store करना हो, तो हम **char** प्रकार के Data Type को Use करते हैं, जबकि यदि हमें पता ही ना हो कि हम किस प्रकार के

Data को Computer में Store करेंगे, तो हम **void** प्रकार का Identifier Declare करते हैं। यानी हमें किस प्रकार के मान को किस प्रकार के Data Type के Identifier में Store करना है, यह बात Program की Requirement पर निर्भर करता है।

"C" Language हमें विभिन्न प्रकार के Data Types को अलग–अलग Store व Manage करने की सुविधा इसलिए Provide करता है, ताकि सारा काम Systematically हो सके और प्रोग्राम को विश्वसनीय व तेजी से Run होने वाला बनाया जा सके। ''सी'' भाषा में कुल चार तरह के Fundamental Data Types हैं, जिन्हे निम्नानुसार समझाया गया हैः

Integer

जब प्रोग्राम में सिर्फ पूर्णांक संख्याओं को Store करने के लिए ही Memory Reserve करनी होती है, तब Identifier को Integer प्रकार का Declare किया जाता है। इसमें भिन्नांक संख्याएं नहीं हो सकती है। किसी Variable को Integer प्रकार का Declare करने के लिए Identifier के नाम के साथ **int** Keyword का प्रयोग करके "C" Compiler को बताया जाता है कि वह Identifier केवल पूर्णांक संख्याओं को ही Memory में Store कर सकेगा। int प्रकार के Identifier में हम + व – दोनों तरह के मान रख सकते हैं।

जब हम किसी Identifier को Declare करते समय किसी भी Modifier का प्रयोग नहीं करते हैं, तब Create होने वाला Identifier By Default ***signed*** होता है। इसलिए यदि हमें ऐसे मान को Store करने के लिए Identifier Create करना हो, जो Positive या Negative किसी भी प्रकार के मान को Store कर सकता है, तो हमें ***signed*** Keyword का प्रयोग करने की जरूरत नहीं होती है, लेकिन यदि हमें केवल Positive मान को Store करने के लिए ही Identifier Create करना हो, तो उस स्थिति में हमें Data Type के साथ ***unsigned*** Modifier का प्रयोग करना जरूरी होता है। ''सी'' मे Integer प्रकार के Identifier को निम्न भागों में बांटा गया हैः

int OR signed int

यदि हम 16 – Bit Compiler का प्रयोग करते हैं, तो इस प्रकार का Identifier Memory में दो Byte का Storage Space Reserve करता है जबकि यदि हम 32 – Bit का Compiler Use करते हैं तो इस प्रकार का Identifier Memory में 4 Bytes का Storage Space Reserve करता है। int प्रकार का Identifier 16 – Bit Compiler Use करने पर –32768 से 32767 मान तक की संख्या को स्टोर कर सकता है।

जब Identifier के साथ – चिन्ह होता है या संख्या का मान ऋणात्मक हो सकता है, तो Identifier के साथ ***signed int*** लिखते हैं। जैसे **signed int** *total* लेकिन यदि हम **signed** Key word का प्रयोग नहीं करते हैं तो भी int प्रकार का Identifier Negative Values को Hold कर सकता है। इस प्रकार के Data को Memory Space Allocate करने के लिए हमें निम्नानुसार Declaration करना होता हैः

signed int *bankDues*; or **int** *bankDues*;

unsigned int

यदि हम 16 – Bit Compiler का प्रयोग करते हैं तो इस प्रकार का Identifier Memory में दो Byte का Storage Space Reserve करता है जबकि यदि हम 32 – Bit का Compiler Use करते हैं तो इस प्रकार का Identifier Memory में 4 Bytes का Storage Space Reserve करता है। इसमें int Keyword के पहले **unsigned** Keyword Use किया जाता है।

unsigned int प्रकार का Identifier 0 से 65535 तक के मान की संख्या को स्टोर कर सकता है, क्योंकि इसमें हमेंशा एक Positive पूर्णांक मान ही Store हो सकता है। इसलिए इसकी कुल क्षमता 32768 + 32767 = 0 से 65535 तक की संख्या Store हो जाती है। इस प्रकार के Data को Store करने के लिए हमें निम्नानुसार Declaration करना होता हैः

unsigned int *villagePopulation*;

short OR signed short int

जब हमें काफी छोटी संख्या को Store करने के लिए Memory Reserve करनी होती है और Store की जाने वाली संख्या पूर्णांक ही होती है, तब Identifier के नाम के साथ **short** Modifier का प्रयोग करते है। यदि हम **signed** Modifier का प्रयोग ना भी करें, तो भी हम इस प्रकार के Identifier में Minus Sign के मान को Store कर सकते हैं। लेकिन यदि हम चाहें तो **short** Keyword के साथ **signed** Modifier का प्रयोग भी कर सकते हैं। इस प्रकार का Identifier मेमोरी में हमेंशा 2 Byte की Storage Space Reserve करता है, फिर चाहे हम 16 – Bit Compiler Use कर रहे हों या 32 – Bit Compiler Use कर रहे हों।

सामान्यतया **short** प्रकार के Identifier का प्रयोग तब किया जाता है, जब हम 32 – Bit Compiler को Use कर रहे होते हैं और हमें छोटे मान को Store करना होता है। चूंकि 32 – Bit Compiler में Integer 4 Byte का होता है, इसलिए छोटी Integer संख्याओं को Store करने के लिए हम **short** प्रकार के Identifier को Use करके 2 Bytes की बचत कर सकते हैं। 16 – Bit Compiler में हम चाहे **int** Type का Identifier Declare करें या **short** Type का, दोनों ही Memory में 2 Byte का ही Storage Space Reserve करते हैं। इस प्रकार के Identifier को हम निम्नानुसार Declare कर सकते हैंः

signed short int *normalTemprature*;or
short int normalTemprature;

short Modifier का प्रयोग हमेंशा Integer प्रकार के Identifier के साथ ही किया जाता है, इसलिए यदि हम चाहें तो उपरोक्त दोनों Declaration निम्नानुसार बिना ***int*** Keyword का प्रयोग किए हुए भी कर सकते हैं:

signed short *normalTemprature*; or
short normalTemprature;

unsigned short int

जब हम 16 – Bit Compiler को Use करते हैं और हमें छोटी लेकिन केवल Positive संख्या को ही Computer में Store करने के लिए Storage Space Reserve करना होता है, तब हम **unsigned short int** प्रकार का Identifier Declare करते हैं। ये Identifier भी Memory में 2 Bytes की Storage Space ही Reserve करता है और इसे Declare करने का तरीका **signed short** Type के Identifier को Declare करने के समान ही होता है। यानी:

unsigned short int *salary*; or
unsigned short *salary*;

long OR signed long int

जब हमें काफी बडी संख्या का प्रयोग करना होता है तब हम इस Data Type का चयन करते हैं। यह मेमोरी में 4 Byte की Storage Space Reserve करता है और -2,147,483,648 से 2,147,483,647 मान तक की संख्या को Store कर सकता है। इस Data Type का प्रयोग अक्सर वैज्ञानिक गणनाओं में किया जाता है जहां काफी बडी संख्याओं की गणना करनी होती है। इस प्रकार का Identifier Declare करने के लिए हम निम्न में से किसी भी तरीके को Use कर सकते हैं:

long *velocity*;
long int *velocity*;
signed long *velocity*;
signed long int *velocity*;

long प्रकार का Identifier हमेंशा By Default Integer प्रकार का ही होता है, इसलिए यदि हम **long** के साथ **int** Keyword प्रयोग नहीं भी करते हैं, तब भी Create होने वाला Identifier long int प्रकार का ही होता है।

unsigned long int

जब हमें काफी बडी लेकिन केवल धनात्मक संख्या को Store करने के लिए ही Memory में Storage Space Create करना होता है, तब हम इस Data Type को Use करते हैं। यह भी

मेमोरी मे 4 Byte की Storage Space Reserve करता है, लेकिन इस प्रकार के Identifier में Minus की संख्या को Store नहीं किया जा सकता है। इस प्रकार के Data Type के Identifier की संख्या Store करने की Limit 0 से 4,294,967,295 है। इस प्रकार का Identifier Define करने के लिए हम निम्न तरीकों से Statements को Use कर सकते हैं:

unsigned long *population*;
unsigned long int *population*;

Float

जब हमें प्रोग्राम में भिन्नात्मक व दशमलव वाली संख्याओं को Store करने के लिए Memory की जरूरत होती है, तब हम float प्रकार का Identifier Declare करते हैं। ये Identifier मेमोरी मे 4 Bytes की Storage Space Reserve करता है और भिन्न या घातांक रूपों में 3.4E-38 से 3.4E+38 मान तक की संख्या को Store कर सकता है। इस प्रकार के Identifier के साथ **unsigned, signed, short** या **long** किसी भी Modifier का प्रयोग नहीं किया जा सकता है। इस तरह का Identifier निम्नानुसार तरीके से Declare कर सकते हैं:

float *lightSpeed*;

जब हम किसी Float प्रकार के Variable में कोई मान Store करना चाहते हैं, तब मान के साथ हमें **f** या **F** Character को Post-Fix के रूप में Use करना जरूरी होता है। यदि हम ऐसा नहीं करते हैं, तो Float प्रकार के Identifier में Store होने वाला मान Float प्रकार का नहीं बल्कि Double प्रकार का होता है। यानी "C" Language में हर Real Number *By Default* Double प्रकार का होता है। इसलिए यदि हम lightSpeed Variable में कोई मान Store करना चाहें, तो हमें ये मान निम्नानुसार Statement द्वारा Store करना होगा:

lightSpeed = 38000000000*f*; OR
lightSpeed = 38000000000*F*;

Double

जब हमें प्रोग्राम में इतनी बडी भिन्नात्मक या घातांक संख्या के साथ प्रक्रिया करनी होती है, जो की ***float*** की Range से भी ज्यादा हो, तब हम इस Data Type का प्रयोग करके Identifier Declare करते है। इन्हें भी हम दो भागों में बांट सकते हैं:

Double

ये मेमोरी में 8 Byte की Storage Space Reserve करता है और 1.7E-308 से 1.7E+308 मान तक की संख्या को Store कर सकता है। इस प्रकार के Identifier को हम निम्नानुसार Declare कर सकते हैं:

double *lightMovementIn1Year*;

long double

जब Double के साथ **long** Key word लगा दिया जाता है यानी जब **long double** प्रकार का Data Type Use करते है तब वह Identifier बडी से बडी संख्या को Store कर सकता है। यह मेमोरी में 10 Byte की Storage Space Reserve करता है और 3.4E-4932 से 3.4E+4932 मान तक की संख्या Store कर सकता है। इस तरह का Identifier भी हम निम्नानुसार Declare कर सकते हैं:

double *lightMovementIn100Year*;

Character

जब हमें Computer में ''सी'' Characterset के किसी Character को Store करने के लिए Memory को Reserve करना होता है, तब हम Character प्रकार के Data Type का प्रयोग करके Identifier Create करते हैं। इस प्रकार का Identifier Create करने के लिए हमें "C" Language के ***char*** Keyword का प्रयोग करना होता है। इस प्रकार का Identifier मेमोरी में 1 Byte की Space Reserve करता है। char प्रकार के Identifiers में हम केवल एक ही Character Store कर सकते हैं। हम char प्रकार के Identifier में संख्या भी Store कर सकते हैं। इस Data Type को भी दो भागों में बांटा गया है:

signed char or char

Computer की Memory में हम कभी भी किसी Character को Store नहीं करते हैं। यदि हम किसी Character को Store भी करते हैं, तो वह Character किसी ना किसी अंक के रूप में ही Computer में Store होता है। Computer में हर Character का एक ASCII Code होता है।

यदि हम किसी Character को Computer की किसी Memory Location पर Store करते हैं, तो वास्तव में हम एक Integer मान को ही Computer की Memory में Store कर रहे होते हैं। लेकिन उस Memory Location पर Stored Character को Output में Display करने के तरीके पर निर्भर करता है, कि वह Character हमें एक Character के रूप में दिखाई देगा या फिर एक अंक के रूप में।

चूंकि ***char*** प्रकार का Data Type Memory में केवल एक Byte की ही Space लेता है, इसलिए जब हमें Computer में बहुत ही छोटी लेकिन चिन्ह वाली संख्या को Store करना होता है, तब इस Data Type के Identifier का प्रयोग कर सकते हैं। इस प्रकार के Identifier में हम -128 से 127 तक की संख्या Store कर सकते हैं। किसी Character प्रकार के Identifier को Declare करने के लिए हमें निम्न तरीके को Use करना होता हैः

```
char studentAge;          or
signed char studentAge;
```

यदि हम इस Identifier में किसी Character को Store करना चाहें, तो Store किए जाने वाले Character को हमें Single Quote में लिखना होता है। यानीः

```
studentAge = '9';
```

इस Statement में हम Variable में 9 Store कर रहे हैं, लेकिन वास्तव में हम यहां पर अंक 9 Store नहीं कर रहे हैं, बल्कि अंक 9 की ASCII Value **57** या Character **9** Store कर रहे हैं। यदि हम इस Variable में अंक 9 Store करना चाहें, तो हमें ये Statement निम्नानुसार लिखना होगाः

```
studentAge = 9;
```

unsigned char

जब हमें Computer में ऐसे मान को Store करना होता है, जो कि बहुत छोटा तो होता है साथ ही कभी भी Minus में नहीं हो सकता है, तब हम इस के Identifier को Declare करते हैं।

उदाहरण के लिए किसी भी Student की Age Minus में नहीं हो सकती है, इसलिए Age को **signed** प्रकार का Declare करने की जरूरत नहीं है, बल्कि Age को **unsigned** प्रकार का Declare किया जाना चाहिए साथ ही चूंकि किसी भी व्यक्ति की Age सामान्यतया 255 साल से अधिक नहीं हो सकती है, इसलिए Age Integer होने के बावजूद Age को ***int*** प्रकार का Declare करने की जरूरत नहीं है। क्योंकि **unsigned char** प्रकार का Identifier **0** से **255** तक के मान की संख्या को Store कर सकता है। unsigned char प्रकार का Identifier Create करने के लिए हम निम्नानुसार Statement लिख सकते हैंः

```
unsigned char studentAge = 20;
```

अलग–अलग प्रकार के Data Type में जो मेमोरी Space बताया गया है, उसका अर्थ यही है कि ज्यादा मेमोरी Space लेने वाले Identifier में बडी संख्या व कम Storage Space लेने वाले Identifier में छोटी संख्या को Store किया जा सकता है।

Data Types Modifiers

ये मानक डाटा टाइप की साईज बदल देते हैं यानी ये डाटा टाइप के आकार में परिवर्तन कर देते हैं। ये कुल चार प्रकार के होते हैं:

- **Signed**
- **Unsigned**
- **Short**
- **Long**

हमने ऊपर इनके प्रयोग से देखा है कि कैसे **double** प्रकार का Identifier Memory में 8 Byte का Space लेता है और double के साथ **short** Modifier Use करने से वह Identifier 10 Byte की जगह Reserve कर लेता है।

विभिन्न प्रकार के Identifier Memory में कितनी Space Reserve करते हैं, इस बात की पूरी जानकारी "C" Language की Library में उपस्थित ***limits.h*** व ***float.h*** नाम की Header Files में दी गई है। लेकिन इन Header Files से इन जानकारियों को Screen पर Display करवाने के लिए पहले हमें "C" Language के Output Function ***printf***() व इसमें प्रयोग किए जाने वाले विभिन्न प्रकार के Control Strings को ठीक से समझना होगा।

Exercise:

1 Data किसे कहते हैं ? Real World में मूल रूप से Data कितने प्रकार के होते हैं?

2 "C" Language में विभिन्न प्रकार के Data को Represent करने के लिए Data को किन दो भागों में बांटा गया है? इन दोनों भागों द्वारा किन–किन Data Types को Represent किया जाता है?

3 किस प्रकार के Real World Data को "C" Language के किस Data Type के Identifier में Store किया जाना चाहिए, इस बात का Decision किस प्रकार से लिया जाता है ?

4 निम्न Data Types के आपसी अन्तर को समझाईएः

A	**int**	**float**
B	**short**	**long**
C	**signed**	**unsigned**

5 Data Type modifiers से आप क्या समझते हैं ? इनका प्रयोग क्यों किया जाता है ? समझाईए।

6 सभी प्रकार के Data types का एक–एक उचित **Variable** व **Constant** Declare कीजिए।

7 निम्न Declarations में के हर Identifier के मान को यदि Output में Display किया जाए, तो इन Identifiers में Store किए गए सभी मानों को ज्यों का त्यों Output में प्राप्त करने के लिए हमें इन में से किन–किन Declarations किस तरह से Modify करना होगाः

A	const int age;	B	signed speed = 125.50
C	short velocity = 1.2e+4	D	long lightSpeed = 3.8e+10
E	unsigned float x = 1.5	F	unsigned long double p=1.5
G	const singed char ;	H	char x = 254;

Control String

जिस तरह से हम "C" Language में विभिन्न प्रकार के Data को Store करने के लिए अलग–अलग Keywords का प्रयोग करके अलग–अलग **Limit** की Memory Location को Reserve किया जाता है, ठीक इसी तरह से अलग–अलग प्रकार के मानों को Access करने के लिए भी हमें अलग–अलग तरह के Control Strings का प्रयोग करना होता है। Control String कुछ ऐसे Characters होते हैं, जिन्हें **%** के साथ Use किया जाता है।

उदाहरण के लिए यदि हम किसी Integer संख्या को Memory में Store करते हैं, तो उस Integer संख्या को Screen पर Display करने के लिए हमें **%d** Control String का प्रयोग करना होता है। इसी तरह से यदि हम Character प्रकार के किसी Data को Screen पर Print करना चाहें, तो हमें **%c** Control String का प्रयोग करना होता है। विभिन्न प्रकार के Data Type के Data को Screen पर Display करने के लिए **printf() Function** के साथ Use किए जाने वाले **Control String** को हम निम्न सारणी द्वारा समझ सकते हैं:

%d Integer Data Type के मान को Display करने के लिए।
%c Character Data Type के मान को Display करने के लिए।
%f Real Number Data Type के मान को Display करने के लिए।
%g Floating Point Real Data Type के मान को दसमलव के बाद केवल एक Digit तक के **Round Off** Form में Display करने के लिए
%i Signed Decimal Integer Data Type के मान को Display करने के लिए।
%u Unsigned Decimal Integer Data Type के मान को Display करने के लिए।
%o Octal Integer Data Type के मान को Display करने के लिए।
%s String Data Type के मान को Display करने के लिए।
%x Hexadecimal Data Type के मान को Display करने के लिए।
%e Real Number Data Type के मान को Display करने के लिए, जबकि संख्या का मान घातांक रूप में हो

विभिन्न प्रकार के Data Type के मानों को Access करने के लिए हमें विभिन्न प्रकार के Control Strings का प्रयोग करना पडता है। किस प्रकार के Identifier को Access करने के लिए किस Control String को Use करना चाहिए, इस बात की जानकारी निम्न सारणी द्वारा प्राप्त की जा सकती है:

Data Type	Control String
signed char|unsigned char	%c
short signed int|signed int	%d
short unsigned int|unsigned int	%u
long signed int	%ld
long unsigned int	%lu

float	%f / %e
double	%lf / %le
long double	%Lf / %Le

float, double या ***long double*** Type के मानों को यदि Normal Form में Display करना हो, तो क्रमशः **%f, %lf** व **%Lf** Control Strings का प्रयोग करते हैं, जबकि यदि इनके मानों को घातांक रूप में Display करना हो, तो इनके लिए हमें क्रमशः **%e, %le** व **%Le** Control Strings का प्रयोग करना होता है।

printf() Function का प्रयोग हम किसी भी प्रकार के Numerical या Alphanumerical मान को Monitor पर Display करने के लिए करते हैं। इस Function में हमें जो भी Message Screen पर Display करना होता है, उस Message को हम String के रूप में Double Quotes के बीच में लिखते हैं। Double Quotes के बीच में लिखा गया Message ज्यों का त्यों Screen पर Display हो जाता है। उदाहरण के लिए यदि हमें Screen पर "**Hello World**" Print करना हो, तो हमें printf() Function में इस Message को निम्नानुसार लिखना होता है:

```
printf("Hello World");
```

इस Statement का Output हमें निम्नानुसार प्राप्त होता है:

```
Hello World
```

यदि हम इसी Statement को निम्नानुसार लिखते हैं:

```
printf("        Hello                    World");
```

जो इस Statement का Output भी हमें निम्नानुसार प्राप्त होता है:

```
        Hello                    World
```

यानी Printf() Statement में हम String को जिस Format में लिखते हैं, Output में हमें वह String उसी Format में दिखाई देता है। लेकिन विभिन्न प्रकार की Calculations के बाद प्राप्त होने वाले Result को Display करने के लिए भी हमें printf() Function का ही प्रयोग करना होता है। इस स्थिति में हमें Display किए जाने वाले Data के Data Type के आधार पर किसी ना किसी Control String का प्रयोग करना पडता है।

जब हम Control String का प्रयोग करके किसी Calculated मान को Screen पर Display करना चाहते हैं, तब हमें हमेंशा Data के Source व Data के Target दोनों को printf() Function में

Specify करना जरूरी होता है, जहां Source वह मान होता है, जिसे Monitor पर Display करना होता है, जबकि Target वह स्थान होता है, जहां पर Data के मान को Display करना है। Target के स्थान पर Display किए जाने वाले Data के Data Type के Control String को Specify करना होता है। इस तरह से यदि हम printf() Function का पूर्ण Syntax देखें तो वह Syntax निम्नानुसार होता है:

Syntax:

printf("Message **cntrlStr1** Message ***cntrlStr2***...Message *cntrlStrN*",
value/Identifir1, ***value/Identifier2*** ... *value/IdentifierN*)

इस Syntax में ***Message*** के स्थान पर हम उस String को लिखते हैं, जिसे ज्यों का त्यों Screen पर Display करना होता है, जबकि ***cntrlStr*** के स्थान पर हम उस Control String का प्रयोग करते हैं, जो **value/Identifier** में Stored Data Type के मान को Display करने में सक्षम होता है।

cntrlStr व value/Identifier दोनों एक दूसरे के समानान्तर होते हैं। यानी cntrlStr1 के स्थान पर value/Ddentifier1 का मान ही Display होगा, cntrlStr2 के स्थान पर value/Ddentifier2 का मान ही Display होगा और cntrlStrN के स्थान पर value/IdentifierN का मान ही Display होगा। इनके क्रम में किसी प्रकार का कोई परिवर्तन नहीं किया जा सकता है।

यानी यदि हम चाहें कि **cntrlStr1** के स्थान पर **value/Identifier2** का मान Display हो, तो बिना printf() Statement में Change किए हुए हम ऐसा नहीं कर सकते हैं। यदि हमें cntrlStr1 के स्थान पर value/Identifier2 का मान Display करना हो, तो हमें ***printf***() Syntax निम्नानुसार लिखना होगा:

Syntax:

printf("Message **cntrlStr1** Message cntrlStr2...Message cntrlStrN",
value/Identifir2, value/Identifier1 ... value/IdentifierN)

निम्न Program द्वारा हम विभिन्न प्रकार के Control Strings को Use करने की प्रक्रिया को समझ सकते हैं:

Program:

```
#include <stdio.h>
#include <conio.h>
main()
{
  printf("\n Integer   = %d", 10);
  printf("\n Character = %c", 'X');
  printf("\n Float     = %f", 13.2);
```

```
    printf("\n Double   = %e", 12365.599999);
    printf("\n Double   = %g", 12365.599999);
    printf("\n String   = %s", "Hello World");
    getch();
  }
```

Output:

```
Integer   = 10
Character = X
Float     = 13.200000
Double    = 1.236560e+04
Double    = 12365.6
String    = Hello World
```

इस Program में Monitor पर String को Display करने की सुविधा प्राप्त करने के लिए हमने **stdio.h** नाम की Header File को अपने Source Program में Include किया है, क्योंकि printf() Function को इसी Header File में Define किया गया है, जो कि Monitor पर Output को Display करने का काम करता है।

getch() Function को **conio.h** नाम की Header File में Define किया गया है, इसलिए हमने getch() की सुविधा को प्राप्त करने के लिए इस Header File को भी अपने Program में Include किया है।

फिर हमने main() Function बनाया है, क्योंकि यही वह Function होता है, जिसमें Computer को दिए जाने वाले विभिन्न Instructions को लिखा जाता है।

हम देख सकते हैं कि सभी printf() Functions में हमने **"\n"** को Use किया है। इस Character को **Backslash Character Constant** कहते हैं। ये Constant Computer को हर Message Print करने के पहले एक **New Line** लेने का Instruction देता है, ताकि Display होने वाला हर Statement एक नई Line में Display हो। यदि हम printf() Function में इस Character Constant का प्रयोग ना करें, तो इस Program का Output हमें एक ही Line में प्राप्त होगा।

इस Program में हम देख सकते हैं कि हर Statement में जिस स्थान पर Control String का प्रयोग किया गया है, Output में उसी स्थान पर वह मान Display हो रहा है, जो मान Source के रूप में Specify किया गया है।

इस Program में हमने हर मान को बिना Memory Allocate किए ही Directly Screen पर Display करने के लिए भेज दिया है। यदि हम चाहें तो सभी प्रकार के मानों को Display करने से पहले उन्हें Memory प्रदान कर सकते हैं और हर Memory को Refer करने के लिए Identifier

Set कर सकते हैं। इस प्रक्रिया को Use करते हुए हम पिछले Program को ही निम्नानुसार Modify कर सकते हैं:

Program:

```
#include <stdio.h>
#include <conio.h>

main()
{
    int Integer = 10;
    char Character = 'X';
    float Float = 13.2;
    double Double = 12365.599999;
    char String[] = "Hello World";

    clrscr();

  printf("\n Integer   = %d", Integer);
  printf("\n Character = %c", Character);
  printf("\n Float     = %f", Float);
  printf("\n Double    = %e", Double);
  printf("\n Double    = %g", Double);
  printf("\n String    = %s", String);
  getch();
}
```

इस Program का Output भी हमें वही प्राप्त होता है, जो पिछले Program का प्राप्त हुआ है। लेकिन इस Program में विभिन्न प्रकार के मान Computer की Memory में प्रत्यक्ष रूप से विद्यमान हैं, जिन्हें किसी दूसरी प्रक्रिया के लिए भी Use किया जा सकता है।

"C" Language में Characters के समूह यानी String को Memory में Store करने के लिए किसी प्रकार का कोई Primary Data Type नहीं है, बल्कि String को Computer की Memory में Store करने के लिए हमें Character प्रकार का एक ***One-Dimensional Array*** Create करना होता है।

Preprocessor Directive

कई बार हमें ऐसी जरूरत होती है जिसमें हम चाहते हैं कि हमारा Source Program Compiler पर Compile होने के लिए Processor पर जाने से पहले कुछ काम करे। इस प्रकार के कामों को परिभाषित करने के लिए हम Preprocessor Directives का प्रयोग करते हैं। Preprocessor Directives की शुरूआत हमेंशा # से होती है और इन्हें हमेंशा Header Files को Include करने वाले Statement के Just नीचे लिखा जाता है। Preprocessor Directives को समझने के लिए हम एक Program देखते हैं, जिसमें हम Output में "**Hello World**" Print करना चाहते हैं। ये Program निम्नानुसार है:

Program:

```
#include <stdio.h>
#include <conio.h>

#define START      main() {
#define PRINT      printf("Hello World");
#define PAUSE      getch();
#define END        }

START              /* Start the program. */
PRINT              /* Display message on the screen.*/
PAUSE              /* Pause the screen to display output. */
END                /* Terminate the program. */
```

Output:

```
Hello World
```

इस Program को Compile करने पर भी हमें वही Output प्राप्त होता है, जो Output हमें पिछले अध्याय में प्राप्त हुआ था। ऐसा इसलिए होता है, क्योंकि जब भी हम इस Program को Compile करते हैं, इस Program में Define किए गए सभी Preprocessor Directives Program के Processor पर Compile होने के लिए जाने से पहले Expand होकर मूल Codes में Convert हो जाते हैं। जब सभी Directives Expand हो जाते हैं, तब ये Program निम्नानुसार Normal Form में आ जाता है:

Program:

```
#include <stdio.h>
#include <conio.h>
main() {                                    /* START */
    printf("Hello World");                  /* PRINT */
```

```
    getch();                                    /* PAUSE */
}                                               /* END  */
```

Compiler अब इस Normal Form में Converted Program को Processor पर Compile होने के लिए भेजता है। चूंकि **#define** के साथ Use किए जाने वाले Directives Program के Processor पर Compile होने के लिए जाने से **पहले** Expand होते हैं, इसलिए इन Directives को ***Preprocessor Directives*** कहा जाता है। विभिन्न प्रकार के Preprocessor Directives के बारे में हम आगे विस्तार से जानेंगे।

Preprocessor Directives का प्रयोग Header Files को Develop करते समय ही सबसे ज्यादा किया जाता है। "C" Language की Library में विभिन्न प्रकार के तकनीकी मानों को सरल रूप में Represent करने के लिए उन्हें Preprocessor Directives का प्रयोग करके एक सरल नाम दे दिया जाता है, ताकि इन मानों को सरलता से याद रखा जा सके व Use किया जा सके।

Preprocessor Directives का प्रयोग करके विभिन्न प्रकार के बडे–बडे तकनीकी नामों व मानों को सरल व छोटे Identifier के रूप में परिभाषित किया जा सकता है। एक बार किसी मान का कोई नाम दे देने के बाद हम उस मान को उसके नाम से Refer कर सकते हैं।

उदाहरण के लिए मानलो कि हमें किसी Program में बार–बार PI के मान **3.142857142857142** की जरूरत पडती है। अब इतने बडे मान को बार–बार विभिन्न Statements में बिना गलती के लिखना, नामुमकिन है। किसी ना किसी Statement में इसको Type करने में Mistake हो ही जाएगी।

इस स्थिति में हम एक Preprocessor Directive का प्रयोग करके इस मान को एक नाम प्रदान कर सकते हैं। इस मान को एक नाम प्रदान कर देने के बाद हमें जिस किसी भी स्थान पर Calculation के लिए इस मान की जरूरत हो, हम उस नाम को Use कर लेते हैं।

जब Program को Compile करते हैं, तब Program Compile होने से पहले उन सभी स्थानों पर, जहां पर Preprocessor का प्रयोग किया गया है, Preprocessor को उसके मान से Replace कर देता है। इस तरह से Program में Typing की वजह से होने वाली गलतियों से बचा जा सकता है। इस प्रक्रिया को निम्न Program में Implement किया गया है।

Program:

```
#include <stdio.h>
#include <conio.h>
#define PI   3.142857142857142

main()
```

```
{
    printf("Value of PI is = %.15e", PI);
    getch();
}
```

जब इस Program को Run किया जाता है, तब Output में **%2.15e** Control String के स्थान पर मान **3.142857142857142** को Represent करने वाले Identifier PI के स्थान पर ये मान Expand हो जाता है और Screen पर घातांक रूप में Display हो जाता है।

चूंकि Float प्रकार का Control String दसमलव के बाद केवल 6 अंकों तक की संख्या को ही Display कर सकता है, इसलिए हमने इस Program के printf() Function में **%.15e** Control String का प्रयोग किया है। ये Control String Compiler को ये Instruction देता है, कि Screen पर Display किए जाने वाले मान में दसमलव के बाद कुल 15 Digits Display होने चाहिएं। ये Program जब Compile किया जाता है, तब Compile होने से पहले निम्नानुसार Form में Convert हो जाता है:

Program:

```
#include <stdio.h>
#include <conio.h>

main()
{
    printf("Value of PI is = %.15e", 3.142857142857142);
    getch();
}
```

जब हम अपनी जरूरत के आधार पर विभिन्न प्रकार के Identifier Declare करते हैं, तब किस प्रकार का Identifier Memory में Minimum व Maximum कितने मान तक की संख्या को Hold कर सकता है, इस बात की जानकारी "C" Language के साथ मिलने वाली "**limits.h**" नाम की Header File में दी गई है।

यदि हम चाहें तो इस Header File को जो कि ...TC\Include नाम के Folder में होती है, Open करके विभिन्न Data Types द्वारा Store की जा सकने वाली Minimum व Maximum Range का पता लगा सकते हैं। इस Header File में विभिन्न Data Type द्वारा प्रदान की जाने वाली Minimum व Maximum Range को कुछ Preprocessor Directives के रूप में Define किया गया है।

इसलिए यदि हम Header File को Open करके ना देखना चाहें, तो निम्नानुसार एक Program बना कर भी हम विभिन्न Data Types द्वारा प्रदान की जाने वाली Minimum व Maximum Range का पता लगा सकते हैं।

चूंकि इन विभिन्न प्रकार के Directives को "**limits.h**" नाम की Header File में Define किया गया है, इसलिए इन Directives को Access करने के लिए हमें "limits.h" नाम की Header File को अपने Program में Include करना जरूरी होता है।

Program:

```
#include <stdio.h>
#include <conio.h>
#include <limits.h>
#include <float.h>
#define _ printf("\n Minimum
#define __ printf("\n Maximum

void main(){
    _ short|short int|signed short|signed short int : %d ", SHRT_MIN );
    __ short|short int|signed short|signed short int : %d ", SHRT_MAX );
    _ unsigned short|unsigned short int : %u", 0 );
    __ unsigned short|unsigned short int : %u ", USHRT_MAX );
    _ int|signed int : %d ", INT_MIN );
    __ int|signed int : %d ", INT_MAX );
    _ unsigned int : %u ", 0 );
    __ unsigned int : %u ", UINT_MAX );
    _ long|long int|signed long|signed long int : %ld ", LONG_MIN );
    __ long|long int|signed long|signed long int : %ld ", LONG_MAX );
    _ unsigned long|unsigned long int : %lu ", 0 );
    __ unsigned long|unsigned long int : %lu", ULONG_MAX );
    _ float : %e", FLT_MIN);
    __ float : %e", FLT_MAX);
    _ double : %e", DBL_MIN);
    __ double : %e", DBL_MAX);
    _ long double : %Le", LDBL_MIN);
    __ long double : %Le", LDBL_MAX);
    }
```

ये Program देखने में बहुत अजीब लग सकता है, लेकिन इस Program में हमने Underscore व Double Underscore Symbol से printf() Function के कुछ Part को Directive के रूप में

परिभाषित कर लिया है। जब इस Program को Compile करते हैं, तब Program Compile होने से पहले Underscore (_) Symbol के स्थान पर "**printf("\n Minimum**" String को व Double Underscore Symbol (__) के स्थान पर "**printf("\n Maximum**" String Replace कर देता है। Preprocess होने के बाद जब Program Compile होकर Run होता है, तब हमें इस Program का Output निम्नानुसार प्राप्त होता है:

Output:

Minimum√ short|short int|signed short|signed short int	: -32768
Maximum√ short|short int|signed short|signed short int	: 32767
Minimum√ unsigned short|unsigned short int	: 0
Maximum√ unsigned short|unsigned short int	: 65535
Minimum√ int|signed int	: -2147483648
Maximum√ int|signed int	: 2147483647
Minimum√ unsigned int	: 0
Maximum√ unsigned int	: 4294967295
Minimum√ long|long int|signed long|signed long int	: -2147483648
Maximum√ long|long int|signed long|signed long int	: 2147483647
Minimum√ unsigned long|unsigned long int	: 0
Maximum√ unsigned long|unsigned long int	: 4294967295
Minimum√ float	: 1.175494e-38
Maximum√ float	: 3.402823e+38
Minimum√ double	: 2.225074e-308
Maximum√ double	: 1.797693e+308
Minimum√ long double	: 3.362103e-4932
Maximum√ long double	: 1.189731e+4932

limits.h नाम की Header File में विभिन्न प्रकार के Integers से सम्बंधित Range की जानकारी होती है, उसी तरह से Float से सम्बंधित विभिन्न प्रकार के Range की जानकारी के लिए हम "**float.h**" नाम की Header File को Open करके देख सकते हैं। इसीलिए हमने हमारे Program में Float व Double से सम्बंधित Range की जानकारी के लिए "**float.h**" नाम की Header File को भी अपने Program में Include किया है।

यदि हम ये जानना चाहें कि विभिन्न प्रकार के Data Type के Identifiers Memory में कितने Bytes की Space Reserve करते हैं, तो इस बात का पता लगाने के लिए हम ***sizeof***() Operator का प्रयोग कर सकते हैं। ये Operator Argument के रूप में उस Identifier या Data Type को लेता है, जिसकी Size को हम जानना चाहते हैं और हमें उस Data Type या Identifier की Size Return करता है। यानी इस Operator के Bracket के बीच में हम जिस Identifier या Data Type को लिख देते हैं, हमें उसी Data Type की Size का पता चल जाता है।

सामान्यतया Integer Data Type के अलावा सभी Data Types सभी प्रकार के Computers में समान Memory Occupy करते हैं, जबकि **Integer,** Memory में Compiler के Register की Size के बराबर Space Reserve करता है।

यदि हम 16-Bit Compiler में 16-Bit Processor पर Program Develop करते या Run करते हैं, तो Integer 16-Bit System में 2-Bytes का होता है जबकि 32-Bit System में Integer की Size 4-Bytes होती है। हम जिस Compiler को Use कर रहे हैं, उस Compiler द्वारा विभिन्न प्रकार के Basic Data Type द्वारा Occupy की जा रही Memory का पता हम निम्न Program द्वारा लगा सकते हैं:

Program

```
#include <stdio.h>
#include <conio.h>

void main()
{
    printf("char       :  %d Bytes\n",  sizeof(char));
    printf("short      :  %d Bytes\n",  sizeof(short));
    printf("int        :  %d Bytes\n",  sizeof(int));
    printf("long       :  %d Bytes\n\n", sizeof(long));

    printf("signed char  :  %d Bytes\n",  sizeof(signed char));
    printf("signed short :  %d Bytes\n",  sizeof(signed short));
    printf("signed int   :  %d Bytes\n",  sizeof(signed int));
    printf("signed long  :  %d Bytes\n\n", sizeof(signed long));

    printf("unsigned char :  %d Bytes\n",  sizeof(unsigned char));
    printf("unsigned short:  %d Bytes\n",  sizeof(unsigned short));
    printf("unsigned int  :  %d Bytes\n",  sizeof(unsigned int));
    printf("unsigned long :  %d Bytes\n\n", sizeof(unsigned long));

    printf("float      :  %d Bytes\n", sizeof(float));
    printf("double     :  %d Bytes\n", sizeof(double));
    printf("long double  :  %d Bytes\n", sizeof(long double));

    getch();
    }
```

Output:

16 – Bit Compiler's Output			32 – Bit Compiler's Output		
char	:	1 Bytes	char	:	1 Bytes
short	:	2 Bytes	short	:	2 Bytes
int	:	**2 Bytes**	**int**	:	**4 Bytes**
long	:	4 Bytes	long	:	4 Bytes
signed char	:	1 Bytes	signed char	:	1 Bytes
signed short	:	2 Bytes	signed short	:	2 Bytes
signed int	:	**2 Bytes**	**signed int**	:	**4 Bytes**
signed long	:	4 Bytes	signed long	:	4 Bytes
unsigned char	:	1 Bytes	unsigned char	:	1 Bytes
unsigned short	:	2 Bytes	unsigned short	:	2 Bytes
unsigned int	:	**2 Bytes**	**unsigned int**	:	**4 Bytes**
unsigned long	:	4 Bytes	unsigned long	:	4 Bytes
float	:	4 Bytes	float	:	4 Bytes
double	:	8 Bytes	double	:	8 Bytes
long double	:	10 Bytes	long double	:	10Bytes

Exercise:

1 Control Strings से आप क्या समझते हैं? विभिन्न प्रकार के Control Strings को समझाते हुए इसे Use करने के तरीके का एक Program द्वारा वर्णन कीजिए।

2 Preprocessor Directives किसे कहते हैं? ये किस प्रकार से काम करता है और एक "C" Program में किस प्रकार से Use किया जा सकता है? एक उदाहरण देकर समझाईए।

3 "C" Language में Supported विभिन्न Data Types की **Range Limit** तथा हर Data Type द्वारा Reserve की जाने वाली **Memory Size** को एक Program द्वारा Screen पर Display कीजिए। साथ ही Program के Flow को भी समझाईए।

Literal

जब हम Computer में किसी Program को Develop करते हैं, तब Program में हमें कई ऐसे Data को भी Store व Access करना होता है, जिनका मान हमेंशा स्थिर रहता है। इस प्रकार के Data को **Literal** या **Constant** कहते हैं। उदाहरण के लिए

1 एक Week में हमेंशा **7** दिन होते हैं।
2 PI का मान हमेंशा **22/7** होता है।
3 एक साल में हमेंशा **12** महीने होते हैं।
4 एक दिन में हमेंशा **24** Hours होते हैं।
5 एक Hour में हमेंशा **60** मिनट होते हैं।

प्रोग्राम के Execution के दौरान Literals के मान में कोई परिवर्तन नहीं होता है। इनका मान सम्पूर्ण प्रोग्राम में स्थिर रहता है। चूंकि Literals हमेंशा स्थिर Data को Represent करते हैं, इसलिए इन मानों को Store करने के लिए Memory में Space Reserve करते समय ही इन मानों को Memory में Store कर दिया जाता है और ऐसी व्यवस्था कर दी जाती है, ताकि इनका मान पूरे Program में किसी भी स्थिति में Change ना किया जा सके। "C" Language में Literals या Constants को तीन भागों में बांटा गया हैः

Integer Constant

इन्हें पूर्णांक संख्याएं भी कहते हैं, क्योंकि इनमें दशमलव वाली संख्याएं नहीं होती हैं। इस प्रकार के Constant में +/- चिन्ह हो सकते हैं। जिस अंक पर कोई चिन्ह न हो वह संख्या Positive होती हैं। जैसे 124, 3223, 545, 23 आदि Positive Integer Constant के उदाहरण हैं। Programming के दौरान बडी संख्याओं को Represent करने के लिए संख्याओं के बीच कोमा का प्रयोग नही किया जाता है। जैसे 1233,33,000 एक गलत स्थिरांक है। Integer Constants को भी मुख्यतः तीन भागों में बांटा जा सकता हैः

Decimal Constant

जब हम Computer में किसी संख्या को 0 से 9 तक की Digits का प्रयोग करके Represent करते हैं, तब हम इस प्रकार के Constant को Decimal Integer Constant कहते हैं। इस तरीके को Number की Decimal Form कहा जाता है। हम हमारे दैनिक जीवन में संख्याओं को इसी रूप में उपयोग में लेते हैं। जब हम किसी Identifier में इस प्रकार के Literal को Assign करना चाहते हैं, तब हमें निम्नानुसार Syntax लिखना होता हैः

```
const int speed = 120;
```

const Keyword का प्रयोग इसलिए किया जाता है, क्योंकि const Keyword का प्रयोग करने पर Create होने वाला Identifier **Constant Identifier** बन जाता है, जबकि const का प्रयोग ना करने पर बनने वाला Identifier **Variable Identifier** होता है। इस Statement में 120 एक **Decimal Literal** है।

Octal Constant

जब हम Computer में किसी संख्या को 0 से 7 तक की Digits का प्रयोग करके Represent करते हैं, तब हम इस प्रकार के Constant को Octal Integer Constant कहते हैं। इस तरीके को Number की Octal Form कहा जाता है और इस तरीके का प्रयोग केवल Electronic Devices जैसे कि Calculator, VCD Player, DVD Player, Remote Control आदि में Number को Represent करने के लिए किया जाता है।

जब हम किसी Number को इस तरीके का प्रयोग करके Represent करते हैं, तब इस मान के Number से पहले English Alphabet के एक Character **0** का प्रयोग किया जाता है, जो बताता है कि Represent होने वाली संख्या Octal Form में है।

हम हमारे दैनिक जीवन में इस तरीके का प्रयोग करके किसी Number को Represent नहीं करते हैं। जब हम किसी Identifier में इस प्रकार के Literal को Assign करना चाहते हैं, तब हमें निम्नानुसार Syntax लिखना होता हैः

```
const int speed = 0120;
```

चूंकि जब हम किसी मान को Octal Value के रूप में Store या Access करना चाहते हैं, तब हमें उस मान के आगे उपसर्ग के रूप में **0** (**Zero**) Add करना जरूरी होता है। इसीलिए हमने मान 120 से पहले Prefix के रूप में **0** (**Zero**) का प्रयोग किया है। इस Statement में **0**120 एक **Octal Literal** है।

Hexadecimal

जब हम Computer में किसी संख्या को 0 से 9 तक की Digits English के a/A, b/B, c/C, d/D, e/E या f/F Characters का प्रयोग करके Represent करते हैं, तब हम इस प्रकार के Constant को Hexadecimal Integer Constant कहते हैं।

इस तरीके को Number की Hexadecimal Form कहा जाता है और इस तरीके का प्रयोग Computer जैसी बडी Digital Electronic Devices में संख्याओं को Represent करने के लिए किया जाता है।

जब हम किसी Number को इस तरीके का प्रयोग करके Represent करते हैं, तब इस मान के Number से पहले **0X/0x** का प्रयोग किया जाता है, जो बताता है कि Represent होने वाली संख्या Hexadecimal Form में है।

हम हमारे दैनिक जीवन में इस तरीके का प्रयोग करके भी किसी Number को Represent नहीं करते हैं। जब हम किसी Identifier में इस प्रकार के Literal को Assign करना चाहते हैं, तब हमें निम्नानुसार Syntax लिखना होता है:

```
const int speed = 0x120;
            or
const int speed = 0X120;
```

चूंकि जब हम किसी मान को Hexadecimal Value के रूप में Store या Access करना चाहते हैं, तब हमें उस मान के आगे उपसर्ग के रूप में **0x** (**Zero with x/X**) Add करना जरूरी होता है। इसीलिए हमने मान 120 से पहले Prefix के रूप में **0x** (**Zero with x/X**) का प्रयोग किया है। इस Statement में **0**120 एक **Hexadecimal Literal** है।

Rules for Representing Integer Constants in a PROGRAM

किसी Program में जब भी हम किसी Integer Constant मान को Represent करते हैं, तब हमें कुछ नियमों का ध्यान रखना होता है। किसी Integer Constant को Represent करने के नियम निम्नानुसार हैं:

- किसी भी Integer Constant में कम से कम एक Digit को होना जरूरी होता है।
- किसी Integer Constant में कोई दसमलव नहीं होता है।
- Integer Constant **Positive** या **Negative** किसी भी प्रकार का हो सकता है।
- यदि किसी Integer Constant के साथ जब किसी चिन्ह का प्रयोग नहीं किया गया होता है, तब By Default वह Integer एक Positive Integer Constant होता है।
- किसी Integer Constant में अंकों को अलग करने के लिए **Blank Space** या **Comma** का प्रयोग नहीं किया जा सकता है।

इन नियमों को ध्यान में रखते हुए ही हमें हमारे Program में किसी Integer Constant को Represent करना होता है। इनमें से किसी भी नियम को Avoid करने पर "C" का Compiler **Compile Time Error** Generate करता है।

जब किसी Program को Compile करते समय Source Code में की गई किसी Typing Mistake के कारण कोई Error Generate होती है, तो इस Error को ***Compile Time Error*** कहते हैं।

हम जिस किसी भी रूप में जो भी स्थिर Integer मान Represent करते हैं, वह मान **Integer Literal** या **Integer Constant** कहलाता है। उदाहरण के लिए आगे दिए जा रहे सभी मान एक Integer Constant मान हैं:

Decimal Form	Octal Form	Hexadecimal Form
123	0123	0x124
+2568	+02568	0x5698
-7812	-01235	0x-4589
-5698	-124589	0x-7895

Floating Point Constant

जब हमें Computer में ऐसे Constant मान को Hold करना होता है, जिसमें दसमलव का प्रयोग होता है, तो इस प्रकार की संख्या को *Floating Point Constant* कहा जाता है। इस प्रकार की संख्या को Real Number Constant या वास्तविक स्थिरांक भी कहते हैं। "C" Language में इसे भी दो रूपों में Represent किया जाता है :

Fractional Form

जब किसी संख्या में स्थित दसमलव से पहले व दसमलव के बाद में दोनों ओर कम से कम एक अंक हो, तो इस प्रकार की संख्या को *Fractional Form Floating Point Constant* कहा जाता है। जैसे 12122.122, 11.22 आदि। इस प्रकार का Literal किसी Constant Identifier को Assign करने के लिए हमें निम्नानुसार Declaration करना होता है:

```
const float lightSpeed = 300000000.00;
```

इस Statement में 300000000.00 एक **Fractional Form Literal** है। चूंकि बडी संख्याओं को हमेंशा सरलता से Use करने के लिए घातांक रूप में Convert करके Use किया जाता है। इसलिए इस Literal को भी हम Exponent Form में Convert करके Use कर सकते हैं।

Exponent Form

जब हम किसी संख्या को घातांक के रूप में Computer में Represent करते हैं, तो उस प्रकार की संख्या को *Exponent Form Floating Point Constant* कहा जाता है। ऐसी संख्या के हमेंशा निम्नानुसार दो भाग होते हैं:

1) **Mantissa** व 2) **Exponent**

इस तरीके का प्रयोग करके बडी–बडी संख्याओं को घातांक के रूप में दर्शाया जाता है। जैसे 1200000000 को घातांक रूप में हम निम्नानुसार भी लिख सकते हैं। 1200000000 = 1.200000000 * 10^{10} जहां 1.2 Mantissa वाला भाग होगा व 10^{10} Exponent वाला भाग हो जाएगा। किसी भी Fractional Form मान को Exponent Form में Convert करने के लिए निम्न सूत्र का प्रयोग कर सकते हैं:

Value	=	Mantissa	e/E	Exponent		
1200000000	=	1.2	+E	10	=	1.2+E10

इस तरह से किसी भी संख्या को घातांक रूप प्राप्त किया जा सकता है। यदि घातांक धनात्मक हो तो +e या +E आता है अन्यथा -e या -E आता है। इस प्रकार का Literal किसी Constant Identifier को Assign करने के लिए हम निम्नानुसार Declaration कर सकते हैं:

```
const float lightSpeed = 3.0+E10;
```

इस Statement में 300000000.00 एक **Exponent Form Literal** है।

Rules for Representing Real Constants in a PROGRAM

किसी Program में जब भी हम किसी Real Constant मान को Represent करते हैं, तब हमें कुछ नियमों का ध्यान रखना होता है। किसी Real Constant को Represent करने के नियम निम्नानुसार हैं:

- किसी भी Real Constant में कम से कम एक Digit को होना जरूरी होता है।
- किसी Real Constant में हमेंशा एक दसमलव होता है और दसमलव के बाद कम से कम एक Digit का होना जरूरी होता है।
- Real Constant भी **Positive** या **Negative** किसी प्रकार का हो सकता है।
- यदि किसी Real Constant के साथ जब किसी चिन्ह का प्रयोग नहीं किया गया होता है, तब By Default वह Real Constant एक Positive Real Constant होता है।
- किसी Real Constant में अंकों को अलग करने के लिए **Blank Space** या **Comma** का प्रयोग नहीं किया जा सकता है।

इन नियमों को ध्यान में रखते हुए ही हमें हमारे Program में किसी Integer Constant को Represent करना होता है। इनमें से किसी भी नियम को Avoid करने पर "C" का Compiler **Compile Time Error** Generate करता है।

हम जिस किसी भी रूप में जो भी स्थिर **Real** मान Represent करते हैं, वह मान **Real Literal** या **Real Constant** कहलाता है। उदाहरण के लिए आगे दिए जा रहे सभी मान Real Constant मान हैं:

Fractional Form	Exponential Form
123.32	12.3e+12
+2568.23	+2.568+e32
-7812.12	-123.5+e5
-5698.21	-1245.89-e10

हम किसी Literal को सामान्य Form में Store करके उसे किसी भी दूसरे Form में भी Display करवा सकते हैं। उदाहरण के लिए यदि हम किसी Integer मान Decimal Form में Store करते हैं, तो उस मान को Display करवाते समय %d, %i, %o या %x Control Strings का प्रयोग करके Decimal, Octal या Hexadecimal Form में Display करवा सकते हैं।

इसी तरह से यदि हम किसी Floating Point Value को किसी Identifier में सामान्य रूप में Store करते हैं, तब भी हम उस मान को सामान्य दसमलव वाली संख्या व घातांक वाली संख्या दोनों ही रूपों में Output में Display करवा सकते हैं। इस पूरी प्रक्रिया को निम्न Program द्वारा समझा जा सकता है:

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    const int Integer = 12345;
    const float Float = 12345.6789;

    printf("\n Integer in Decimal Form     = %d ", Integer);
    printf("\n Integer in Octal Form       = %o ", Integer);
    printf("\n Integer in Hexadecimal Form = %x ", Integer);
    putchar('\n');
    printf("\n Float in Fractional Form  = %f ", Float);
    printf("\n Float in Exponential Form = %e ", Float);

    getch();
}
```

Output

```
Integer in Decimal Form     = 12345
Integer in Octal Form       = 30071
Integer in Hexadecimal Form = 3039

Float in Fractional Form  = 12345.678711
Float in Exponential Form = 1.234568e+04
```

इस Program में हमने केवल एक ही Integer मान को तीन तरीकों से व एक ही Floating Point मान को दो तरीकों से Output में Display करने के लिए अलग–अलग **Control Strings** का प्रयोग किया है।

Output में हम देख सकते हैं कि Control String को बदल देने से हमें Output में प्राप्त होने वाला मान किसी दूसरे Form में दिखाई देने लगता है, जबकि हमें Actual Identifier के मान को Change करने की जरूरत नहीं होती है।

इस Discussion का सारांश ये है कि हम चाहे किसी भी Form में Calculation को Perform करें, उसे Output में Display करने पर हमें किस Form में Output चाहिए, इस बात को printf() Function में अलग–अलग Control String का प्रयोग करके तय किया जा सकता है।

इस Program में हमने ***putchar***() नाम का एक नया Function Use किया है। इस Function को भी **stdio.h** नाम की Header File में ही Define किया गया है। इस Function में हम Argument के रूप में एक Character Pass करते हैं और ये Function उस Character को Screen या Output Device पर भेज देता है।

चूंकि हमने इस Function में Output Screen पर एक New Line प्राप्त करने के लिए **'\n'** Backslash Character Constant को Argument के रूप में Pass किया है, इसलिए ये Function हमें Output में एक New Line Provide करता है। यदि हम Argument के रूप में किसी अन्य Character को इस Function में Pass करते, तो वह Character भी Output में ज्यों का त्यों Print हो जाता है।

चूंकि हमें इस Function में हमेंशा एक ही Character को Argument के रूप में भेजना होता है, इसलिए इस Function में Argument के रूप में Pass किए जाने वाले Character को Single Quotes के बीच लिखकर भेजना होता है।

Character Constant

कई बार हमें कुछ ऐसे Data को Computer में Store करना होता है, जो एक या एक से अधिक Alphanumeric Character के होते हैं। इस प्रकार के Constant मान को *Character Constant* कहा जाता हैं। "C" Language में Character Constant भी तीन तरह के होते हैं:

Single Character Constant

जब कभी हमें Computer में ऐसे सवालों का जवाब Store करना होता है, जो केवल True/False या Yes/No के रूप में होते हैं, तब हम इस प्रकार के सवालों के जवाब को Represent करने के लिए एक Single Character का प्रयोग करते हैं। इस प्रकार के Constant को *Single Character Constant* कहा जाता है।

इसे हमेशा Single Quote द्वारा Represent करते हैं। उदाहरण के लिए मानलो कि हमें किसी Character Identifier में एक Character को Store करना है। इस काम को करने के लिए हमें निम्नानुसार Statement लिखना होगा:

```
char isTrue = 'y';
```

इस Statement में **'y'** एक Character Literal है, जिसे Single Quote में Specify किया गया है।

String Constant

जब हमें Computer में कुछ Characters के एक समूह को Store करना होता है, जो कि एक स्थिर मान को Represent करता है, तो उस स्थिति में हम Alphanumerical Characters के एक समूह Computer में Store करते हैं। इस Characters के समूह को *String Constant* कहा जाता है।

String को हमेंशा Double Quotes के बीच में लिखते हैं। इस प्रकार का Identifier Declare करने के लिए हमें निम्नानुसार एक **One-Dimensional** Array बनाना होता है, क्योंकि "C" Language में String Constant को Hold करने के लिए किसी प्रकार का कोई Standard Data Type नहीं है:

```
const char firstDayOfWeek [ ] = "MONDAY";
const char firstMonthOfYear [ ] = "January";
const char independenceDayOfIndia [ ] = "15-Aug-1947";
```

इन तीनों Statements में "**MONDAY**", "**January**" व "**15-Aug-1947**" String Literals हैं।

Back slash Character Constant

जब हम कोई Program Develop करते हैं, तब उसका Output अच्छे Format में दिखाने के लिए हम कुछ विशेष प्रकार के Character Constants का प्रयोग कर सकते हैं, जिन्हें Back Slash के साथ उपयोग में लिया जाता है। इस प्रकार के Character Constants को *Backslash Character Constant* कहा जाता है। "C" Language में Support किए जाने वाले विभिन्न Backslash Character Constants निम्नानुसार हैं:

'\a'	Bell	**'\b'**	Back Slash
'\f'	Form Feed	**'\n'**	New Line
'\r'	Carriage Return	**'\t'**	Horizontal Tab
'\v'	Vertical Tab	**'\''**	Single Quote
'\"'	Double Quote	**'\?'**	Question Mark
'\\'	Back Slash	**'\0'**	NULL Character

Rules for Representing Character Constants in a PROGRAM

किसी Program में जब भी हम किसी Character Constant मान को Represent करते हैं, तब हमें कुछ नियमों का ध्यान रखना होता है। किसी Character Constant को Represent करने के नियम निम्नानुसार हैं:

- किसी भी Character Constant में अधिकतम एक ही Character को Represent किया जा सकता है।
- किसी Character Constant में Represent किए जाने वाले Character को हमेंशा एक Opening Single Quote के बीच ही लिखा जाता है। Character Constant के दोनों तरफ प्रयोग किया जाने वाला ये Quote हमेंशा Opening Quote ही होना चाहिए।
- Character Constant के रूप में "C" Characterset के किसी भी Character को Represent किया जा सकता है।

इन नियमों को ध्यान में रखते हुए ही हमें हमारे Program में किसी Character Constant को Represent करना होता है। इनमें से किसी भी नियम को Avoid करने पर "C" का Compiler **Compile Time Error** Generate करता है।

हम जिस किसी भी रूप में जो भी स्थिर **Character** मान Represent करते हैं, वह मान **Character Literal** या **Character Constant** कहलाता है। उदाहरण के लिए आगे दिए जा रहे सभी मान Character Constant मान हैं:

```
'A', '2', '\0', '\t', 'y', 'n', '$', '#', '-', '='
```

विभिन्न प्रकार के Backslash Character Constants का प्रयोग सामान्यतया printf() जैसे Output Functions में किया जाता है। इनका प्रयोग करने से हमें Output Screen पर दिखाई देने वाले Output को कुछ हद तक Format करने की सुविधा प्राप्त होती है।

विभिन्न प्रकार के Backslash Character Constants का प्रयोग विभिन्न प्रकार की स्थितियों में किया जाता है। उदाहरण के लिए यदि Output Screen पर New Line की जरूरत हो, तो हम '\n' Character Constant को Use करते हैं।
यदि हमें किसी Error को High Light करना हो, तो हम '\a' Character Constant का प्रयोग कर सकते हैं। इसी तरह से यदि हमें Output Screen पर Horizontal Tab की जरूरत हो, तो हम '\t' Character Constant को Use कर सकते हैं।

Exercise:

1 वर्णन करते हुए Literal को परिभाषित कीजिए।

2 Compile Time Error किसे कहते हैं और ये कब Generate होती है ?

3 Integer Literals को कितने तरीकों से Represent किया जा सकता है ? सभी तरीकों से चार–चार उचित Literals को Represent कीजिए।

4 किसी Program में Integer Constants को Represent करने के नियमों का वर्णन कीजिए।

5 Floating Point Constant मानों को कितने तरीकों से Represent किया जा सकता है ? विभिन्न तरीकों से दो–दो उचित Literals Represent कीजिए।

6 निम्न Floating Point Constants को Exponential Form में Convert कीजिए:

A 12536.369 B -4589.2

C 789.124587369 D 7889.2356

7 निम्न Floating Point Constants को Fractional Form में Convert कीजिए:

A 1.2369e+1 B 8920000e-10

C 7.9E+9 D 2.356E-6

8 किसी Real Constant को Program में Represent करने के नियमों का वर्णन कीजिए।

9 123.54 व 897 को इनके विभिन्न रूपों में Display करने का Program बनाईए।

10 Character Constants कितने प्रकार के होते हैं ? इनका प्रयोग कब किया जाता है ?

11 एक Program द्वारा विभिन्न प्रकार के Backslash Character Constants को समझाईए।

12 Character Constant Represent करते समय किन नियमों को ध्यान में रखना होता है ?

13 विभिन्न प्रकार के Character Constants के चार–चार उदाहरण दीजिए।

Punctuation

कुछ Special Symbols का प्रयोग प्रोग्राम में शब्दों व वाक्यों को अलग करने के लिए किया जाता है। इन्हे Punctuation या Separator कहते हैं। इनका मुख्य काम Program के एक Statement को दूसरे Statement व एक हिस्से को दूसरे हिस्से से अलग करने का होता है।

[ ] Array की Size Define करने में उपयोग होता है।
{ } सभी Functions के Executables Code इन्ही कोष्ठको के बीच लिखे जाते हैं।
( ) ये चिन्ह बताता है कि Use हो रहा Statement एक Function है।
, इसे Separator की तरह Use करते हैं।
; हर Executable Statement का अन्त Semi Colon से ही होता है।
: Label Statement में Use होता है।
***** Pointer Variable के साथ Use होता है।
# Preprocessor Directive है।

इन विभिन्न प्रकार के चिन्हों का प्रयोग हमें समय–समय पर जरूरत के आधार पर करना पडता है। उदाहरण के लिए जब भी हमें Header Files को Include करना होता है, हम # Preprocessor Directive का प्रयोग करते हैं। जब हम कोई नया Function Define करते हैं या किसी Function को Call करते हैं, तब हमें Function के नाम के साथ **( ) Braces** का प्रयोग करना होता है, जैसाकि हम पिछले Programs में करते आ रहे हैं।

जब हम कोई Function Define करते हैं, तब इस कोष्ठक के बीच में Argument List को Specify किया जाता है और इसके बाद Semicolon का प्रयोग नहीं किया जाता है, जबकि किसी Function को Call करते समय हमें इस कोष्ठक में Argument Pass करना होता है और कोष्ठक के बाद Statement का अन्त दर्शाने के लिए Semicolon का प्रयोग करना जरूरी होता है।

ठीक इसी तरह से किसी भी Function के Body की शुरूआत व अन्त को Represent करने के लिए Opening व Closing **Curly Braces { }** का प्रयोग किया जाता है। इन Braces के बीच लिखे गए Statements के समूह को Block Statement भी कहा जाता है। Block Statement की विशेषता ये होती है, कि या तो Block के सभी Statement Execute होते हैं या फिर एक भी Statement Execute नहीं होता है।

Operators

किसी भी प्रोग्रामिंग भाषा में विभिन्न प्रकार के Results प्राप्त करने के लिए विभिन्न प्रकार के Mathematical व Logical Calculations करने पडते हैं। इन विभिन्न प्रकार के Mathematical व Logical Calculations को Perform करने के लिए कुछ Special Symbols का प्रयोग किया जाता है। ये Special Symbols कम्प्यूटर को विभिन्न प्रकार के Calculations करने के लिए निर्देशित करते हैं।

विभिन्न प्रकार के Calculations को Perform करने के लिए Computer को निर्देशित करने वाले चिन्हों को **Operators** कहा जाता है। साथ ही Data को Refer करने वाले जिन Identifiers के साथ ये प्रक्रिया करते हैं, उन Identifiers को इन Operators का **Operand** कहा जाता है। Operators दो तरह के होते हैं:

Unary Operator

कुछ Operators ऐसे होते हैं, जिन्हें कोई Operation Perform करने के लिए केवल एक Operand की जरूरत होती है। ऐसे Operator **Unary Operator** कहलाते हैं। जैसे Minus **(-)** एक Unary Operator है। जिस किसी भी संख्या के साथ ये चिन्ह लगा दिया जाता है, उस संख्या का मान बदल जाता है। जैसे 6 के साथ - चिन्ह लगा देने से संख्या -6 हो जाती है। "C" Language में Support किए गए Unary Operators निम्नानुसार हैं।

&	Address Operator
*****	Indirection Operator
+	Unary Plus
-	Unary Minus
~	Bit wise Operator
++	Unary Increment Operator
--	Unary Decrement Operator
!	Logical Operator

Binary Operators

जिन Operators को काम करने के लिए दो Operands की जरूरत होती है, उन्हें **Binary Operators** कहते हैं। जैसे 2 + 3 को जोडने के लिए Addition Operator (+) को दो **Operands** की जरूरत होती है, अतः Plus एक Binary Operator भी है।

''सी'' Language में विभिन्न प्रकार के Operators को उनके काम के आधार पर निम्न Categories में बांटा गया है:

Arithmetic Operators

इनका उपयोग गणित के संख्यात्मक मानों की गणना करने के लिए किया जाता है। इन Operators की कुल संख्या पांच होती है, जो कि निम्नानुसार है:

```
// A = 10, B = 3, C = ?
//-------------------------------------------------------------
```

1 Addition Operator (+)

ये Operator दो Operands को जोड कर उनका **योगफल** Return करता है। जैसे

C = A + B
C = 10 + 3
C = **13**

2 Subtraction Operator (-)

ये Operator पहले Operand के मान में से दूसरे Operands के मान को घटाने पर प्राप्त होने वाले घटान या **घटाफल** को Return करता है। जैसे

C = A - B
C = 10 - 3
C = **7**

3 Multiplication Operator (*)

ये Operator दोनों Operands के मानों को गुणा करके प्राप्त होने वाले **गुणनफल** को Return करता है। जैसे

C = A * B
C = 10 * 3
C = **30**

4 Division Operator (/)

ये Operator पहले Operands के मान में दूसरे Operand के मान का भाग देकर प्राप्त होने वाले **भागफल** को Return करता है। जैसे

C = A / B
C = 10 / 3
C = **3**

5 Modules OR Reminder Operator (%)

ये Operator पहले Operands के मान में दूसरे Operand के मान का भाग देकर प्राप्त होने वाले **शेषफल** को Return करता है। जैसे

C = A % B
C = 10 % 3
C = **1**

हम विभिन्न प्रकार के Arithmetical Operators को निम्न Program द्वारा Use करके उनके काम करने के तरीके को समझ सकते हैं:

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    int A = 10, B = 3, C;

    C = A + B ;
    printf("\n Addition       = %d", C);
    C = A - B ;
    printf("\n Subtraction     = %d", C);
    C = A * B ;
    printf("\n Multiplication  = %d", C);
    C = A / B ;
    printf("\n Division       = %d", C);
    C = A % B ;
    printf("\n Modules|Reminder = %d", C);
    getch();
}
```

Output:

```
Addition       = 13
Subtraction     = 7
Multiplication  = 30
Division       = 3
Modules|Reminder = 1
```

जब भी हम किसी प्रकार की कोई Calculation करते हैं, Calculation के बाद किसी ना किसी प्रकार का कोई मान Generate होता है। इस मान को Hold करने के लिए हम हमेंशा किसी तीसरे Identifier को Use करते हैं।

जब हमें किसी Calculation से Generate होने वाले मान को किसी Identifier में Store करना होता है, तब हम उस Target Identifier को ***Equal To*** (**=**) Symbol के Left Side में लिखते हैं और Result Generate करने वाली Calculation में भाग ले रहे Identifiers के Expression को Equal To Symbol के Right Side में Specify करते हैं। Equal To Symbol को "C" Language में ***Assignment Operator*** कहा जाता है।

ये Operator अपने Right Side में Perform होने वाली Calculation से Generate होने वाले Resulting मान को अपने Left Side के Identifier में Store करने का काम करता है।

इस Program में सबसे पहले Variable Identifier **A** व **B** के बीच Addition, Subtraction आदि की प्रक्रिया होती है, जिससे कोई ना कोई Resultant मान Generate होता है। मान Generate होने के बाद उस मान को ***Equal To*** Operator Identifier **C** में Assign कर देता है, यानी Resultant मान Identifier **C** में Store हो जाता है। फिर printf() Function द्वारा Identifier **C** में Stored इस Resultant मान को Output में Display कर दिया जाता है।

Exercise:

1 Operator किसे कहते हैं ? Unary व Binary Operators के बीच क्या अन्तर है?

2 Initialization व Assignment के बीच के अन्तर को एक उदाहरण द्वारा समझाईए।

Relational Operators

Real World यानी वास्तविक जीवन में भी हम हमेंशा देखते हैं कि हर काम के साथ किसी ना किसी तरह की कोई शर्त जरूर Associated होती है। उदाहरण के लिए लोग आसानी से चल सकें, इसके लिए Road बनाया जाता है। लेकिन लोग रोड के बीच में नहीं चल सकते हैं। रोड पर चलने के साथ शर्त ये है कि लोगों को हमेंशा Road के Left Side में ही चलना चाहिए।

ठीक इसी तरह से जब हम कोई Program Develop करते हैं, तब हमेंशा ये जरूरी नहीं होता है कि विभिन्न प्रकार के काम करने के लिए सभी Statements को एक क्रम में ही Execute करना होगा। कई बार ऐसी परिस्थितियां होती हैं, जिनमें किसी एक परिस्थिति में किसी एक Statement को Execute करना होता है, जबकि दूसरी परिस्थिति में किसी अन्य Statement को Execute करने की जरूरत होती है।

यानी शर्त (**Condition**) के आधार पर एक ही Program में एक ही Control को एक Statement से दूसरे Statement पर भेजने की जरूरत पड सकती है। ठीक इसी तरह से किसी एक ही Statement को किसी विशेष परिस्थिति (Condition) में बार–बार Execute करना पड सकता है। Programming में इस प्रकार की Situations को Handle करने के लिए कुछ अन्य Operators को Define किया गया है, जिन्हें **Relational Operators** कहते हैं।

जब प्रोग्राम में किसी शर्त के आधार पर दो अलग Statements को Execute करने की जरूरत होती है, जहां पहली स्थिति में किसी एक Statement को Execute करना होता है, जबकि दूसरी स्थिति में किसी दूसरे Statement को Execute करना होता है, तब इस परिस्थिति में दो अलग मानों की आपस में तुलना की जाती है। तुलना करने पर यदि पहली Condition सही होती है, तो पहले Statement को Execute किया जाता है, जबकि पहली Condition गलत होने की स्थिति में किसी दूसरे Statement को Execute किया जाता है।

जब Program में किसी Condition के आधार पर Execute होने वाले Statements का चुनाव करना होता है, तब Condition को Specify करने के लिए हम इन Relational Operators का प्रयोग करते हैं। किसी प्रोग्राम में इन Operators का प्रयोग करके हम ये पता लगाते हैं कि कोई Condition सही है या नहीं। यदि Statement सही (***True***) होती है, तो ये Operators **1** Return करते हैं और यदि Condition सही नहीं होती है (***False***) तो ये Operators **0** Return करते हैं। Relational Operators निम्न हैंः

Operator	Mathematical Symbol	"C" Symbol
Equal to	=	==
Not Equal to	<>	!=
Less then	<	<
Greater then	>	>
Less then or Equal to	<_	<=
Greater then or Equal to	>_	>=

ये Relational Operators के काम करने के तरीके को हम निम्न Program द्वारा समझ सकते हैं। इस Program में हम देख सकते हैं कि जब Condition ***True*** होती है, तब **1** Return होता है और जब Condition ***False*** होती है, तब **0** Return होता है। Condition के आधार पर जिस प्रकार का मान Return होता है, उसे हमने इस Program में Output में Display किया है।

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    printf("\n 10 is equal to 10  [10 == 10] : %d", 10==10);
    printf("\n 10 is less than 100 [10 < 100] : %d", 10<100);
    printf("\n 100 is greater than 10 [100 > 10] : %d", 100>10);
    printf("\n 10 is less than or equal to 10 [10 <= 10] : %d", 10<=10);
    printf("\n 10 is greater than or equal to 10 [10 >= 10] : %d", 10>=10);
    printf("\n 11 is not equal to 10 [11 != 10] : %d", 11!=10);

    putchar('\n');

    printf("\n 10 is equal to 11 [10 == 11] : %d", 10==11);
    printf("\n 100 is less than 10 [100 < 10] : %d", 100<10);
    printf("\n 10 is greater than 100 [10 > 100] : %d", 10>100);
    printf("\n 11 is less than or equal to 10 [11 <= 10] : %d", 11<=10);
    printf("\n 10 is greater than or equal to 11 [10 >= 11] : %d", 10>=11);
    printf("\n 10 is not equal to 10 [10 != 10] : %d", 10!=10);
}
```

Output:

```
10 is equal to 10 [10 == 10] : 1
 10 is less than 100 [10 < 100] : 1
 100 is greater than 10 [100 > 10] : 1
 10 is less than or equal to 10 [10 <= 10] : 1
 10 is greater than or equal to 10 [10 >= 10] : 1
 11 is not equal to 10 [11 != 10] : 1

 10 is equal to 11 [10 == 11] : 0
```

```
100 is less than 10 [100 < 10] : 0
10 is greater than 100 [10 > 100] : 0
11 is less than or equal to 10 [11 <= 10] : 0
10 is greater than or equal to 11 [10 >= 11] : 0
10 is not equal to 10 [10 != 10] : 0
```

हालांकि इस Program को सरल बनाए रखने के लिए हमने Literals का प्रयोग किया है। लेकिन यदि हम चाहें तो विभिन्न प्रकार के मानों को विभिन्न प्रकार के Variables या Constant Identifiers में Store करके उन Identifiers की भी आपस में तुलना कर सकते हैं। ऐसा करने पर भी प्राप्त होने वाले परिणाम में किसी प्रकार का कोई Change नहीं होता है।

उदाहरण के लिए यदि इसी Program में हम तीन Integer प्रकार के Variables **A, B,** व **C** Create करें और उनमें क्रमशः **10, 11,** व **100** Store कर दें, और फिर पिछले Program में हमने जहां–जहां Integer Literal **10** को Use किया है, वहां Identifier **A** को, जहां–जहां Integer Literal **11** को Use किया है, वहां–वहां Identifier **B** को व जहां–जहां Integer Literal **100** को Use किया है, वहां–वहां Identifier **C** को Replace कर दें, तो भी हमें प्राप्त होने वाला Output वही प्राप्त होगा, जो इस Program से प्राप्त हो रहा ळें

Ex/ercise:

1 Relations Operators को समझाईए। ये Operators किस तरह से काम करते हैं और इनका प्रयोग क्यों किया जाता है?

2 एक Program में विभिन्न प्रकार के Relational Operators को Use कीजिए व समझाईए कि ये किस प्रकार से किसी Condition के आधार पर True या False Return करते हैं।

Conditional Operators / Ternary Operator

यह **if . . . else** Conditional Statement का संक्षिप्त रूप है, जिसके बारे में हम अगले Chapter में विस्तार से पढेंगे। इसका Syntax निम्नानुसार होता है:

```
Target = (Condition) ? A : B
```

इस Operator को Use करने पर यदि Braces में दी गई Condition से True Return होता है, तो Target Identifier में Identifier **A** का मान Store हो जाता है। लेकिन यदि Braces में दी गई Condition ***True*** के स्थान पर ***False*** Return करता है, तो **Target** Identifier में Identifier **B** का मान Store हो जाता है।

यदि हम दो संख्याओं में से बडी संख्या प्राप्त करने का Algorithm बनाना चाहें, तो ये Algorithm हम निम्नानुसार बना सकते हैं, जहां Identifier **A** व Identifier **B** वे मान हैं, जिनकी आपस में तुलना करनी है और **Target** Identifier वह Identifier है, जो Condition के आधार पर Identifier A या Identifier B से Return होने वाले मान को Hold करता है।

Algorithm:

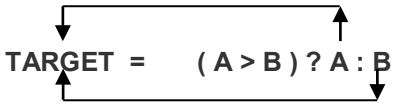
```
CONDITIONAL_OPERATOR(A, B, TARGET)

IF A is greater thane B THEN
    TARGET = A
ELSE
    TARGET = B
```

इसी Algorithm के आधार पर यदि हम Ternary Operator को Define करें, तो हमें Ternary Operator के लिए निम्नानुसार Algorithm प्राप्त होता है:

```
TARGET = (A > B) ? A : B
```

मानलो यदि A = 2, B = 3 व TARGET = ? हो, तो Ternary Operator में इन Identifiers को Place करने पर हमें TARGT Identifier में **3** प्राप्त होगा, क्योंकि Condition (**A>B**) के Execute होने पर ***True*** यानी **1** Return होगा और Condition के ***False*** होने की स्थिति में Identifier **A** का मान Target में Store हो जाएगा। यदि एक चित्र द्वारा Ternary Operator के काम करने के तरीके को Represent करें, तो बनने वाला चित्र निम्नानुसार होगा:



यदि Ternary Operator का प्रयोग करके हम एक ऐसा Program बनाना चाहें, जो दो संख्याओं में से बडी संख्या को प्राप्त करके Output में Print करें, तो हम ये Program निम्नानुसार बना सकते हैं:

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    int num1 = 10, num2 = 30, big;

    big = num1 > num2 ? num1 : num2;

    printf("Biggest Number in %d and %d is = %d", num1, num2, big);

    getch();
}
```

Output:

```
Biggest Number in 10 and 30 is = 30
```

इस Program में हमने printf() Function को अलग तरीके से Use किया है और तरीके का Effect हम Program के Output में देख सकते हैं। Function के पहले Control String के स्थान पर **num1** का, दूसरे Control String के स्थान पर **num2** का व तीसरे Control String के स्थान पर Variable Identifier **big** का मान Print हो रहा है।

सामान्यतया जब हम किसी Variable Type के Identifier को Declare करते हैं, तब उसे Variable नाम से ही सम्बोधित करते हैं और जब हम **const** Keyword का प्रयोग करके Constant Identifier Declare करते हैं, तब उसे केवल Constant नाम से ही सम्बोधित करते हैं। इसलिए अब यहां से हम भी इस प्रकार के Identifiers को इन्हीं सम्बोधनों को उपयोग में लेंगे।

इस Program में printf() Function को इस तरह से Use करने का कारण ये है, कि यदि हम इसी Program में num1 का मान बदल कर **20** व num2 का मान बदल कर **10** कर दें, यानी Program के main() Function के सबसे पहले Statement को यदि हम निम्नानुसार Modify कर दें:

```
int num1 = 20, num2 = 10, big;
```

तो हमें इसी Program से प्राप्त होने वाला Output निम्नानुसार प्राप्त होगा, जो कि पिछले Program के Output से अलग है व ज्यादा सभी तरीके से Information दे रहा है:

Output:

```
Biggest Number in 20 and 10 is = 30
```

यानी printf() Function को इस तरह से Use करके हम एक ही Printf() Function द्वारा अलग–अलग प्रकार के Output प्राप्त कर सकते हैं।

Software Programming के क्षैत्र में हमेंशा दो तरह के लोग होते हैं। एक वे जो विभिन्न प्रकार की समस्याओं को Solve करने के लिए विभिन्न प्रकार के Programs Develop करते हैं। इस प्रकार के लोगों को **Software Programmer** कहा जाता है, जबकि दूसरे प्रकार के लोग वे लोग होते हैं, जो किसी समस्या का समाधान प्राप्त करने के लिए पहले प्रकार के लोगों यानी Programmers द्वारा Develop किए गए Programs को Use करते हैं। Programs को Use करने वाले इस प्रकार के लोगों को **Program User** कहा जाता है।

एक Software Programmer कभी भी किसी User को अपने Source Codes नहीं देता है, बल्कि वह User को केवल Executable Codes प्रदान करता है, ताकि User उसके Program को Use तो कर सके, लेकिन उसमें किसी प्रकार का कोई Modification ना कर सके।

यदि हम इस तरह से सोचें कि हम एक Programmer हैं और हम जो Program बना रहे हैं, उसे कोई User केवल Use ही कर सकेगा, तो हमने अभी तक जितने भी Programs बनाए हैं, उन में से किसी भी Program से User को कोई फायदा नहीं होने वाला है। क्योंकि अभी तक हमने कोई भी ऐसा Program नहीं बनाया है, जिसमें User अपनी जरूरत के आधार पर किसी प्रकार का कोई Result प्राप्त कर सके।

उदाहरण के लिए यदि हम हमारे पिछले Program को ही लें, तो इस Program से उसी स्थिति में दो अलग संख्याओं का Comparison करके सबसे बडी संख्या को Output में Display किया जा सकता है, जब दोनों मानों को Program के Source Codes में Modify करके Program को फिर से Recompile किया जाए, जबकि Source Codes तो हम किसी User को देंगे ही नहीं।

इसलिए ये Program किसी User के लिए तब तक बेकार है, जब तक कि Program के Run होते समय User की जरूरत के आधार पर ये Program User से **Run Time** में Data प्राप्त करके उन्हें Process ना कर सके और सबसे बडी संख्या को Output में Display ना कर सके।

यदि हम सारांश में कहें तो अभी तक के जितने भी Programs हमने बनाए हैं, उन में से किसी भी Program में Input की सुविधा को हमने Add नहीं किया है और बिना Input की सुविधा के एक Program केवल उन्हीं मानों के साथ प्रक्रिया कर सकता है, जिन्हें Program Develop करते समय Initialize किया गया होता है। ऐसे Programs हमेंशा एक ही प्रकार का Output प्रदान करते हैं। ये Programs Game की तरह होते हैं, जो हमेंशा एक ही तरह से Run होते हैं।

Program में Interactivity लाने के लिए व एक ही Program द्वारा विभिन्न प्रकार के मानों के साथ प्रक्रिया करने की क्षमता प्राप्त करने के लिए Program में Input की सुविधा को भी Add करना जरूरी होता है। "C" Language में जिस तरह से Output को Display करने के लिए printf() Function को **stdio.h** नाम की Header File में Define किया गया है, ठीक उसी तरह से Input की सुविधा को प्राप्त करने के लिए ***scanf*()** नाम का एक Function भी इसी Header File में Define किया गया है।

scanf() Function

Computer से जितने भी Devices Connected होते हैं, उन सभी Devices की अपनी स्वयं की Memory होती है, जिसे **Temporary Buffer** कहा जाता है। Keyboard, Monitor, Mouse, Printer आदि सबका अपना Temporary Buffer होता है। हम Keyboard से जब भी किसी Key को Press करते हैं, तो उस Key की Information Directly Computer की RAM में जा कर Store नहीं होती है, बल्कि उस Key की Information सबसे पहले Keyboard के Memory Buffer में Store होती है, जहां से हमारे Computer का CPU उस Key की Information को Computer के RAM में Store करता है।

ठीक इसी तरह से जब हम हमारे Computer के Monitor पर किसी Message को Print करना चाहते हैं, तो वास्तव में हम Message को Print करने के लिए Computer के Monitor नहीं भेज रहे होते हैं, बल्कि हम उस Printable Message को Computer के **Graphics Buffer** में भेज रहे होते हैं, जहां से हमारा Monitor Printable Message की Information को प्राप्त करके Monitor पर Display कर देता है।

जब हम printf() Function को Use करके किसी Message को Monitor पर Display करना चाहते हैं, तब हम उस Message या Data को printf() Function में एक String Argument के रूप में भेज देते हैं। printf() Function उस String Message को Computer की Memory से प्राप्त करके Monitor के **Graphics Buffer** में Store देता है और इस Graphics Buffer में Stored Data को हमारा Monitor अपने Screen पर Display कर देता है।

इसी तरह से जब हम Keyboard से किसी Input को प्राप्त करना चाहते हैं, ***scanf***() Function Keyboard पर Press की गई Keys की Information को Keyboard के **Buffer** से प्राप्त करता है और उन Keys की Information को ***scanf***() Function में Specify किए गए Variable Identifier की Storage Location पर Store कर देता है।

दूसरे शब्दों में कहें तो जब हम Memory में Stored किसी Data को Monitor पर Display करना चाहते हैं, तब printf() Function में विभिन्न Identifiers को Specify करके हम हमारे Computer को ये बताते हैं कि हमें Memory की किस Location पर Stored Data को Screen पर Display करना है और विभिन्न प्रकार के Control Strings का प्रयोग करके हम हमारे Computer को ये

बताते हैं कि विभिन्न Identifiers द्वारा Specify किए जा रहे Data को Monitor के Screen की किस Location पर व किस Format में Display करना है।

इसी तरह से जब हम Keyboard से किसी Data को Input के रूप में प्राप्त करके किसी Memory Location पर Store करना चाहते हैं, तब जिस Data Type के Data को Keyboard से Receive करना चाहते हैं, उस Data Type के Control String को ***scanf***() Function में Specify करते हैं और Keyboard से आने वाले Data को Memory के जिस Storage Location पर Store करना चाहते हैं, ***scanf***() function में उस Storage Location के Variable Identifier का नाम Address Operator (**&**) के साथ Specify करते हैं।

जिस तरह printf() Function से साथ हम विभिन्न प्रकार के Control Strings का प्रयोग करके विभिन्न प्रकार के Identifiers के मानों को Output में Print करते हैं, उसी तरह से विभिन्न प्रकार के Data Type के मानों को Keyboard Buffer से प्राप्त करके विभिन्न प्रकार के Identifiers में Store करने के लिए भी हम विभिन्न प्रकार के Control Strings का प्रयोग कर सकते हैं। printf() Function के साथ जो Control String जिस Data Type से Related होता है, scanf() Function में भी वह Control String उसी Data Type से Associated होता है। scanf() Function के साथ Use किए जा सकने वाले Control Strings निम्नानुसार हैं:

%d Keyboard से Integer Data Type के मान को प्राप्त करने के लिए
%c Keyboard से Character Data Type के मान को प्राप्त करने के लिए
%f Keyboard से Floating Point Real Data Type के मान को प्राप्त करने के लिए
%g Keyboard से Floating Point Real Data Type के मान को प्राप्त करने के लिए
%e Keyboard से Floating Point Real Data Type के मान को प्राप्त करने के लिए
%i Keyboard से Signed Decimal Integer Data Type के मान को प्राप्त करने के लिए
%u Keyboard से Unsigned Decimal Integer Data Type के मान को प्राप्त करने के लिए
%o Keyboard से Octal Integer Data Type के मान को प्राप्त करने के लिए
%s Keyboard से String को प्राप्त करने के लिए
%x Keyboard से Hexadecimal Data Type के मान को प्राप्त करने के लिए
%[...] Keyboard से String को प्राप्त करने के लिए

scanf() Function ***printf***() function की तुलना में एक अधिक Control String को Support करता है। ***scanf***() function का Syntax निम्नानुसार होता है:

Syntax:

scanf("cntrlStr1 cntrlStr2 cntrlStrN", &Identifier1, &Identifier2, &IdentifierN)

इस Syntax में **cntrlStr** ये तय करती हैं कि Keyboard से किसी Data Type का Data ***scanf***() function Receive करेगा, जबकि **Identifier** उस Memory Location को Represent करता है,

जहां पर Keyboard से आने वाले Data को Store करना है। इस Function में भी Control Strings जिस क्रम में Specify किए जाते हैं, उसी क्रम में आने वाले Data भी Memory में Store होते हैं।

उदाहरण के लिए cntrlStr1 Identifier1 से, cntrlStr2 Identifier2 से व cntrlStrN IdentifierN से Associated है, इसलिए Keyboard से आने वाला cntrlStr1 Type का सबसे पहला मान Identifier1 की Storage Location पर Store होगा, दूसरे Number पर आने वाला cntrlStr2 Type का मान Identifier2 के Memory Location पर Store होगा और सबसे बाद में आने वाला cntrlStrN Type का मान IdentifierN नाम के Identifier द्वारा Represent होने वाली Memory Location पर Store होगा।

& Operator को **Address Operator** कहा जाता है। ये एक Unary Operator है। ये Operator हमेंशा उस Identifier के Memory Location का Address Return करता है, जिसके साथ इसे Use किया जाता है।

जब हम Program के **Run Time** के Keyboard से किसी Data को Receive करके उस पर Processing करना चाहते हैं, तब ***scanf***() Function द्वारा Computer को हमें दो बातें बतानी पडती हैंः पहली ये कि हम Keyboard से किस प्रकार के Data को Read करना चाहते हैं। Keyboard से Read किए जाने वाले Data के Data Type को Specify करने के लिए उपयुक्त Control String का प्रयोग किया जाता है।

Computer को दूसरी बात ये बतानी होती है, कि Keyboard से आने वाला Data Memory की किस Location पर Store होगा। यानी दूसरी बात के रूप में हमें Computer को उस Memory Location का Address बताना होता है, जहां पर हम Keyboard से आने वाले Data को Store करना चाहते हैं।

जैसाकि हमने पहले भी कहा कि **&** Operator किसी भी Identifier का Address Return करने का काम करता है, इसलिए हमें जिस Identifier की Memory Location पर Keyboard से आने वाले Data को Store करना होता है, उस Identifier के नाम के पहले हम & Operator का प्रयोग उस Identifier को ***scanf***() Function में Specify कर देते हैं।

चूंकि ***scanf***() Function का प्रयोग Keyboard से Input प्राप्त करने के लिए किया जाता है, इसलिए इस Function का प्रयोग करने से पहले हमें उस Data Type का एक Variable Identifier Create करना जरूरी होता है, जिसमें हमें scanf() Function द्वारा Keyboard से आने वाले Data को Store करना चाहते हैं।

बिना Variable Create किए हुए, हम scanf() Function का प्रयोग नहीं कर सकते हैं, क्योंकि इस Function में हमें उस Identifier का नाम Address Operator के साथ Specify करना पडता है, जिसकी Memory location पर हम Keyboard से आने वाले मान को Store करना चाहते हैं। यदि

हम बिना Variable Create किए हुए scanf() Function को Use करते हैं, तो "C" का Compiler Compile Time Error Generate करके हमें ऐसा करने से रोक देता है।

चलिए, एक उदाहरण द्वारा scanf() Function को Use करना सीखते हैं। मानलो कि हम Keyboard से किसी Student की **Age** को Read करना चाहते हैं और उस Age में 10 जोडकर Resultant मान को Screen पर Display करना चाहते हैं। इस समस्या का Algorithm निम्नानुसार बनाया जा सकता है:

Algorithm :

```
SIMPLE_INPUT(AGE, RESULT)
Where:
    AGE is the age of student and
    RESULT is the modified age of the student.

1   START                          [Start the program.]
2   READ AGE                       [Get AGE from keyboard.]
3   PROCESS RESULT = AGE + 10
4   PRINT RESULT
5   END                            [End the program.]
```

इसी Algorithm के आधार पर हम "C" भाषा में Program भी बना सकते हैं, जिसमें सबसे पहले हमें ये तय करना है कि हमें किस प्रकार का Data Computer की Memory में Process करने के लिए Store करना है। चूंकि Age एक **Unsigned** Type का मान होता है, जो कि कभी भी Minus में या Negative Type में नहीं हो सकता है, साथ ही Age एक ऐसा मान होता है, जो बहुत ही छोटा होता है, क्योंकि किसी भी व्यक्ति की Normal Age 100-150 साल से ज्यादा नहीं हो सकती है, इसलिए हम Age को Store करने के लिए Unsigned Character Type का Variable Identifier Create कर सकते हैं, क्योंकि इस Data Type के Identifier की Range 0 से 255 तक होती है, जिसमें किसी की भी Age आसानी से Store हो सकती है। अब यदि हम इस समस्या का "C" Program बनाना चाहें, तो ये Program निम्नानुसार होगा:

Program

```c
#include <conio.h>
#include <stdio.h>

main()
{
    /* Declaration Section */
    unsigned char age, result;
    clrscr();
```

```
    /* Input Section */
    printf("Enter Age of the student : ");
    scanf("%u", &age);

    /* Process Section */
    result = age + 10;

    /* Output Section */
    printf("After 10 year, student will be %u years old", result);

    getch();
}
```

हमें एक "C" Program में जितने भी Identifiers को Use करना होता है, उन सभी Identifiers को Declaration Section में ही Declare करना जरूरी होता है। चूंकि हम हमारे इस Program में Keyboard से Input लेना चाहते हैं, इसलिए User को एक Message देकर ये बताना जरूरी होता है, कि Program को काम करने के लिए किस प्रकार के मान की जरूरत है।

इसीलिए Input Section में scanf() Function को Use करने से पहले हमने एक printf() Statement को Use करके User को Student की Age Input करने का Message दिया है। जब हम इस Program को Compile करके Run करते हैं, तो Program के Run होते ही User को निम्नानुसार ये Message दिखाई देता है और Data प्राप्त करने के लिए Cursor Blink करने लगता है:

```
Enter Age of the student : _
```

यदि printf() Statement द्वारा ये Message Print ना करें, तो Output में Black Screen पर केवल Cursor Blink करता हुआ ही दिखाई देता है और User को पता ही नहीं लगता कि उसे करना क्या है। जहां पर Cursor Blink कर रहा है, वहां पर User जो भी मान Input करता है, उस मान को scanf() Function Scan करता है।

मानलो User ने इस स्थान पर **15** Input किया, तो scanf() function इस 15 को Scan करेगा और इस मान को उस **age** नाम के Variable के Memory Location पर भेज देगा, जिसका नाम **&** Address Operator के साथ scanf() Function में Specify किया गया है।

scanf() Function जैसे ही Keyboard से आने वाले मान को Computer की उस Memory Location पर Store करता है, जिसका नाम **age** है, वैसे ही Input का काम समाप्त हो जाता है। उसके बाद Computer Program के अगले Statement को Execute करके Age में 10 जोडता है

और इसके बाद के printf() Statement द्वारा Resultant मान को Screen पर निम्नानुसार Form में Print कर दिया जाता है:

```
After 10 year, student will be 25 years old
```

जब ये Program पूरी तरह से Run हो जाता है, तब इसका Output हमें निम्नानुसार प्राप्त होता है:

Output:

```
Enter Age of the student : 15
After 10 year, student will be 25 years old
```

अब यदि हम दो संख्याओं में से बडी संख्या निकालने वाले Program को Modify करना चाहें, जिसमें User स्वयं अपनी इच्छानुसार दोनों संख्याओं को Input करे और Program, Input की गई दोनों संख्याओं को Compare करके बडी संख्या को Output में Print करे, तो इस Program को Develop करने के लिए हम निम्न Algorithm का प्रयोग कर सकते हैं:

Algorithm:

```
BIG_IN_2(A, B, BIG) Where:
A is the first number.
B is the second number. and
BIG is the biggest number between A and B.

START
READ A, B              [Get values from keyboard to be compare.]
IF A is greater than B THEN  [Process: Compare to get biggest. ]
    BIG = A
ELSE
    BIG = B
PRINT BIG              [Display biggest value on the monitor. ]
EXIT
```

इस Algorithm के आधार पर यदि हम "C" Program बनाना चाहें, तो Program को निम्नानुसार बनाया जा सकता है:

Program

```
#include <stdio.h>
#include <conio.h>

main()
```

```
{
    /* Declaration Section */
    long double A, B, BIG;

    /* Input Section */
    printf("Enter first value :");
    scanf("%Lf", &A);
    printf("Enter second value  :");
    scanf("%Lf", &B);

    /* Process Section */
    BIG = (A > B) ? A : B;

    /* Output Section */
    printf("\n Biggest Value is   : %Lf", BIG);
    getch();
}
```

Output:

```
Enter first value  : 12457889562312.323232
Enter second value : 1223564574898.121212

Biggest Value is   : 12457889562312.323230
```

इस Program में हमने **long double** Type के Variable Declare किए हैं, इसलिए इनमें Value Input करने के लिए हमें ***scanf***() Function **%Lf** Control String की जरूरत होती है। इसी तरह से इन Variables में Stored Values को Display करने के लिए भी हमें printf() Function में **%Lf** Control String को Use करना होता है।

यदि हम इन Identifiers में घातांक रूप में मानों को Input करना चाहें या फिर इन Identifiers में Stored मानों को Output में Display करने के लिए घातांक रूप का प्रयोग करना चाहें तो, दोनों ही स्थितियों में हमें **%Lf** Control String के स्थान पर **%Le** Control String का प्रयोग करना जरूरी होता है।

इस Program में हमने **long double** प्रकार के Identifiers इसलिए लिए हैं, ताकि हम बडी से बडी संख्या को इसमें Store कर सकें। इस Program में भी Program के Run Time में Keyboard से Input प्राप्त करने के लिए हमने उसी Process को Use किया है, जिस Process को पिछले Program में Use किया था।

यानी सबसे पहले एक printf() Statement द्वारा User को ये Message प्रदान किया है, कि वह पहला मान Input करे। फिर ***scanf***() Function का प्रयोग करके User द्वारा प्रदान किए गए Input को Accept करके उस **A** नाम के Variable में Store किया जिसका प्रयोग **&** Address Operator के साथ किया गया है। इसी तरह से एक और Message दे कर दूसरे Variable के लिए भी User से Input प्राप्त किया।

scanf() Function का प्रयोग करके हम एक ही बार में एक से ज्यादा Variables में मान Store कर सकते हैं। scanf() Function को इस प्रकार से Use करने की जरूरत तब पडती है, जब कई मान एक साथ एक Group के रूप में किसी विशेष सूचना को Represent करते हैं।

उदाहरण के लिए यदि Keyboard से **Date** या **Time** Input करना हो, तो Date या Time को हम अलग–अलग टुकडों में Input नहीं कर सकते हैं। ऐसे ***Group of Data*** को हमें एक साथ Input करना होता है। एक Date या Time में हमेंशा तीन हिस्से होते हैं, जो क्रमशः ***Day, Month, Year*** या ***Hour, Minutes, Seconds*** को Represent करते हैं। Keyboard से जब इस प्रकार के Data को Read करना होता है, तब Data या Time Input करने का एक ही Message दिया जाता है और तीनों मानों को एक साथ Input कर दिया जाता है।

निम्न Program द्वारा scanf() Function को इस प्रकार Use करने की कार्य–विधि को ज्यादा अच्छे तरीके से समझा जा सकता है। ये Program User से उसकी Date Of Birth (DOB) व Current Date Input करने के लिए कहता है। जब User उसकी DOB व Current Date Input कर देता है, तब Program Output के रूप में उस User की Current Age Display करता है।

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    /* Declaration Section */
    int dob_DD, dob_MM, dob_YYYY;
    int cur_DD, cur_MM, cur_YYYY;
    char dummy;
    int age;

    /* Input Section */
    printf("Enter Date of Birth in DD/MM/YYYY Format : ");
    fflush(stdin);
    scanf("%2d%1c%2d%1c%4d", &dob_DD, &dummy, &dob_MM, &dummy,
&dob_YYYY);
```

```
    printf("Enter Today's Date in DD/MM/YYYY Format  : ");
    fflush(stdin);
    scanf("%2d%1c%2d%1c%4d", &cur_DD, &dummy, &cur_MM, &dummy,
&cur_YYYY);

    /* Process Section */
    age = cur_YYYY - dob_YYYY;

    /* Output Section */
    printf("\n Your Date of Birth is : ");
    printf("%d%c%d%c%d", dob_DD, '/', dob_MM, '/', dob_YYYY);
    printf("\n And you are %d years old now", age);
}
```

Output:

```
Enter Date of Birth in DD/MM/YYYY Format : 06/03/1982
Enter Today's Date in DD/MM/YYYY Format  : 11-04-2008

 Your Date of Birth is : 6/3/1982
 And you are 26 years old now
```

Description:

इस Program को Run करते ही ये Program हमे DD/MM/YYYY Format में Birth Date Input करने के लिए कहता है। जैसे ही हम Birth Date Input करते हैं, ये Program DD/MM/YYYY Format में ही हमसे Current Date Input करने के लिए कहता है। जैसे ही हम Current Date भी Input करते हैं, ये Program हमें हमारा **Date Of Birth** व हमारी Current Age Screen पर Display कर देता है।

इस Program में हमने कई नए Concepts Use किए हैं, लेकिन ये Program पूरी तरह से **Error Proof** नहीं है, क्योंकि ये जिस Format में Date Input करने के लिए कहता है, हमें उस Format को पूरी तरह से Follow करना पडता है। यानी हम Date Of Birth **06/03/1982** को **6/3/1982** Format में Input नहीं कर सकते हैं। यदि हम ऐसा करते हैं, तो हमारा Program हमें सही Output नहीं देता है।

जैसाकि हम देख सकते हैं, कि Date एक ऐसा Data है, जिसके होते तो तीन हिस्से हैं, लेकिन इसके तीनों हिस्सों को एक ही बार में Input करना जरूरी होता है। हम देख सकते हैं कि इस Date में Day, Month व Year के अलावा एक और चौथा हिस्सा भी है, जो Day, Month व Year को आपस में एक दूसरे से अलग रखता है।

scanf() Function जब एक ही बार में एक से ज्यादा मानों को Input के रूप में प्राप्त करना चाहता है, तब एक scanf() द्वारा जितने Data Computer की Memory में Store करने होते हैं, उन सभी मानों के Control Strings के साथ उनके Variables को scanf() Function में ठीक उसी तरह से Specify किया जाता है, जिस तरह से printf() Function द्वारा एक से अधिक Identifiers के मानों को Output में Display करने के लिए किया जाता है।

इन दोनों Functions में अन्तर केवल इतना होता है कि scanf() Function में Specify किए जाने वाले सभी Identifiers Keyboard Buffer से अपना मान प्राप्त करते हैं, और सभी मानों को Variables की Reserved Memory Location पर भेजने के लिए इन Variables के साथ Address Operator का प्रयोग किया जाता है।

इस Program में हमने scanf() Function में Use किए जाने वाले Control Strings को थोडा अलग तरीके से Use किया है। scanf() Function के इस तरीके से Input लेने की प्रक्रिया को **Formatted Input** कहते हैं।

चूंकि एक Date के पहले दो अंक Day को Represent करते हैं, इसलिए Input किए जाने वाले Date के पहले दो Characters को ही हमें **dob_DD** व **cur_DD** Variable में Store करना होता है।

इस जरूरत को पूरा करने के लिए हमने पहले Control String के साथ एक Digit 2 का प्रयोग %2s के रूप में किया है। जब हम इस तरह से Control String Use करते हैं, तब Compiler Keyboard से आने वाले Input में से केवल पहले दो अंकों को ही dob_DD व cur_DD में Store करता है।

चूंकि तीसरा Character एक Separator के रूप में काम कर रहा है जो Day को Month की Digit से अलग करता है, इसलिए %1c Control String का प्रयोग करके इस तीसरे Character को हमने dummy नाम के एक Character प्रकार के Variable में Store कर दिया है।

अब Input के रूप में आने वाले अगले दो Digits Month को Represent करते हैं। केवल इन दो Digits को प्राप्त करके **dob_MM** व **cur_MM** में Store करने के लिए हमने फिर से %2d का प्रयोग किया है और Month को Year से Separate करने वाले Separator को फिर से %1c Control String द्वारा dummy नाम के Variable में Store कर लिया है। फिर अन्तिम 4 Digits को **dob_YYYY** व **cur_YYYY** Variable में Store करने के लिए हमने %4d Control String का प्रयोग किया है।

इस Program में हमने निम्नानुसार एक Statement का scanf() Function से पहले प्रयोग किया है:

```
fflush(stdin);
```

ये Function एक विशेष काम करता है। जब हम Keyboard से Keys को Press करते हैं, तब जरूरत के आधार पर विभिन्न Characters विभिन्न Variables में Store हो जाते हैं। लेकिन कई बार जब हम Formatted Input का प्रयोग करते हैं, तब Keyboard से चाहे जितने Characters Input किए जाएं, Variable में Control String में Use किए गए मान के अनुसार कुछ ही Characters Store होते हैं, शेष Characters Keyboard के Buffer में ही पडे रहते हैं।

यदि हम Keyboard के Buffer में पिछले Input के बचे हुए Characters को Clear किए बिना ही scanf() Function को Use करते हैं, तो कई बार scanf() Function User से कोई मान Input करने के लिए नहीं कहता है, बल्कि Keyboard के Buffer में Stored Characters को ही Use कर लेता है, जिससे Program का Output सही नहीं आता। इस स्थिति में ये Statement Keyboard के Buffer में Stored बचे हुए Characters को Clear करने का काम करता है, ताकि User को सही Output प्राप्त हो।

कई बार हमें ऐसी जरूरत भी पड जाती है, जिसमें हम एक ही scanf() Function द्वारा एक से ज्यादा Variables में मान तो Store करना चाहते हैं, लेकिन किसी Formatted Input Process को Use करना नहीं चाहते हैं। इस स्थिति में विभिन्न Identifiers में मानों को Store करने के लिए भी scanf() Function को तो समान तरीके से ही Use किया जाता है। अन्तर केवल इतना होता है कि scanf() Function में Use किए जाने वाले Control Strings को Simple ही रखा जाता है।

चूंकि scanf() Function की एक विशेषता ये है कि ये Function Blank Space से Terminate हो जाता है। इसलिए यदि हम किसी मान को Input करते समय Space या Enter Key द्वारा कई मानों को अलग–अलग कर दें, तो Input किया गया मान scanf() Function में Specify किए गए विभिन्न Variables में Store हो जाते हैं। उदाहरण के लिए हम यहां दो संख्याओं को जोडने का एक Program बना रहे हैं, जिसमें एक ही scanf() Function द्वारा दोनों मानों को Input किया जा रहा है।

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    /* Declaration Section */
    int firstVal, secondVal, result;

    /* Input Section */
    printf("Enter First and Second Values ");
    scanf("%d%d", &firstVal, &secondVal);
```

```
    /* Process Section */
    result = firstVal + secondVal;

    /* Output Section */
    printf("\n Total of %d and %d is = %d ", firstVal, secondVal, result);

    getch();
}
```

Output 1st :

```
Enter First and Second Values :  10 20 (Blank Space between values)
Total of 10 and 20 is 30
```

Output 2nd :

```
Enter First and Second Values :  10 (Pressed Enter between values)
20
Total of 10 and 20 is 30
```

Exercise:

1 printf() व scanf() Function के अन्तर व समानताओं का वर्णन कीजिए, तथा दोनों की कार्यप्रणाली को एक उचित उदाहरण द्वारा विस्तार से समझाईए।

2 एक Program बनाईए जो Input के रूप में User से Year प्राप्त करे और Output के रूप में User को ये बताए कि Input किया गया Year Leap Year है या नहीं। Leap Year एक ऐसा Year होता है, जिसमें हर चौथे साल February **29** दिन की होती है।

3 एक Program बनाईए जिसमें Input के रूप में User से एक अंक प्राप्त करता है और Output के रूप में User को ये बताता है कि Input किया गया मान सम है या विषम। जिस संख्या में दो का भाग पूरा–पूरा चला जाता है, वह संख्या सम संख्या होती है।

4 किसी शहर के Temperature को Centigrade के रूप में Input करो और इस मान के Fahrenheit मान को Output में Print करो। **(Fahrenheit = 1.8 * Centigrade + 32;)**

5 एक Program बनाईए जिसमें User **Days** की संख्या Input करता है और Program उन Days को **Month** व **Reminder Days** में Convert करता है। उदाहरण के लिए यदि User Program में 50 Input करता है, तो Output में "**1 Month 20 Days**" Display होना चाहिए।

6 दो राज्यों के बीच की दूरी को KM में Input करो और इस Input किए गए इस मान को Meters, Centimeters, Feets व Inches के रूप में Convert करके Screen पर Print करो।

7 किसी Employee की Basic Salary Keyboard से Input करो और इस Basic Salary के आधार पर 20% Provident Fund(PF), 30% Dearness Allowance (DA) व 15% House Rent Allowance (HRA) Calculate करो। अब इस Calculation से प्राप्त Result के आधार पर उस Employee की Gross Salary ज्ञात करो। जब Basic Salary में विभिन्न प्रकार के Allowances, Funds आदि को जोड दिया जाता है, तब प्राप्त होने वाली Salary को **Gross Salary** कहते हैं।

8 एक आयत की लम्बाई व चौडाई Keyboard से Input करो और इस आयत का क्षैत्रफल तथा परिमाप ज्ञात करने का Program Algorithm की मदद से बनाओ।

9 एक वृत का क्षैत्रफल व परिमाप ज्ञात करने का Algorithm बनाओ और इस Algorithm के आधार पर Program Create करो जबकि Circle का Radius Keyboard से Input किया जाए।

10 किसी त्रिभुज का क्षैत्रफल ज्ञात करने का Program बनाओ जिसकी भुजाएं क्रमशः A, B व C हैं तथा त्रिभुज का क्षैत्रफल ज्ञात करने का सूत्र निम्नानुसार हैः

$$\text{Area} = \sqrt{S(S-A)(S-A)(S-C)} \qquad \text{Where } S = A + B + C / 2$$

11 Keyboard से एक चार Digit की संख्या Input करो और उस संख्या के First व Last Digit के योग को Output में Print करने का Program लिखों

12 यदि Keyboard से दो संख्याओं को Input किया जाए, तो दोनों संख्याओं को Exchange या Swap करने का Program लिखो। साथ ही इस Program के Algorithm को विस्तार से समझाओ। जब दो मानों को आपस में एक दूसरे के स्थान पर Exchange करके Store किया जाता है, तो इस प्रक्रिया को **Swapping** करना कहते हैं।

Logical Operators

Program में कोई ऐसी स्थिति होती है, जिसमें किसी एक Condition के आधार पर दो में से किसी एक काम को पूरा करना होता है, तब हम Relational Operators का प्रयोग करके Conditions को Check करते हैं।

लेकिन कई बार Program में ऐसी परिस्थितियां बन जाती हैं, जिसमें एक से अधिक Conditions के आधार पर किसी एक काम को पूरा करना होता है। जब किसी Program में इस प्रकार की परिस्थिति पैदा हो जाती है, जिसमें दो या दो से अधिक Conditions के साथ प्रक्रिया करके परिणाम प्राप्त करना होता है, तब Logical Operators का उपयोग किया जाता है।

"C" Language में मुख्यतः तीन Logical Operators होते हैं। चूंकि Logical Operators Binary Operators होते हैं, इसलिए इन Operators के साथ हमेंशा दो Operands होते हैं, साथ ही Logical Operators जिन दो Operands के आधार पर Operation Perform करके Result Generate करना चाहते हैं, उनमें भी कोई ना कोई Relational Operator Included होता है।

AND (&&)

जब Logical Operator के दोनों तरफ की Condition ***True*** होती है, तब ये Logical Operator ***True*** या **1** Return करता है। यदि Logical Operator के दोनों तरफ की Conditions में से किसी एक भी Condition द्वारा **0** या ***False*** Return हो रहा हो, तो ये Logical Operator भी ***False*** Return करता है। जैसेः

```
X = (10 > 5) && (5 > 3)
```

ये Statement Identifier X में 1 यानी True Store करेगा, क्योंकि इस Statement के Execute होने पर सबसे पहले Logical AND Operator के Left Hand Side में स्थित Expression (10 > 5) Execute होगा, जो केवल उस स्थिति में True Return करता है, जब 10 का मान 5 के मान से बडा होता है।

चूंकि 10 हमेंशा ही 5 से बडा होता है, इसलिए ये Expression True Return करता है। फिर Logical AND Operator के Right Side की Condition (5 > 3) Check होती है, जो उस स्थिति में True या 1 Return करता है, जब 5 का मान 3 से ज्यादा होता है।

चूंकि यहां भी 5 का मान हमेंशा ही 3 से ज्यादा होता है, इसलिए ये Expression भी True या 1 Return करता है। अब यदि हम Logical Operator के उपरोक्त Expression को Represent करें, तो इस Statement को निम्नानुसार Represent कर सकते हैंः

```
X = 1 && 1
```

"C" Language में 0 के अलावा किसी भी संख्या को True ही माना जाता है, फिर चाहे संख्या Positive हो या Negative, इसलिए इस Expression में यदि हम देखें तो Logical AND Operator के दोनों और True या 1 है, अतः ये Logical AND Operator भी True या 1 ही Return करेगा और Variable Identifier X में **1** यानी **True** Store हो जाएगा।

OR (||)

इस Logical Operator के दोनों तरफ की Condition में से यदि किसी एक तरफ की Condition भी ***True*** होती है, तब भी ये Logical Operator ***True*** या **1** Return करता है। यदि Logical Operator केवल एक ही स्थिति में ***False*** Return करता है, जब इस Logical Operator के Left Hand Side व Right Hand Side दोनों तरफ की Conditions ***False*** होती हैं। जैसेः

```
X = (10 < 5) || (5 < 3)
```

ये Statement Identifier X में 0 यानी False Store करेगा, क्योंकि इस Statement के Execute होने पर सबसे पहले Logical OR Operator के Left Hand Side के Expression (10 < 5) का Execution होता है और ये Expression उस स्थिति में True Return करता है, जब 10 का मान 5 के मान से छोटा होता है।

चूंकि 10 हमेंशा ही 5 से बडा होता है, इसलिए ये Expression False Return करता है। फिर Logical OR Operator के Right Side की Condition Check होती है, जो उस स्थिति में True या 1 Return करता है, जब 5 का मान 3 से कम होता है। चूंकि यहां भी 5 का मान हमेंशा ही 3 से ज्यादा होता है, इसलिए ये Expression भी False या 0 Return करता है। अब यदि हम Logical Operator के उपरोक्त Expression को Represent करें, तो इस Statement को निम्नानुसार Represent कर सकते हैंः

```
X = 0 || 0
```

इस Expression में Logical OR Operator के दोनों और False या 0 है, अतः ये Logical OR Operator False या 0 ही Return करेगा और Variable Identifier X में **0** यानी **False** Store हो जाएगा।

NOT (!)

ये एक ऐसा Unary Logical Operator है। इस Operator को काम करने के लिए केवल एक ही Operand की जरूरत होती है। जिस Identifier के साथ इस Operator को Use किया जाता है, ये Operator उस Identifier की Condition को Invert कर देता है।

यानी यदि किसी Expression से ***True*** Return हो रहा हो, तो इस Operator का प्रयोग करने से वह ***False*** Return करने लगेगा और यदि किसी Expression से ***False*** Return हो रहा हो, तो उस Expression में इस Operator का प्रयोग करने पर वह Expression ***True*** Return करने लगेगा। इस प्रक्रिया को हम निम्नानुसार Expression द्वारा समझ सकते हैंः माना

```
int A = 6;
int B ;
B = !A
```

यदि किसी Program में हम इस Expression को Execute करें और Variable **B** के मान को Print करें, तो हमें Output में **0** या ***False*** प्राप्त होता है। ऐसा इसलिए होता है क्योंकि Variable **A** में **6** Store है, जो कि एक True मान है, लेकिन जब हम इसके साथ NOT Logical Operator का प्रयोग करके Return होने वाले मान को Variable B में Store करते हैं, तो ये Operator Variable **A** के ***True*** मान को False में Convert कर देता है। इसलिए यदि हम Variable B को Output में Print करते हैं, तो हमें Output में **0** प्राप्त होता है, जो कि False को Represent करता है।

Assignment Operators

किसी भी Program में हमें विभिन्न प्रकार के Identifiers को समय–समय पर विभिन्न प्रकार के मान Initialize या Assign करने की जरूरत पडती है। इस जरूरत को पूरा करने के लिए हमें जिस Operator का प्रयोग करना होता है, उसे **Assignment Operator** कहते हैं।

हालांकि Assignment Operator तो केवल एक ही है, लेकिन इसे कई अन्य तरीकों से भी Use कर सकते हैं। Assignment Operator को जिन अन्य तरीकों से Use किया जाता है, उन तरीकों को **Short Hand** तरीके कहा जाता है। "C" Language में निम्नानुसार 6 तरीकों से किसी Assignment Operator का प्रयोग किया जा सकता हैः

Operator	Declaration	Example	Example Explanation
=	Assignment	A = 10	A = 10
+=	Assigning Sum	A += 10	A = A + 10
-=	Assigning Difference	A -= 10	A = A – 10
*=	Assigning Product	A *= 10	A = A * 10
/=	Assigning Dividend	A /= 10	A = A / 10
%=	Assigning Reminder	A %= 10	A = A % 10

कई बार हमें ऐसी जरूरत होती है, जिसमें किसी एक ही Identifier के मान के साथ किसी प्रकार की प्रक्रिया करने के बाद Generate होने वाले मान को वापस उसी Identifier में Store करना होता है।

इस प्रकार का काम करने के लिए हम **Short Hand Assignment Operators** का प्रयोग करते हैं।

उदाहरण के लिए मानलो कि किसी Identifier **A** का मान **10** है और हम चाहते हैं कि इस Identifier में **20** जोड कर प्राप्त होने वाले मान **30** को फिर से Identifier **A** में ही Store कर दिया जाए। इस काम को पूरा करने के लिए सामान्यतया हमें निम्नानुसार Statement लिखना होता हैः

```
A = A + 20;
```

इसी Statement द्वारा पूरे होने वाले काम को यदि हम और छोटे रूप में लिखना चाहें, तो निम्नानुसार लिख सकते हैंः

```
A += 20;
```

ये Statement भी वही काम करता है, जो पिछला वाला Statement कर रहा है। यानी A के मान में 20 जोड कर प्राप्त होने वाले मान 30 को फिर से A में Store कर देता है। इसी तरह से हम अन्य Assignment Operators का भी प्रयोग कर सकते हैं, जिन्हें उपरोक्त सारणी में उदाहरण के रूप में दर्शाया गया है।

Exercise:

Explain the Short Hand Assignment Operators using appropriate example?

Increment and Decrement Operators

कई बार हमें हमारे Program में क्रम से एक–एक बढने या घटने वाली संख्याओं को Generate करने की जरूरत पडती है। इस प्रकार की जरूरत को पूरा करने के लिए हमें Increment (++) या Decrement (--) Operators का प्रयोग करना पडता है। वेरियेबल के साथ इनकी दिशा बदल देने से इनके स्वभाव में परिवर्तन आ जाता है।

जब किसी Variable के मान में क्रम से कोई संख्या जोड कर वापस उसी Variable में Store कर देते हैं, तो उस Variable का मान उस जोडी गई संख्या के अनुसार उसी क्रम में बढता जाता हैं, इस प्रक्रिया को Variable का **Increment** होना कहते हैं।

उदाहरण के लिए माना एक Variable x = 0 है और हम चाहते हैं कि इसका मान क्रम से एक–एक बढता जाए। इस जरूरत को पूरा करने के लिए हम निम्नानुसार Statement लिख सकते हैं:

```
x = x + 1
```

हम इसी Statement को x = x + 1 लिखने के बजाय संक्षिप्त रूप में **x++** भी लिख सकते हैं।

इसी तरह से जब Variable के मान में से क्रम से कोई संख्या घटा कर प्राप्त मान वापस उसी Variable में Store कर देते हैं, तो इस प्रक्रिया को Variable का **Decrement** होना कहते हैं।

उदाहरण के लिए माना x = 10 है व हम क्रम से x का मान 1 कम करना चाहते हैं। इस जरूरत को पूरा करने के लिए हम निम्नानुसार Statement लिख सकते हैं:

```
x = x – 1
```

हम इसी Statement को x = x – 1 लिखने के बजाय संक्षिप्त रूप में **x--** भी लिख सकते हैं।

जब हमें किसी Variable के मान को एक–एक के क्रम में ही बढाना या घटाना होता है, या एक–एक के क्रम में ही ***Increment*** या ***Decrement*** करना होता है, तब हम जिन दो Operators को Use करते हैं, उन्हें Increment (++) व Decrement (--) Operators कहते हैं। इन दोनों Operators को भी दो–दो तरीकों से Use किया जाता है, जो कि निम्नानुसार हैं:

1 Pre – Increment

जब किसी Variable के पहले Increment ++ का चिन्ह लगाया जाता है, तब उस Variable का मान पहले Increase होता है, उसके बाद वह Variable उस Expression में भाग लेता है, जिसमें उस Variable को Use किया गया है। जैसे

```
int x = 0, y = 10, Result;
Result = ++x + y
```

इस Code Segment में पहले x का मान Increment हो कर 0 से 1 हो जाता है, उसके बाद x का मान 1 y के मान 10 में जुड कर 11 Return करता है और Result में 11 Store हो जाता है। अब यदि x, y व Result तीनों को Print किया जाए, तो तीनों का मान क्रमशः 1, 10 व 11 Print होगा।

2 Post – Increment

जब किसी Variable के बाद में Increment चिन्ह लगाया जाता है, तो वह Variable पहले उस Expression में भाग लेता है, जिसमें उसे Use किया गया है, उसके बाद उस Variable का मान Increment होता है। जैसेः

```
int x = 0, y = 10, Result;
Result = x++ + y
```

इस Code Segment में पहले (x + y) Expression Execute होगा और इस Expression से Generate होने वाला Resultant मान **10** Variable Result में Store होगा। उसके बाद x का मान Increment होकर 1 होगा। इस Statement के Execute होने के बाद यदि हम x, y व Result तीनों के मानों को Screen पर Display करें, तो हमें क्रमशः 1, 10 व 10 प्राप्त होगा।

3 Pre – Decrement

Pre-Increment की तरह ही जब किसी Variable के पहले Decrement का चिन्ह लगाया जाता है, तब उस Variable का मान पहले Decrease होता है, उसके बाद वह Variable उस Expression में भाग लेता है, जिसमें उस Variable को Use किया गया है। जैसे

```
int x = 10, y = 20, Result;
Result = --x + y
```

इस Code Segment में पहले x का मान Decrement हो कर 10 से 9 हो जाता है, उसके बाद x का मान 9 y के मान 20 में जुड कर 29 Return करता है और Result में 29 Store हो जाता है। अब यदि x, y व Result तीनों को Print किया जाए, तो तीनों का मान क्रमशः 9, 20 व 21 Print होगा।

4 Post – Decrement

Post-Increment की तरह ही जब किसी Variable के बाद में Decrement चिन्ह लगाया जाता है, तो वह Variable पहले उस Expression में भाग लेता है, जिसमें उसे Use किया गया है, उसके बाद उस Variable का मान Decrement होता है। जैसेः

```
int x = 10, y = 20, Result;
Result = x-- + y
```

इस Code Segment में पहले (x + y) Expression Execute होगा और इस Expression से Generate होने वाला Resultant मान **30** Variable Result में Store होगा। उसके बाद x का मान Decrement होकर 9 होगा। इस Statement के Execute होने के बाद यदि हम x, y व Result तीनों के मानों को Screen पर Display करें, तो हमें क्रमशः 9, 20 व 30 प्राप्त होगा।

चलिए, एक Program में इन चारों Operators को Practically Use करके Result देखते हैं। Program निम्नानुसार है:

Program

```
#include <stdio.h>
main()
{
    int x = 10, y = 20, z = 30;
    printf("\n x = 10, y = 20, z = 30 \n");

    printf("\n ++x + y = %d", ++x + y);
    printf("\t x = %d, y = %d, z = %d", x, y, z);
    printf("\n y++ + z = %d", y++ + z);
    printf("\t x = %d, y = %d, z = %d", x, y, z);
    printf("\n --z + x = %d", --z + x);
    printf("\t x = %d, y = %d, z = %d", x, y, z);
    printf("\n y-- + x = %d", y-- + x);
    printf("\t x = %d, y = %d, z = %d", x, y, z);
}
```

Output

```
x = 10, y = 20, z = 30

++x + y = 31    x = 11, y = 20, z = 30
y++ + z = 50    x = 11, y = 21, z = 30
--z + x = 40    x = 11, y = 21, z = 29
y-- + x = 32    x = 11, y = 20, z = 29
```

Exercise:

Explain the flow of this program?

Bit wise Operators

अभी तक हमने जितने भी Operators के बारे में जाना है, वे सभी Operators किसी Identifier के पूरे Byte पर Operation Perform करते हैं। लेकिन "C" Language में कुछ ऐसे Operators भी Provide करता है, जिनका प्रयोग हम किसी Identifier की Bits पर कर सकते हैं। इस Operator का उपयोग सीधे ही किसी Identifier की Bits पर काम करने के लिए किया जाता है। ये Operator हमेंशा Integer प्रकार के Data Type के साथ ही Use होता है, यानी Bitwise Operators को केवल Integer प्रकार के Data Type के Identifier के साथ ही प्रक्रिया करने के लिए Use किया जा सकता है। "C" Language में इनकी कुल संख्या छः होती हैः

- **&** Bitwise **AND** Operator
- **!** Bitwise **OR** Operator
- **^** Bitwise **Exclusive OR** Operator
- **<<** Bitwise **SHIFT LEFT** Operator
- **>>** Bitwise **SHIFT RIGHT** Operator
- **~** Bitwise **Ones Compliment** Operator

जैसाकि हमने पहले भी कहा है कि Computer की memory में किसी मान को Store किए बिना हम उस मान के साथ किसी प्रकार की कोई प्रक्रिया नहीं कर सकते हैं। लेकिन चूंकि Computer केवल एक Electronic Machine है, इसलिए हम इसमें जिस किसी भी मान को Store करते हैं, वह मान Binary Digits के रूप में ही Store होता है।

जब हम किसी Identifier का प्रयोग करके किसी Memory Location को Access करते हैं, तब वास्तव में हम उस Identifiers से Associated Memory Block के पूरे Byte को Use कर रहे होते हैं। लेकिन "C" हमें कुछ **Bitwise Operator** भी Provide करता है, जिनका प्रयोग करके हम किसी Identifier से Associated पूरे Memory Byte को Access करने के बजाय Memory Byte के किसी भी Bit को Access करने की क्षमता प्राप्त करते हैं। ,

इन Bitwise Operators का यदि ठीक तरह से प्रयोग किया जाए, तो हम हमारे Program की Speed को बहुत तेज कर सकते हैं, क्योंकि Bytes को Access करने की तुलना में Bits को Access करने में Computer को कम समय लगता है। साथ ही इन Operators का प्रयोग करके Directly Memory Location के Bits को Access कर पाने के कारण हम Computer के Hardware को भी Directly Access कर पाने में सक्षम हो सकते हैं।

Bitwise Operators को समझने से पहले हमें ये समझना होगा कि Computer विभिन्न प्रकार के Character व Integer मानों को Computer की Memory में किस तरह से Store करता है। Computer में 8 Bits के समूह को Byte, 16 Bits के समूह को Word तथा 32 Bits के समूह को DWord कहा जाता है।

हम जब भी Character प्रकार का कोई Identifier Create करते हैं, तो Computer एक Byte की Space को Reserve करता है, जो कि 8 Bits का एक समूह होता है। इस Byte को हम निम्नानुसार चित्र द्वारा Represent कर सकते हैं:

7	6	5	4	3	2	1	0

Bit-Pattern of Byte

किसी भी Byte या Word के Bits की Position को हमेंशा Right Side से Count किया जाता है। इस Byte के हर Bit में केवल ***True*** या ***False*** यानी **0** या **1** ही Store हो सकता है। किसी Byte के पहले Bit (0th Position Bit) को **Least Significant Bit** (LSB) व अन्तिम Bit (7th Position Bit) को **Most Significant Bit** (MSB) कहा जाता है।

किसी Signed Character Data Type के Identifier में Most Significant Bit हमेंशा Sign को Represent करता है। इस Bit का मान यदि 0 होता है, तो इसमें Stored Bit Pattern किसी Positive संख्या को Represent करता है, जबकि यदि इस Bit का मान 1 हो, तो इसमें Stored Bit Pattern किसी Negative संख्या को Represent करता है।

किसी Byte में स्थित हर Bit का उसकी Position के आधार पर एक Unique मान होता है। किसी Byte के Least Significant Bit में Store हो सकने वाली संख्या का अधिकतम मान 1 हो सकता है और हम जैसे–जैसे Least Significant Bit से Most Significant Bit की तरफ बढते जाते हैं, वैसे–वैसे उन Bits के मान Store करने की क्षमता पिछले मान की तुलना में Double होती जाती है।

उदाहरण के लिए किसी भी Byte के **0th** Position में Store हो सकने वाला अधिकतम मान **1** हो सकता है, इसलिए **1st** Position पर Store हो सकने वाली संख्या का मान पिछले Bit की अधिकतम Capacity से दुगुना, यानी **2** हो सकता है।

इसी तरह से **3rd** Position पर Store हो सकने वाली संख्या का अधिकतम मान **4** हो सकता है इसी तरह से आगे भी यही क्रम जारी रहता है। यदि हम इस प्रक्रिया को चित्र द्वारा Represent करें, तो निम्न चित्र में हम देख सकते हैं कि किस प्रकार से हर Bit **LSB** से **MSB** की तरफ बढते हुए दुगुने मान को Represent करने लगता है:

7th	6th	5th	4th	3rd	2nd	1st	0th
128	64	32	16	8	4	2	1

चूंकि Character Type का Variable Memory में 1 Byte या 8 Bit लेता है, इसलिए हमने इस प्रक्रिया को एक Character प्रकार के Identifier पर Apply करके समझाया है। लेकिन ये प्रक्रिया एक Integer प्रकार के Word पर भी पूरी तरह से Apply होती है, अन्तर केवल इतना है कि एक

Integer में कम से कम 16 Bits होते हैं, इसलिए Bits की संख्या व उनकी Location के आधार पर एक Integer में Data को Store करने की क्षमता काफी बढ जाती है, जिसे हम निम्न चित्र द्वारा समझ सकते हैं:

7th	6th	5th	4th	3rd	2nd	1st	0th
128	64	32	16	8	4	2	1

15th	14th	13th	12th	11th	10th	9th	8th
32768	16384	8192	4096	2048	1024	512	256

इस तरह से यदि किसी Unsigned Character प्रकार के Identifier के सभी Bits में निम्न चित्रानुसार ***False*** यानी **0** Store हो, तो ये Byte Numerical मान 0 को Represent करता है।

7th	6th	5th	4th	3rd	2nd	1st	0th
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
128	64	32	16	8	4	2	1

इसी तरह से यदि इस Byte के सभी Bits का मान ***True*** या **1** हो, तो ये Byte निम्न चित्रानुसार Numerical मान **255** को Represent करता है।

7th	6th	5th	4th	3rd	2nd	1st	0th
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**
128	64	32	16	8	4	2	1

Byte में Stored Binary Bits के आधार पर Decimal Number प्राप्त करने का तरीका ये है कि Byte में जिस–जिस Bit Position पर ***True*** या ***1*** Stored होता है, उन्हें आपस में जोड लिया जाता है। इस जोड से प्राप्त होने वाला मान उस Binary Bit-Pattern का Decimal मान होता है। उदाहरण के लिए निम्न Byte Representation में **0th, 1st** व **3rd** Bit Position पर True या 1 Stored है, इसलिए इस Binary का Decimal मान **1 + 2 + 8 = 11** होगा।

7th	6th	5th	4th	3rd	2nd	1st	0th
0	**0**	**0**	**0**	**1**	**0**	**1**	**1**
~~128~~	~~64~~	~~32~~	~~16~~	**8**	**4**	**2**	**1**

Decimal Value of this Binary [00001011] = **8 + 2 + 1 = 11**

हम जानते हैं कि किसी भी Byte में उसकी Bit Position पर **0** या **1** ही हो सकता है, जहां **0** Lowest Value को Represent करता है, जबकि **1** Highest Values को Represent करता है। इसलिए यदि इस तरीके के आधार पर हम किसी Unsigned Character प्रकार के Identifier के सभी Bits में **0** Store कर दें, तो Byte में Store हो सकने वाला Minimum मान **0** ही हो सकता है,

जबकि यदि सभी Bit Position को उनकी High Value यानी **1** से Fill कर दें, तो Byte में Store हो सकने वाला Maximum मान [1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255] ही हो सकता है। यही वजह है कि किसी Unsigned Character प्रकार के Data Type में हम इससे बडी संख्या को Store नहीं कर सकते हैं।

जब हम Unsigned Byte में Store हो सकने वाले मान की गणना करते हैं, तब हमें 0 से 255 की Range प्राप्त होती है, लेकिन जब हम एक Byte में Signed प्रकार के मान के Store होने की Range ज्ञात करना चाहते हैं, तब Byte का **Most Significant Bit** Sign की Information को Hold करने लगता है। यदि इस Bit का मान 0 होता है, तो इसमें Stored संख्या Positive होती है, जबकि इस Bit में Stored मान **1** होने की स्थिति में इसमें Stored Bit-Pattern के आधार पर Generate होने वाली संख्या Negative हो जाती है। इस स्थिति में यदि हम किसी Sign वाली संख्या की Minimum व Maximum Range ज्ञात करना चाहें, तो Minimum संख्या का Bit-Pattern निम्नानुसार बनेगाः

7th	6th	5th	4th	3rd	2nd	1st	0th
1	0	0	0	0	0	0	0
128	~~64~~	~~32~~	~~16~~	~~8~~	~~4~~	~~2~~	~~1~~

हम देख सकते हैं कि इस Bit Pattern में केवल Most Significant Bit यानी **Sign Bit** ही **1** है जो बता रहा है, कि ये संख्या एक Negative संख्या है, साथ ही हम ये भी देख सकते हैं कि इस **MSB** की Position पर ***True*** या **1** Store होने का मतलब ये भी है कि इस संख्या का Decimal मान **128** है। यानी इस संख्या का वास्तविक मान **-128** है। अब यदि हम इस Byte के Bit-Pattern को **Invert** कर दें, यानी True को False व False को True कर दें, तो हमें निम्नानुसार Bit-Pattern प्राप्त होता हैः

7th	6th	5th	4th	3rd	2nd	1st	0th
0	1	1	1	1	1	1	1
~~128~~	64	32	16	8	4	2	1

इस Bit-Pattern में **MSB** का मान **0** है जो बताता है, कि इस Byte में Stored संख्या एक Positive संख्या है। अब यदि हम इस संख्या का Decimal मान ज्ञात करें, तो हमें [1 + 2 + 4 + 8 + 16 + 32 + 64 = 127] प्राप्त होता है, जो कि इस Byte में Store हो सकने वाला Maximum मान है।

जिस तरह से हमने एक Byte की Bit Position पर Stored **0** या **1** के आधार पर एक Byte में Store हो सकने वाले न्यूनतम व अधिकतम मान को Calculate किया है, उसी तरह से हम **Integer** व **Long Integer** Type के Identifier में Store हो सकने वाले न्यूनतम व अधिकतम मान को भी ज्ञात कर सकते हैं।

इसके लिए हमें केवल ये ध्यान रखना होता है, कि 2 Byte के Integer में 16 Bit होते हैं, इसलिए 2 Byte के Integer की Limit ज्ञात करने के लिए हमें 16 Bits के Pattern के आधार पर Range ज्ञात करना पडेगा। जबकि Long Integer प्रकार का Identifier Memory में 4 Byte या 32 Bit Reserve करता है, इसलिए Long Integer प्रकार के Identifier में Store हो सकने वाली Minimum व Maximum संख्या का मान प्राप्त करने के लिए हमें 32 Bit के Pattern बनाने पडेंगे।

एक बात हमेंशा ध्यान रखें, कि जब किसी Identifier में Store होने वाली संख्या Unsigned होती है, तब Byte, Word व DWord के सभी Bits के आधार पर उसमें Store हो सकने वाली Maximum व Minimum संख्या तय करते हैं, लेकिन जब किसी Byte, Word या DWord में Store होने वाली संख्या Sign वाली होती है, तब Byte, Word व DWord के सभी Bits मिलकर उनमें Store होने वाली संख्या की Range तय नहीं करते हैं, बल्कि इन सभी का MSB संख्या का Sign तय करने का काम करते हैं।

इस कारण से एक Unsigned Identifier जितना मान Store कर सकता है, Sign वाला Identifier उसका आधा ही Store करता है। यानी Sign वाले Identifier की Range Positive संख्या Store करने के लिए आधी कम हो जाती है, लेकिन Negative संख्या Store करने के लिए आधी बढ जाती है।

दसमलव वाले मान भी Computer में Bit-Pattern के रूप में ही Store होते हैं, लेकिन उनके Store होने के तरीके में थोडा अन्तर होता है और अन्तर ये होता है कि हर दसमलव वाली संख्या Sign वाली संख्या ही होती है। यानी Float, Double या Long Double संख्या कभी भी Unsigned नहीं होती है। साथ ही इनके Bit-Pattern के कुछ Bits दसमलव के बाद वाली संख्या को Represent करने का काम करते हैं, जबकि कुछ Bits दसमलव से पहले वाली संख्या को Represent करने का काम करते हैं।

जिस तरह से हम किसी Byte में Stored Bit-Pattern के आधार पर Decimal संख्या ज्ञात कर सकते हैं, उसी तरह से हम किसी Decimal संख्या का Bit Pattern भी बना सकते हैं। उदाहरण के लिए मानलो हमें मान **57** का Bit-Pattern ज्ञात करना है। इस Bit-Pattern को ज्ञात करने के लिए हमें निम्न क्रम को Use करना होता है:

7th	6th	5th	4th	3rd	2nd	1st	0th
128	64	32	16	8	4	2	1

1 सबसे पहले हमें ये पता लगाना होता है, कि हम जिस संख्या का Bit-Pattern ज्ञात करना चाहते हैं, उस संख्या को Represent करने वाला कोई Bit Byte के Bit-Pattern में उपलब्ध है या नहीं। यदि ज्ञात की जाने वाली संख्या का मान Bit-Pattern में ना हो, तो ज्ञात की जाने वाली संख्या से छोटी संख्या के Bit को ***True*** या ***1*** कर देना चाहिए।

उपरोक्त चित्र में हम देख सकते हैं कि इस Byte के Bit-Pattern में 1, 2, 4, 8, 16, 32, 64 व 128 हैं, लेकिन **57** नहीं है। इस स्थिति में 57 से Just छोटा मान 32 है, इसलिए हमें Byte के Bit Pattern में इसी मान को True या 1 करना होता है, जैसाकि हमने निम्न चित्र में किया है।

7th	6th	5th	4th	3rd	2nd	1st	0th
		1					
128	64	32	16	8	4	2	1

2 अब ज्ञात की जाने वाली संख्या के मान में से Set किए गए Bit की संख्या को घटाना होता है।

चूंकि हमने 32 के मान के Bit को Set किया है, इसलिए हमें **57** में से **32** को घटाना होता है। 57 में से 32 को घटाने पर 25 बचता है, इसलिए अब हमें इस 25 की Binary Set करनी है। इसके लिए हमें फिर से पिछले Step को Use करना होता है।

चूंकि हमारे Byte के Pattern में मान 25 के लिए भी कोई संख्या नहीं है, इसलिए हमें 25 से छोटे मान के Bit को Set करना होता है, जो कि हमारे इस उदाहरण में 16 है। इस Bit को **1** कर देने पर हमें निम्नानुसार Pattern प्राप्त होता है:

7th	6th	5th	4th	3rd	2nd	1st	0th
		1	1				
128	64	32	16	8	4	2	1

3 अब हमें फिर से बची हुई संख्या को ज्ञात करना होता है। इसके लिए हमें Current संख्या में से Set की गई Bit की संख्या को घटाना होता है।

चूंकि हमारी Current संख्या 25 है, जिसमें से हमने 16 को Set किया है, इसलिए अब हमें 25 में से 16 को घटाना होता है। ऐसा करने पर हमें Resultant मान **25 – 16 = 9** प्राप्त होता है, जिसके लिए हमें Bit को Set करना होता है।

यहां फिर हमें Step1 को Use करना होता है, जिससे हमें मान 8 मिलता है, जिसे Set करना होता है। क्योंकि मान 8 ही मान 9 से सबसे कम छोटा मान है। मान 8 के Bit को Set करने पर हमें निम्नानुसार Bit-Pattern प्राप्त होता है:

7th	6th	5th	4th	3rd	2nd	1st	0th
		1	1	1			
128	64	32	16	8	4	2	1

4 Current मान 9 में से Currently Set किए गए Bit के मान 8 को घटाने पर हमें 1 प्राप्त होता है, और अब हमें केवल मान 1 के लिए Bit को Set करना है। चूंकि मान 1 को Represent करने वाला bit 0th Position पर है, इसलिए केवल इस Bit को Set कर देने पर हमें 57 का Binary Bit-Pattern प्राप्त हो जाएगा, जो कि निम्नानुसार है:

7th	6th	5th	4th	3rd	2nd	1st	0th
		1	1	1			1
128	64	32	16	8	4	2	1

चूंकि हमें हमारे Required मान की Binary Bit-Pattern प्राप्त हो चुकी है, इसलिए जिन भी Bits की Position Blank है, उनमें 0 Fill कर देने से हमें हमारे Required मान की Actual Bit-Pattern प्राप्त हो जाती है, जो कि निम्नानुसार है:

7th	6th	5th	4th	3rd	2nd	1st	0th
0	0	1	1	1	0	0	1
128	64	32	16	8	4	2	1

हमें मान 57 का Binary Bit Pattern **00111001** प्राप्त हुआ है। ये Bit Pattern सही है या नहीं इस बात की जांच करने के लिए हम उन Bits के मानों को जोड सकते हैं, जिनमें ***True*** या **1** Stored है। हमारे Bit-Pattern में 0th, 3rd, 4th व 5th Position के Bits **On** है, जिनके मानों का Total [1 + 8 + 16 + 32 = 57] है, जो कि वही मान है, जिसका Bit-Pattern हम बनाना चाहते थे, इसलिए हमारा Bit-Pattern सही है।

इस तरह से हम किसी भी Bit-Pattern का Decimal मान ज्ञात कर सकते हैं और किसी भी Decimal संख्या का Bit Pattern बना सकते हैं।

Bitwise AND Operator (&)

ये Operator Use करके हम दो Identifier के Bits पर AND Masking की प्रक्रिया को Apply करते हैं। AND Masking में दोनों Identifiers के Bits आपस में AND Form में Compare होते हैं। यदि दोनों Identifiers में समान Position पर Bit का मान 1 हो यानी Bit ***True*** हो तो Resultant Bit भी True होता है, अन्यथा Resultant Bit False हो जाता है। उदाहरण के लिए **19** की Binary 10011 होती है और 21 की Binary 10101 होती है। अब यदि निम्नानुसार दो Variables में ये दोनों मान Stored हों:

```
int firstValue = 19;        //Binary : 10011
int secondValue = 21;       //Binary : 10101
int resultValue;
```

और यदि इन दोनों Identifiers पर निम्नानुसार AND Masking करके Resultant मान को result नाम के Variable में Store किया जाए, तो Result निम्नानुसार प्राप्त होगा :

```
result = firstValue & secondValue;
```

```
firstValue's Binary      :   10011        // Decimal Value = 19
secondValue's Binary     :   10101        // Decimal Value = 21
----------------------------------------
resultValue's Binary     :   10001        // Decimal Value = 17
----------------------------------------
```

AND Masking में निम्नानुसार Table के अनुसार Bits पर प्रक्रिया होती है, जिसमें यदि दोनों Identifiers के समान Position के दोनों Bits का Comparison होता है और समान Position पर ही Resultant Bit Return होता है।

AND Mask	0	1
0	0	0
1	0	1

इस Bitwise Operator का प्रयोग अक्सर ये जानने के लिए किया जाता है, कि किसी Operand का कोई अमुक Bit ON (1) है या OFF (0) किसी Operand का कोई Bit On है या Off, ये जानने के लिए हमें एक अन्य Operand लेना होता है और उस Operand में उस Bit को On रखा जाता है, जिसे प्रथम Operand में Check करना होता है।

उदाहरण के लिए माना एक Operand का Bit Pattern 11000111 है और हम जानना चाहते हैं कि इस Pattern में चौथा Bit ON है या नहीं। ये जानने के लिए हमें एक दूसरा Operand लेना होगा और उस Operand के Bit Pattern में चौथे Bit को ON(1) व शेष Bits को OFF(0) रखना होगा। इस प्रकार से हमें दूसरे Operand का जो Bit Pattern प्राप्त होगा वह 00001000 होगा, जिसका चौथा Bit ON है।

किसी Operand के Bit Pattern के किसी Bit की स्थिति पता करने के लिए दूसरा Bit Pattern लेकर जो प्रक्रिया की जाती है, उसे **Masking** कहते हैं और जब इस प्रक्रिया में Bitwise Operator & का प्रयोग किया जाता है, तब इसे **AND Mask** कहते हैं।

Trick ये है कि जब हम प्रथम Operand को **&** Operator द्वारा दूसरे Operand के Bit Pattern से Compare करते हैं तब यदि प्रथम Bit Pattern में चौथा Bit ON होता है, तो ही Comparison से प्राप्त Resultant Bit Pattern में भी चौथा Bit ON होता है अन्यथा चौथा Bit Off होता है। इस Masking को हम निम्नानुसार Represent कर सकते हैं:

```
First Operand      :   11000111
Second Operand     :   00001000
---------------------------------------------
```

```
AND MASK          :    00000000
-----------------------------------------------
```

इस उदाहरण में हम देख सकते हैं कि पहले Bit-Pattern का चौथा Bit Off है, इसलिए Resultant Bit-Pattern में भी चौथा Bit Off है। अब निम्न Fragments को देखिए:

```
First Operand     :    11001001
Second Operand    :    00001000
-----------------------------------------------
AND MASK          :    00001000
-----------------------------------------------
```

इस Fragment में हम देख सकते हैं कि पहले Bit-Pattern का चौथा Bit On है और यही जानने के लिए कि पहले Operand का चौथा Bit On है या नहीं, हमने एक **Mask Bit Pattern** Create किया है, जिसके चौथे Bit को On रखा है। इस स्थिति में Resultant Bit-Pattern का चौथा Bit केवल उसी स्थिति में On हो सकता है, जब Check किए जा रहे Operand के Bit-Pattern में चौथा Bit On हो। इस तरह से AND Masking के उपयोग द्वारा हमें पहले Operand के चौथे Bit की स्थिति का पता चल जाता है।

किसी भी Bit-Pattern में हर Bit की Position का एक मान होता है। इस Position के मान द्वारा हम Directly उस Bit को Refer कर सकते हैं। उदाहरण के लिए निम्न चित्र को देखिए:

7th	6th	5th	4th	3rd	2nd	1st	0th
128	64	32	16	8	4	2	1

इस चित्र में हर Bit Position के साथ एक Number Associated है। यदि हम किसी Bit-Pattern के चौथे Bit को Refer करना चाहते हैं, तो हमें मान 16 को Use करना होता है। इसी तरह से यदि हमें किसी Bit Pattern के छठे Bit को Access करना हो, तो हमें इस Bit Position से Associated मान 64 को Use करना होता है।

चलिए, अब हम एक उदाहरण द्वारा Logical AND Operator को Use करके किसी Identifier के किसी Particular Bit की Status को Check करते हैं कि वह Bit **On** है या नहीं।

इस उदाहरण में हमने एक Identifier **x** में एक मान 150 Store किया है, जिसका Bit-Pattern **1001011** होता है। हम इस Bit-Pattern के पांचवे व छठे Position के Bit की ON/OFF Status जानना चाहते हैं। चूंकि हम किसी भी मान के Bit-Pattern को Binary Form में Use नहीं कर सकते हैं, इसलिए किसी Bit Position को Refer करने के लिए हमें उसके साथ Associated Decimal Number को Use करना होता है।

Program

```
#include <stdio.h>

main()
{
    int x = 150;            // Bit-Pattern of 150 = 10010110
    int j;
    clrscr();

    printf("\n Value of x is %d ", x);

    j = x & 16;
    (j == 0) ?  printf("\n Fifth Bit of value %d is Off", x) :
    printf("\n Fifth Bit of value %d is On", x);

    j = x & 32;
    (j == 0) ?  printf("\n Sixth Bit of value %d is Off", x) :
    printf("\n Sixth Bit of value %d is On", x);
}
```

Output

```
Value of x is 150
Fifth Bit of value 150 is On
Sixth Bit of value 150 is Off
```

जब ये Program Run होता है, तब निम्नानुसार Form में **j = x & 16;** व **j = x & 32;** Statements को Execute करता है:

For Fifth Bit of the value of x: **j = x & 16;**

```
Bit-Pattern of x        :   10010110
Bit-Pattern of Mask     :   00010000
-------------------------------
AND MASK                :   00010000
-------------------------------
```

For Sixth Bit of the value of x: **j = x & 32;**

```
Bit-Pattern of x        :   10010110
Bit-Pattern of Mask     :   00100000
```

```
-------------------------------
AND MASK              :    00000000
-------------------------------
```

चूंकि जब हम पांचवे Bit को Check करते हैं, तब मान 150 का पांचवा Bit On होने की वजह से Making Process से 1 Generate होता है और ये **1** Variable **j** में Store हो जाता है। फिर Ternary Operator में (j==0) Expression Execute होता है, जो कि False हो जाता है, क्योंकि j का मान **1** है और (1==0) नहीं होता है। इस वजह से Ternary Operator के दूसरे Statement का Execution हो जाता है, जो Output में निम्नानुसार Message प्रदान करता है:

```
Fifth Bit of value 150 is On
```

लेकिन जब हम छठे Bit को Check करते हैं, तब मान 150 का छठा Bit **Off** होने की वजह से Masking Process से 0 Generate होता है और ये **0** Variable **j** में Store हो जाता है। अगले Statement में फिर से (j==0) Expression Execute होता है, जो इस बार True होने की वजह से Ternary Statement के पहले Statement का Execution कर देता है और हमें निम्नानुसार Output प्राप्त होता है:

```
Sixth Bit of value 150 is Off
```

इस प्रकार से हम किसी भी Identifier के मान के किसी Particular Bit को On/Off Status की जानकारी प्राप्त करने के लिए Bitwise AND Operator का प्रयोग कर सकते हैं। सामान्यतया विभिन्न प्रकार के Bitwise Operators का प्रयोग विभिन्न प्रकार के Hardware Devices के साथ प्रक्रिया करने के लिए ही करते हैं।

Bitwise OR Operator (|)

ये Operator Use करके हम दो Identifier के Bits पर OR Masking की प्रक्रिया को Apply करते हैं। OR Masking में दोनों Identifiers के Bits आपस में OR Form में Compare होते हैं। यदि दोनों Identifiers में से किसी एक भी Identifier में समान Position पर Bit का मान 1 हो यानी Bit ***True*** हो तो Resultant Bit भी True होता है, अन्यथा Resultant Bit False हो जाता है। OR Masking को हम निम्नानुसार समझ सकते हैं, जहां दो Variables में **19** व **21** मान Stored है:

```
int firstValue = 19;        //Binary : 10011
int secondValue = 21;       //Binary : 10101
int resultValue;
```

इन दोनों Identifiers पर OR Masking की प्रक्रिया को Apply करने पर निम्नानुसार Operation Perform होते हैं:

```
result = firstValue | secondValue;

firstValue's Binary      :   10011          // Decimal Value = 19
secondValue's Binary     :   10101          // Decimal Value = 21
---------------------------------------------
resultValue's Binary     :   10111          // Decimal Value = 23
---------------------------------------------
```

OR Masking में Identifier के Bits पर निम्नानुसार Table के अनुसार पर प्रक्रिया होती है, जिसमें दोनों Identifiers के समान Position के दोनों Bits का आपस में Comparison होता है और तीसरे Identifier में समान Position पर ही Resultant Bit Return होता है।

OR Mask	0	1
0	0	1
1	1	1

Bitwise OR Operator का प्रयोग किसी Bit Pattern में स्थित किसी खास Bit को ON करने के लिए किया जाता है। जब हमें किसी अमुक Bit को किसी Bit-Pattern में ON करना होता है, तब हमें एक और Bit-Pattern की जरूरत होती है। इस दूसरे Bit-Pattern को **OR Mask** Bit Pattern कहते हैं। इस OR Mask में हमें केवल उसी Bit का मान **1** रखना होता है, जिसे हम हमारे प्रथम Bit-Pattern में **ON** करना चाहते हैं।

चलिए, एक उदाहरण द्वारा OR Masking की प्रक्रिया को समझते हैं। मानलो कि हम Bit-Pattern 10100110 (Value = **150**) की चौथी Bit को On करना चाहते हैं। इस जरूरत को पूरा करने के लिए हमें OR Mask के रूप में Bit-Pattern **00001000** को Use करना होगा। जब हमें इस पर OR Masking की प्रक्रिया करनी हो, तो ये प्रक्रिया निम्नानुसार होगीः

```
result = firstValue | secondValue;

firstValue's Binary      :   10010110          // Decimal Value = 150
secondValue's Binary     :   00001000
-----------------------------------------------------
resultValue's Binary     :   10011110          // Decimal Value = 158
-----------------------------------------------------
```

Resultant मान में हम देख सकते हैं, कि इसके केवल चौथे Bit का मान ही 0 से 1 हुआ है। चूंकि चौथे Bit-Position का मान 8 होता है, इसलिए Resultant मान 158 प्राप्त हो रहा है, जो कि Original मान 150 से केवल 8 ही ज्यादा है। इस समस्या का Program निम्नानुसार है:

Program

```
#include <stdio.h>
#include <conio.h>
main()
{
    int x = 150, j;
    clrscr();
    printf("\n Value of x is %d ", x);

    j = x | 8;
    printf("\n Forth Bit of value %d is Now On", x);
    printf("\n Now the Value of x is %d " , j);

    getch();
}
```

Output:

```
Value of x is 150
Forth Bit of value 150 is Now On
Now the Value of x is 158
```

Bitwise XOR (Exclusive OR) Operator (|)

ये Operator Use करके हम दो Identifier के Bits पर XOR Masking की प्रक्रिया को Apply करते हैं। XOR Masking में दोनों Identifiers के Bits आपस में XOR Form में Compare होते हैं। यदि दोनों Identifiers में से किसी एक भी Identifier में समान Position पर Bit का मान 1 हो यानी Bit ***True*** हो तो Resultant Bit भी True होता है, लेकिन यदि दोनों ही Identifiers में समान Position पर समान Bit हो, तो Resultant Bit ***False*** हो जाता है। XOR Masking को समझने के लिए हम पिछले उदाहरण को ही Use कर रहे हैं, जिसमें दो Variables में मान **19** व **21** Stored हैं:

```
int firstValue = 19;        //Binary : 10011
int secondValue = 21;       //Binary : 10101
int resultValue;
```

इन दोनों Identifiers पर XOR Masking की प्रक्रिया को Apply करने पर निम्नानुसार Operation Perform होते हैं:

```
result = firstValue ^ secondValue;

firstValue's Binary      :   10011        // Decimal Value = 19
secondValue's Binary     :   10101        // Decimal Value = 21
-----------------------------------------------
resultValue's Binary     :   00110        // Decimal Value = 6
-----------------------------------------------
```

XOR Masking में निम्नानुसार Table के अनुसार Bits पर प्रक्रिया होती है, जिसमें दोनों Identifiers के समान Position के दोनों Bits का आपस में Comparison होता है और समान Position पर ही Resultant Bit Return होता है।

XOR Mask	0	1
0	0	1
1	1	0

इस Operator का प्रयोग करके हम किसी Identifier की Bits को Toggle तरीके से बार–बार On/Off कर सकते हैं। यानी जब हमें किसी Identifier के Bits को Toggle तरीके से On/Off करना होता है, तब हम इस Bitwise Operator का प्रयोग करते हैं। इस Operator को हम निम्न Program के अनुसार Use कर सकते हैं:

Program

```
#include <stdio.h>

main()
{
    int x = 50, k=10;
    clrscr();
    printf("\n Value of x is %d ", x);
    printf("\n\n Value of k is %d ", k);

    printf("\n\n k = %d is Masking the value of x = %d \n", k, x);
    x = x ^ k;
    printf("\n After XOR Masking the Value of x is %d \n" , x);
```

```
    printf("\n k = %d is Masking the Changed Value of x = %d again", k, x);
    x = x ^ k;
    printf("\n\n Now the Value of x is changed again to %d " , x);
}
```

Output

Value of x is 50	//	Bit-Pattern : **00110010**
Value of k is 10	//	Bit-Pattern : **00001010**
k = 10 is Masking the value of x = 50		
After XOR Masking the Value of x is 56	// Resultant	: **00111000**
k = 10 is Masking the Changed Value of x = 56 again		
Now the Value of x is changed again to 50	// Resultant	: **00110010**

One's Complement Bitwise Operator (~)

इस Operator का प्रयोग करके हम किसी भी Identifier के मान की Bits को Invert कर सकते हैं। जब किसी मान की Bits को Invert कर दिया जाता है, तब Generate होने वाले मान का चिन्ह बदल जाता है। इस प्रक्रिया को हम निम्नानुसार समझ सकते हैं:

x = 150	//Bit-Pattern : 10010110	=	150
~x	//Bit-Pattern : 01101001	=	-151

One's Complement को समझने के लिए हम निम्नानुसार एक Program बना सकते हैं:

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    int j = 150, k;
    clrscr();
    k = ~j;
    printf("\n Original Value is     = %d", j);
    printf("\n Complemented Value is = %d", k);
    getch();
}
```

Output:

```
Original Value is     = 150
Complemented Value is = -151
```

Right Shift Operator (>>)

ये Operator, Operand के Bits को Right में Shift करने का काम करता है। हमें किसी Operand के Bits को जितना Shift करना होता है, हम इस Operator के बाद वह संख्या लिख देते हैं। जैसे val का मान 128 है और हमें इसके Bits orientation को 2 अंक Right में Shift करना हो तो हम val >> 2 लिखते हैं। इस Statement से val में Stored Bits 10000000 दो Bit Right में Shift हो जाता है और हमें 00100000 प्राप्त होता है। हम जितने Bits Right में Shift करते हैं, Bit-Pattern में Left side में उतने ही 0 fill हो जाते हैं। इसे एक उदाहरण द्वारा देखते हैं।

Program

```c
#include <stdio.h>
#include <conio.h>

main()
{
    int k = 1028, l;
    clrscr();
    printf("\n Value of Identifier K is %d \n", k);
    l = k >> 2;
    printf("\n After 2-Bits Right Shifting \n");
    printf(" The Value of K is %d \n", l);
    getch();
}
```

Output

```
Value of Identifier K is 1028
After 2-Bits Right Shifting
The Value of K is 257
```

इस प्रोग्राम में हम देख सकते हैं कि Operand k में Stored Bits, Right में Shift हो रहे हैं। जैसे–जैसे Bits Right में Shift होते हैं तो k का मान भी बदलता जाता है। Right Shifting से एक और बहुत महत्वपूर्ण तथ्य सामने आता है, जो ये है कि यदि हम क्रम से किसी संख्या को एक–एक Bit Right Shift करते जाते हैं, तो Operand का मान भी क्रम से आधा होता जाता है।

यानी Right Shifting से हम जितने Bits Right में Shift करते हैं, उतनी ही बार Operand का मान आधा हो जाता है। जैसे इस Program में हुआ है। 1028 को यदि 4 Bit Right में Shift किया जाए तो ये कहा जा सकता है कि 1028 में चार बार दो का भाग दिया गया है। यानी

```
1028/ 2    =    514
514 / 2    =    257
```

हमने जैसा कि पहले बताया कि हम Bitwise Operators का प्रयोग केवल char या int प्रकार के Operand के साथ ही कर सकते हैं और यहां int प्रकार के Operand k के साथ प्रक्रिया की है। यहां ये सवाल दिमाग में आ सकता है कि 257 / 2 = 128.5 होना चाहिये था फिर 256 क्यों हुआ। इसकी वजह यही है कि int प्रकार का मान पूर्णांक में ही हो सकता है। int में दसमलव संख्याएं मान्य नहीं हैं और Bitwise Operators **float** या **double** को मान्य नहीं करते, वे केवल int या char को मान्य करते हैं। इसलिए यहां 128.5 ना हो कर 128 ही हुआ है।

Left Shift Operator (<<)

Left Shift Operator के काम करने का तरीका बिल्कुल वही है जो Right Shift Operator का है। लेकिन दोनों के काम करने का क्रम बिल्कुल विपरीत है। ये किसी Operand के Bits को Left में Shift करता है और Right में खाली हुए स्थान को 0 से भर देता है। इसे समझने के लिए हम ऊपर के ही उदाहरण में केवल इतना बदलाव कर रहें हैं, यानी जहां पर Right Shift Operator का प्रयोग किया था, वहां पर Left Shift Operator का प्रयोग कर रहे हैं और k का मान 1028 से बदल कर 128 कर रहे हैं।

Program

```
#include <stdio.h>
main()
{
    int k = 128, l;
    clrscr();

    printf("\n Value of Identifier K is %d \n", k);
    l = k << 2;
    printf("\n After 2-Bits Right Shifting \n");
    printf(" The Value of K is %d \n", l);
}
```

Output

```
Value of Identifier K is 128
```

```
After 2-Bits Right Shifting
The Value of K is 512
```

जैसाकि हमने अभी बताया कि Left Shift Operator, Right Shift Operator से विपरीत Output देता है। जिस प्रकार से Right Shifting में Operand का मान आधा होता जाता है, उसी प्रकार से Left Shifting में Operand का मान पूर्व मान से दुगुना होता जाता है। यही वजह है कि मान 128 को 2 Bit Left Shift करने से उसका मान 512 हो गया है। किसी Operand के 2 Bits को Left में Shift करने का मतलब है, उस संख्या को दो बार दुगुना करना । इस प्रक्रिया को हम निम्नानुसार समझ सकते हैं:

```
128 *   2   =   256
256 *   2   =   512
```

Comments

"C" Language में Program लिखते समय विभिन्न प्रकार के Comments दिए जा सकते हैं। ये Comments Programmer अपनी सुविधा के लिए लिखता है। विभिन्न प्रकार के Comments द्वारा एक Programmer Program के Flow को तथा Program में Use किए जाने वाले Special Tricks को Specify करता है, जिससे Program Readable हो जाता है। सामान्यतया Comments को Program के Documentation Section में लिखा जाता है, लेकिन एक Programmer Program में किसी भी स्थान पर Comments लिख सकता है।

"C" Language Program में Comments को लिखने के लिए /* … */ का प्रयोग किया जाता है। इस Symbol के बीच लिखे जाने वाले Statements केवल Source File में ही उपयोगी होते हैं। Comments कभी भी Compile नहीं होते हैं। Compiler किसी Source File में लिखे गए विभिन्न Comments को Compilation के समय हमेंशा Ignore कर देता है, इसलिए Comments की वजह से कभी भी Executable File की Size में कोई फर्क नहीं पडता है।

हम एक Program में किसी भी स्थान पर Comment लिख सकते हैं। लेकिन किसी एक Comment के अन्दर दूसरे Comment की Nesting नहीं कर सकते हैं। जैसे

```
/* This is my first C Program */
```

ये एक सामान्य Comment है। लेकिन

```
/* This is my /*first*/ C Program */
```

ये एक गलत Comment है, क्योंकि इसमें एक Comment के अन्दर दूसरे Comment को Nest किया गया है। हम printf() या scanf() जैसे किसी Function में भी Comment को नहीं लिख सकते हैं। यदि हम ऐसा करते हैं, तो Program तो Compile होता है, लेकिन Output में वह Comment भी Print हो जाता है। जैसेः

```
printf("/*This is my first printf() function */ Hello");
```

Output:

```
/*This is my first printf() function */ Hello
```

हम देख सकते हैं कि इस Statement के Output में Comment भी Compile हो रहा है।

Exercise:

1 Comments से आप क्या समझते हैं? किसी Program में इसका प्रयोग क्यों किया जाना चाहिए? यदि इसका प्रयोग ना किया जाए, तो Program पर क्या असर पडेगा?

2 किसी Integer संख्या का Equivalent Binary Bit-Pattern ज्ञात करने का Program बनाओ।

3 निम्न Expressions क्या Result Generate करेंगे, जहां A = 10, B = 20 व C = 30 हैं:

```
I      A != 10 && B > 30          II     10 <= C && !(A) == B
III    C || A != B                IV     (A && B) == (B && C)
V      C = A++ + ++ B             VI     C = --A - --B + ++C -C++
VII    A += A++ + ++A             VIII   B /= (A * ++B) - --C - B–
```

4 Left Shift व Right Shift Operator के अन्तर को समझाईए। इन्हें किस परिस्थिति में Use करना चाहिए।

5 Logical **AND/OR** तथा Bitwise **AND/OR** Operators एक दूसरे से किस प्रकार भिन्न हैं?

6 Logical **NOT** Operator व **One's Complement** Operator के काम करने के तरीके को समझाईए। इन्हें एक दूसरे के स्थान पर Use करने के लिए हमें Program में किस प्रकार का Change करना पडता है? एक उदाहरण द्वारा समझाईए।

7 Increment/Decrement Operators को समझाईए। Pre व Post के अन्तर को उदाहरण द्वारा स्पष्ट कीजिए।

8 "C" का Compiler किसी भी Non-Zero मान को True मानता है। ये तथ्य सही है या गलत?

Types of Instructions

"C" Language में किसी Program File में हम मुख्यतया तीन तरह के Instructions लिखते हैं। इन तीनों प्रकार के Instructions का एक विशेष काम होता है और हर प्रकार Instruction अपने उस विशेष काम को पूरा करता है। ये तीनों Instructions निम्नानुसार होते हैं:

Type Declaration Instruction

ये वे Instructions होते जिनका प्रयोग करके हम विभिन्न प्रकार के Data को Computer की Memory में Store करने के लिए Memory Reserve करते हैं। हम जिस किसी भी Data को Program में Process करना चाहते हैं, उस Data को Store करने के लिए हमें Memory की जरूरत होती है, जहां उन Process किए जाने वाले Data को Hold करके रखना होता है।

Required Data के आधार पर हमें Memory में कुछ जगह Reserve करने के लिए जिन Instructions का प्रयोग करना होता है, उन्हें Type Declaration Instructions कहते हैं। इन Instructions का प्रयोग करके हम विभिन्न प्रकार के Variables Declare करते हैं।

एक "C" Program में हम कई तरह से Variables Create कर सकते हैं। Variables Declare करते समय ही हम उन Variables को Data Initialize कर सकते हैं। जैसे:

```
int i = 19, j = 23 * 3/2-1;
```

जब हम किसी Variable को Create करते समय ही उसमें कोई एक निश्चित मान Initialize कर देते हैं, तो इस प्रक्रिया को **Implicit Initialization** कहते हैं। उदाहरण के लिए उपरोक्त Instruction में Variable **i** को Declare करते समय ही उसमें मान **19** को Initialize कर दिया गया है, जो कि ***Implicit Initialization*** का उदाहरण है।

जब हम किसी Variable को Declare करते समय उसमें किसी प्रकार की Calculation से प्राप्त मान को Initialize करते हैं, तो इस प्रक्रिया को **Explicit** या **Dynamic Initialization** कहते हैं। उदाहरण के लिए उपरोक्त Statement में Variable **j** को Create करते समय उसमें जो मान Initialize किया जा रहा है, वह मान एक Calculation से Generate हो रहा है, इसलिए ये एक ***Explicit*** या ***Dynamic Initialization*** का उदाहरण है।

हम किसी Variable को Create करने के बाद यदि कोई दूसरा Variable Create करते हैं, तो उस दूसरे Variable में पहले Variable के मान को भी Initialize कर सकते हैं। जैसे:

```
int i = 19, j = i;
```

इस Statement में हमने जो मान Variable **i** में Store किया है, वही मान हमने Variable **j** में भी Store किया है। लेकिन यदि हम इस Declaration के क्रम को निम्नानुसार Change कर दें:

```
int j = i, i = 19;
```

तो Compiler हमें निम्नानुसार Error प्रदान करता है:

```
Error:  Undefined symbol 'i'
Error:  Multiple declaration for 'i'
```

ऐसा इसलिए होता है, क्योंकि "C" का Compiler हर Instruction को **Up to Down** व **Left to Right** Execute करता है। इस स्थिति में Compiler सबसे पहले `int j = i`; Instruction को Execute करता है।

इस Instruction के आधार पर Compiler जब Variable **j** के मान में Variable **i** का मान Initialize करने के लिए Variable **i** की Memory Location को खोजता है, तो उसे ऐसा कोई Memory Location प्राप्त नहीं होता है, जिसका नाम **i** है, क्योंकि Compiler ने अभी तक Variable **i** के लिए Memory में किसी Location को Reserve ही नहीं किया है और Compiler जब किसी ऐसे Variable को Memory में खोजता है, जिसे उसने किसी Memory Block के साथ Associate करके Define ही नहीं किया है, तो वह "**Undefined symbol**" का Error Message Generate करता है।

Compiler हमें दूसरी Error इसलिए Display करता है, क्योंकि Compiler जिस Variable **i** को पहले Memory में खोज चुका होता है, उसी नाम का Variable हम बाद में Define करने की कोशिश करते हैं। इस स्थिति में पहले Instruction के लिए तो Compiler ये समझता है, कि हमने variable **i** को Define नहीं किया है, जबकि दूसरे Instruction के लिए Compiler ये सोंचता है कि हम एक ही नाम के एक से ज्यादा Variables Define करने की कोशिश कर रहे हैं।

इस तरह से एक **Misplaced Instruction** एक से ज्यादा प्रकार की Errors को Generate कर रहा है। जब किसी Program में कोई एक गलत Instruction एक से ज्यादा प्रकार की Errors को Generate करने में सक्षम होता है, तो इस प्रकार के Error Instruction को "**Error Generator Source Instruction**" कहा जाता है।

कई बार हमें ऐसी जरूरत पडती है कि एक ही मान को एक से ज्यादा Variables में Assign या Initialize करना होता है। इस प्रकार की जरूरत को हम निम्नानुसार पूरा कर सकते हैं:

```
int x, y, z;
x = y = z = 100;
```

लेकिन यदि इन दोनों Instructions को हम एक Instruction के रूप में निम्नानुसार Use करें:

```
int x = y = z = 100;
```

तो ये एक गलत Instruction होगा और हमें फिर से "**Undefined Symbol**" की Error प्राप्त होगी। क्योंकि यहां फिर से हम उस Variable **y** का मान Variable **x** में Initialize करने की कोशिश कर रहे हैं, जिसे अभी तक Memory Allot नहीं किया गया है।

Arithmetical Instruction

एक Arithmetical Instruction में हमेंशा एक Assignment Operator का प्रयोग करके किसी Calculation से प्राप्त परिणाम को किसी Variable में Store करना होता है। Assignment Operator के Left Side में हमेंशा एक Variable ही हो सकता है, जबकि इसके Right Side में Calculation में भाग लेने वाले Variable व Constants का पूरा एक समूह हो सकता है।

किसी Assignment Operator के Left Side में हम कभी भी किसी Constant Identifier को Place नहीं कर सकते हैं। यदि हम ऐसा करते हैं, तो Compiler हमें "**Lvalue Required**" नाम का एक Error प्रदान करता है। यानी

```
int x, y = 10;
x = y + 2;
```

यदि हम उपरोक्त Instruction लिखते हैं, तो Program में किसी तरह का कोई Error Generate नहीं होगा और **y + 2** Expression से Generate होने वाला Resultant मान 12 Variable **x** में Store हो जाएगा। लेकिन यदि हम इसी Instruction के मानों की Position को निम्नानुसार Exchange कर दें, तो हमें "Lvalue Required" का Error Message प्राप्त होता है:

```
int x, y = 10;
y + 2 = x;              // Error: Lvalue required
```

ये Error हमें इसीलिए प्राप्त होता है, क्योंकि Assignment Operator (=) हमेंशा **Right To Left** Direction में Execute होता है और अपना पूरे Expression के Execute होने के बाद अपना काम अन्तिम Operation के रूप में करता है।

यानी जिस Expression में Assignment Operator का प्रयोग किया जाता है, उस Expression में सबसे पहले Assignment Operator के **Right Side** के सभी Operations Perform होते है और इन Operations से जो Resultant मान Generate होता है, Assignment Operator उस मान को Expression के सबसे अन्तिम Operation के रूप में अपने Left Side में Place किए गए Variable में Store कर देता है।

अब यदि किसी Expression में Assignment Operator के Left Side में किसी Variable के स्थान पर कोई Constant हो, तो Assignment Operator कभी भी किसी मान को किसी Constant में Store नहीं कर सकता है, क्योंकि Constant पूरे Program में स्थिर होते है, जिनका मान Change नहीं हो सकता है, जबकि Assignment Operator अपने Left Side के Variable का मान Change करने का ही काम करता है।

जब हम "C" Program में Division की प्रक्रिया को Perform करते हैं, तब प्राप्त होने वाले Result का चिन्ह निम्नानुसार होता है:

```
-20/4   =   -5
20/-4   =   +5
```

यानी यदि हम किसी Negative Sign वाली संख्या में किसी Positive Sign वाली संख्या का भाग देते हैं, तो Resultant मान भी Negative ही प्राप्त होता है। जबकि यदि हम किसी Positive Sign वाली संख्या में Negative Sing की संख्या का भाग देते हैं, तो प्राप्त होने वाला मान भी Positive Sign का होता है। सारांश ये है कि किसी "C" Program में भाग की प्रक्रिया करने पर Resultant मान का Sign Numerator या अंश के Sign के समान होता है।

हम हमारी जरूरत के आधार पर विभिन्न Characters को Computer की Memory में Store करते हैं। लेकिन Computer में कोई भी मान Character के रूप में Store नहीं होता है। हम जितने भी Character किसी Character Variable में Store करते हैं, वे सभी Characters Computer की Memory में Integers के रूप में ही Store होते हैं।

Computer में विभिन्न Characters को Represent करने के लिए एक Standard Code System Develop किया गया है। इस Code System में हर Character को एक Integer मान Assign किया गया है। हम जब भी किसी Character Identifier में किसी Character को Store करते हैं, वह Character उस Integer मान से Replace हो जाता है, और Computer की Memory में उस Character को Represent करने वाला Integer मान Store हो जाता है।

ठीक इसी तरह से जब हम उस Identifier में Stored Character को Display करना चाहते हैं, तब उस Character Identifier की Memory Location पर Stored Integer Code फिर से Character में Convert हो जाता है और हमें उस Character से Associated Code के स्थान पर वही Character दिखाई देता है। हर Character के साथ Associated Code को उस Character का **ASCII Code** कहते हैं।

उदाहरण के लिए यदि हम किसी Character प्रकार के Identifier में एक Character **A** Store करते हैं, तो वास्तव में Character **A** Computer की Memory में Store नहीं होता है, बल्कि ये Character **A** इसकी ASCII Code Value **65** से Replace हो जाता है और Identifier की Memory Location पर ये ASCII मान 65 Store हो जाता है।

जब हम फिर से उस Character को Access करना चाहते हैं, तब Computer इस ASCII Code **65** को Character **A** से Replace कर देता है और हमें Screen पर Character **A** दिखाई देता है ना कि ASCII Code 65,

वास्तव में एक Character प्रकार का Identifier एक छोटे Size के Integer मान को ही Computer की Memory में Store करता है। इसलिए यदि हम किसी Character Type के Identifier के मान को **%c** Control String का प्रयोग करके Display करते हैं, तो हमें उस Identifier में Stored Character Screen पर Display होता है, जबकि यदि हम **%d** Control String का प्रयोग करके उस Identifier के मान को Display करते हैं, तो हमें उस Character का ASCII Code Screen पर Display होता है। इस प्रक्रिया को हम निम्न Program द्वारा समझ सकते हैं:

Program:

```
#include <stdio.h>
#include <conio.h>

main()
{
    char x = 'A';

  printf("\n Character in the Identifier x is  = %c ", x);
  printf("\n ASCII Code of the Character %c is = %d ", x, x);

  getch();
}
```

Output:

```
Character in the Identifier x is = A
 ASCII Code of the Character A is = 65
```

हालांकि हम Integer व Float प्रकार के Identifiers के साथ विभिन्न प्रकार की Calculations करते हैं, लेकिन हम किसी Character प्रकार के Identifier के साथ भी विभिन्न प्रकार की प्रक्रियाएं कर सकते हैं। अन्तर केवल इतना है, कि जब हम किसी Character प्रकार के Identifier में किसी Character को Store करके उस पर किसी प्रकार की Calculation को Apply करते हैं, तो वह Calculation Character पर Perform नहीं होती है, बल्कि उस Character प्रकार के Identifier में Stored Character के ASCII Code पर Perform होती है। इस पूरी प्रक्रिया को हम निम्न Program द्वारा समझ सकते हैं:

Program:

```
#include <stdio.h>
main()
{
    char x = 'A', y = 'p';

    printf("\n x - y = %c ", y - x);
    printf("\n x += 37 = %c ", x += 37);
    printf("\n (++y + 53)-(x += 12) = %c", (++y + 53)-(x += 12));
    printf("\n x * 2 - 60 = %c", x * 2 - 60);
    printf("\n y / 2 + 20 = %c", y / 2 + 20);
}
```

Output:

```
x - y = /
x += 37 = f
(++y + 53)-(x += 12) = 4
x * 2 - 15 = L
y / 2 + 20 = L
```

Exercise:

Explain the flow of this program.

हालांकि हम विभिन्न प्रकार के Mathematical Calculations को एक Computer Program द्वारा Perform करते हैं, लेकिन जिस तरह से हम Real Life के Mathematical Expressions को लिखते हैं, "C" Language में उस तरह से लिए गए Expression Run नहीं होते हैं। उदाहरण के लिए एक Real World Situation में निम्न Statement एक Valid Statement है:

```
x = (p.n.y.3-2)(l.3/m+33)
```

यदि इसी Expression को हमें "C" Program में लिखना हो, तो हमें हर Symbol को Specify करना जरूरी होगा और इस Statement को निम्नानुसार लिखना पडेगा:

```
x = (p * n * y * 3 -2 ) * (l * 3 / m + 33)
```

Control Instruction

तीसरे प्रकार के Instructions को Control Instructions कहा जाता है। ये Instructions Computer को विभिन्न प्रकार के काम करने के लिए तथा Program के Flow को मनचाही दिशा देने के लिए Use किए जाते हैं। विभिन्न प्रकार के **Decision Making** Instructions द्वारा हम Computer को विभिन्न प्रकार के Decision लेने की क्षमता प्रदान करते हैं, तथा Iterative Instructions का प्रयोग करके हम Computer को Repetitive या बार–बार दोहराए जाने वाले कामों को करने के लिए Instructions देते हैं। Control Instructions मुख्यतया चार प्रकार के होते हैं, जो अग्रानुसार है और इन्हें हम अगले अध्याय में विस्तार से समझेंगे।

1 ***Sequence Control Instructions***
2 ***Selection or Decision Control Instructions***
3 ***Repetition or Loop Control Instructions***
4 ***Case Controls Instructions***

Precedence of Operators

''सी'' भाषा में हर Operator का एक प्राथमिकता का क्रम होता है। जिससे गणनाओं में होने वाली परेशानियां समाप्त हो जाती हैं। उदाहरण के लिए निम्न Expression देखें:

```
11 + 23+20/6 – 4 = 5
11 + 23+20/6 – 4 = 27
11 + 23+20/6 – 4 = 33.333
```

जिस परिणाम की हमें जरूरत है, वह इनमें से कौनसा है, ये पता लगाना बहुत ही मुश्किल हो जाएगा क्योंकि तीनों ही मान सही हैं। ऐसे में एक ऐसे क्रम की आवश्यकता हुई, जिससे यह पता चल सके कि पहले कौनसी गणना होगी व बाद में कौनसी, ताकि हमें प्राप्त होने वाला मान वही हो जो हम चाहते हैं।

''सी'' में इस प्रकार की परेशानी से बचने के लिए Operators को एक प्राथमिकता क्रम में व्यवस्थित किया गया। इसमें गणनाए इसी क्रम में होती हैं। जो Operator प्राथमिकता क्रम में पहले आता है, उसके Operands की गणना पहले होती है और जो Operator प्राथमिकता क्रम में बाद में आता है, उसके Operands की गणना बाद में होती है। ये प्राथमिकता क्रम सारणी निम्नानुसार है:

Category	Operator	Operations	Precedence	Associatively
Highest Precedence	( ) [ ] -> .	Function Call Array Subscript Reference Dot Operator	1	Left To Right
Unary	!	Logical Negation	2	

	~ + - ++ -- & * sizeof type	Bit wise 1st complement Unary Plus Unary Minus Increment Decrement Address Indirection Size of an Object Type Cast Conversion		Right To Left
Multiplication	* / %	Multiply Divide Reminder	3	Left To Right
Additive	+ -	Binary Plus Binary Minus	4	Left To Right
Shift	<< >>	Shift Left Shift Right	5	Left To Right
Relational	< <= > >=	Less Than Less Than or equal to Greater Than Greater Than or Equal to	6	Left To Right
Equality	= = !=	Equal To Not Equal To	7	Left To Right
Bit wise AND	&	Bit wise AND	8	L to R
Bit wise XOR	^	Bit wise XOR	9	L to R
Bit wise OR	|	Bit wise OR	10	L to R
Logical AND	&&	Logical AND	11	L to R
Logical OR	| |	Logical OR	12	L to R
Conditional	? :	Ternary Operator	13	L to R
Assignment	= *= /= %= += - = & = ^ = | = << >>	Simple Assignment Assign Product Assign Quotient Assign Reminder Assign Sum Assign Difference Assign Bit wise AND Assign Bit wise XOR Assign Bit wise OR Assign Left Shift Assign Right Shift	14	R TO L

Comma	,	Evaluate	15	L to R

Computer में जब किसी Arithmetical Expression Calculation के लिए Perform होता है, तब उस Expression में Use किए गए विभिन्न प्रकार के Operators अपने Direction व प्राथमिकता क्रम के अनुसार Execute होते हैं। कोई Expression जिस तरह से विभिन्न Steps में प्राथमिकता क्रम व Direction (***Precedence and Associativity***) के आधार पर Perform होकर एक Result Generate करता है, उन Steps की श्रृंखला को Operations की Hierarchy (***Hierarchy of Operations***) कहते हैं।

कोई Expression किस क्रम व दिशा के आधार पर Execute होकर Accurate Result Generate करेगा, इसके लिए "C" Language में कुछ नियम निर्धारित किए गए हैं। ये नियम निम्नानुसार हैं:

1 किसी Expression में यदि Parenthesis का प्रयोग किया गया हो, तो सबसे पहले उस Parenthesis का Expression Calculate होता है। साथ ही यदि एक से ज्यादा Parenthesis Nested हों, तो सबसे पहले Inner Most यानी सबसे अन्दर का Parenthesis Solve होता है और फिर क्रम से बाहर के Parenthesis Solve होते हैं।

2 जो Operator प्राथमिकता क्रम की सारणी में पहले आता है, वह Operator अपना Calculation पहले करता है तथा जो Operator प्राथमिकता क्रम सारणी में बाद में आता है, वह अपना Operation बाद में Perform करता है। ठीक इसी तरह से सभी Operators उसी दिशा में Operations Perform करते हैं, जिस दिशा में उसे प्राथमिकता क्रम सारणी में दर्शाया गया है।

Example:

	x = (11 * 2) / 3 + 5 / 5 – 1 + 6 * 6 – 4	**Expression**
Step 1	x = **(22 / 3)** + 5 / 5 – 1 + 6 * 6 – 4	Operation *
Step 2	x = 7 + **(5 / 5)** – 1 + 6 * 6 – 4	Operation /
Step 3	x = 7 + 1 – 1 + **(6 * 6)** – 4	Operation /
Step 4	x = **(7 + 1)** – 1 + 36 – 4	Operation *
Step 5	x = **(8 – 1)** + 36 – 4	Operation +
Step 6	x = **(7 + 36)** – 4	Operation –
Step 7	x = **(43 – 4)**	Operation +
Step 8	**(x = 39)**	Operation -

किसी भी Arithmetical Expression में यदि *, / या % में से कोई Operator हो, तो सबसे पहले वही Operator अपना काम करता है। उसके बाद यदि Expression में + या – में से कोई Operator हो तो वह Operator अपना काम करता है और सबसे अन्त में = Operator अपना काम करता है। प्राथमिकता सारणी में हम देख सकते हैं कि केवल **Unary** व **Assignment** Operators ही **Right To Left** Operation Perform करते हैं, शेष सभी Operators **Left To Right** दिशा में अपना काम करते हैं।

उदाहरण वाले Expression में हम देख सकते हैं कि इसमें विभिन्न प्रकार के Operators Result Generate करने के लिए Involved हैं। चूंकि, **Unary** व **Assignment** Operators के अलावा सभी प्रकार के Operators **Left To Right** दिशा में Operation Perform करते हैं, इसलिए इस Expression सबसे पहले **(11 * 2)** का Expression Execute होता है, जो मान **22** Generate करता है।

अगले Step में **(22 / 3)** Expression Execute होता है, जो मान **7** Generate करता है। चूंकि इस Expression में अभी भी * व / Operators हैं, इसलिए अगले Statement में **Left To Right** चलते हुए जो Operator सबसे पहले मिलता है वह **(5 / 5)** Expression में होता है, इसलिए अब ये Expression Execute होता है।

फिर **Left To Right** आगे बढते हुए अगले Operation में **(6 * 6)** मिलता है, और अपना काम पूरा करके **36** Generate करता है। अब हम देख सकते हैं कि चौथे Step वाले Expression में *, / या % में से कोई भी Operator नहीं है, इसलिए फिर से **Left To Right** चलते हुए अब + व – का Expression Execute होता है और इस Expression में सबसे पहले **(7 + 1)** Execute होकर **8** Generate करता है। फिर पांचवे Statement में **(8 – 1)** से **7** Generate होता है। फिर छठे Step में **(7 + 36)** से **43** Generate होता है और सातवें Step में **(43 – 4)** से **39** Generate होता है।

चूंकि Assignment Operator सबसे बाद में Execute होता है और **Right To Left** Execute होता है, इसलिए अन्तिम Step में **(x = 39)** Expression Execute होता है और Variable **x** में मान **39** Store हो जाता है।

Type Conversion in Expressions

Computer में विभिन्न Data Types के मानों को Store करके Process किया जाता है। विभिन्न प्रकार के मानों के साथ प्रक्रिया करते समय कई बार ऐसी स्थितियां पैदा हो जाती हैं, जिसमें किसी एक Data Type के मान को किसी दूसरे Data Type के मान में Convert करके किसी Calculation को Perform करना होता है।

इस प्रकार की स्थितियों से निपटने के लिए "C" Language हमें दो तरह से **Type Conversion Mechanism** Provide करता है, जिसके आधार पर हम एक प्रकार के मान को दूसरे Type के मान में Convert करके अपनी Requirement को पूरा करते हैं। ये दोनों तरीके निम्नानुसार हैं:

Automatic Type Conversion

''सी'' भाषा में विभिन्न प्रकार के Variables व Constant की Mixing कर सकते हैं, लेकिन Execution के समय ये Expressions एक विशेष नियम का पालन करते हैं। हम जानते हैं कि Expressions में Compiler दो या दो से अधिक Operands के साथ क्रिया करके एक Operand में उसके मान को Store करता है। यदि Operand अलग–अलग Data Type के हों तो Lower Type का Operand Upper Type के Operand में बदल जाता है और Result हमेंशा Higher Type का प्राप्त होता है। इस सम्बंध में ''सी'' कुछ नियमों का पालन करता है जो निम्नानुसार है:

1 सभी **short** व **char** प्रकार के Variables Automatically int प्रकार में Convert हो जाते हैं।
2 यदि एक Operand **Long Double** प्रकार का हो तो Execution के दौरान दूसरा Operand भी Long Double प्रकार के Operand में Convert हो जाता है और Result Long Double प्रकार का प्राप्त होता है।
3 यदि एक Operand **Double** प्रकार का हो तो दूसरा भी Double में Convert हो जाता है और Result Double प्रकार का प्राप्त होता है।
4 यदि एक Operand **Float** प्रकार का हो तो दूसरा भी Float प्रकार में बदल कर Output Float प्रकार का प्राप्त होता है।
5 यदि एक Operand **unsigned long int** है तो दूसरा भी unsigned long int में Convert हो जाएगा और Result unsigned long int में ही प्राप्त होता है।
6 **long int** में बदल कर long int का Result प्राप्त किया जा सकता है या फिर दोनों Operand unsigned long int में बदल जाएंगे व Result unsigned long int में प्राप्त होगा।
7 यदि एक Operand **unsigned long int** है तो दूसरा भी unsigned में बदल कर Result unsigned int में प्राप्त होगा।

Assignment Operator के Left Side में जिस प्रकार के Data Type का Variable होता है, Expression उसी प्रकार का Data Return करेगा। उदाहरण से समझें तो माना तीन वेरियेबल्स् क्रमशः int, long व Double प्रकार के हैं:

```
int a;
long b;
double c;
```

अब यदि a = b * c; Expression Use करें तो b का मान long से double Type के Data में Convert हो जाएगा व प्राप्त मान a में Store हो जाएगा। लेकिन a int प्रकार का है इसलिए a में केवल पूर्णांक मान ही Store होगा ना कि double प्रकार का क्योंकि int प्रकार का Identifier केवल पूर्णांक ही Store कर सकता है। इसलिए किसी भी Expression का Final Result Expression के Left Side के Identifier के Data Type पर निर्भर रहता है।

Manual Type Conversion OR Casting

जब हमें विशेष प्रकार की गणनाएं करनी होती है, तब हम Automatic Type Conversion का प्रयोग नहीं कर सकते हैं। ऐसे में हमें विशेष प्रकार से Type Conversion करना पडता है। इस प्रकार से किसी Expression में किसी Identifier के Declare किये गए Data Type को बदल कर नए Data Type में Convert करके गणना करते हैं। इस प्रकार के Conversion को **Casting** करना कहते हैं।

किसी Variable के Data Type को किसी दूसरे Data Type में Convert करके Calculation Perform करने के लिए हमें उस Variable से पहले उस Data Type को Specify करना होता है, जिसमें Variable के मान को Convert करना है। उदाहरण के लिए निम्न Program देखिए:

Program

```
#include<stdio.h>
main()
{
    int a = 15, b = 2;
    float c;

    c = a/b;

    printf("C is %f ",c);
    getch();
}
```

इस Program को जब Run किया जाता है, तब हमें Output में 7.000000 मान प्राप्त होता है, जबकि हमें 7.500000 मान प्राप्त होना चाहिए। ऐसा इसलिए होता है, क्योंकि Variable **a** व **b** दोनों ही Integer प्रकार के हैं। हालांकि Variable **c** एक Float प्रकार का Variable है, लेकिन फिर भी ये Variable वही मान Store कर सकता है, जो मान **a/b** Expression से Generate होता है और **a** व **b** दोनों ही Integer होने की वजह से दसमलव वाला मान 7.5 Generate नहीं कर सकते हैं, इसलिए Variable **c** में Store होने वाला Resultant मान 7 ही होता है, जो एक Float प्रकार के Variable में Store होने की वजह से 7.000000 में Convert हो जाता है।

Variable **c** में **7.5** तभी Store हो सकता है, जब **a** या **b** में से कोई Float प्रकार का हो। इस प्रकार की स्थिति में हमें इस Expression के लिए **a** या **b** में से किसी एक Variable के मान को Float में Convert करना जरूरी होता है।

किसी Variable के मान को किसी एक Expression के लिए एक Type से दूसरे Type में Convert करने के लिए हमें उस Variable की **Manual Casting** करनी पडती है और किसी

Variable की Casting करने के लिए हमें उस Variable से पहले **Braces** के बीच में उस Data Type को Specify करना होता है, जिस Data Type में हम उस Variable के मान को Change करना चाहते हैं। पिछले Program के आधार पर इस प्रक्रिया को हम निम्नानुसार समझ सकते हैं:

```
c = (float)a / b;
```

पिछले Program में जब हम इस Statement को पहले लिखे गए Statement से Replace करते हैं, तो Output में हमें Variable **c** का मान **7.5000000** प्राप्त होता है। ऐसा इसलिए होता है, क्योंकि इस Statement के Execute होने से पहले Variable **a** जो कि Integer प्रकार का है, Float प्रकार के मान में Convert हो जाता है।

चूंकि कभी भी दो अलग प्रकार के Variables के बीच किसी प्रकार की Calculation Perform नहीं होती है, इसलिए जब Integer प्रकार के Variable **a** को Float प्रकार में Convert कर दिया जाता है, तब Integer प्रकार का दूसरा Variable **b** की भी **Automatic Type Casting** होती है और वह भी Float प्रकार के मान में Convert हो जाता है। अब **a** व **b** दोनों Float प्रकार के मान के आधार पर Calculation Perform करते हैं, जिससे Float प्रकार का मान **7.5** Generate होता है।

Assignment Operator इस मान **7.5** को Variable **c** में Store कर देता है और जब हम Variable **c** के मान को Screen पर Display करते हैं, तब हमें Variable **c** का मान **7.5000000** प्राप्त होता है। यदि हम चाहें तो उपरोक्त Statement को निम्नानुसार भी लिख सकते हैं:

```
c = a / (float) b;
```

Statement को इस तरह से लिखने पर भी Program के Output पर कोई फर्क नहीं पडता है और हमें वही Output प्राप्त होता है, जो पिछले Statement का प्राप्त होता है। Modified Program को हम निम्नानुसार लिख सकते हैं:

Program

```
#include<stdio.h>
main()
{
    int a = 15, b = 2;
    float c;
    c = (float) a / b;
    printf("C is %f ",c);
}
```

Exercise:

What would be the flow of the program if we replace the statement "**c = (float) a / b**" with "**c = a / (float) b**". Explain.

Function Calling and Function Arguments

किसी पहले से बने हुए Function की सुविधा को प्राप्त करने के लिए हमें उस Function को Call करना होता है। किसी Function को किसी भी दूसरे Function में Call करने के लिए हमें केवल Source Calling Function का नाम Call किए जाने वाले Target Function में लिखना होता है।

Computer में विभिन्न प्रकार के Functions को विभिन्न प्रकार के कामों को पूरा करने के लिए बनाया जाता है। बनाया गया हर Function किसी एक ही काम को Perfectly पूरा करने के लिए बनाया गया होता है।

इसलिए कई बार किसी Call किए जा रहे Function से किसी प्रकार का काम पूरा करवाने के लिए उसे Target Function से कुछ मान प्रदान किए जाते हैं। ये मान Call किए जा रहे Function के Parenthesis में Specify किए जाते हैं। किसी Function को Call करते समय उसके कोष्ठक के बीच Specify किए जाने वाले मान को **Argument** कहा जाता है।

उदाहरण के लिए अभी तक हमने **printf(), scanf(), clrscr()** व **getch()** Functions को main() Function में कई बार Call किया है। हम देख सकते हैं कि **clrscr()** व **getch()** Function के कोष्ठक में हमने किसी भी Program में कोई Data प्रदान नहीं किया है, यानी इन दोनों Functions को main() Function में Call करते समय हमने किसी भी Program में इन्हें कोई **Argument Pass** नहीं किया है।

जबकि printf() व scanf() Function को हमने जितनी बार भी Use किया है, हर बार उसमें कम से कम एक String को तो Specify किया ही है। printf() व scanf() Function में हम जिस String को Specify करते हैं, उसे ही printf() Function का **Argument** कहते हैं।

सारांश ये कि किसी एक Function **X** में किसी दूसरे Function **Y** का नाम लिखने की प्रक्रिया को Function **X** में Function **Y** को Call करना कहते हैं, जबकि यदि इस Function **Y** में किसी Data को Specify किया जाए, तो ये Data Function **Y** का Argument कहलाता है।

Exercise:

1 "C" Language में कोई Program Develop करते समय हम किन–किन Instructions का प्रयोग करके Program Create करते हैं?

2 **Implicit** व **Explicit** Initialization में क्या अन्तर है ? एक उदाहरण द्वारा समझाईए।

3 Operators की **Precedence** व **Associativity** से आप क्या समझते हैं?

4 Operators कितने प्रकार के होते हैं, संक्षेप में वर्णन करो।

5 Hierarchy of Operators से आप क्या समझते हैं ? निम्न Expressions की Hierarchy of Operations के Steps को प्रदर्शित कीजिए:

A result = 3 / 2 * 4 + 3 / 8 + 2 – 6 % 5

B result = 2 * 3 / 4 + 4 / 4 + 8 – 2 % 3 + 2 – 5 * 3

C result = a % 3 – 2 * b + (3 – c) / 4 (c * 3 / 4 % 3) + 4

(Where a = 9, b = 5 and c = 2)

D result = p / 2.9 * q + r / 8.8 + 9.9 – s * 6 % 5.6

(Where p = 1.9, q = 5.1, r = 2.0 and s = 1.2)

6 Type Conversion कितने प्रकार के होते हैं? सभी को एक–एक उदाहरण द्वारा समझाईए।

7 Function किसे कहते हैं? ये कितने प्रकार के होते हैं ? समझाईए।

8 Function Arguments से आप क्या समझते हैं ? किसी एक Function को दूसरे Function में किस प्रकार से Call किया जाता है ? एक उदाहरण द्वारा समझाईए।

String and Character Functions

"C" Language की Library में हमें Strings व Characters के साथ प्रक्रिया करने से सम्बंधित विभिन्न प्रकार के Functions प्रदान किए गए हैं, जिनका प्रयोग हमें विभिन्न प्रकार की परिस्थितियों में करना होता है।

Working with String

जब किसी वेरियेबल में ''सी'' Characterset का पूरा एक समूह Input किया जाता है, जिसका एक निश्चित अर्थ होता है, तो इसे **String** कहते हैं। जैसे "Krishna", "Mohan", "Govind" आदि String हैं क्योंकि इनमें "C" **Characterset** के Characters का पूरा एक समूह है और इनका एक विशेष अर्थ भी है।

हम ऐसे किसी Computer Program की कल्पना भी नहीं कर सकते हैं, जिसमें String का प्रयोग ना किया गया हो। User को किसी ना किसी तरह की Information देने के लिए ही हम विभिन्न प्रकार के Programs Develop करते हैं और Information हमेंशा Characters के एक समूह यानी String के रूप में ही Represent हो सकता है। इसलिए **String Manipulation** किसी भी Program का एक बहुत ही महत्वपूर्ण हिस्सा होता है।

scanf() Function किसी String को Accept करने में पूरी तरह से सक्षम नहीं होता है, क्योंकि scanf() function को जैसे ही ***Blank Space, Carriage Return, New Line Character, Tab, Form Feed*** Character प्राप्त होता है, scanf() Function Terminate हो जाता है। इसलिए अगर हमें प्रोग्राम में किसी String के साथ कोई प्रक्रिया करनी हो, तो हमें ऐसे Functions की जरूरत होती है, जो पूरे String को Accept कर सके और मेमोरी में Store कर सके।

"C" Language में Characters व Strings पर काम करने के लिए कुछ अलग Functions बनाए गए हैं, जो scanf() Function की कमियों को पूरा करते हैं। एक और ध्यान देने वाली बात ये है कि जब भी हमें किसी प्रोग्राम में String के साथ किसी प्रकार की प्रक्रिया करनी होती है, तब हमें उस String को Computer की Memory में Store करना पडता है।

हम जिस तरह से किसी एक Character या किसी एक Numerical मान को Computer की Memory में Store करके Manipulate व Process करते हैं, ठीक उसी तरह से हम किसी String को Computer की Memory में Store करके Manipulate नहीं कर सकते हैं, क्योंकि एक String कभी भी एक Single Data Item नहीं होता है, बल्कि Character प्रकार के बहुत सारे Data Items का पूरा एक समूह होता है। इसलिए एक String को Computer की Memory में Store करने के लिए हमें हमेंशा कम से कम एक **One-Dimensional Array** बनाना पडता है।

एक उदाहरण द्वारा समझते हैं कि scanf() Function String पर काम करने के लिए क्यों उपयुक्त नहीं है। हम एक प्रोग्राम बनाते हैं जिसमें किसी व्यक्ति का पूरा नाम व उसकी उम्र Input किया

जाएगा और Output में वापस उसका नाम और ये Message कि अगले साल उसकी उम्र एक साल अधिक हो जाएगी, Print होगा। इस प्रोग्राम में दो वेरियेबल Declare करने होंगे, जिसमें एक वेरियेबल नाम Accept करेगा व दूसरा वेरियेबल उम्र Accept करेगा।

इस प्रोग्राम में हमने पहला वेरियेबल ***Age* int** प्रकार का लिया है, क्योंकि उम्र का मान संख्यात्मक होता है। दूसरा वेरियेबल name लिया है। यह Character प्रकार का है क्योंकि नाम में Characters ही होते है। char प्रकार का एक वेरियेबल केवल एक ही Character Accept कर सकता है और String Input करने के लिए किसी अन्य प्रकार का Data Type "C" में नहीं है। इसलिए name को एक One Dimensional Array के रूप में Declare किया गया है।

Array का प्रयोग तब किया जाता है, जब एक ही Data Type के ढेर सारे Data किसी एक ही Variable में Store करने होते हैं।

चूंकि इस Program में हम एक नाम Computer की Memory में Store करना चाहते हैं, और एक नाम में Characters का पूरा एक समूह होता है, इसलिए नाम को Store करने के लिए हमने इस Program में **name** नाम का एक Array Declare किया है।

Program

```
/* Inputting Name and Age & Getting Name and Age one Year Incremented in output */
#include<stdio.h>
main()
{
    /* Declaration Section */
    int age;
    char name[30];
    clrscr();

    /* Input Section */
    printf("\n Enter Name");
    scanf("%s", name);
    printf("\n Enter Age");
    scanf("%d", &age);

    /* Process Section */
    age=age+1;

    /* Output Section */
    printf("Your Name is %s", name);
    printf("Your Age Will Be %d Next Year", age);
```

```
        getch();
    }
```

इस Program में यदि हम Array का प्रयोग ना करें, तो नाम के हर अक्षर को Store करने के लिए एक अलग वेरियेबल Declare करना पडेगा, क्योंकि char प्रकार का Data Type एक वेरियेबल में केवल एक ही अक्षर Store कर सकता है। ऐसे में प्रोग्राम काफी बडा और जटिल हो जाएगा, क्योंकि हर अक्षर के लिए हमें एक printf() Function द्वारा Message देना होगा और हर Input किये गये Character को पढने के लिए एक scanf() Function लिखना पडेगा। इस समस्या से बचने के लिए ''सी'' में एरे की व्यवस्था की गई।

एरे की विशेषता ये है कि इसके Bracket में जितनी संख्या लिखी जाती है, हम उतने ही Data इसमें Store कर सकते हैं और जिस Data Type का Array Declare किया जाता है, वह Array उसी तरह के मान मेमोरी में Store करता है।

इस प्रोग्राम में हमने char प्रकार का Array Declare किया है और Bracket में Array की Size 30 दी है, इसलिए इस Array में हम केवल Characters Store कर सकते हैं। और इस Array में कुल अक्षरों की संख्या 30 हो सकती है।

यहां हमने केवल एक ही Bracket मे Size दी है इसलिए इसे **One-Dimensional Array** कहते हैं। यदि एक और Bracket बना कर कोई Size उसमें लिख दें, तो यह **Two Dimensional Array** बन जाता है, और यही प्रक्रिया यदि आगे भी अपनाई जाए तो Declare होने वाला Array, **Multi Dimensional** हो जाता है। जैसे int value[10][12] एक Two Dimensional Array का उदाहरण है और int sum[2][3][4] एक Multi Dimensional Array का उदाहरण है।

हम जानते हैं कि जब हम किसी Identifier को Declare करते हैं, तो वह Identifier अपने Data Type के अनुसार मेमोरी में कुछ जगह Reserve करता है। ठीक इसी तरह से एक Array भी Memory में अपनी Size के अनुसार कुछ Space Reserve कर लेता है, लेकिन Array की विशेषता ये है कि हम स्वयं ये तय करते हैं, कि एक Array Memory में कितनी जगह Reserve करेगा, जबकि सामान्य Identifier के सम्बंध में Identifier Memory में कितनी जगह Reserve करेगा, ये Use किए जा रहे Identifier के Data Type पर निर्भर होता है। Array एक ऐसा Identifier होता है, जिसमें हम एक ही प्रकार के एक से अधिक Data Store कर सकते हैं।

इस प्रोग्राम के अनुसार यदि हम **char name;** लिखते हैं, तो यह Statement Memory में एक Byte की Space लेगा और इसमें हम सिर्फ एक Character Store कर सकेंगे। लेकिन एक नाम में हमेंशा एक से ज्यादा Characters होते हैं, इसलिए हमें इस name नाम के Identifier में एक से अधिक Characters Store करने हैं।

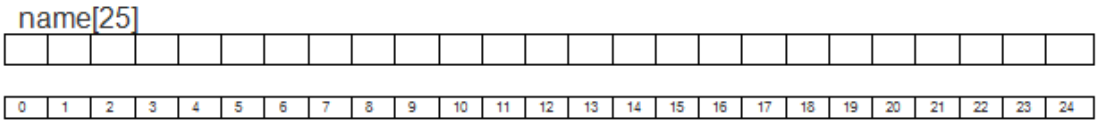
इस स्थिति में एक name नाम के Character प्रकार के Array में हम कितने Characters Store करना चाहते हैं, ये Size भी हम Array के साथ Specify कर देते हैं और Statement को **char**

name[size]; लिखते हैं, जहां size उस मान को Specify करता है, जितने Characters हम नाम के रूप में Memory में Store करना चाहते हैं।

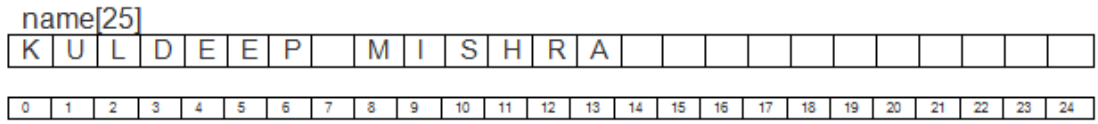
हमनें इस Program में **name** Identifier की Size एक Bracket में char name[25] Statement द्वारा Specify कर दिया है। अब name नाम का Identifier Memory में 25 Byte की जगह एक क्रम में Reserve कर लेता है और name नाम के Identifier की Size 25 Byte हो जाती है।

name की Size 25 Byte इसलिए हो जाती है, क्योंकि char प्रकार का एक Data Memory में एक Byte की Space Reserve करता है इसलिए char प्रकार के 25 Data Memory में 25 Byte की Space लेंगे।

यदि Data Type char प्रकार से बदलकर int प्रकार का कर दें व Array को char name[25] से बदलकर int name[25] कर दें, तो यह Array Memory में 25 * 2 = 50 Byte की Space Reserve करेगा क्योंकि int प्रकार का एक Data Memory में दो Byte की जगह लेता है। इसलिए int प्रकार के 25 Data Memory में 50 Byte की जगह घेरेंगे। name नाम का Array मेमोरी में निम्न चित्रानुसार Space Reserve करता है:



अब यदि हम इसमें कोई नाम जैसे KULDEEP MISHRA Store करें तो वह निम्नानुसार Memory में Array की विभिन्न Locations पर Store होगा:



इस तरह से हर अक्षर Memory की अलग–अलग Locations पर जा कर Store होता है। हम यदि 25 से कम अक्षर भी Input करेंगे, तो भी **name** 25 Byte तो Reserve रखेगा ही। ध्यान दें की Blank Space भी एक Character है।

सिर्फ char प्रकार के Array में हमें केवल Characters या String Input करने का Statement देना होता है, जबकि यदि हम किसी और प्रकार का Array Declare करें तो Array की हर Locations पर जा कर मान Store करना पडता है और मान पुनः प्राप्त करने के लिए भी हर Location पर जाना पडता है।

char प्रकार के Array में हम जैसे ही एक Character Input करते हैं, प्रोग्राम Control स्वयं ही Array की अगली Location पर चला जाता है और Input होने वाले मान को Store करता जाता है साथ ही Array में जब String या Character Input करना होता है तब हमें **&** Address Operator का प्रयोग नहीं करना होता है क्योंकि Array के नाम से ही "C" Compiler समझ जाता है कि Input किए जाने वाले Character या String को किस Identifier के किस Location में Store करना है। यदि हम & Address Operator का प्रयोग करते हैं तो प्रोग्राम सही तरीके से काम नही करता है।

एक बात हमेंशा ध्यान रखें कि हमें जितने Characters Input करवाने हैं, Array की Size हमेंशा उससे एक अधिक रखनी चाहिये, क्योंकि "C" Compiler हर एरे का अन्त वहां समझता है जहां उसे '**\0**' **(NULL)** Character प्राप्त होता है। जब हम String Input करवाते हैं, तब "C" Compiler स्वयं ही String के अंत में ये NULL Character लगा देता है।

यदि हम Array की Size कम लेते हैं और उसमें String Input करवाने के बाद यदि NULL के लिए कोई जगह नहीं बचती है, तो "C" Compiler स्वयं ही Input किये गए String के अन्तिम Character के स्थान पर NULL Character को Replace कर देता है। इसलिए यदि हमें 10 Character का String Input करना है, तो हमारे Array की Size 11 होनी चाहिये।

इस प्रोग्राम को जैसे ही Run किया जाता है, प्रोग्राम कंट्रोल को **clrscr**() Function प्राप्त होता है। ये Function Screen पर पहले से लिखे हुए Messages को साफ कर देता है। अब printf() Function Run होता है और हमसे नाम मांगता है।

माना हम "**KULDEEP MISHRA**" नाम का String Input करके Enter Key Press करते हैं, तो प्रोग्राम का दूसरा Statement Execute नहीं होता, बल्कि सीधे ही एक गलत Result प्राप्त हो जाता है जिसमें KULDEEP String ही प्राप्त होती है। Space के बाद का नाम भी नहीं आता और Program User से Age भी नहीं मांगता है।

जैसाकि हमने पहले भी कहा कि ऐसा इसलिए होता है, क्योंकि scanf() Function ***Blank Space, Form Feed, Tab Key*** या ***New Line Character Constant*** मिलते ही Terminate हो जाता है, यानी इनके बाद Input किये जा रहे Characters Array में नहीं जाते हैं बल्कि **Garbage Value** के रूप में Memory में Store हो जाते हैं और यही Garbage Value अगले Identifier **age** को प्राप्त हो जाती है। चूंकि ***age* int** प्रकार का है, इसलिए इसको जब Character प्राप्त होता है, तो यह Output में अजीब से Symbols प्रिंट कर देता है।

gets(Array_Identifier) Function

scanf() Function की कमी को पूरा करने के लिए "C" Library में ***gets***() नाम का एक Function Provide किया गया है। ये फंक्शन Keyboard से आने वाली String को **Array_Identifier** में Store करने का काम करता है।

इस Function का प्रयोग करके हम मनचाही लम्बाई की String को किसी Array_Identifier में Input कर सकते हैं। ये Function तब Terminate होता है, जब हम String Input करने के बाद Keyboard के Enter Key को Press करते हैं। इस Function को समझने के लिए हमने पिछले Program को ही निम्नानुसार थोडा सा Modify किया है। अब ये Program केवल New Line Character से ही Terminate होता है और New Line Character तब Generate होता है, जब हम Keyboard पर स्थित Enter Key को Press करते हैं।

Program

```
#include<stdio.h>
main()
{
    int age;
    char name[30];
    clrscr();

    printf("\n Enter Name");
    gets(name);
    printf("\n Enter Age");
    scanf("%d", &age);

    age=age+1;

    printf("Your Name is %s", name);
    printf("Your Age Will Be %d Next Year", age);
    getch();
}
```

इस Function में कोष्ठक के अंदर हमें उस Array का नाम देना होता है, जिसमें हम String को Store करना चाहते हैं। इस Function में किसी भी Control String का प्रयोग नहीं किया जाता है, क्योंकि ये Function केवल Keyboard से String प्राप्त करने का काम करता है और String हमेंशा Array में Store होती है।

puts (Identifier name) Function

यह Function **gets**() Function का Complementary Function है। जिस तरह scanf() Function से प्राप्त मान को किसी Identifier में Store किया जाता है व प्रक्रिया के पश्चात उस Identifier में Stored मान को Screen पर printf() Function द्वारा प्रिंट किया जाता है, उसी प्रकार scanf() Function या gets() Function द्वारा प्राप्त String को **puts**() Function द्वारा Screen पर प्रिंट किया जा सकता है। यह String हम printf() Function द्वारा भी प्रिंट कर सकते हैं।

Program

```
#include<stdio.h>
main()
{
    int age;
    char name[30];

    puts("Enter Name");
    gets(name);
    puts("Enter Age");
    scanf("%d", &age);

    age=age+1;

    puts("Your Name is ");
    puts(name);

    //OR printf("Your Name is %s", name);
    printf("Your Age Will Be %d Next Year", age);
}
```

यह Function बिल्कुल उसी तरह प्रयोग किया जाता है, जिस तरह से **gets**() Function को Use किया जाता है। फर्क सिर्फ इतना है कि gets() Function में कोष्ठक में हमें उस Identifier का नाम लिखना होता है, जिसमें Input किया जाने वाला मान Store होना होता है, जबकि **puts**() Function में कोष्ठक में उस Identifier का नाम लिखना होता है, जिसमें Stored String को Screen पर Print करना है अथवा Double Quote के अन्दर वह Message लिखा जाता है, जिसे Screen पर प्रिंट करना होता है।

इस तरह से हम puts() Function को printf () Function के स्थान पर Message देने में प्रयोग कर सकते हैं और किसी Identifier में स्थित String को Screen पर प्रिंट भी करवा सकते हैं।

puts() Function की एक विशेषता यह भी है कि इस Function के उपयोग के समय हमें New Line के लिए '\n' Character Constant का प्रयोग नहीं करना पडता है, बल्कि यह Function स्वयं New Line में ही Message को प्रिंट करता है।

इस Function की कमी यह है कि इसके द्वारा हम Message व किसी Identifier में स्थित String दोनों को एक साथ प्रिंट नहीं करवा सकते हैं। यदि हमें Message को भी प्रिंट करना हो, तो पहले एक puts() Function द्वारा उस Message को प्रिंट करना होगा फिर दूसरे puts() Function द्वारा किसी Identifier में स्थित String को प्रिंट करना होगा।

इस प्रकार दो बार इस Function को Use करने के बजाय अक्सर हम ये काम एक ही printf() Function द्वारा कर लेते हैं। gets() Function व puts() Function केवल String पर ही काम करते हैं।

Working with Characters

''सी'' में कुछ Functions सिर्फ एक Character पर काम करने के लिए बनाए गए हैं। इनका प्रयोग तब किया जाता है, जब User को कई Options में से सिर्फ एक Option को Choose करना होता है। जैसे कि Menu Driven Programs में किसी खास Option को Choose करने के लिए किसी खास **Highlighted** Key को Press किया जाता है। ''सी'' में मुख्यतया निम्न Functions हैं, जो सिर्फ Characters पर काम करने के लिए प्रयोग किये जाते हैं:

getchar() Function

यह Function Keyboard से प्राप्त केवल एक अक्षर को Read करता है। इस Function को किसी भी तरह के Argument की जरूरत नहीं होती है और इसका कोष्ठक खाली ही रखा जाता है। जब इस Function का उपयोग किया जाता है और हम कोई Key Press करते हैं, तो यह Function उस अक्षर को Integer में बदल देता है, इसलिए Input किये गए Character को Use करने के लिए उस Character को किसी Identifier में Assign करना जरूरी होता है।

जिस Identifier में Character Store होता है, उस Identifier को हम दो तरह से Use कर सकते हैं: एक तो उस Identifier में Input किया गया Character होता है व दूसरा उसी Identifier में उस Character की ASCII Value रहती है।

जब हम कोई Character Input नहीं करते हैं व Enter Key Press कर देते हैं, तो Assign किये गए Identifier में Enter Key की ASCII Value Store हो जाती है और Output में कोई Character Print नहीं होता, बल्कि ASCII Value के रूप में अंक 10 print हो जाता है, जो कि Enter Key की ASCII Value है।

getchar() Function **getc**() Function का एक Macro होता है, यानी यदि हम चाहें तो getchar() Function के स्थान पर getc() Function का प्रयोग कर सकते हैं। इस Function का स्वयं का एक Built-In Buffer होता है। यानी इस Function का प्रयोग करने पर ये जरूरी नहीं होता है, कि हम केवल एक ही Character Input करें।

ये Function Keyboard से आने वाले Characters को तब तक अपने Buffer में Store करता रहता है, जब तक कि हम Keyboard पर स्थित Enter Key का प्रयोग नहीं करते हैं। लेकिन ये Function अपने Buffer में Stored Characters की पूरी Stream में से एक समय में केवल एक ही Character को Read करता है और उस Character की ASCII Value Return करता है।

Program

```
/*   Use of getchar Function          */
#include<stdio.h>

main()
{
    int asc;
    printf("Type a Character and press Enter");
    asc = getchar();
    printf("\n The Key is %c", asc);
    printf("\n Ascii of the key is %d", asc);
    getch();
}
```

इस प्रोग्राम को जब रन करते हैं तो Screen पर निम्न Message आता है:

```
Type a Character and press Enter
```

जब हम कोई Key Press करते हैं, तो Input होने वाले Character की ASCII Value **getchar**() Function द्वारा **asc** नाम के int प्रकार के Variable में Store हो जाती है। यदि हम एक से अधिक Character भी Input कर देते हैं, तो भी इस Function द्वारा **asc** Variable में केवल पहला Character ही Store होता है, शेष Character का कोई उपयोग नहीं होता।

अब यदि हम asc को **Character** Control String का प्रयोग करके प्रिंट करें, तो Input किया गया Character Print होता है और यदि **Integer** Control String द्वारा प्रिंट करें, तो उस Character की ASCII Value प्रिंट होती है। ध्यान दें कि हमने asc Variable int प्रकार का लिया है। यदि यह int प्रकार का ना लेकर char प्रकार का लेते हैं तो भी प्रोग्राम पर किसी प्रकार का कोई फर्क नहीं पडता है।

putchar() Function

यह Function एक Character को Screen पर प्रिंट करने का काम करता है। इसका कोष्ठक खाली नहीं रखा जाता है, बल्कि Argument के रूप में इसमें या तो वह Identifier देना पडता है जिसमें कोई Character लिखा हो या फिर Single Quote के अन्दर कोई Character लिखा जाता है और Output में वही Character Print हो जाता है। इसे समझने के लिए निम्न प्रोग्राम देखें, जिसमें:

- पहला *putchar()* Function Variable **a** में Store अक्षर K को प्रिंट करेगा।
- दूसरा *putchar()* Function Variable एक नई लाइन प्रिंट करेगा।
- तीसरा *putchar()* Function Variable नई लाइन में U प्रिंट करेगा।

Program

```
#include<stdio.h>
main()
{
    int asc;
    char a = 'K';
    putchar(a);
    putchar('\n');
    putchar('U');
}
```

Output

```
K
U
```

चलिए, अब हम इन दोनों Functions को Use करके एक Program बनाते हैं। ये Program User से कुछ Characters Input करने की Request करता है। User जब कुछ Character Input कर देता है, तब Program उस नाम के पहले 6 Characters को Screen पर Print कर देता है।

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    int character;

    printf("Enter some characters");
```

```
        character = getchar();

        printf("You have entered :");
        putchar(character);
        character = getchar();
        putchar(character);

        character = getchar();
        putchar(character);

        character = getchar();
        putchar(character);

        character = getchar();
        putchar(character);

        character = getchar();
        putchar(character);

        getch();
    }
```

Output

```
Enter some characters : Manohar
You have entered :Manoha
```

जब इस Program को Run किया जाता है, तब Program User से एक String Input करने के लिए Request करता है। जब User कोई String Input करता है, तब getchar() Function Keyboard से आने वाली String को अपने Buffer में Store कर लेता है। जब User String को Terminate करने के लिए Enter Key Press करता है, तब getchar() Function अपने Buffer में स्थित String में से सिर्फ पहले Character को Read करता है और उसकी ASCII Value को **character** नाम के Variable में Store कर देता है।

अगले Character को Read करने के लिए Program में फिर से getchar() Function को Use किया है, लेकिन इस बार ये Function Keyboard से Input प्राप्त नहीं करता है, बल्कि अपने Buffer में Stored String के ही अगले Character को Read करता है और उसकी ASCII Value को फिर से character नाम के Variable में Store कर देता है, जिसे फिर से putchar() Function Screen पर Display कर देता है।

इस Program के Output में हम देख सकते हैं कि हमने String के रूप में "**Manohar**" Input किया है, लेकिन getchar() व putchar() Function के Pair को केवल 6 बार लिखा है, इसलिए ये Program Keyboard से बहुत सारे Characters को Read करके getchar() Function के Buffer में Store तो कर देता है, लेकिन उस Buffer से केवल 6 Characters यानी "**Manoha**" को ही Output में Print कर पाता है।

इसी Program को अगले Chapter में थोडा सा Modify करके हम Input की जा रही String के कुल Characters की संख्या को ज्ञात करने का Program बनाएंगे। इस Program को अगले Chapter में इसलिए बनाएंगे, क्योंकि इस Program में हमें Looping Statements का प्रयोग करना होगा और Looping Statement को अगले अध्याय में समझाया गया है।

getch() Function

यह Function भी वैसा ही काम करता है जैसा कि getchar() करता है, लेकिन इस Function में Input किया गया अक्षर Input करते समय Screen पर दिखाई नहीं देता है। इस Function का प्रयोग हम उस समय भी कर सकते हैं, जब हम User से Password प्राप्त करना चाहते हैं, क्योंकि Password कभी भी Screen पर दिखाई देते हुए Input नही किया जाता है।

इस Function का उपयोग हम तब भी करते हैं, जब Program Execution के दौरान Output Screen को देखने के लिए रोक कर रखना होता है। हमारा परिणाम हमें तब तक दिखाई देता रहता है जब तक कि हम कोई Key Press नहीं करते। ऊपर के Programs में हमने इस Function को Use किया है। किसी भी प्रोग्राम में से इस Function को हटा कर Output प्राप्त करें और इस Function को Use करके Output देखें, दोनों में अन्तर स्वयं ही पता चल जाएगा।

Turbo C के IDE में ^F9 एक ऐसा Key Combination है जिसके द्वारा प्रोग्राम Compile भी होता है और Execute भी। जब बिना getch() Function के हम प्रोग्राम रन करते है, तब प्रोग्राम रन होने के बाद हमें Alt + F5 Key Combination द्वारा Output को देखना होता है, जबकि यदि इस Function को Use किया है तो Program Execution के साथ ही हमें प्रोग्राम का परिणाम भी प्राप्त हो जाता है।

Formatted Input

scanf() व printf() Functions के साथ हम कुछ **Flags** का प्रयोग करके Formatted या मनचाहे प्रकार में मान Input कर सकते हैं व Output में मनचाहे रूप में परिणाम प्रिंट कर सकते हैं। ये काम Control Strings के मध्य कुछ Flags को Use करके किया जाता है और ये Flags सभी प्रकार के Control Strings चाहे वह Integer के लिए हो, Float के लिए हो या Char या string के लिए हो, सभी के साथ प्रयोग किया जा सकता है। इस प्रकार Control Strings के बीच Flags निम्न सुत्र द्वारा लगाए जाते हैं:

%Flage Type Specifier (Control String)
Flage = w.d

यहाँ **w** एक पूर्णांक संख्या है, जो Input किये जाने वाले अंकों की संख्या बताती कि कितने अंकों तक मान Input होकर उस Identifier में जाएगा जिसका Address दिया गया है और .d यह बताता है कि दशमलव के बाद कितने अंक Store होंगे। d एक अंक होता है। **Type Specifier,** Input किये जा रहे Data का प्रकार बताता है। इसे समझने के लिए निम्न प्रोग्राम देखें:

Program

```
#include<stdio.h>
main()
{
    int a, b, c;
    clrscr();

    printf("Enter Three Integers");
    scanf("%2d %4d  %3d", &a,  &b, &c);

    printf("\nThe Value of First Integer is %d " , a );
    printf("\nThe Value of Second Integer is %d" , b );
    printf("\nThe Value of Third Integer is %d" ,c );
    getch();
}
```

इस प्रोग्राम में हम प्रथम Identifier **a** में दो अंको तक की संख्या Store कर सकते हैं। b में चार अंको व Identifier c में तीन अंकों तक का Store कर सकते हैं। ये प्रोग्राम बना कर रन करें और निम्न संख्याएं Input करें:

12
3456
78

Output

```
The Value of First Integer is 12
The Value of Second Integer is 3456
The Value of Third Integer is 78
```

यह वही Output है, जो हमने Input किया था। लेकिन यदि Input किए जा रहे मानों के क्रम को निम्नानुसार कर दें, तो हमें Output गलत प्राप्त होता है:

1234
12
233

जब हम ये मान Input करने की कोशिश करते हैं, तो Program हमसे केवल दो ही Input लेता है, तीसरा मान 233 Accept करने से पहले ही प्रोग्राम निम्न परिणाम दे देता है:

The Value of First Integer is 12
The Value of Second Integer is 34
The Value of Third Integer is 12

हम देखते हैं कि तीसरा मान हमने Input नहीं किया था फिर भी हमें परिणाम में 12 प्राप्त हो रहा है। ऐसा क्यों हुआ?

हमने Value Input करने में Formatted Input का प्रयोग किया है, यानी %2d का प्रयोग करके प्रोग्राम कंट्रोल को बताया है कि प्रथम Identifier a में केवल दो अंकों तक का ही मान Store होगा और हमने इसमें चार अंकों की संख्या को Input कर दिया है। इसलिए "C" Compiler दो अंकों तक के मान को Variable a में Store कर देता है और शेष रहे 34 को दूसरे Variable b में Store कर देता है।

चूंकि हमने %4d Flag द्वारा "C" Compiler को बताया है कि दूसरे Variable में चार अंकों तक का मान Store होगा, लेकिन फिर भी इस Variable में केवल बचे हुए दो ही अंक Store हुए हैं। ऐसा इसलिए हुआ है, क्योंकि **scanf**() Function Space या Enter Key के मिलते ही Terminate हो जाता है, और 1234 के बाद हमने Enter Key को Press किया है। इसलिए बचा हुआ 34 दूसरे Variable को मिल गया।

दोनों Variables को मान मिल जाने के बाद दिया जाने वाला दूसरा मान तीसरे Variable को मिल जाता है, और Program हमसे Input करने के लिए तीसरा मान नहीं मांगता, बल्कि सीधे ही Output दे देता है। ऐसा इसलिए होता है, क्योंकि अब किसी भी अन्य Variable के लिए मान Accept होना बाकी नहीं रह जाता है।

इस तरह हमें उपरोक्त मान परिणाम के रूप में प्राप्त होते हैं। इस प्रक्रिया का प्रयोग करके हम यह निश्चित कर सकते हैं कि किस Variable में किस संख्या तक मान को Input किया जा सकता है।

इसी प्रोग्राम में हम एक और Concept Use कर सकते हैं। जब किसी Control String के साथ * का प्रयोग किया जाता है, तब Compiler उस Control String के लिए Keyboard से आने वाले

मान को Neglect कर देता है और अगले मान को दूसरे वेरियेबल में Store कर देता है। तीसरे Variable में Garbage Value आ जाती है। इसी प्रोग्राम को दूसरे तरीके से Use करके नया प्रोग्राम बनाते हैं। इस प्रोग्राम में हमने दूसरे वेरियेबल में Control String के साथ * का प्रयोग किया है। इस प्रोग्राम को रन करें और निम्न मान Input करें:

```
111
222
333
```

Program

```
#include<stdio.h>
main()
{
    int a, b, c;
    clrscr();

    printf("Enter Three Integers");
    scanf("%d %*d  %d", &a,  &b, &c);

    printf("\nThe Value of First Integer is %d " , a );
    printf("\nThe Value of Second Integer is %d" , b );
    printf("\nThe Value of Third Integer is %d" ,c );
    getch();
}
```

Output

```
Value of First Integer is 111
Value of Second Integer is 333
Value of Third Integer is 2344
```

ध्यान दें कि हमने दूसरे Variable का मान 222 दिया था व 333 तीसरे Variable को दिया था लेकिन दूसरे Variable का मान 333 हो गया व तीसरे Variable में वह मान प्राप्त हुआ जो हमने कभी दिया ही नहीं है। ऐसा दूसरे Variable के Control String के साथ * का प्रयोग करने के कारण हुआ है।

जब किसी Control String के साथ * Flage का प्रयोग किया जाता है, तब "C" Compiler उस Variable के लिए Keyboard से Input होने वाले मान को Neglect कर देता है और जो अगला मान, अगले Variable के लिए Input किया जाता है वह मान पिछले Variable को Assign हो

जाता है। इस प्रोग्राम में तीसरे Variable के लिए कोई मान नहीं बचा इसलिए Compiler ने Memory की Garbage Value को इसमें Store करके दिखा दिया।

Program

```
#include<stdio.h>
main()
{
    int a, b, c, x, y;
    int p, q, r;
    clrscr();
    printf("Enter three integer number \n");
    scanf("%d %*d %d", &a, &b, &c);
    printf("%d %d %d", a, b, c);
    printf("\n\n Enter two 4-digit number \n");
    scanf("%2d %d    ",&x, &y);
    printf("\n %d %d", x, y);
    printf(" Enter two integers\n");
    scanf("%d %d", &a, &x);
    printf("%d %d \n\n", a, x);
    printf("Enter a nine digit number \n");
    scanf("%3d %4d %3d  ",&p, &q, &r);
    printf("%d %d %d" ,p, q, r);
    printf("Enter two three digit number \n");
    scanf("%d %d", &x, &y);
    printf("%d %d" ,x ,y);

    getch();
}
```

Output

```
Enter three integer number
1 2 3
1 3 –3577
Enter two 4-digit number
6789 4321
67 89
Enter two integers
44 66
4321 44
```

```
Enter a nine digit number
123456789
66 1234 567
Enter two three digit number
123 456
89 123
```

Program

```
#include<stdio.h>
main()
{
    int a, b, c, tot;

    printf("\nEnter 5-digit first number:");
    scanf("%2d ",&a);
    fflush(stdin);

    printf("\n Enter 4-digit second number:");
    scanf("%3d",&b);
    fflush(stdin);

    printf("\n Enter 2-digit third number:");
    scanf("%4d",&c);
    fflush(stdin);

    tot=a + b + c;

    printf("\n First value is % d" ,a);
    printf("\n Second value is % d" ,b);
    printf("\n Third value is % d" ,c);
    printf("\n Total of three values is %d" ,tot);
    getch();
}
```

Output

```
Enter 5-digit first number:12345
Enter 4-digit second number:5678
Enter 2-digit third number:23
```

```
First value is 12
Second value is 568
Third value is 23
Total of three values is 603
```

इस प्रोग्राम में हमने एक नए Function fflush(stdin) का प्रयोग किया है। ये Function Keyboard से Input किए जाने वाले मान से बचे मान को, जो कि Memory में रह जाता है, साफ करने का काम करता है। यानी जैसे हमनें printf("\nEnter 5-digit first number:"); statement द्वारा मान मांगा और scanf("%2d ",&a); statement द्वारा Keyboard से प्राप्त मान को Variable a को दिया, तो Variable a को केवल आगे के दो अंक ही प्राप्त होंगे। शेष अंक Memory में पडे रहेंगे। fflush(stdin) Function Memory में पडे इन अंकों को साफ कर देता है।

Formatted Output

जिस तरह विशेष Format में Data Input किया जा सकता है उसी तरह हम Control Strings के साथ कुछ **flags** का प्रयोग करके Output को भी विशेष Format में प्राप्त कर सकते हैं। इन विशेष Flags को **Escape Sequence Characters** कहते हैं।

Working With Integer Numbers

इसका Syntax निम्नानुसार होता है–

Syntax:- **% w.p Type Specifier**

W	यह बताता है कि कुल कितने Columns में Output Print होगा।
P	यह बताता है कि दशमलव के बाद कितने अंकों तक मान प्राप्त होगा।
Type Specifier	यह बताता है कि किस Data Type का Data Input किया जा रहा है। Integer, Float, Character या String प्रकार का।

जब हम Integers के साथ काम करते हैं, तब **W** वह न्यूनतम **Width** बताता है, जितने में Output प्राप्त होना है। जैसे निम्न उदाहरण देखिये:

माना a = 12345 है तो निम्न Statements निम्न Output Print करेंगे–

printf("%d", a);

यह Output में 12345 पांचो अंक print करेगा।

1	2	3	4	5

printf("%3d", a);

यह मान को उसी प्रकार प्रिंट करेगा जिस तरह ऊपर प्रिंट हुआ है।

1	2	3	4	5

printf("%10d", a);

यह Screen पर 10 Column Reserve करेगा और Right Side से पांचो अंक प्रिंट करेगा क्योंकि मान हमेंशा Right Side से ही Screen पर प्रिंट होता है। इसे निम्न चित्रानुसार समझा जा सकता है:

					1	2	3	4	5

printf("%010d", a);

इस Statement से Compiler screen की दस column Reserve करेगा और Right Side से मान प्रिंट करेगा लेकिन Left Side के जो पांच स्थान खाली बचते हैं उसमें पांच 0 भर देगा। देखें निम्न चित्र:

0	0	0	0	0	1	2	3	4	5

printf("%-10d", a);

यह Screen पर 10 columns Reserve करेगा लेकिन संख्याएं Left Side से ही Print करेगा क्योंकि Flage के रूप में Minus भी लिया गया है। इस Statement का Output निम्नानुसार प्रिंट होगा:

1	2	3	4	5					

Program

```
/*   Integer Formatted Output        */
#include<stdio.h>

main()
{
    int a;
```

```
    a = 12345;
    clrscr();

    printf("\n A is %d", a);
    printf("\n A is %3d",a);
    printf("\n A is %10d",a);
    printf("\n A is %-10d",a);
    printf("\n A is %010d",a);
    getch();
}
```

Output

```
A is 12345
A is 12345
A is     12345
A is 12345
A is 0000012345
```

Working With Real Numbers

जब Float प्रकार की संख्याओं को विशेष Formatting में Screen पर प्रिंट करना होता है, तब हमें निम्न Syntax को printf() Function के Control String के साथ प्रयोग करना होता है:

For Real Numbers	**%w.p f**	
For Exponent Numbers	**%w.p e**	यहां

W इसमें वह संख्या दी जाती है जितने अंकों तक का मान हमें screen पर print करवाना है। जैसे हमें कुल 10 अंकों में Value को print करना हो तो % के बाद 10 लिखा जाता है।

P इसमें यह बताया जाता है कि दशमलव के बाद कितने अंकों तक के मान को Screen पर print करना है। जैसे दसमलव के बाद 4 अंकों तक के मान को Screen पर Print करवाना हो व कुल दस अंकों में मान Print करवाना हो तो Control String के रूप में हमें %10.4 लिखना होगा।

F यह Compiler को बताता है कि Print किया जाने वाला मान Float प्रकार के Data Type का मान है।

E यह बताता है कि print होने वाला मान घातांक रूप में प्रिंट होगा।

Float प्रकार के Data Type का Variable जब Screen पर Print करते हैं तो Print होने वाला मान हमेंशा दशमलव के बाद 6 अंकों तक के मान को Print करता है। साथ ही Print होने वाला हर

मान यहां भी Right Justified रूप में ही Print होता है। यदि हमें Left Side से Value को Print करना हो तो (**Minus**) – Flag का प्रयोग करना पडता है। मानलो **x =12345.6789** है, तो निम्न अलग–अलग Statements निम्न Format में Output Print करेंगेः

printf("%10.4f", x);

ये Statement कुल 10 अंको का मान Output में Print करेगा और दशमलव के बाद के चार अंक Print करेगा।

1	2	3	4	5	.	6	7	8	9

printf("%10.2f", x);

यहां दशमलव के बाद कुल दो अंक प्रिंट होंगे और Output में कुल दस अंक Print हो सकेंगे।

1	2	3	4	5	.	6	7		

जैसा कि चित्र में दिखाया गया है कुल Space तो 10 Reserve होंगे लेकिन Output में केवल आठ ही अंक Print होंगे क्योंकि दशमलव के बाद केवल दो अंक ही प्रिंट होंगे। इसलिए आगे के दो अंको की जगह खाली ही रहेगी।

printf("%-10.2f", x);

यहां पर कुल दस Space Reserve होंगे और दशमलव के बाद दो अंकों तक संख्या Print होगी शेष जगह खाली रहेगी, लेकिन यहां हमने – चिन्ह प्रयोग किया है इसलिए संख्या Left Justified Print होगी। देखें निम्न चित्र

1	2	3	4	5	.	6	7		

printf("%f", x);

यह Statement x की पूरी Value को print करेगा। चूंकि Float Variable दशमलव के बाद 6 अंकों तक के मान को Screen पर Print करता है और यहां पर दशमलव के बाद केवल 4 ही अंक हैं इसलिए बाकी के अंक Garbage Value के प्राप्त हो जाते हैं।

1	2	3	4	5	.	6	7	8	9

printf("%*.*f", w, p, x);

यह Statement बिल्कुल उसी प्रकार है जिस प्रकार अन्य Statements हैं। फर्क बस इतना ही है कि इस Statement में width व precision को Control String के साथ ना लिख कर Variable के साथ लिखा जाता है। इसमें यदि w की जगह 10 व p की जगह 2 लिख दिया जाए तो Output निम्नानुसार प्राप्त होगा जो कि %10.2f के समान ही है।

		1	2	3	4	5	.	6	7

जिस तरह Float प्रकार के Variables को विभिन्न Format में प्रिंट करते हैं, उसी तरह हम घातांक मानों को भी विभिन्न प्रकार से प्रिंट कर सकते हैं। जब हमें संख्या को घातांक रूप में प्रिंट करना होता है तब मात्र f के स्थान पर e का प्रयोग करते हैं और बाकी की सारी Formatting समान रखी जाती है। निम्नानुसार कुछ उदाहरण बताए जा रहे हैं लेकिन इन्हीं उदाहरणों में मामूली से बदलाव करके विभिन्न Format प्राप्त किये जा सकते हैं:

printf("%e", x);

यह Statement निम्न Output देगा, क्योंकि सारा मान घातांक रूप में बदल जाएगा।

1	.	2	3	4	5	6	8	e	+	0	4

printf("%10.2e", x);

यह Statement दस Space Reserve करेगा और दशमलव के बाद दो अंक दर्शाएगा।

				1	.	2	3	e	+	0	4

printf("%-10.2e", x);

यह Statement दस Space Reserve करेगा और दशमलव के बाद दो अंक Print करेगा लेकिन Print होने वाला मान Left Justified Print होगा क्योंकि Control String के साथ – Flage का प्रयोग किया गया है।

1	.	2	3	e	+	0	4				

printf("%10.2e", -x);

यह Statement दस Space reserve करेगा और दशमलव के बाद के दो अंकों को Print करेगा साथ ही संख्या को ऋणात्मक Format में प्रिंट करेगा क्योंकि Variable के सामने Minus का चिन्ह प्रयोग किया गया है।

			-	1	.	2	3	e	+	0	4

इन सभी उदाहरणों को निम्नानुसार एक ही प्रोग्राम द्वारा समझा जा सकता है। ये प्रोग्राम बना कर रन कीजिये और देखिये कि किस प्रकार का Formatted Output प्राप्त होता है:

Program

```
/*   Formatted Output with Float and Exponential Values      */
#include<stdio.h>
main()
{
    float x = 12345.6789;
    clrscr();

    printf("\n X is %f", x);
    printf("\n X is %e", x);
    printf("\n X is %10.2f", x);
    printf("\n X is %10.2e", x);
    printf("\n X is %-10.2f", x);
    printf("\n X is %-10.2e", x);
    printf("\n X is %*.*f", 10, 2, x);
    printf("\n X is %e", 10, 2, x);
    printf("\n X is %f", -x);
    printf("\n X is %e", -x);
    getch();
}
```

Output

```
X is 12345.678989
X is 1.234568e+04
X is     12345.67
X is       1.23e+04
X is 12345.67
X is 1.23e+04
X is     12345.67
X is       1.23e+04
X is -12345.678989
X is -1.234568e+04
```

Working With Characters

जब हमें एक Single Character पर प्रक्रिया करके विशेष Formatting प्राप्त करनी होती है, तब हम निम्न Syntax द्वारा Character Print करते हैं।

```
%wc
```

यहां **w** Column की संख्या है और **c** Character प्रकार के Data Type को Use करने का Control String है। इसे समझने के लिए हम निम्न Format में एक Character को प्रिंट करने का प्रोग्राम बनाते हैं:

```
K
  K
    K
  K
K
```

इस प्रोग्राम में प्रथम Character Column संख्या एक पर स्थित है। दूसरा Character अगली पंक्ति में Column संख्या 2 पर स्थित है, तीसरा Character अगली पंक्ति में तीसरे Column पर है व चौथा Character अगली पंक्ति में चौथे Column पर है। फिर यह Column संख्या हर पंक्ति में उसी क्रम में कम होती जा रही है जिस क्रम में बढी थी। इस Format को हम निम्न प्रोग्राम द्वारा प्राप्त कर सकते हैं:

Program

```
/*   Using Of Character  Formatting  */

#include<stdio.h>

main()
{
    char c= 'K';
    clrscr();

    printf("\n %c", c);
    printf("\n %2c", c);
    printf("\n %3c", c);
    printf("\n %4c", c);
    printf("\n %3c", c);
    printf("\n %2c", c);
```

```
        printf("\n %c", c);
        getch();
    }
```

Working With Strings

जब हम String को विभिन्न प्रकार के Format में Display करना चाहते हैं, तब हमें String को Display करने के लिए Control String को निम्न Format में Use करना होता है:

%w.p s

यहां

w कुल Print होने वाले Characters की संख्या बताता है।
p String की शुरूआत के कुल Printable Characters की संख्या बताता है।

किसी भी String को जब Print किया जाता है, तब Print होने वाले सारे Characters **Right Justified** Format में Print होते हैं। जब String को Left से Print करना होता है, तब Width Flag से पहले Prefix के रूप में हमेंशा की तरह Minus का चिन्ह प्रयोग करना पडता है।

निम्न उदाहरण द्वारा हम एक ही String "MADHUSUDAN" को अपनी आवश्यकतानुसार विभिन्न रूपों में Print करवा सकते हैं।

printf("%s", x);

यह Statement पूरा का पूरा नाम ज्यों का त्यों निम्नानुसार Left Justified Format में Output में Print कर देता है:

M	A	D	H	U	S	U	D	A	N										

printf("%20s", x);

यह Statement 20 Columns Reserve करता है और String को निम्नानुसार Right Justified Format में Output में Print कर देता है:

										M	A	D	H	U	S	U	D	A	N

printf("%20.8s", x);

यह Statement 20 Columns Reserve करता है और String के आगे के दस Characters को Output में Screen Right Justified Format में निम्नानुसार Print कर देता है:

												M	A	D	H	U	S	U	D

printf("%.5s", x);

यह Statement किसी भी प्रकार का कोई Column Reserve नहीं करता है। इसलिए परिणाम Left Side से Screen पर Print होता है लेकिन दसमलव के बाद 5 लिखा है इसलिए String के आगे के केवल पांच Characters को ही Screen पर Output के रूप में निम्नानुसार Print करता है:

M	A	D	H	U

printf("%-20.8s",x);

यह Statement 20 Column Reserve करता है व दशमलव के बाद 10 लिखा है, इसलिए ये String के शुरूआत के दस Characters को ही Print करता है। Control String के साथ Minus चिन्ह का प्रयोग किया गया है, इसलिए Print होने वाला Format Left Side से Print होता है।

M	A	D	H	U	S	U	D	A	N										

Program

```c
#include <stdio.h>
main()
{
    char x[20] = "MADHUSUDAN";
    clrscr();
    printf("\n %s", x);
    printf("\n %20s", x);
    printf("\n %20.8s", x);
    printf("\n %.5s", x);
    printf("\n %-20.8s", x);
}
```

Output

```
MADHUSUDAN
            MADHUSUDAN
              MADHUSUD
MADHU
   MADHUSUD
```

ध्यान दें कि % के साथ # का चिन्ह लगा देने से Hexadecimal संख्याओं के पहले 0x व Octal संख्याओं के पहले 0 (Zero) लिखा जाता है और सभी Floating Point अंकों के साथ दशमलव संख्या आ जाती है चाहे संख्या पूर्णांक ही क्यों ना हो।

Exercise:

1 Is there any error in the following statements?

a. long = 323.32 / 232.4

b. father_name = 'Mohan Lal"

c. char zero = '0';

d. "Hello World" = 3.14 * r * r;

e. 23 * r = r * 22;

f. x = (a * y) (ab / 20) / 3+p.y;

g. int date[] = "10-May-2009";

h. simple interest = time * principle * rate / 100;

i. area = 22/7 * (r **2);

j. cube_volume+circle_volume = length * width * depth + 22/7 * r ^ h;

k. double x = y = z = 10;

l. x = y = z = 190 = 32;

m. increment = decrement-- + ++increment;

2 किसी Student द्वारा पांच विषयों में प्राप्त किए गए Marks को Input करो। इस Input के आधार पर उस Student की Mark – Sheet Develop करो, जिसमें Student का नाम, Address, पांचों Subjects के Marks की Total, Percent व Grade को Output में Screen पर Display किया गया हो। हम यहां पर ये मान रहे हैं कि Student के हर Subjects की Marking 100 में से की जा रही है।

3 String किसे कहते हैं ? String को Store करने के लिए हमें One-Dimensional Array का प्रयोग क्यों करना पडता है ?

4 क्या हम **gets**() Function के स्थान पर **scanf**() Function को Use करके एक Line की String को Keyboard से प्राप्त कर सकते हैं ? यदि हां तो बताईए हम ऐसा कैसे कर सकते हैं?

5 **gets**() व **scanf**() Function तथा **puts**() व **printf**() Function के बीच के अन्तर को एक Example Program द्वारा विस्तार से समझाईए।

6 **getc**(), **getch**() व **getchar**() Functions तथा **putchar**() व **putch**() Function की कार्यप्रणाली को एक Program द्वारा समझाईए।

7 **fflush(stdin);** किसी Program में इस Statement को कब व क्यों Use किया जाता है ? एक उदाहरण Program द्वारा इस Function की उपयोगिता को साबित कीजिए।

8 एक Program बनाईए जो User से Input के रूप में **HH:MM:SS** Format में एक Time लेता है आर उस Time में **1** घण्टा **2** मिनट **3** सेकण्ड जोडकर बनने वाले नए Time को फिर से **HH:MM:SS** Format में ही Monitor पर Display कर देता है।

9 सरल ब्याज ज्ञात करने का एक Program बनाईए, जो Input के रूप में User से Principal, Rate, व Time लेता है, तथा Output के रूप में Interest Calculate करके Display करता है। Output में Display होने वाले मान Principal, Rate व Interest के मान में दसमलव के बाद केवल दो संख्याएं ही Display होनी चाहिए और Display होने वाला Amount Right Justified Format में Display होना चाहिए।

DECISION MAKING AND LOOPING STATEMENTS

Control Statement and Looping

हम हमारे वास्तविक जीवन में भी हमेंशा निर्णय लेते रहते हैं। जैसे कि यदि हमें बाजार जाना है तोः

- किस दिन बाजार जाया जाएगा?
- किस समय बाजार जाया जाएगा?
- किस काम के सम्बंध में बाजार जाया जाएगा?
- यदि वह काम पूरा नहीं होता तो फिर कौनसा दूसरा काम बाजार में पूरा किया जाएगा?

इस तरह हर समय हमारा दिमाग अपनी जरूरतों के अनुसार निर्णय लेता रहता है।

जिस तरह हम हमारे दैनिक जीवन में विभिन्न निर्णय लेते रहते हैं, उसी तरह कम्प्यूटर पर प्रोग्राम बनाते समय भी हमें विभिन्न प्रकार के निर्णय लेने होते हैं, कि किस Statement के बाद कौनसा Statement Execute होगा।

उदाहरण के लिए यदि जिस काम के लिए Statement लिखा गया है, वह काम नहीं होता है तो फिर कौनसा Statement Execute होगा और यदि वह Statement Execute हो जाता है तो फिर कौनसा Statement Execute होगा? आदि।

Program Control

हम जानते हैं कि कोई भी प्रोग्राम Statements का एक समूह होता है, जिन्हें सामान्यतया जिस क्रम में Source File में लिखा जाता है, वे उसी क्रम में Execute होते हैं। किसी प्रोग्राम में लिखे गए Statements का Execution, जिस क्रम में होता है, उस क्रम को **Flow Of Control** कहा जाता है।

कभी–कभी हमारे सामने इस तरह की स्थितियां होती हैं, जिनके कारण हमें Statements के क्रम को Conditions के आधार पर बदलना पडता है और Condition के सत्य नहीं होने तक किन्हीं Statement के समूह को बार–बार दोहराना पडता है।

''सी'' भाषा में इस प्रकार के कुछ Statement उपलब्ध हैं, जिनका प्रयोग करके हम Program के Control को अपनी सुविधा अनुसार उस स्थान पर भेज सकते हैं, जहां भेजना चाहते हैं। इस प्रकार के Statements में हमेंशा एक Condition होती है, जिसके आधार पर ये तय किया जाता है, कि Program के Control को किस Statement पर भेजना है। ये विशेष प्रकार के Statements **Control Statements** कहलाते हैं।

Types Of Control Statement

Control Statements को हम मुख्यतया तीन भागों में बांट सकते हैं, जो निम्नानुसार हैंः

Sequential Statements

जिन Statements का Execution होने के बाद क्रम से अगली पंक्ति में लिखे Statements का Execution होता है, **Sequential Statement** कहलाते हैं। अभी तक हमने जितने भी Programs बनाए हैं, उन सभी में केवल Sequential Statements का ही प्रयोग किया है।

जब कोई Program क्रम से लिखे गए Statements का Execution उसी क्रम में करता है, तो इसे Control का **Normal Flow** कहा जाता है, क्योंकि इस प्रकार के Execution में Program Control का Flow एक क्रम में चलता रहता है और Program Control को किसी अन्य Point पर जाने की जरूरत नहीं पडती है। Sequential Statements हर main() function में होते हैं और किसी भी प्रोग्राम में ये जरूर होते हैं।

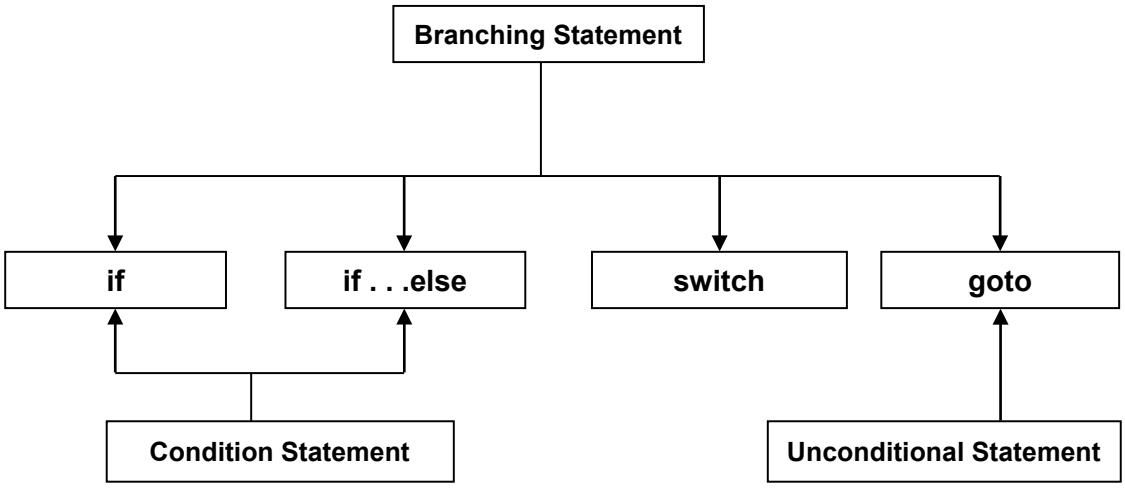
Conditional Statements

प्रोग्राम में कई जगह पर किसी Condition के आधार पर Control के सामान्य प्रवाह को छोड कर किसी भिन्न Point से Statement का Execution करना पडता है। इस प्रकार के चयनात्मक Execution के लिए प्रयुक्त Statements को **Conditional Statements** या **Branching Statements** कहा जाता है।

यानी जब किसी समस्या के किसी शर्त के अनुसार दो या दो से अधिक परिणाम संभावित होते हैं, तब किसी निश्चित परिणाम पर पहुंचने के लिए प्रोग्राम को अपना Normal Flow छोड कर किसी भिन्न बिंदु से Program को Execute करना पडता है।

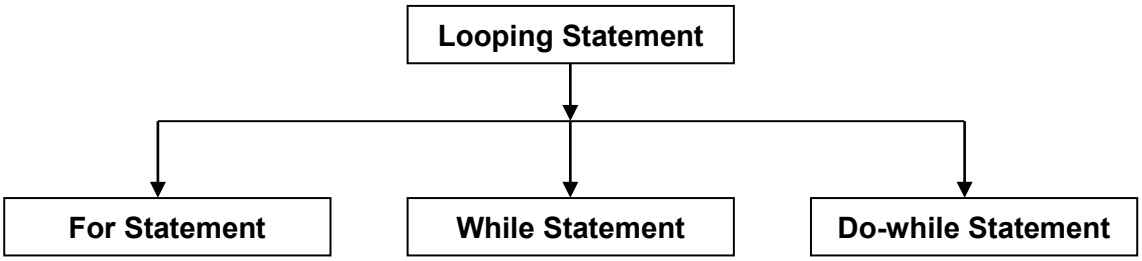
इस प्रक्रिया को प्रोग्राम की **Branching** होना कहते है और **goto Statement** बिना Condition का एक ऐसा Control Statement है, जिसे हम **Looping** व **Branching** दोनों रूपों में प्रयोग कर सकते हैं।

"C" Language में मुख्यतया चार **Conditional Control Statements** हैं, जिन्हें निम्न चित्र द्वारा दर्शाया गया हैः



terative Statements

प्रोग्राम में कुछ Statements के क्रम किसी Condition पर निर्भर करते हुए जब बार–बार दोहराने की आवश्यकता होती है, तो पुनरावर्ती या **Iterative Statements** का प्रयोग किया जाता है। "C" में मुख्यतः तीन Iterative Statements होते हैं, जिसे निम्न चित्र द्वारा दर्शाया गया है:



Compound Statement or Statement Block

जब Statement के एक समूह को इकाई के रूप में उपयोग में लेना होता है, तो उसे **Block** या **Compound Statement** कहते हैं। एक Block में ढेर सारे Statements होते हैं, जो किसी खास Condition के सत्य होने पर या फिर Program Control के Normal Flow में, एक साथ Execute होते हैं। Block के Statements मंझले कोष्ठक (**Curly Braces**) के बीच में लिखे जाते हैं। एक Block के अन्दर कई Definitions, Declarations व Statements का समूह हो सकता है। जैसे:

```
{
    int a, b;
    c = a + b;
    … … … …
}
```

if statement

सभी Control Statements में से if Statement सबसे शक्तिशाली Statement है, जिसके द्वारा हम Statements के Execution के Flow को Control कर सकते हैं। यह एक द्विमार्गी (Two Way) Statement है, जिसमें Condition के सत्य या असत्य होने के आधार पर निर्भर करते हुए प्रोग्राम का Control दो अलग–अलग बिंदुओं पर पहुंचता है।

इस तरह If Statement के अनुसार प्रोग्राम के पास दो रास्ते होते हैं, एक Condition के सत्य होने की स्थिति वाला रास्ता व दूसरा Condition के असत्य होने की स्थिति वाला रास्ता। जब हमें Condition के सत्य होने पर केवल एक Statement का Execution करना होता है, तब हम निम्न प्रकार के Syntax में if Condition को Use कर सकते हैं, जिसमें जब if Condition सत्य होती है तब **Statement 1** Execute हो जाता हैः

```
if ( Expression and Condition )
    Statement 1;
```

जब हमें किसी Condition के सत्य होने पर एक से अधिक Statements का Execution करना होता है, तब हमें सभी Statements मंझले कोष्ठक में लिखने पडते हैं। यदि हम ऐसा नहीं करते व ऊपर बताए Syntax के अनुसार ही प्रोग्राम लिख देते हैं, तो Condition के सत्य होने पर "C" Compiler पहले Statement का तो Execution कर देता है, लेकिन शेष Statements को Program की Condition के अनुसार Execute नहीं करता बल्कि उन्हें Program के Normal Flow के अनुसार Execute करता है।

इस स्थिति में पहले Statement को छोडकर बाकी के सभी Statements हमेंशा Execute होते हैं, चाहे if Condition सत्य हो चाहे असत्य। इसलिए यदि if Condition के प्रोग्राम में एक से अधिक Statements का Execution करना हो तो हमें निम्न if Syntax Use करना पडता हैः

```
if ( Expression and Condition )
{
    Statement 1;
    Statement 2;
    "   "   "
```

```
                    "    "    "
                    Statement n;
            }
        Other Statements
```

इसमें जब Condition सत्य होती है, तो Block के अन्दर लिखे सारे Statements का Execution हो जाता है, उसके बाद **Other Statements** का यानी Block से बाहर के Statements का Execution होता है। लेकिन यदि Condition असत्य होती है, तो प्रोग्राम Control, if Condition के Block को छोड कर सीधे ही Other Statements यानी Block के बाहर के Statements का Execution कर देता है। Condition सत्य हो या असत्य फिर भी **Other Statements** यानी Block से बाहर के Statement का Execution होता ही है, क्योंकि ये Statement **Sequential Flow** में होते है।

Example :

यदि कोई Employee 25 दिन से अधिक काम करता है, तो उसे उसकी payment का 10% Bonus प्राप्त होगा। Employee को प्राप्त होने वाला कुल पैसा ज्ञात करो। इस प्रोग्राम में निम्न Variables Define करने होंगे:

1. Employee की Payment का मान Float में लेने के लिए एक **float** Integer प्रकार का Variable माना **payment**
2. प्राप्त Bonus को Store करने के लिए एक **float** प्रकार का Variable माना **bonus**
3. कुल कितने दिन काम किया इसे Store करने के लिए एक **int** प्रकार का Variable माना **days**

Algorithm

1. *payment* व *days* Read करेंगे।
2. यदि काम 25 दिन से अधिक किया है, तो 10 प्रतिशत bonus दिया जाएगा और bonus को payment में जोड कर कुल payment Output में Print किया जाएगा।
3. यदि काम 25 से कम किया है, तो केवल payment को Output में print किया जाएगा।
4. समाप्त।

Program

```
/*   If Condition Example        */
#include<stdio.h>
main()
{
  float payment=0,bonus=0;
```

```
    int days;
    clrscr();

    printf("\n Enter Payment of Employee");
    scanf("%f",& payment);
    printf("\n Total Working Days");
    scanf("%d",& days);

    if(days>=25)
    {
      bonus = payment * 10/100;
      payment = payment +bonus;
    }

    printf("\n Total Payment Of The Employee is %6.2f \t", payment);
    printf("\n Bonus Gained By Employee is %6.2f \t", bonus);

    getch();
  }
```

इस प्रोग्राम में जब Input किये गए दिन 25 से ज्यादा होते हैं, तो if Condition सत्य हो जाती है। तब if Statement Block का Execution होता है और bonus की गणना होती है तथा bonus payment में जुड जाता है। जब if Condition Block से Program Control बाहर आता है, तब Total Payment व bonus Print हो जाते हैं।

यदि Input किये दिन 25 से कम होते है, तो if Condition असत्य हो जाती है व प्रोग्राम Control, if Block में नहीं जाता, बल्कि सीधे ही Input की गई payment व bonus को Print कर देता है, जिसमें bonus शुन्य होता है। इस प्रोग्राम में Control String के साथ **Flags** का प्रयोग किया गया है, जिससे Output में प्राप्त होने वाला परिणाम दशमलव के बाद दो अंकों तक का ही मान प्रदान करता है। हम if Statement में एक साथ दो Condition भी दे सकते हैं।

इसे समझने के लिए हम उपरोक्त प्रोग्राम में ही थोडा सा बदलाव कर रहे हैं। जिसमें किसी Employee को Bonus तब प्राप्त होना चाहिये,, जब उसकी Payment 3000 से अधिक हो व Employee कम से कम 25 दिन काम करता हो।

इस प्रश्न में दो शर्तें हो गई हैं। पहली ये, कि उसी Employee को 10% Bonus प्राप्त होगा, जिसकी Payment कम से कम 3000 हो और दूसरी ये कि Employee को कम से कम 25 दिन काम करना होगा।

इस समस्या को हल करने के लिए हम if Conditional कोष्ठक में Logical Operator **AND** का प्रयोग कर सकते हैं, क्योंकि यह Operator तभी प्रोग्राम Control को दूसरे बिन्दु पर भेजता है, जब दी गई दोनों शर्तें सत्य हों। अब if Condition तभी सत्य होगी जब Employee की Payment 3000 या 3000 से अधिक हो और Employee 25 या 25 से अधिक दिन काम करे।

Logical Operator के सम्बंध में हमने बताया था कि AND Operator का प्रयोग करने पर यदि Input की गई दोनों Condition सत्य हों तो प्रोग्राम 1 **Return** करता है अन्यथा प्रोग्राम 0 **Return** करता है। 1 **Return** करने का मतलब है कि Condition सत्य है और पहले वह काम होगा जिस काम के लिए Conditional Operator का प्रयोग किया गया है, और 0 **Return** होने का मतलब है कि जिस काम के लिए शर्त रखी गई है, वह काम नहीं होगा बल्कि उसके आगे के Statements का Execution होगा। इस Modified प्रोग्राम को आगे दिखाया गया है:

Program

```
/*   If Condition Example                    */
#include<stdio.h>
main()
{
   float payment=0,bonus=0;
   int days;
   clrscr();

   printf("\n Enter Payment of Employee");
   scanf("%f",& payment);
   printf("\n Total Working Days");
   scanf("%d",& days);

   if(days>=25 && payment >= 3000)
   {
      bonus = payment * 10/100;
      payment = payment +bonus;
   }

   printf("\n Total Payment Of The Employee is %6.2f \t", payment);
   printf("\n Bonus Gained By Employee is %6.2f \t", bonus);

   getch();
}
```

हम if Condition Block में **OR** व **NOT** Condition का भी अपनी आवश्यकता के अनुसार प्रयोग कर सकते हैं और अपनी जरूरत के अनुसार विभिन्न प्रकार के Expressions **if** के कोष्ठक में प्रयोग कर सकते हैं।

Exercise:

1. Control Statements से आप क्या समझते हैं तथा Flow of Control का क्या मतलब होता है ?
2. विभिन्न प्रकार के Control Statements का वर्णन कीजिए।
3. Compound Statements या Block Statements किसे कहते हैं?
4. if Statement का Structure बनाकर इसके काम करने के तरीके को समझाईए।
5. दो संख्याओं में से बडी संख्या ज्ञात करने का Program बनाओ।
6. Input की गई संख्या Even है या Odd, इस बात का पता लगाने वाला एक Program बनाओ।

if – else statement

जब हमें दो या दो से अधिक शर्तों के आधार पर कोई निर्णय लेना होता है, या प्रोग्राम से कोई खास काम करवाना होता है, तब हम if - else Statement का प्रयोग करते हैं। यह साधाराण if - else का विस्तृत रूप है। इसका Syntax नीचे दिखाया गया है:

```
if ( Expression and Condition )
{
        Statement 1;
        Statement 2;
        " " "
        Statement n;
}
else
{
        Statement 3;
        Statement 4;
        " " "
        Statement m;
}
Sequential Statement a;
```

इस Syntax के अनुसार जब **if** Condition सत्य होगी, तो Statement 1, Statement 2, से Statement n तक का Execution होगा और यदि if Condition असत्य होगी, तो प्रोग्राम Control, if Statement Block को छोड देगा और Default रूप से **else** Condition की Statements का Execution हो जाएगा।

इस तरह से Statement 3, Statement 4 से Statement m तक का Execution होगा। यदि if Condition सत्य होती है, तो else Block के Statements का Execution नहीं होता है। Input किये गए मान के आधार पर if या else Condition का Execution होने के बाद प्रोग्राम Control Sequential Statements का Execution करता है। Sequential Statements का तो Execution होता ही है, क्योंकि ये main() Function Block में लिखे गए हैं, और Sequential क्रम में हैं।

Example:

दो संख्याएं Input करके उनमें से बडी संख्या ज्ञात करने का एक प्रोग्राम नीचे दिया जा रहा है, जिसको **if-else** Condition द्वारा हल किया गया है।

Algorithm

1 संख्याएं Input करो।
2 यदि पहली संख्या बडी है तो उसे Print करो।
3 यदि पहली संख्या बडी नहीं है तो दूसरी संख्या Print करो।
4 समाप्त।

इस प्रोग्राम में Integer प्रकार की दो संख्याएं Input की गई हैं। माना हमने पहली संख्या 13 व दूसरी संख्या 12 Input की। यह मान Input करने पर पहली संख्या digit1, दूसरी संख्या digit2 से बडी होती है, इसलिए if Condition सत्य हो जाती है और Output में निम्न Message प्राप्त होता है:

```
Digit1 = 13  is Greater Than Digit2 = 12
Thanks for using This Program
```

लेकिन यदि यही मान हम उल्टे क्रम में दें, यानी पहले 12 फिर 13 Input करे तो if Condition असत्य हो जाती है। इसलिए else Condition के Statement का Execution हो जाता है और Output में हमें निम्न Message प्राप्त होता है–

```
Digit2 = 13  is Greater Than Digit1 = 12
Thanks for using This Program
```

ध्यान दें कि यहां पर हमने किसी प्रकार के Statement Block का प्रयोग, ना तो if Condition में किया है, ना ही **else** Condition में। ऐसा इसलिए, क्योंकि दोनों ही स्थिति में केवल एक ही Statement का Execution होना है। यदि हमें एक से अधिक Statements का Execution करना होता, तो हमें Block का प्रयोग करना जरूरी हो जाता।

Program

```
/*   If - else Condition Example        */
#include<stdio.h>

main()
{
    int digit1,digit2;
    clrscr();

    printf("\n Enter Value of First Digit");
    scanf("%d",& digit1);
```

```
    printf("\n Enter Value of Second Digit");
    scanf("%d",& digit2);

    if(digit1>digit2)
        printf("\n Digit1 = %d  is Greater Than Digit2 = %d",digit1, digit2);
    else
        printf("\n Digit2 = %d  is Greater Than Digit1 = %d",digit2,digit1);

    printf("\n Thanks for using This Program   \t");
    getch();
}
```

Exercise:

1. एक Program बनाओ तथा इसमें Input के रूप में एक संख्या Input करो और बताओ कि वह संख्या Leap Year है अथवा नहीं। Program का Algorithm भी बनाईए और Program के **Flow** को समझाईए।
2. Algorithm के आधार पर एक Program बनाओ जिसमें जब कोई User किसी सामान की **Purchasing** व **Selling** Price Input करे, तो Program User को ये बताए कि उसे कितने रूपये का Profit या Loss हुआ है।
3. एक तीन अंकों की संख्या Input कीजिए और पता कीजिए कि वह संख्या Palindrome है, अथवा नहीं। Program का Algorithm भी बनाईए और Program के Flow को विस्तार से समझाईए।

Nested if else statement

जब एक if Condition के Statement Block में एक और if Condition या if else Condition के Statement Block का प्रयोग किया जाता है, तो इसे if Condition की **Nesting** करना कहते हैं। हम विभिन्न if Conditions की,आवश्यकता के अनुसार Nesting कर सकते हैं, यानी हम एक if Condition के Statement Block में दूसरा if या if else Condition का Statement Block, दूसरे में तीसरा if या if else Condition का Statement Block, तीसरे में चौथा आदि कितनी भी संख्या में if Condition Statements Blocks की Nesting कर सकते हैं। इसका Syntax निम्नानुसार होता है:

```
if ( Expression and Condition 1 )
{
    if ( Expression and Condition 2 )
    {
        Statement 1;
        Statement 2;
        "    ""
        Statement l;
    }
    else
    {
        Statement 3;
        Statement 4;
        "    ""
        Statement m;
    }
    Inner Sequential Statement n;
}
else
{
    Statement 5;
    Statement 6;
    "    "    "
    Statement o;
}
Outer Sequential Statement a;
```

इस Syntax में यह बताया गया है, कि यदि प्रथम if Condition सत्य होती है, तो प्रोग्राम Control प्रथम if Condition के Statement Block में जाएगा। वहां प्रोग्राम Control को दूसरी if Condition मिलेगी। यदि ये दूसरी if Condition भी सत्य है, तो प्रोग्राम Control, Inner if Condition Statement Block में जाएगा और Statement 1, Statement2 से Statement n तक के Statements का Execution करेगा।

फिर Inner if Statement Block के बाहर आकर Outer if Condition Block के Inner Sequential Statement n का Execution करेगा और अंत में प्रोग्राम Control दोनों if Condition Statement Block से बाहर आकर Outer Sequential Statement a का Execution करेगा।

लेकिन यदि Inner if Condition सत्य ना हो, तो प्रोग्राम Control, Inner else Statement Block के Statement 3 से लेकर Statement m तक के Statements का Execution करेगा और Inner else से बाहर आकर Outer if Condition के Inner Statement, Statement n का Execution करेगा।

अगर दोनों ही if Conditions असत्य हो जाती है, तो प्रोग्राम Control सीधे ही Outer else Condition के Statement Block का Execution कर देता है और else Statement Block से बाहर आकर Outer Sequential Statement a का Execution करता है।

सारांश के रूप में हम यह कह सकते हैं कि जब If Condition के Statement Block का Execution होता है, तब else के Statement Block का Execution नहीं होता, और else Statement Block का Execution तभी होता है, जब if Condition असत्य हो जाती है। इसे अच्छी तरह से समझने के लिए हम निम्न उदाहरण देखते हैं:

Example:

तीन संख्याएं Input करके उनमें से सबसे बडी संख्या ज्ञात करने का एक प्रोग्राम Nested if Condition द्वारा हल कीजिये।

इस प्रोग्राम में हमें तीन संख्याएं Input करनी है, इसलिए हमने int प्रकार के तीन Variable लिए हैं। प्रोग्राम रन करके हम तीन Input क्रमशः 1, 2 व 3 देते हैं। अब देखते हैं कि प्रोग्राम किस प्रकार हमें Output देगा या Program का Execution Flow किस प्रकार होगा?

ये तीनों मान क्रम से *digit1* = 1, *digit2* = 2 व *digit3* = 3 को मिल जाएंगे। अब प्रथम if Condition Check होगी, जिसमें यदि digit1, digit 2 से बडा हो तो प्रोग्राम Control if Condition के Statement Block में जाएगा।

यहां digit1 का मान 1 है व ये digit2 से बडा नहीं है क्योंकि digit2 का मान 2 है, इसलिए Outer if Condition असत्य हो जाएगी और प्रोग्राम Control if Statement Block में नहीं जाएगा, बल्कि

Outer if Condition के else Statement Block का Execution होगा, क्योंकि जब if Condition असत्य हो जाती है, तब else Statement Block का Execution होता है।

जब प्रोग्राम Control बाहर के else Statement Block में प्रवेश करता है, तो वहां उसे एक और if Condition मिलती है। इस if Condition के कोष्ठक में प्रोग्राम Control check करता है, कि क्या digit3 का मान digit2 के मान से बडा है या नहीं।

digit3 का मान 3 है व digit2 का मान 2 है इसलिए यहां if Condition सत्य हो जाती है व प्रोग्राम Control else के अन्दर के if Condition के Statement Block का Execution कर देता है। इस प्रकार से digit3 सबसे बडा है यह Output में Print हो जाता है। प्रोग्राम निम्नानुसार है:

Program

```
/*   Use of Nested if  else3 Control Statement Example        */
#include<stdio.h>
main()
{
    int digit1, digit2, digit3;
    clrscr()

    printf("\n Enter Three Integers");
    scanf("%d %d %d", digit1, digit2, digit3);

    if(digit1 > digit2)
    {
        if(digit1 > digit3)
        {
            printf("\t Digit1 is Largest");
        }
        else
        {
            printf("\t digit3 is largest");
        }
    }
    else
    {
        if(digit3 > digit2)
        {
            printf("\t Digit3 is Largest");
        }
```

```
            else
            {
                printf("\t Digit2 is Largest");
            }
        }
        getch();
    }
```

यदि हम तीनो variables को क्रम से 3, 2 व 1 अंक Input कर दें तो प्रोग्राम द्वारा सबसे बडा मान इस प्रकार प्राप्त होगा।

digit1 = 3 digit2 = 2 digit3 = 1

सबसे पहले प्रथम if Condition check होगी, जिसमें ये check होगा कि digit1, digit2 से बडा है या नहीं। यहां Condition सत्य होती है और प्रोग्राम Control प्रथम if Condition के Statement Block में प्रवेश करता है। यहां वापस Inner if Condition मिलती है जहां check होता है, कि क्या digit1, digit3 से भी बडा है या नहीं। हमने digit1 का मान सबसे बडा रखा है, इसलिए यह Condition भी सत्य हो जाती है और प्रोग्राम Control inner if के Statement Block में प्रवेश करता है, और निम्न Statement का Execution कर देता है कि digit1 सबसे बडा है।

Digit1 is Largest

अब यदि हम तीनो variables को क्रम से 2, 3 व 1 अंक Input कर दें तो प्रोग्राम द्वारा सबसे बडा मान इस प्रकार प्राप्त होगा।

digit1 = 2 digit2 = 3 digit3 = 1

सर्वप्रथम यह check होता है कि digit1, digit2 से बडा है या नहीं। यहां digit1, digit2 से बडा नहीं है। इसलिए प्रोग्राम Control, if Statement Block में प्रवेश नहीं करता, क्योंकि प्रथम If Condition ही असत्य हो जाती है। इसलिए Outer if के else का Execution होता है। जैसे ही प्रोग्राम Control, else Statement Block में प्रवेश करता है, तो उसे एक और Inner if Condition मिलती है, जहां ये check होता है, कि क्या digit3, digit2 से बडा है या नहीं। हमने digit3 का मान digit2 से छोटा रखा है, इसलिए यह if Condition असत्य हो जाती है और प्रोग्राम Control इस if Condition को छोड कर Inner Else पर चला जाता है और वहां के Statement Block को Execute कर देता है और निम्न Message Print करता है:

Digit2 is Largest

अब यदि हम variables को निम्नानुसार मान प्रदान कर दें तो निम्न प्रकार से प्रोग्राम का Execution होगा।

digit1 = 2 digit2 = 1 digit3 = 3

जब प्रोग्राम का Execution होगा तब सबसे पहले if Condition में check किया जाएगा कि क्या digit1, digit2 से बडा है। यहां digit1, digit2 से बडा है इसलिए प्रोग्राम Control, if Condition के Statement Block में प्रवेश करेगा।

यहां फिर Check होगा कि क्या digit1, digit3 से भी बडा है या नहीं। यहां ditit1, digit3 से बडा नहीं है इसलिए inner If Condition असत्य हो जाती है और प्रोग्राम Control, inner else statement Block का Execution कर देता है और Output में हमें निम्न Output प्राप्त हो जाता है:

Digit3 is Largest

Exercise:

1 Keyboard से Input की गई एक तीन अंकों की संख्या को Reverse Order में Print करने का Program बनाइए।

2 Ternary Operator का प्रयोग करते हुए तीन संख्याओं में से बडी संख्या ज्ञात करने का Program बनाइए, साथ ही Program का Algorithm भी लिखिए।

3 एक Program बनाओ जिसमें Input के रूप में किसी Triangle के तीनों कोणों को Input किया जाता है। Input किए गए मानों के आधार पर ज्ञात कीजिए कि इन तीन कोणों के आधार पर बनने वाला Triangle एक Valid Triangle है अथवा नहीं। किसी Triangle के तीनों कोणों का योग हमेंशा **180** Degree होता है।

4 Keyboard से Input की गई तीन संख्याओं में से बडी संख्या ज्ञात करने का Program बनाते हुए Nested if . . . else Control Statement को विस्तार से समझाईए।

5 Keyboard से Input किए गए किसी भी Number के Absolute मान को Output में Screen पर Print करो।

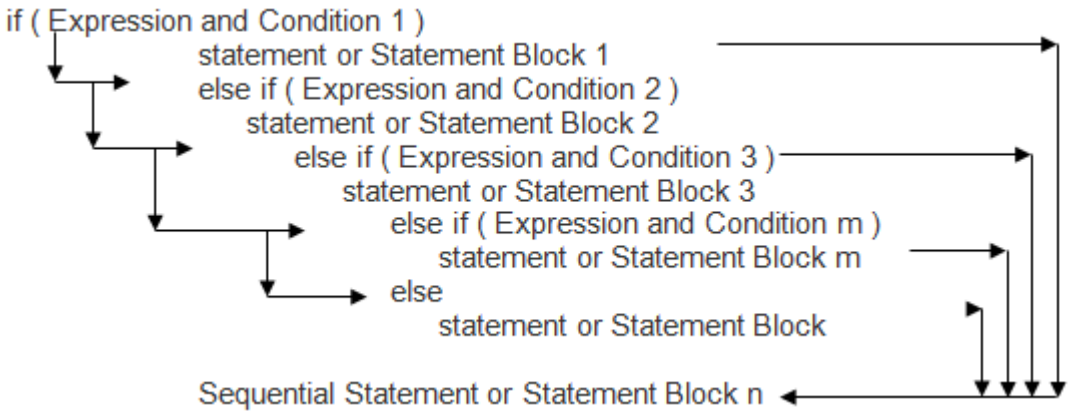
6 एक Program बनाओ जिसमें Input के रूप में किसी Triangle की तीनों भुजाओं के नाप को Input किया जाए। अब ये बताओ कि Input की गई भुजाओं के आधार पर बनने वाला Triangle समकोण होगा अथवा नहीं। जबकि किसी Triangle की दो भुजाओं के वर्ग का योग यदि तीसरी भुजा के वर्ग के बराबर हो, तो बनने वाला Triangle समकोण होता है।

7 एक द्विघातीय समीकरण Ax2 + Bx + C = 0 के वास्तविक मूल (Real Roots) ज्ञात करने के लिए एक Program लिखिए। जबकि b^2-4ac का मान यदि 0 से कम हो, तो प्राप्त होने वाले दोनों Roots Imaginary होते हैं। लेकिन यदि Roots Imaginary ना हों, तो दोनों Roots निम्नानुसार Formula द्वारा प्राप्त होते हैं:

Root1 = $-b + \sqrt{(b^2-4ac)} / 2a$ Root2 = $-b - \sqrt{(b^2-4ac)} / 2a$

if – else if – else Ladder statement

जब हमारे पास ऐसी समस्या होती है कि ढेर सारी Conditions में से कोई एक ही सही हो तब हम इस Control Statement का प्रयोग करते हैं। इसमें क्रम से else के साथ कई if Conditions दी गई होती हैं और प्रोग्राम Control इन सभी Conditions को क्रम से check करता है और जहां भी if Condition सत्य हो जाती है, प्रोग्राम Control उस if Condition के Statement Block का Execution कर देता है। शेष if Condition को प्रोग्राम Control check नहीं करता हैं। इसका Syntax निम्नानुसार है:



Example :

एक प्रोग्राम बनाओ जिसमें यदि Input किया गया अंक 79 से अधिक हो तो Output में **HOUNORS**, 59 से अधिक हो तो **FIRST DIVISION**, 49 से अधिक हो तो **SECOND DIVISION**, 39 से अधिक हो तो **THIRD DIVISION** व अन्य **FAIL** print हो।

इस प्रोग्राम में जब Marks Input कर देते हैं, माना हमने 50 Input किया तो प्रोग्राम Control सर्वप्रथम if Condition में check करता है कि Marks का मान 79 से अधिक है या नहीं। यदि Marks 79 से अधिक है तो प्रथम Condition सत्य हो जाती और Output में HOUNORS print होता लेकिन यहां Marks 50 है।

इसलिए अगली Condition check होती है कि Marks 59 से अधिक है या नहीं। यह Condition भी असत्य हो जाती है। अब दूसरी else if Condition check होती है। यहां पर Condition सत्य हो जाती है क्योंकि Marks का मान 49 से अधिक है और Output में SECOND DIVISION print हो जाता है।

Program

```
/*  Example of if – else if – else Condition Statement          */
    #include<stdio.h>
    main()
```

```
{
    int marks;
    printf("\n Enter Marks");
    scanf("%d", &marks);

    if(marks > 79)
        printf("\n HOUNORS");

    else if(marks > 59)
        printf("\n FIRST DIVISION");

    else if(marks > 49)
        printf("\n SECOND DIVISION");

    else if(marks > 39)
        printf("\n THIRD DIVISION");

    else
        printf("\n Fail");

    getch();
}
```

Example :

यदि unit 200 से कम या बराबर हो तो प्रति unit 50 पैसे के हिसाब से Charge किया जाएगा। यदि unit 200 से अधिक व 400 से कम हो तो 200 से उपर जितने भी unit हों उनका charge 65 पैसे प्रति Unit लिया जाएगा और साथ ही 100 रू extra लिया जाएगा। यदि unit 400 से अधिक व 600 से कम हो तो 400 से अधिक जितने भी unit हों उनका 80 पैसे प्रति unit के हिसाब से charge लिया जाएगा साथ ही 230 रू अधिक देने होंगे और यदि unit 600 से अधिक हों तो जितने unit अधिक होंगे उतने रूपये और 390 रू अधिक देने होंगे। एक प्रोग्राम लिखो जिसमें unit के हिसाब से कुल भुगतान की राशि ज्ञात हो।

Program

```
#include<stdio.h>
main()
{
    int units, customer;
```

```
    float charges;

    clrscr();
    printf("Enter CUSTOMER NO. and UNITS consumed\n");
    scanf("%d %d", &customer, &units);

    if(units<=200)
        charges=units * 0.05;
    else if(units<=400)
        charges=100+0.65 * (units-200);
    else if (units<=600)
        charges=230+0.8 *(units-400);
    else
        charges=390+(units-600);

    printf("\n\n customer no: %d\n charges = %.2f\n",customer,charges);
    getch();
}
```

Exercise:

1 Keyboard से किसी Character को Enter कीजिए और Program द्वारा Output में एक Message Display कीजिए, जिसके द्वारा ये बताया जाए कि Input किया गया Character Small Case Letter, Capital Letter, Digit या Special Symbol में से कौनसा है।

2 यदि किसी Triangle की तीनों भुजाओं को Input किया जाए, तो Input किए जाने वाले मान के आधार पर बनने वाला Triangle एक Valid Triangle होगा या नहीं, इस बात को तय करने वाला एक Program बनाईए। जबकि एक Valid Triangle की दो भुजाओं के माप का योग हमेंशा तीसरी भुजा के माप से ज्यादा होता है।

3 यदि किसी Triangle के तीनों कोणों को Keyboard से Input किया जाए। तो एक ऐसा Program बनाइए जो ये बताए कि इन तीनों कोणों के आधार पर बनने वाला Triangle समकोण Triangle है, न्यूनकोण हैं अथवा अधिककोण Triangle है। साथ ही Triangle के Valid होने की भी जांच कीजिए। किसी Triangle के तीनों कोणों का योग यदि **180** Degree हो, तो Triangle एक Valid Triangle होता है।

switch statement

जब हम किसी प्रोग्राम में ढेर सारी if Conditions का प्रयोग करते हैं, तो प्रोग्राम बहुत जटिल हो जाता है। इस वजह से प्रोग्राम को समझना व पढना काफी मुश्किल हो जाता है। इस कठिनाई से बचने के लिए हम एक और Control Statement **switch** का प्रयोग करते हैं।

जिस तरह if Condition एक Two - way Condition Statement है, उसी तरह switch एक Multi-way Condition Statement है। यह बिल्कुल **if – else if – else** के जैसा ही काम करता है। इसकी सामान्य संरचना निम्नानुसार होती है:

```
switch ( expression of variable )
{
case value 1:
    Statement or Statement Block 1;
    break;

case value 2:
    Statement or Statement Block 2;
    break;
case value 3:
    Statement or Statement Block 3;
    break;
    "      "      "      "
    "      "      "      "
    "      "      "      "
case value n:
    Statement or Statement Block n;
    break;

default :
    Statement or Statement Block;
    break;
}
sequential Statement a;
```

कार्यप्रणाली – इस संरचना में value1, value2.....value n expression या Variable हैं, जिन्हे **case label** कहा जाता है। इनके बाद **Colon** लगाना जरूरी होता है। ये सभी मान अलग–अलग होने चाहियें। Statement or Statement Block 1, Statement or Statement Block 2, … Statement or Statement Block n Statements का समूह है। इन Statements के समूहों में एक से अधिक Statements होने पर भी मंझले कोष्ठक की जरूरत नहीं होती है। फिर भी यदि ये कोष्ठक लगा दिए जाएं, तो भी "C" Compiler कोई परेशानी नहीं करता है।

switch Statement के Execution के लिए सर्वप्रथम variable या expression के मान की तुलना क्रम से एक–एक करके value1, value2 ..... value n से की जाती है और जहां भी ये मान मिल जाता है, उस case label के अनुसार लिखे गये Statements का Execution हो जाता है।

इस संरचना में सभी Statements Block के बाद **break** लिखा जाता है। यह Statements के समूह का अंत दर्शाता है व Statements Block के Execution के बाद प्रोग्राम Control को switch Statement के बाहर sequential Statement a पर ले जाता है।

यदि यह break ना लिखा जाए तो Statement Block से Execution के बाद भी Program Control **switch Statement Block** के अंदर ही रहता है और आगे के Statement का Execution करता रहता है। जब तक इसे कोई break Statement नहीं मिल जाता, तब तक ये सभी labels के Statement Block का Execution करता रहता है।

switch Statement के लिए दी गई संरचना में **default** case label एक Optional Label है। यदि यह switch Statement में होता है व expression या variable का मान switch Statement में दिये गए किसी भी Statement से मेल नही करता, तो Program Control default case label पर चला जाता है और इसके अंतर्गत दिये गए Statement का Execution कर देता है। switch case labels को हम किसी भी क्रम में रख सकते हैं, यानी चौथे स्थान का case प्रथम स्थान पर, प्रथम स्थान का case तीसरे स्थान पर। default Statement को भी किसी भी स्थान पर रख सकते हैं।

अब हम पिछले प्रोग्राम को ही switch Statement द्वारा लिखते हैं। इस प्रोग्राम में दो variable define किये गए हैं। एक variable marks का Input लेता है। फिर marks मे 10 का भाग दिया है ताकि कुल 10 situations हो जाए और प्राप्त मान को a को assign किया गया है। अब जब कोई मान माना कि हमने 55 Input किया तो 55/10= 5 a को assign हो जाता है।

ध्यान दें कि यहां 55/10 का मान 5.5 होता है, लेकिन a int प्रकार का variable है, इसलिए यह दशमलव के भाग को छोड देता है और केवल पूर्णांक संख्या को ही accept करता है। अब switch Statement में a का मान check होता है कि a का मान किस case से मेल कर रहा है। यहां switch का मान case 5 से मेल करता है, क्योंकि a का मान 5 है।

इसलिए switch Statement के case 5 के Statement Block का Execution हो जाता है और Output में SECOND DIVISION print हो जाता है। Statement Print होते ही Program Control को break मिलता है और Program Control switch Statement Block से बाहर आ जाता है। यदि हम कोई ऐसा अंक जो कि 39 से कम है, देते हैं तो कोई case मेल नही करता और switch के default Statement का Execution हो जाता है।

ये प्रोग्राम बना कर विभिन्न मान दें और प्रोग्राम की testing करें। जब प्रोग्राम को Develop करके पहले से ज्ञात विभिन्न मानों द्वारा प्रोग्राम का Execution करके प्राप्त होने वाले Output को Accuracy के लिए Check किया जाता है, तो इस प्रक्रिया को प्रोग्राम की **Testing** करना कहते हैं।

चलिए, अब हम **switch** Statement को समझने के लिए एक उदाहरण Program बनाते हैं। ये उदाहरण Program किसी Student के Marks के आधार पर Student का Grade Output में Display करता है। Program निम्नानुसार है:

Program

```
/*   Example of  switch Condition Statement          */
#include<stdio.h>
main()
{
    int marks, a;
    clrscr();

    printf("\n Enter Marks");
    scanf("%d", &marks);

    a = marks / 10;

    switch ( a )
    {
        case 10 :
        case 9 :
        case 8 :
            printf("HONOURS");
            break;

        case 7 :
        case 6 :
            printf("FIRST DIVISION");
            break;

        case 5 :
            printf("SECOND DIVISION");
            break;
```

```
            case 4 :
                printf("THIRD DIVISION");
                break;

            default :
                printf("FAIL");
                break;
        }

        getch();
    }
```

Example :

एक प्रोग्राम लिखो जिसमें User जिस भी रंग का प्रथम अक्षर Input करे,, Output में उस रंग का नाम आ जाए।

Program

```
    #include<stdio.h>
    main()
    {
        char a;
        clrscr();

        printf("Enter the character:\n");
        scanf("%c", &a);
        fflush(stdin);

        switch(a)
        {
            case 'R':
            case 'r':
                printf("\n Color is red\n");
                break;

            case 'g':
            case 'G':
                printf("\n Color is green\n");
                break;
```

```
        case 'b':
        case 'B':
            printf("\n Color is blue\n");
            break;

        default:
            printf("color is found");
            break;
    }
    getch();
}
```

Output

```
Enter the character: R
Color is red
```

Exercise:

1. **switch** Statement का Block Structure बनाते हुए इसकी कार्यप्रणाली को समझाईए।
2. **switch** Statement में **break** Keyword के महत्व को एक उचित उदाहरण द्वारा समझाईए।

goto Statement

किसी प्रोग्राम के Execution के Flow को इस Statement का प्रयोग करके तोडा जा सकता है और प्रोग्राम Control को प्रोग्राम में किसी अन्य चाही गई जगह पर भेजा जा सकता है। इसके द्वारा हम आवश्यकता के अनुसार किसी Statement की पुनरावर्ती बार–बार कर सकते हैं।

यह एक ऐसा Statement है, जिसे किसी Condition के साथ प्रयोग करके Looping का काम करवा सकते हैं और इसके द्वारा हम Program में कभी भी किसी भी Point पर जा सकते हैं। इसकी संरचना निम्नानुसार होती हैः

```
goto label;                 label:
    statements;                 statements;
label:                      goto label;
```

इसकी संरचना में एक लेबल होता है, जो यह बताता है कि Program Control प्रोग्राम में कहां जाएगा। लेबल एक वेरियेबल हो सकता है। यह लेबल **goto** से पहले या बाद में कहीं भी आ सकता है। यानी

```
goto again:
 "   "   "
"   "   "
again;
```

इस Code Segment के आधार पर देखें तो प्रोग्राम Control को जैसे ही **goto again** Statement मिलता है, प्रोग्राम Control वहां आ जाता है, जहां **again** लिखा है और प्रोग्राम बीच के Statements को छोड देता है।

यदि **goto** Statement पहले Execute हो चुके Statements से पहले आता है, तो Program Control पुनः उन्हीं Statements का Execution कर देता है। जब किसी परिस्थितिवश किन्हीं Statements का बार–बार Execution होने लगता है, तो इस प्रकार की परिस्थिति को **Looping** कहते हैं।

जब goto Statement किसी पहले से Execute हो चुके Statement को Execute करने के लिए Program के किसी पिछले Label पर जाता है, तो इस प्रक्रिया को **Backward Jump** कहते हैं। लेकिन यदि लेबल goto Statement के बाद में आता है, तो Program Control बीच के Statements को छोड कर वहां से आगे के Statements का Execution करना शुरू कर देता है, जहां पर लेबल होता है। इस प्रक्रिया को **Forward Jump** कहते हैं। इसे समझने के लिए निम्न उदाहरण देखते हैंः

Example :

इस प्रोग्राम में 1 से 10 तक की संख्या को Print किया गया है। यदि साधारण प्रोग्राम द्वारा ये काम करना हो, तो हमें 10 printf() Function लिखने पड़ेंगे, जबकि इस प्रोग्राम में goto Statement का प्रयोग करके अंकों को Print किया है।

Program

```
/*   Use of goto Statement   */
#include<stdio.h>
#include<conio.h>
main()
{
    int i=0;
    clrscr();
    next:
    i++;
    if(i<=10)
    {
        printf("\n \t \t %d", i);
        goto next;
    }
    getch();
    return(0);
}
```

इस प्रोग्राम में एक int प्रकार का Variable i लिया है और **goto** Statement के साथ एक लेबल next का प्रयोग किया है। Program Control क्रम से एक int प्रकार का वेरियेबल i Define करता है और इसका प्रारम्भिक मान 0 कर देता है। फिर Program Control **clrscr();** Function पर जाता है।

ये Function Output Screen को साफ करने का काम करता है। इसके बाद next नाम का एक लेबल Program Control को मिलता है। यह लेबल यहां पर कोई काम नहीं करता। किसी भी goto Statement के प्रोग्राम में लेबल का मतलब इतना ही होता है, कि प्रोग्राम इस लेबल के आगे के Statements का Execution करेगा, यानी लेबल "C" Compiler को मात्र उस Point पर ले जाता है, जहां से आगे के Statements का Execution होना है। Program Control इस लेबल के आगे के प्रथम Statement i++ का Execution करता है और i का मान एक बढा देता है यानी 0 से बढा कर 1 कर देता है। फिर if कोष्ठक में यह Check किया जाता है, कि i का मान 10 से कम है या नहीं।

यहां Condition सत्य होती है, क्योंकि i का मान पहले चक्र में 1 है। Condition सत्य होने के कारण Program Control if Statement Block के Statements का Execution करता है। यहां एक printf() Function द्वारा i का मान जो कि 1 है, print किया जाता है। अगली पंक्ति में Program Control को goto next Statement मिलता है और प्रोग्राम कंट्रोल सारे Executions छोड कर पुनः वहां चला जाता है, जहां next लेबल होता है और वहां से आगे के Statements का पुनः Execution शुरू कर देता है।

यहां वापस Program Control को i++ मिलता है, जो कि i का मान एक और बढा कर 2 कर देता है। वापस if Condition check होती है और i का मान 2 Print हो जाता है। फिर वापस goto next Statement मिलता है और Program Control वापस next लेबल पर चला जाता है। i का मान पुनः एक बढ कर 3 हो जाता है। पुनः i को Print किया जाता है। इस प्रकार यह प्रक्रिया तब तक चलती रहती है जब तक कि i का मान 10 से अधिक नहीं हो जाता। i का मान 11 होते ही if Condition असत्य हो जाती है और Program Control if Condition पर ना जाकर सीधे ही getch(); statement पर चला जाता है और हमें Output में 1 से 10 तक की संख्या प्राप्त हो जाती है।

Exercise:

1. एक Program बनाते हुए goto Statement को समझाईए।
2. **goto** Statement को Looping Statement की तरह Use करते हुए 10 का पहाडा Print करने का Program बनाईए और इस Program द्वारा goto Statement को समझाते हुए ये भी बताईए कि Programming में goto Statement का प्रयोग क्यों नहीं किया जाता है ?
3. **Backward Jump** व **Forward Jump** से आप क्या समझते हैं ? 100 से 90 तक की Reverse गिनती Print करने का Program बनाते हुए समझाईए।

Looping Statements

ये तीसरे प्रकार के Control Statements होते हैं। जब प्रोग्राम में हमें किसी प्रक्रिया को बार–बार दोहराना होता है, तब हम Looping Control Statements का प्रयोग करते हैं। किसी भी Loop में हमेंशा तीन बातें निश्चित करनी होती हैं:

1 इसे Loop का **Initial Part** कहा जाता है। इसमें Loop को Iterate करने वाले Variable को प्रारम्भिक मान दिया जाता है, जो ये तय करता है कि Loop की शुरूआत कब से होगी। यहां हमेंशा Assignment Operator = का प्रयोग किया जाता है।

2 इसे **Test Condition** कहा जाता है। किसी भी Loop में यह Part ये तय करता है कि Loop किस Condition में execute होगा। जब तक Test Condition सत्य होती है, तभी तक Loop का Iteration चलता है। इसलिए Loop के इस भाग को Define करना बहुत ही जरूरी होता है। यही भाग किसी Loop को बताता है कि Loop को कहां तक चलना है। इस भाग में Conditional व Logical Operators का प्रयोग किया जाता है। यहां एक Valid Condition देना बहुत ही जरूरी होता है, क्योंकि यदि यहां पर एक Valid Condition define नहीं करते हैं तो कई बार Loop Infinite हो जाता है।

3 Loop के इस में **Step Size** बताना होता है। यानी यहां Loop को ये बताना होता है कि Loop किस क्रम में आगे बढेगा, घटते क्रम में या बढते क्रम में। यहां हमेंशा Increment या Decrement Operator का प्रयोग किया जाता है। यदि हमें Loop को एक के क्रम में ना बढा कर किसी और क्रम में बढाना या घटाना होता है, तो यह काम Assignment Operators का प्रयोग करके किया जाता है। जैसे b = b + 2 यह expression Loop को दो–दो के क्रम में Increase करेगा। इसके स्थान पर हम इस Expression का संक्षिप्त रूप b += 2 का प्रयोग भी कर सकते हैं, जैसा कि पिछले अध्याय में **Assignment Operators Heading** के अन्तर्गत बताया गया है।

Loop मुख्यतः तीन प्रकार के होते हैं। इनकी अपनी–अपनी, अलग–अलग विशेषता व उपयोग है, जिन्हें निम्नानुसार समझाया गया है:

for Loop

यह सर्वाधिक प्रयोग होने वाला Loop है। इस Loop में "C" के **for** Key Word का प्रयोग होता है। इस Loop में ऊपर बताए गए तीनों ही भाग एक ही कोष्ठक में लिखने होते हैं। इस Loop की विशेषता यह है, कि इसके जितने भी Statement होते हैं, उन्हे for Loop लिखने के बाद उसके नीचे मंझले कोष्ठक के एक Block में लिखा जाता है और ये Statements Block तभी Execute होता है, जब for Condition सत्य होती है। for Loop का Syntax निम्नानुसार होता है–

```
for( Initial Part; Conditional Part; Step Size Part)
{
```

```
        Statements Block;
    }
```

जब for Loop का Execution होता है, तो सर्व प्रथम Loop का Variable Initialize होता है और फिर Condition Check होती है। यदि Condition सत्य होती है, तो Program Control for Loop के Statement Block में जाता है और वहां के Statements का Execution करता है।

जब For Loop Statement Block के सभी Statements का Execution कर देता है तो Block से बाहर आने से पहले Loop के Step Size Part का Execution करता है और बताई गई Size के अनुसार Variable का मान Increment या Decrement करता है।

फिर वापस Condition Check करता है यदि Condition सत्य होती है तो वापस Statement Block में जाता है और सभी Statements का Execution करने के बाद वापस Step Size Part का Execution करता है।

ये क्रम तब तक चलता रहता है जब तक कि for Loop की Condition सत्य रहती है। Loop का Initialization केवल एक बार ही होता है जब पहली बार Program Control For Loop में प्रवेश करता है। for Loop का Execution हमेंशा इसी क्रम में होता है।

Example :

इस Loop द्वारा हम एक प्रोग्राम बनाते हैं जिसमें 1 से 7 तक की गिनती को Output में निम्नानुसार Print करवाना हैं।

```
1
2
3
4
5
6
7
```

इस प्रोग्राम को हम पहले गणितीय रूप में लिखते हैं। हम int प्रकार का एक Variable i लेते हैं। इस प्रोग्राम में गिनती का प्रारम्भिक मान 1 है अतः i का प्रारम्भिक मान i = 1 कर सकते हैं। यह इस प्रोग्राम के **Initial Part** का Declaration है।

Print होने वाली गिनती का अधिकतम मान 7 है, इसलिए Condition के रूप में हम ये कह सकते हैं कि Loop तब तक चलना चाहिये जब तक कि i का मान 7 नहीं हो जाता। इसलिए हम इसे गणितीय रूप में i <= 7 लिख सकते हैं। यह Expression प्रोग्राम को बताता है कि Condition तब तक सत्य रहेगी जब तक कि i का मान 7 से कम या 7 के बराबर नहीं हो जाता। यह इस प्रोग्राम के **Conditional Part** का Declaration है।

इस प्रोग्राम में हर पहली संख्या हर दूसरी संख्या से बडी है। इसलिए Loop के Step Size Part में हमें Increment Operator का प्रयोग करना होगा और साथ ही हर संख्या एक के क्रम में बढ रही है इसलिए हमें Variable के मान को हर Iteration में एक के क्रम में बढाना होगा। इसे गणितीय रूप में i = i +1 या i++ भी लिख सकते हैं। यह इस प्रोग्राम के **Step Size Part** का Declaration है। इस प्रकार हमें for Loop के तीनों भाग प्राप्त हो गए हैं जो निम्नानुसार हैं:

```
i = 1
i <= 7
i ++ or i = i + 1
```

इन तीनों को for Loop में रख देते हैं तो निम्न प्रकार से for Loop का कोष्ठक बन जाता है। ध्यान दें कि ये तीनों Expression अलग–अलग हैं। इसलिए इन्हे ; (Semicolon) से अलग करके लिखा गया है–

```
for(i = 1; i <= 7; i = i + 1)      or      for(i = 1; i <= 7; i++)
```

अब for Loop का Statement Block लिखना है। i का मान क्रम से एक–एक बढ रहा है। इसलिए Statement Block में यदि i का मान Print कर दिया जाए, तो हमें हमारा Output प्राप्त हो जाएगा। इसे हम निम्नानुसार लिख सकते हैं–

```
{
    printf("%d ", i );
}
```

इस प्रोग्राम में पहला अंक Print होने के बाद दूसरा अंक अगली पंक्ति में Print होना चाहिए। इसलिए हमें नई लाइन के लिए एक New Line Statement और लिखना होगा। साथ ही ये Statement भी for Statement Block के अंदर ही लिखना होगा ताकि जैसे ही Program Control एक Iteration के बाद दूसरे Iteration के लिए जाए, उससे पहले एक New Line Print कर दे ताकि अगला अंक नई लाइन में Print हो। हम printf() Function के अंदर ही एक New Line Character Constant लिख सकते हैं या फिर एक और printf() Function लिख कर उसमें New Line Character Constant लिख सकते हैं। ये दोनों Statement नीचे लिखे गए हैं। हम जिसे चाहें उसे Use कर सकते हैं।

```
printf("%d \n", i);      or      printf("\n");
```

Program

```
/*   Using Of For Loop        */
```

```
#include<stdio.h>
main()
{
    int i;
    clrscr();

    for(i = 1; i <= 7;i++)
    {
        printf("%d \n", i );
    }
    getch();
}
```

इस प्रोग्राम में जब प्रथम बार for Loop का Execution होता है, तब i = 1 होता है। इसलिए Condition सत्य होती है, साथ ही i का मान Increment हो कर 2 हो जाता है। Program Control, Block Statement को Execute कर देता है और Output में संख्या 1 Print हो जाती है। Program Control वापस i का मान Check करता है और Condition वापस सत्य होती है। i का मान पुनः Incremented हो कर 3 हो जाता है और Output में संख्या 2 Print हो जाती है।

पुनः Condition Check होती है और यह क्रम तब तक चलता है जब तक कि i का मान Increment हो कर 11 नहीं हो जाता। जैसे ही i का मान 11 होता है for Condition असत्य हो जाती है और Program Control for Statement Block को Execute नहीं करता बल्कि सीधे ही getch() Function पर पहुंच कर Output को Print कर देता है।

Example :

1 से 100 तक की सम संख्याओं को Print करने का प्रोग्राम बनाओ। हर संख्या एक नई पंक्ति में Print होनी चाहिये।

इस प्रोग्राम में हमें एक Loop चलाना होगा और Loop की Step Size का मान यदि दो–दो के क्रम में बढाया जाए तो प्रोग्राम के Output में हमें केवल सम संख्याएं ही प्राप्त होंगी। नीचे इस समस्या के समाधान के लिए प्रोग्राम बनाया गया है। इस प्रोग्राम में int प्रकार का एक Variable b लिया है। इस समस्या का प्रोग्राम निम्नानुसार है:

Program

```
/*  Printing Odd Numbers In Output Using Loop      */
#include<stdio.h>
main()
{
```

```
        int b;
        clrscr();
        for(b=0; b<=100; b+=2)
        {
            printf("%d \n", b);
        }
        getch();
    }
```

जब प्रोग्राम को Execute किया जाता है, तो Program हमेंशा की तरह Main() Function पर जाता है और Program के Statement Block में प्रवेश करता है। यहां Compiler को int प्रकार का एक Variable b प्राप्त होता है।

clrscr() Function के बाद Compiler को for Loop मिलता है। यहां Compiler तीन काम करता है। पहला, int प्रकार के Variable को प्रारम्भिक मान 0 assign करता है। दूसरा, Condition Check करता है कि b का मान 100 से कम है या नहीं और जब Statement Block के सभी Statements का Execution करके वापस Condition Check करने के लिए for Loop के कोष्ठक पर आता है, तो Block से बाहर निकलने से पहले तीसरा काम करता है यानी b का मान उसे Initialized प्रारम्भिक मान से दो अंक Increment करके बढा देता है।

b का मान 0 होने से Condition सत्य हो जाती है, इसलिए Program Control, for Statement Block में प्रवेश करता है। यहां उसे printf() Function मिलता है और वह उसे Execute करके b का मान Output में Print करता है। Statement Block को Execute करने के बाद पुनः Condition Check होती है कि b का मान अब 100 से कम है या नहीं। Condition वापस सत्य होती है क्योंकि अब b का प्रारम्भिक मान 2 है जो कि 100 से कम है। Condition पुनः सत्य होने से पुनः Statement Block Execute होता है और b का दूसरा मान 2 Print हो जाता है।

पुनः Program Control b का मान Check करने से पहले b का मान बढ कर 4 हो जाता है। पुनः Condition Check होती है कि b का मान 100 से कम या बराबर है या नहीं पुनः Condition सत्य होती है। इस प्रकार यह क्रम तब तक चलता रहता है जब तक कि b का मान Increment हो कर 102 नहीं हो जाता।

जैसे ही b का मान बढ कर 102 होता है तो for की Condition असत्य हो जाती है और Program Control, for Statement Block में प्रवेश नहीं करता बल्कि सीधे ही getch() Function पर चला जाता है और Output Screen पर Show हो जाता है। इस प्रकार से हमें Output में केवल सम संख्याएं ही मिलती हैं।

Example :

20 ये 10 तक की सभी सम संख्याएं अवरोही क्रम में Print करो।

Program

```
#include<stdio.h>
main()
{
    int a;
    clrscr();

    for(a=10; a>=5; a--)
    {
        printf("\n %d", a*2);
    }
    getch();
}
```

Output

```
20
18
16
14
12
10
```

Example :

एक प्रोग्राम बनाओ जिसमें 100 से 200 तक की संख्याओं का योग Output में Print हो।

Program

```
#include<stdio.h>
main()
{
    int i, sum = 0;
    clrscr();

    for(i=100; i<=200; i++)
    {
```

```
            sum = sum + i;
        }
        printf("\n sum = %d", sum);
        getch();
    }
```

Example :

एक प्रोग्राम बनाओ जिसमें n संख्याओं का Factorial Output में Print हो।

Program

```
    #include<stdio.h>
    main()
    {
        int i, n, fact = 1;
        clrscr();
        printf("enter the n:");
        scanf("%d", &n);

        for(i=n; i>0; i--)
            fact=fact*i;

        printf("fact = %d", fact);
        getch();
    }
```

Output

```
    Fact =120
```

Example :

एक प्रोग्राम बनाओ जिसमें n संख्याओं की Fibonacci Series Output में Print हो।

Program

```
    #include<stdio.h>
    main()
    {
        int a=0,b=1,sum,n,c;
        clrscr();
```

```
        printf("Enter the Fibonacci series:");
        scanf("%d", &n);
        printf("%d\n", b);

        for(c=1;c<=n; c++)
        {
            sum = a + b;
            a=b;
            b=sum;
            printf("%3d", sum);
        }
        getch();
    }
```

Output

```
Enter the Fibonacci serise:5
1  1  2  3  5  8
```

Example :

एक प्रोग्राम बनाओ जिसमें 100 से 200 के बीच की उन संख्याओं का योग व संख्याएं Output में Print हों, जिनमें 7 का भाग पूरा–पूरा जाता है।

Program

```
#include<stdio.h>
main()
{
    int a, sum=0;
    clrscr();

    for(a=100;a<=200;a++)
    {
        if(a%7==0)
        {
            printf("%d", a);
            sum = sum + a;
        }
    }
    printf("\n sum = %5d\n", sum);
```

```
    getch();
}
```

Output

```
105  112  119  126  133  140  147  154  161  168  175  182  189  196
sum = 2107
```

for Loop की एक और विशेषता ये भी है कि हम एक ही for Loop कोष्ठक में आवश्यकता के अनुसार कई Loop चला सकते हैं। इसका syntax निम्नानुसार होता है–

```
for( Initial1, Initial 2,..., Initial Part n ;  ConditionalPart; Step1, Stept2, ..., StepN )
{
    Statements Block;
}
```

एक उदाहरण द्वारा इसे समझते हैं। माना हमें निम्न Format में गिनती Print करनी है।

```
1     10    1     10
2     9     2     9
3     8     3     8
4     7     4     7
5     6     5     6
6     5     6     5
7     4     7     4
8     3     8     3
9     2     9     2
10    1     10    1
```

इस प्रोग्राम में चार पंक्तियों में अंक Print हुए हैं इसलिए हमें चार अंकों के लिए चार Variable लेने होंगे। माना हमने चार Variable b, c, d, f लिए। इस प्रोग्राम में दो Variables को तो मान वैसे ही दिया जाएगा जिस तरह Loop के पिछले प्रोग्राम में दिया है, यानी निम्नानुसार–

```
b = 1          d = 1
b <= 10        d <= 10
b++            d++
```

लेकिन दो Variables के साथ इसकी उल्टी प्रक्रिया करनी होगी यानी Loop का मान प्रथम मान 10 व अन्तिम मान 1 होना चाहिये तथा प्रोग्राम में Variables का Decrement होना चाहिये। इससे Variable c व f का मान निम्नानुसार हो जाएंगे:

```
c = 10          f = 10
c >= 1          f >= 1
c--             f--
```

अब इन चारों variables का मान ऊपर वाले for Loop के Syntax के अनुसार लिख देते हैं, तो हमें में निम्नानुसार Format प्राप्त होता है –

```
for(b=1, c=10, d=1, f=10; b<=10, c>=1, d<=10, f>=1; b++, c--, d++, f--)
```

इस Loop में चार Loop एक साथ चलेंगे। अब हम यदि इन Loops से प्राप्त मानों को Output में Print करें, तो हमें हमारा वांछित परिणाम प्राप्त हो जाएगा। इसके लिए हमें Statement Block में निम्न Statement लिखना होगा:

```
{
    printf("\t %d \t %d \t %d \t %d \t \n", b, c, d, f);
}
```

अब हम पूरा प्रोग्राम लिखते हैं, जो निम्नानुसार होगा:

Program

```
/*  Using Of For Loop with multiple Conditions        */
#include<stdio.h>
main()
{
    int b, c, d, f;
    clrscr();

    for(b=1,c=10,d=1,f=10 ; b<=10,c>=1,d<=10,f>=1; b++, c--, d++, f--)
    {
        printf("%d \t %d \t %d \t %d \t \n", b, c, d, f );
    }
    getch();
}
```

इस प्रोग्राम में पहली बार जब Loop का Execution होता है तब b=1, c=10, d=1 व f=10 होता है। अगले Iteration में b=2, c=9, d=2 व f=9 हो जाता है। इसी प्रकार ये क्रम चलता रहता है और क्रम से Output में हमें मान प्राप्त होता जाता है।

एक खास बात इस Loop की यह है कि इसमें Loop Control के लिए जो Condition दी जाती है, वह Condition सभी को देने की कोई जरूरत नहीं होती है, क्योंकि Program Control केवल एक Condition को ही check करता है, शेष Condition को Check ही नहीं करता।

इसलिए यदि हम चाहें तो केवल एक ही Condition से भी यही Output प्राप्त कर सकते हैं। इस प्रोग्राम में Loop f>=1 की Condition पर काम कर रहा है। इसलिए हम चाहें तो शेष Condition को हटा सकते हैं। ऐसा करने पर प्रोग्राम की for Condition कुछ इस प्रकार से हो जाएगी।

```
for(b=1,c=10,d=1,f=10 ; f>=1; b++, c--, d++, f--)
```

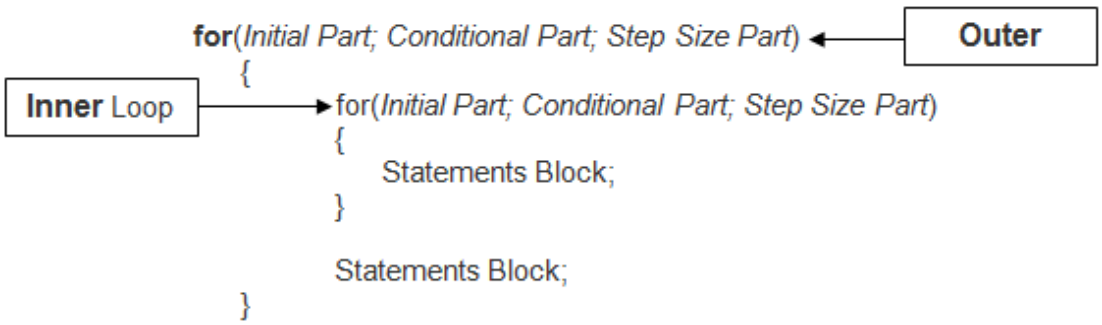
यदि यह Syntax हम for Loop प्रोग्राम में Use किये गए Loop के स्थान पर Use करते हैं, तो भी Output वही प्राप्त होता है जो पहले हुआ था। सारांश के रूप में हम कह सकते हैं कि जब एक for Loop में कई Loop एक साथ Execute करने होते हैं तो हर Loop का प्रारम्भिक मान व Step Size तो सभी को दिया जाता है, लेकिन एक for Loop में Condition सिर्फ एक ही हो सकती है। यदि एक से अधिक Condition देते हैं, तो अन्तिम Loop की Condition के अनुसार ही Loop Execute होता है क्योंकि for Loop **Right To Left** चलता है।

Nesting of Loop

जिस तरह हमने if Conditionals Statements की Nesting की थी उसी तरह से हम Loops की भी Nesting कर सकते हैं। यानी कई बार प्रोग्राम में ऐसी जरूरतें होती हैं, कि उस खास काम को Loop की साधारण प्रक्रिया द्वारा नहीं कर सकते। तब हमें एक Loop के अंदर एक अन्य Loop को Use करना पडता है। जब किसी Loop में वापस Loop का प्रयोग किया जाता है तो इसे Loop की Nesting करना कहते हैं।

Nesting of for Loop

For Loop की जब Nesting की जाती है तब हमेंशा Outer Loop, Inner Loop को Control करता है। Outer Loop Row के लिए व Inner Loop Column के लिए लिखा जाता है। इसका Syntax निम्नानुसार होता है:



जब Program Control, Outer for Loop पर आता है तब check करता है कि Outer For Loop की Condition सत्य है या नहीं। यदि Outer For Loop की Condition सत्य होती है, तो Program Control Inner Loop में प्रवेश करता है। यहां Program Control को एक और for Loop मिलता है और प्रोग्राम Control इस for की Condition को Check करता है।

यदि Inner for Loop की Condition सत्य होती है तो Inner for Loop Iterate होता है और ये Inner Loop तब तक Iterate होता रहता है जब तक कि Inner Loop की Condition सत्य होती है। जैसे ही inner Loop की Condition असत्य होती है, प्रोग्राम Control वापस Outer Loop को check करता है।

यदि वापस Outer Loop की Condition सत्य हो जाती है तो Program Control पुनः inner Loop में प्रवेश करता है और पुनः Inner Loop का तब तक Iteration होता है जब तक कि Inner Loop की condition असत्य नहीं हो जाती।

Program Control इन्हीं दोनों Loops के बीच तब तक Iterate होता रहता है जब तक कि दोनों Loops की Condition असत्य ना हो जाए। यदि Outer Loop की Condition पहली बार में ही असत्य हो जाए तो Inner Loop का Execution ही नहीं होता है।

Inner Loop का Execution तभी होता है जब Outer Loop की Condition सत्य हो। Loop की Nesting को समझने के लिए आगे एक प्रोग्राम बनाया गया है। इससे for Loop की nesting व उसके काम करने तरीके को अच्छी तरह से समझाया गया है।

Example :

एक ऐसा प्रोग्राम बनाओ जो निम्न Format को print करे।

```
*
* *
* * *
* * * *
* * * * *
```

इस प्रोग्राम में दो for Loop चलाए गए हैं और Condition इस प्रकार की रखी गई है कि Inner Loop उतनी ही बार चले जितना बाहर का Loop चलाने वाले variable का मान हो। प्रोग्राम निम्नानुसार है:

Program

```
/*   Example of Nested For Loop              */
#include<stdio.h>
main()
{
    int j, k;
    clrscr();

    for(k=1; k<=5; k++)
    {
        for(l=1; l<=k; l++)
        {
            printf(" *");
        }
        printf("\n");
    }
    getch();
}
```

जैसे ही प्रोग्राम का Execution होता है तो Program Control को for Loop मिलता है। k का प्रारम्भिक मान 1 दिया गया है, इसलिए Condition सत्य हो जाती है और Program Control, Outer Loop के Statement Block में जाता है। यहां Program Control को एक और for Loop मिलता है जिसके Variable का प्रारम्भिक मान 1 है और Condition के रूप में ये शर्त दी गई है कि Inner Loop तब तक Execute होना चाहिये, जब तक कि Inner Loop के Variable का मान, बाहरी Loop के Variable के मान से कम या बराबर हो।

चूंकि बाहरी Loop के Variable k का मान 1 है। यहां k का मान 1 व Variable l का मान भी एक ही है। इसलिए Condition तो सत्य हो गई लेकिन दोनों का मान समान होने से Inner Loop एक ही बार चलता है।

Inner Loop की Condition के सत्य होते ही Program Control, Inner Loop के Statement Block में जाता है। यहां एक printf() Function द्वारा * को Print किया जाता है। अब Inner Loop का मान पुनः Check किया जाता है, तो Condition असत्य हो जाती है, क्योंकि Inner

Loop में I का मान Incremented होकर 2 हो गया है, जबकि k का मान 1 ही है व Condition तभी सत्य होती है, जब I का मान k के मान से कम या बराबर हो।

इसलिए Condition असत्य हो जाती है। Condition के असत्य होते ही Program Control, Inner for Loop से बाहर Outer Loop के Statement Block में आ जाता है। यहां Program Control को एक और printf() Function प्राप्त होता है। यह Function एक New Line Print करता है।

अब दोबारा Outer For Loop Check होती है। पिछले Increment के कारण यहां भी k का मान अब 2 हो चुका होता है। 2 का मान 5 से कम है इसलिए Outer Loop की Condition पुनः सत्य हो जाती है व Program Control पुनः Inner Loop में प्रवेश करता है। Inner Loop में पुनः वही सारी प्रक्रिया होती है जो पिछले Iteration में हुई थी, यानी पुनः b का मान 0 Initialize होता है। पुनः Condition सत्य हो जाती है और साथ ही पुनः b का मान बढ कर 2 हो जाता है।

इस बार Condition I <= k यानी I <= 2 हो जाती है, इसलिए Inner Loop अब दो बार चलता है। पहली बार में I का मान 1 होता है इसलिए Condition सत्य होती है और Program Control, Statement Block में प्रवेश करता है। यहां पर एक * Print होता है। पुनः Inner Condition Check होती है। इस Iteration में I का मान Increment होकर 2 हो जाता है। पुनः Condition check होने पर I <= 2 होने से Condition सत्य हो जाती है।

Condition सत्य होने के साथ ही I का मान एक और बढ जाता है और बढ कर 3 हो जाता है। Condition सत्य होने से पुनः एक * Print होता है। यह * पिछले * के पास में ही Print होता है, क्योंकि हमने Printf() Function में New Line के लिए कोई Character Constant Use नहीं किया है। इस प्रकार दूसरी पंक्ति में दो * Print हो जाते हैं।

अब वापस Inner Loop की Condition check होती है। चूंकि अब I का मान बढ कर 3 हो चुका है इसलिए I <= k Condition असत्य हो जाती है क्योंकि I का मान अब 3 है जबकि k का मान 2 है इस कारण से यदि गणितीय रूप में इस Expression को देखें तो 3 <= 2 Expression बनती है, जो कि सही नहीं है। इसलिए Condition असत्य है।

Condition असत्य होने से Program Control, Inner Loop से बाहर निकल कर वापस Outer Loop में प्रवेश करता है। पिछले Iteration के बाद k के मान का Increment होकर 3 हो चुका है। इसलिए k का मान अब 3 है।

Outer Loop की Condition के अनुसार Condition तभी सत्य होगी जब k का मान 5 से कम या बराबर हो। यहां k का मान 3 है इसलिए Condition पुनः सत्य होती है और Program Control पुनः Inner Loop में प्रवेश करता है। इस प्रकार ये क्रम तब तक चलता रहता है जब तक कि k का मान 6 नहीं हो जाता और k का मान पांचवे Iteration के बाद 6 हो जाता है।

जैसे ही k का मान 6 होता है Outer Loop की Condition असत्य हो जाती है। Outer Loop की Condition असत्य होने से Program Control, Inner Loop में प्रवेश नहीं करता बल्कि सीधे ही **getch**() Function पर जा कर उपरोक्त Format Print कर देता है।

Exercise:

1 Looping Statements किसे कहते हैं ? for Loop का प्रयोग करते हुए 2 का पहाडा Print करने का एक Program बनाईए व for Loop के विभिन्न हिस्सों को समझाईए।

2 Keyboard से Input की गई किसी संख्या **n** तक का Factorial ज्ञात करने का Program बनाइए।

3 Keyboard से Input की गई किसी संख्या **n** तक की Fibonacci Series Print करने का Program बनाइए।

4 0 से 255 तक के ASCII Codes व उनसे Associated Character को Screen पर Print करने का Program बनाईए।

5 Looping का प्रयोग करते हुए निम्न Format को Output में Print कीजिएः

A B C D E F G F E D C B A

6 Keyboard से Input की गई संख्या Armstrong Number है अथवा नहीं, इस बात की जानकारी Output में Print करने वाला Program बनाईए। Armstrong Number एक ऐसा Number होता है, जिसके हर Digit का Cube ज्ञात करके उन्हें आपस में जोड देने पर फिर से मूल Number बन जाता है। उदाहरण के लिए 153 एक Armstrong Number है, क्योंकि इसके तीनों अंकों का घन मान ज्ञात करके उन्हें आपस में जोडने पर हमें फिर से 153 प्राप्त हो जाता है। यानी

```
153 = ( 1 * 1 * 1) + ( 5 * 5 * 5 ) + ( 3 * 3 * 3 )
153 =       1      +      125      +      27
153 = 153
```

7 1 से 100 तक की संख्या के बीच के सभी Prime Numbers को Screen पर Display करो। Prime Number एक ऐसा Number होता है, जिसमें 1 व स्वयं उसी संख्या के अलावा किसी भी अन्य संख्या का भाग पूरा–पूरा नहीं जाता। उदाहरण के लिए 1, 2, 3, 5, 7, 11 आदि

8 Nesting से आप क्या समझते हैं ? विभिन्न प्रकार के Control Statements व Looping Statements की Nesting को उचित उदाहरण Programs द्वारा समझाईए व ये भी बताईए कि किस Control Statement की Nesting कब व क्यों की जाती है ?

9 एक ऐसा प्रोग्राम बनाओ जो Output में Alphabet के Capital Letters को Print करे।

10 एक प्रोग्राम बनाओ जो दस अंक Input ले और उसमें से सबसे बडे अंक को Output में Print करे।

while Loop

For Loop की तरह यह भी किसी Statement के दोहरान का काम करता है, लेकिन फिर भी यह for Loop से काफी अलग है। इस Loop में "C" के Keyword **while** का प्रयोग किया जाता है। while Loop में while कोष्ठक में केवल Condition दी जाती है।

Variable का प्रारम्भिक मान व Step Size while के कोष्ठक का हिस्सा नहीं होते हैं, बल्कि Variable का प्रारम्भिक मान while Loop को शुरू करने से पहले ही Declare व Initialize कर दिया जाता है और Loop की Step Size while Condition के Statement Block का हिस्सा होती है। **while** Loop का Syntax निम्नानुसार होता हैः

```
Variable Declaration;
Value Initialization;

while(Condition )
{
    Statement Block;
    Step Size;
}

Statement 1;
```

while Statement के कोष्ठक के बाद कभी भी ; (Semi Colon) का प्रयोग नहीं किया जाता है। जब while के बाद केवल एक ही Statement का Execution करना होता है, तब हमें मंझले कोष्ठक का प्रयोग करने की जरूरत नहीं रहती है। फिर भी यदि मंझले कोष्ठक का प्रयोग कर लिया जाए तो कोई फर्क नहीं पडता है।

सर्वप्रथम हमें Loop चलाने वाले Variable को प्रारम्भिक मान देना होता है। यह काम while Loop के बाहर ही कर लिया जाता है। जब Program Control, while Loop में प्रवेश करता है तो Program Control, Condition Check करता है।

यदि Condition सत्य होती है तो Program Control while Loop के Statement Block में प्रवेश करता है और Statement Block का Execution करता है। Execution के बाद Statement Size तय करता है, यानी Loop के Variable का मान जरूरत के अनुसार Increment या Decrement करता है।

यह क्रम तब तक चलता रहता है जब तक कि while Condition असत्य नहीं हो जाती है। यदि Condition सत्य नहीं होती है तो Program Control while Loop के Statement Block में प्रवेश नहीं करता, बल्कि सीधे ही Statement 1 पर चला जाता है।

Example :

While Loop का प्रयोग करते हुए एक ऐसा प्रोग्राम बनाओ जिसमें 1 से 100 के बीच वह संख्या Print हो जिसमें 2 व 3 का पूरा–पूरा भाग जाता है।

Program Definition – किसी भी Loop को चलाने के लिए Loop की आवश्यक तीनों बातें हमें पता होनी चाहिए। पहली बात ये कि Loop का प्रारम्भ कहां से होगा, दूसरा ये कि Loop का अंत कहां होना है और तीसरा ये कि Loop को किस क्रम में Increase या Decrease करना है। किसी भी Loop में किसी Variable द्वारा ये तीनों बातें तय की जाती हैं।

इस प्रोग्राम में भी एक Variable x लिया गया है ताकि ये तीनों मान प्राप्त किये जा सकें। इस प्रोग्राम में हमें 1 से 100 के बीच की संख्याओं पर प्रक्रिया करनी है इसलिए Loop के Variable का प्रारम्भिक मान 1 व अन्तिम मान 100 Define किया गया है और Step Size को एक के क्रम में ही बढाया गया है ताकि ये पता लग सके कि कौनसा अंक ऐसा है, जिसमें 2 व 3 का भाग पूरा–पूरा जाता है। इस प्रोग्राम में हमें ये पता करना है कि किस संख्या में 2 व 3 का भाग पूरा–पूरा जाता है। यह काम हम if Condition Statement द्वारा ही कर सकते हैं।

क्योंकि हर Iteration के बाद Loop चलाने वाले Variable का मान बढ जाया करेगा इसलिए हर अंक में 2 व 3 का भाग देना होगा और यदि संख्या में 2 व 3 का भाग पूरा–पूरा चला जाता है तो शेषफल शुन्य प्राप्त होगा। जिस संख्या में 2 व 3 का भाग पूरा–पूरा चला जाएगा वह संख्या Output में Print हो जाएगी। जो संख्याऐं If Condition को सन्तुष्ट नहीं करेंगी।

यानी जिस संख्या में 2 का भाग तो पूरा–पूरा चला जाए लेकिन 3 का भाग ना जाए या फिर जिस संख्या मे 3 का भाग पूरा–पूरा चला जाए लेकिन 2 का ना जाए तो Condition असत्य हो जाएगी और वह संख्या Output में Print नहीं होगी।

if Condition के रूप में ये शर्त देनी होगी कि Output में वही संख्या Print हो जिसमें 2 व 3 का पूरा–पूरा भाग जाता है। इस शर्त को यदि हम ध्यान से समझें तो ये कह सकते हैं कि इस एक शर्त में दो शर्त है।

पहली ये कि किसी संख्या में 2 का भाग पूरा–पूरा जाना चाहिये और दूसरी ये कि उसी संख्या में 3 का भाग भी पूरा–पूरा जाना चाहिये तभी वह Condition पूर्ण रूप से सत्य होगी। इस शर्त को यदि हम दूसरे शब्दों में कहें तो कह सकते हैं कि किसी संख्या में यदि 2 का भाग देने पर शेष फल शुन्य प्राप्त होता है तो Condition सत्य होती है।

यदि इसे गणितीय रूप में लिखें तो x % 2 == 0 लिख सकते हैं। साथ ही उसी संख्या में 3 का भाग भी पूरा–पूरा जाना चाहिये यानी उसी संख्या में 3 का भाग देने पर भी शेषफल शून्य आना चाहिये इसे भी गणितीय रूप में x%3 = = 0 लिख सकते हैं। इस प्रकार दोनों Condition सत्य होने पर ही Statement Block का Execution होना चाहिये।

अतः इन दोनो Condition को Logical Operator AND (&&) द्वारा जोड कर if कोष्ठक में लिखना होगा। इस प्रकार इस Loop की वास्तविक Condition **x % 2 = = 0 && x % 3 = = 0** होगी। इस प्रकार हमें इस प्रोग्राम की सभी आधारभूत चीजें प्राप्त हो जाती हैं जो निम्नानुसार हैं–

- Loop के Variable का प्रारम्भिक मान जिससे Loop का Iteration शुरू होगा यानी 1
- Loop के Variable का अन्तिम मान जिसके बाद Loop का Iteration रूक जाएगा, यानी 100
- Loop के Variable की Step Size जिस क्रम में Loop के Variable का increment होगा।
- एक if Condition जिसमें ये check होगा कि 1 से 100 के बीच कौनसी संख्याएं हैं जिनमें 2 व 3 दोनों का भाग पूरा–पूरा जाता है।

Program

```
/*  While Loop Example          */
#include<stdio.h>
main()
{
    int x;
    clrscr();
    x = 1;

    while(x<=100)
    {
        if( x % 2 = = 0  & &  x % 3 = = 0 )
        printf(" %d \n", x );

        x++;
    }
    getch();
}
```

Program Declaration :– इस प्रोग्राम में int प्रकार का एक Variable x Loop चलाने के लिए लिया गया है। इस Variable के Declaration के बाद clrscr() Function द्वारा Output Screen को साफ किया जाता है। फिर x को प्रारम्भिक मान 1 Initialize किया गया है, क्योंकि while Loop में Loop चलाने वाले Variable को while Loop के बाहर ही प्रारम्भिक मान दिया जाता है।

अब Program Control while Loop को check करता है। यहां ये check होता है कि x का मान 100 से कम या बराबर है या नहीं। यहां Condition सत्य होती है, क्योंकि x का प्रारम्भिक मान 1 है, जो कि 100 से कम है। इसलिए Program Control while Loop के Statement Block में प्रवेश करता है। Statement Block में प्रवेश करते ही Program Control को एक if Statement मिलती है जो यह check करने के लिए है कि x के मान में 2 व 3 का भाग पूरा – पूरा जाता है या नहीं।

यहां x का मान 1 है, इसलिए if Condition असत्य हो जाती है क्योंकि 1 में ना तो 2 का भाग पूरा–पूरा जाता है ना ही 3 का। if Condition के असत्य होने से printf() Function का Execution नहीं होता और Program Control सीधे ही x का मान Increment करने वाले Statement पर पहुंच जाता है।

ध्यान दें कि यहां printf() Function का Execution क्यों नहीं हुआ। यहां printf() Function का Execution इसलिए नहीं हुआ क्योंकि if Condition असत्य हो गई है। आप सोचेंगे कि यदि if Condition असत्य होने से printf() Function का Execution नहीं हुआ तो फिर x के increment Statement का Execution कैसे हुआ ?

ऐसा इसलिए हुआ क्योंकि जब हमें किसी Control Statement के बाद केवल एक ही Statement का Execution करना होता है तब उसके Statement को Block में लिखने की जरूरत नहीं होती है। (यदि उस Statement को Block में लिख दिया जाए तो भी कोई फर्क नहीं पडता है।) क्योंकि Condition सत्य होने पर Program Control सीधे ही उसके आगे लिखे एक Statement का Execution कर देता है। यदि Statement एक से अधिक हों और हम Block का प्रयोग ना करें तो Program Control, Control Statement की Condition सत्य होने पर Control Statement के अगले एक Statement का Execution कर देता है।

शेष Statements को छोड देता है, लेकिन यदि Condition असत्य हो जाए तो Control Statement के बाद के एक Statement को छोड कर शेष Statements का Execution कर देता है। इसलिए Control Statement के बाद यदि एक से अधिक Statements का Execution करना हो तो हमें Block का प्रयोग करना जरूरी हो जाता है।

इस प्रोग्राम में Control Statement if के बाद केवल एक ही Statement लिखा गया है। यदि Condition सत्य होती है तो ये printf() Function Execute हो जाता है और यदि Condition असत्य हो जाती है तो ये printf() Function Execute नहीं होता।

अब प्रोग्राम Control को x++ Statement मिलती है। Program Control x का मान बढा कर अब दो कर देता है। वापस while Condition में x का मान check होता है कि x का मान 100 से कम या बराबर है या नहीं। x = 2 होने से वापस Condition सत्य हो जाती है और Program Control वापस Loop के Statement Block में पहुंचता है। यहां वापस if Condition check होती

है। यहां x = 2 होने से x में 2 का भाग तो पूरा – पूरा जाता है लेकिन 3 का भाग नहीं जाता इसलिए Condition पुनः असत्य हो जाती है और printf() Function Execute नहीं होता है।

इस प्रकार यही क्रम चलता रहता है। जब x का मान Increment हो कर 6 होता है तब if Condition सत्य हो जाती है क्योंकि तब x के मान 6 में 2 व 3 दोनों का भाग पूरा – पूरा जाता है, तब printf() Function Execute होता है और Output में 6 print हो जाता है।

यह क्रम तब तक चलता रहता है जब तक कि x का मान 100 से अधिक नही हो जाता। जैसे ही x का मान 101 होता है, while Loop की Condition असत्य हो जाती है और Program Control सीधे ही getch() Function पर पहुंच कर परिणाम को Output में Print कर देता है।

Exercise:

1 एक Program बनाईए जो User से उतने Input प्राप्त करे, जितने User चाहता है और Program के अन्त में ये Program User को बताए कि User ने कितने Positive मान Input किए हैं, कितने Negative मान Input किए हैं और कितने Zero Input किए हैं ?

2 एक Program बनाइए जिसमें User जो भी संख्या Input करता है, उस संख्या का Octal व Hexadecimal मान Output में Print हो।

3 एक Program बनाईए जिसमें User विभिन्न प्रकार के बहुत सारे मान तब तक Input कर सके जब तब करना चाहे। जब User अपनी इच्छानुसार मान Input कर ले, तब Program के अन्त में Program द्वारा ये बताया जाना चाहिए कि User ने Program के दौरान छोटे से छोटा व बडे से बडा कौनसा मान Input किया है।

4 **for** Loop व **while** Loop में क्या अन्तर है ? आप कैसे तय करेंगे कि किस परिस्थिति में किस Loop को Use करना उचित रहेगा ? क्या विभिन्न प्रकार के Loops को आपस में Exchange करके Use किया जा सकता है ?

5 for Loop व while Loop की Nesting Process को एक–एक उचित उदाहरण Program द्वारा समझाईए।

6 1 से n तक की संख्याओं को आपस में जोडने का Program बनाईए यानी निम्न Equation को Solve करने का Program बनाईए, जिसमें **n** User Input करता है।

$$x = 1 + 2 + 3 + 4 + 5 + 6 \ldots\ldots + n$$

7 एक ऐसा प्रोग्राम बनाओ जो Input की गई संख्या की Binary Value Output में Print करे।

Do…while Loop

यह "C" में प्रयोग होने वाला तीसरा व अन्तिम Loop है। इसमें भी अन्य Loop की तरह ही तीनों आधारभूत Statements की जरूरत होती है यानी Loop के Variable का प्रारम्भिक मान, अन्तिम मान व Step Size. इस Loop की विशेषता ये है कि इस में check होने वाली Condition Loop के अंत में check होती है।

यानी जब हमें ऐसी जरूरत हो कि प्रोग्राम में Loop के Statement या Statement Block का Execution कम से कम एक बार तो करना ही हो, तब हम इस Loop का प्रयोग करते हैं। इस Loop का syntax निम्नानुसार हैः

```
Variable Declaration;
Value Initialization;

do
{
    Statement Block;
    Step Size;
}while(Condition);

Statement 1;
```

इस Loop की शुरूआत **do** key word से होती है और Statement Block के बाद मंझला कोष्ठक बंद करने के बाद while Condition दी जाती है। साथ ही यही एक Loop है, जिसमें while के Condition कोष्ठक के बाद ; (Semi Colon) का प्रयोग किया जाता है। do के बाद कोई भी Colon या Semi Colon प्रयोग नहीं किया जाता है।

इस Loop में प्रोग्राम Control को जैसे ही ***do*** Key word मिलता है तो Program Control सीधे ही do के Statement Block में चला जाता है और उसमें लिखे Statements को Execute कर देता है। फिर Loop चलाने वाले Variable का Step Size Increase या Decrease प्रोग्राम के अनुसार करता है।

Program Control जब इस Block से बाहर आता है तब उसे while Condition प्राप्त होती है। यहां यदि Condition सत्य होती है तो Program Control do से वापस Statements का Execution करता है और यदि Condition असत्य होती है, तो Program Control Loop को वापस Iterate नहीं करता बल्कि सीधे ही Statement 1 पर चला जाता है।

break Statement

Loop के Iteration के समय Statement का दोहरान तब तक होता रहता है जब तक कि Loop की Condition सत्य होती है। लेकिन कई बार हमारे सामने ऐसी परिस्थिति आ जाती है कि किसी खास काम के लिए हमें Loop के कुछ Statements के Execution को छोड कर Loop के बीच में से ही बाहर आना होता है, उस समय हम break Statement का प्रयोग करते हैं।

continue Statement

इस Statement का प्रयोग तब किया जाता है जब हम किसी खास परिस्थिति में Loop के किसी दोहरान में Block के Statements को Execute करना नहीं चाहते है। ध्यान दें कि जहां break का प्रयोग Program Control को Loop से ही बाहर निकाल देता वहां **continue** का प्रयोग हमें Loop से बाहर नहीं निकालता बल्कि केवल उस Condition के सत्य होने पर मात्र हमें Loop के उस Iteration से बाहर निकालता है। नीचे **break** व **continue** Statement के प्रयोग से प्रोग्राम बनाया गया है जिससे इन्हे आसानी से समझा जा सकता है।

Program

```
/*  Use of Break and Condition Statement        */
    #include<stdio.h>
    main()
    {
        int a;
        clrscr();

        for(a=1;a<=20;a++)
        {
            if( a = = 10 )
                break;
            printf("\t %d", a);
        }
        printf("\n");
        for(a=1;a<=10;a++)
        {
            if(a == 10)
                continue;
            printf("\t %d", a);
        }
        getch();
    }
```

इस प्रोग्राम का Output ऊपर बताए अनुसार प्राप्त होता है। हम देखते हैं कि दोनों ही for Loop में if Condition एक समान प्रयोग की गई है लेकिन पहले Loop में जब a का मान Increase हो कर 10 होता है, तब if Condition सत्य हो जाती है और Program Control को break Statement मिलता है, जिससे Program, Control Loop का Execution वहीं पर छोड देता है और Loop आगे नहीं बढता।

जबकि दूसरे Loop में जब a का मान 10 होता है, तो Program Control Loop के इस दसवें Iteration को skip कर देता है यानी Loop के printf Statement को Execute नहीं होने देता, और Program Control को पुनः Loop के शुरूआत में अगले Iteration के लिए लेकर चला जाता है।

इस प्रकार ये Statement Loop के दसवें Iteration के मान 10 को Print नहीं करता लेकिन 11 से 20 तक के अंकों को print कर देता है। जबकि break Condition में Loop केवल 1 से 9 तक के ही अंकों को print करता है शेष को छोड देता है।

Output

```
1  2  3  4  5  6  7  8  9

1  2  3  4  5  6  7  8  9
11 12 13 14 15 16 17 18 19 20
```

चलिए, अब हम एक Program बनाते हैं, जिसमें User जितने Characters Input करता है, Program उन Input किए गए Characters की कुल संख्या Output में Display करता है।

Program

```c
#include <stdio.h>
#include <conio.h>
main()
{
    long numberOfCharacters = 0 ;
    printf(" Enter characters how much you want. ");
    while ( getchar() != '\n')
        ++numberOfCharacters;
    printf("\n You have entered %ld Characters", numberOfCharacters);
    getch();
}
```

इस Program में हमने **Long** Type का एक Variable ***numberOfCharacters*** लिया है और उसमें मान 0 Initialize किया है। इस Variable को Long प्रकार का इसलिए लिया है, क्योंकि User इस Variable में जितने चाहे, उतने Characters Input कर सकता है। साथ ही इसमें 0 इसलिए Initialize किया है, क्योंकि हम इस Variable को एक Counter की तरह Use कर रहे हैं और Counting की शुरूआत से पहले Counter का मान हमेंशा 0 होता है।

Computer की Memory में किसी भी समय विभिन्न प्रकार के मान Store हो सकते हैं। जब भी हम किसी Program को Run करते हैं, वह Program memory में कुछ Space लेता है और उसमें Store हो जाता है। उस Program में विभिन्न प्रकार की प्रक्रियाएं होती हैं और विभिन्न प्रकार की प्रक्रियाओं को पूरा करने के लिए Computer Memory में विभिन्न प्रकार के मानों को Hold करता रहता है।

जब Memory में Stored उस Program से हमारी जरूरत पूरी हो जाती है और हम Program को Terminate करते हैं, तब उस Program द्वारा की गई Processing के लिए Memory में ली गई Space में उस Program से सम्बंधित विभिन्न प्रकार के मान Program के Terminate होने के बाद भी Memory में Stored रहते हैं।

अब यदि हम कोई दूसरा Program Run करते हैं और वह Program भी Memory के उसी हिस्से में जाकर Load होता है, जहां पिछला Program Load था और हम हमारी जरूरत के अनुसार कोई Variable Declare करते हैं, तो उस पिछले Program के मान इस नए Program के Variable में भी Stored रहते हैं, जिसका हमारे Current Program के लिए कोई उपयोग नहीं होता है। इस प्रकार के मान को Computer की भाषा में **Garbage Value** कहा जाता है।

इसीलिए हमने हमारे इस Program में ***numberOfCharacters*** Variable में मान **0** Store किया है, ताकि यदि किसी पिछले Program के किसी Identifier का कोई मान इस Variable की Memory Location पर Stored हो, तो वह मान Clear हो जाए और इस Counter का मान 0 हो जाए।

जब Program Run होता है, तब एक Message Display होता है, जो User को Characters Input करने के लिए कहता है और Program निम्न Statement पर आकर Characters को प्राप्त करने लगता है:

```
while ( getchar() != '\n')
```

getchar() Function User से कुछ Characters प्राप्त करता है और उन्हें अपने Buffer में Store करता है तथा पहले Character के ASCII Code को '\n' Character Constant से Compare करता है।

यदि User ने Enter Key Press नहीं किया होता है, तो Comparison के बाद while Loop True होने की वजह से Computer *numberOfCharacters* Variable को Increment कर देता है और getchar() Function के Buffer से दूसरे Character को Read करता है।

यदि दूसरा Character भी '\n' Character Constant नहीं होता है, तो while Loop फिर से True होता है और *numberOfCharacters* Variable को फिर से Increment कर देता है। ये प्रक्रिया तब तक चलती रहती है, जब तक कि getchar() Function को अपने Buffer में '\n' Character Constant प्राप्त नहीं हो जाता है।

जब while Loop को ये Character Constant प्राप्त होता है, तब while Loop Terminate हो जाता है और एक printf() Statement *numberOfCharacters* के मान को Screen पर Display कर देता है, जो कि Input किए गए कुल Characters की संख्या को Represent करता है। इस तरह से User ने Keyboard से कितने Characters Input किए हैं, इसकी जानकारी Program द्वारा प्राप्त हो जाती है। यदि हम चाहें, तो इस Loop को निम्नानुसार Modify कर सकते हैं:

```
while ( getchar() != '0')
```

यदि हम पिछले Program में while Loop को इस तरह से Modify कर दें, तो getchar() Function Keyboard से प्राप्त Characters में से तब तक Characters की Counting करता है, जब तक कि उसे getchar() Function के Buffer में '0' प्राप्त नहीं हो जाता। हम '0' के स्थान पर किसी अन्य Character को भी Specify कर सकते हैं।

हम इस स्थान पर जिस Character को Specify करते हैं, Computer को getchar() Function के Buffer में जब वही Character प्राप्त होता है, getchar() Function Terminate हो जाता है। हम इस स्थान पर एक Special Constant मान **EOF** को भी Specify कर सकते है। ये एक ऐसा मान है, जो Computer को तब प्राप्त होता है, जब User Keyboard से **Ctrl+Z** Key Combination या Function Key **F6** को Press करता है। यदि हम इस Constant को पिछले while Loop में Use करना चाहें, तो निम्नानुसार Use कर सकते हैं:

```
while ( getchar() != EOF )
```

यदि हम इस Constant मान को पिछले Program के while Loop में Replace करके Program को Run करें, तो ये Program तब तक Run होता रहता है, जब तक कि उसे getchar() के Buffer में EOF (***End Of File***) का Signal प्राप्त नहीं हो जाता, जो कि **Ctrl+Z** Key Combination या Function Key **F6** द्वारा प्राप्त होता है।

चलिए, हम पिछले Program को ही थोडा Modify करते हैं और Characters की Counting के लिए while Loop के स्थान पर **for** Loop को Use करते हैं।

Program

```
#include <stdio.h>
#include <conio.h>
main()
{
    long numberOfCharacters ;
    printf(" Enter characters how much you want. ");

    for(numberOfCharacters=0; getchar()!=EOF; ++numberOfCharacters)
    ;

    printf("\n You have entered %ld Characters",
    numberOfCharacters);
    getch();
}
```

ये Program भी ठीक उसी तरह काम करता है, जिस तरह पिछला Program कर रहा है, लेकिन इस Program में getchar() Function तब तक Keyboard से Characters Read करके अपने Buffer में Store करता रहता है, जब तक कि User Keyboard से EOF का Signal Input नहीं करता है।

इस Program में हमने **for** Loop को थोडा अलग तरीके से Use किया है। इस Program में स्थिति ऐसी है कि Characters की Counting के लिए हमें Loop तो चलाना है, लेकिन for Loop की Body में Execute करने के लिए एक भी Statement की जरूरत नहीं है।

जब किसी for Loop की Body में एक भी Executable Code Statement लिखने की जरूरत नहीं होती है, तब हम for Loop को इस तरह से Use कर सकते हैं। इस Program के for Loop को हम निम्नानुसार Empty Body Statement के रूप में भी लिख सकते हैं:

```
for(numberOfCharacters=0; getchar()!=EOF; ++numberOfCharacters){}
```

यदि हम पिछले Program में ही थोडा सा Modification करें, तो हम एक ऐसा Program बना सकते हैं, जो Keyboard से Input किए गए कुल Characters की संख्या के साथ ये भी बताए कि User ने कुल कितनी Lines Input की है। इस समस्या के समाधान का Logic ये है कि किसी भी Line का अन्त तब होता है, जब Computer को **'\n'** Character Constant प्राप्त होता है।

इस स्थिति में जैसे ही Program को '\n' Character Constant मिलता है, एक Line Counter को Increment किया जा सकता है और Program के अन्त में उस Link Counter के मान को

Screen पर Display करवाया जा सकता है। Display होने वाला ये मान Input की गई कुल Lines की संख्या को Represent करता है।

Program

```
#include <stdio.h>
#include <conio.h>
main()
{
    long numberOfCharacters = 0, numberOfLines = 0, c;
    printf(" Enter characters how much you want. ");
    while((c = getchar()) != EOF)
    {
        ++numberOfCharacters;

        if(c == '\n')
            ++numberOfLines;
    }
    printf("\n You have entered %ld Characters", numberOfCharacters);
    printf("\n You have entered %ld Lines", numberOfLines);
    getch();
}
```

जब ये Program Run होता है और User Characters Input करता है, तब Input किए गए सभी Characters **getchar**() Function के Buffer में Store हो जाते हैं। फिर Buffer में Stored हर Character को Read किया जाता है और उसकी ASCII Value को **c** नाम के एक Variable में Store किया जाता है।

उसके बाद इस **c** में Stored ASCII Code को EOF Constant मान से Compare करवाया जाता है। यदि Variable c में Stored ASCII Code का मान EOF के बराबर नहीं है, तो while Condition True हो जाती है और *numberOfCharacters* नाम के Character Counter Variable के मान को Increment कर दिया जाता है। साथ ये भी Check करवाया जाता है, कि Variable **c** में '\n' Character Constant है या नहीं।

यदि इस Variable में New Line Character '\n' हो, तो if Condition True हो जाती है और *numberOfLines* नाम के Variable का मान Increment हो जाता है। जब getchar() Function के Buffer में EOF Signal मिलता है, तब while Loop Terminate हो जाता है और Screen पर Input किए गए कुल Characters की संख्या व कुल Lines की संख्या Display कर दी जाती है।

Exercise:

1. **while** व **do . . . while** Loop के अन्तर एक उचित Program द्वारा विस्तार से समझाईए?
2. **break** व **continue** Statements के बीच के अन्तर को एक उचित उदाहरण Program द्वारा समझाईए।
3. चक्रवृद्धि ब्याज ज्ञात करने का Program बनाइए, जिसमें **Principal, Time,** व **Rate** को User Keyboard से Input करता है और Program उस User को ***Principal, Time, Rate, Interest*** व ***Total Amount with Interest*** Display करता है।
4. एक Program बनाओ जो Input किए गए Characters, Words, Blank Spaces, Tab व New Lines की कुल संख्या को Screen पर Display करे।
5. एक Program बनाओ जो Input किए गए Characters की Line में से Extra Spaces को Remove करके Line को Screen पर Display करे। दो शब्दों के बीच में हमेंशा एक Space होता है। यदि दो शब्दों के बीच में एक से ज्यादा Space हों, तो उन्हें Extra Space कहते हैं।
6. एक Program बनाओ जिसमें Input किया गया हर शब्द Output में एक New Line में Display हो। जैसेः

Input :	I Line C Language	**Output**:
		I
		Like
		C
		Language

7. ऐसा प्रोग्राम बनाइये, जो Input की गई String के हरेक Character व उसकी ASCII Value को Output में Print करे।

ARRAYS

Arrays

इससे पहले कि हम Array को समझें, हम एक बार फिर समझने की कोशिश करते हैं कि Variable क्या होता है। Computer में विभिन्न प्रकार के मानों को Store करने के लिए Memory में कुछ जगह की आवश्यकता होती है। अलग–अलग प्रकार के मानों को Memory में Store करने के लिए अलग–अलग Size के Memory की आवश्यकता होती है।

जैसे यदि Integer प्रकार के मान को Memory में Store करना हो तो Program को दो Byte की आवश्यकता होती है जबकि एक Character प्रकार के मान को Memory में Store करने के लिए उसे केवल एक Byte की ही जरूरत होती है। मानलो कि हम एक ऐसी Memory Location चाहते हैं जहां Integer प्रकार के मान Store हो सकें। ये काम हम निम्न Statement Code लिख कर सकते हैं:

```
int total;
```

चलिए, समझने की कोशिश करते हैं कि Compiler इस Statement से क्या समझता है और क्या काम करता है। जब Compiler को **int** Keyword मिलता है, तो Compiler Memory में दो Bytes की Free Space खोजता है। Memory में जहां पर भी Compiler को दो Bytes की Space प्राप्त हो जाती है, Compiler उसे Reserve कर लेता है और उस दो Bytes के Memory Block का नाम **total** रख देता है। इस Statement के बाद यदि हम निम्न Statement लिखते हैं:

```
total = 100;
```

तो ये Statement Compiler को बताता है कि जिस Reserved Memory Block का नाम **total** है उस Memory Block पर मान 100 Store कर दें। यानी Compiler उस Memory Location पर 100 Store कर देता है जिसका नाम total है।

सारांश में कहें तो कह सकते हैं कि Variable किसी Reserved Memory Location का एक नाम होता है, जिसका प्रयोग करके Compiler विभिन्न प्रकार के मानों को किसी Memory Location पर Store करता है व विभिन्न प्रकार के मानों को प्राप्त करता है।

मानलो कि किसी Company में 10 Employee काम करते हैं और उन्हें Diwali का Bonus देना है। इस स्थिति में यदि हम सामान्य तरीके से Program बनाते हैं, तो हमें हर Employee की Salary को Store करने के लिए 10 Variables Declare करने होंगे और Calculate होने वाले Bonus को Store करने के लिए भी दस Variables Declare करने होंगे।

हम समझ सकते हैं कि अब हमें कम से कम 20 Variables के साथ प्रक्रिया करनी है। 20 Variables के साथ प्रक्रिया करना हालांकि ज्यादा कठिन नहीं है, लेकिन यदि यहां 10 Employees

की जगह यदि Company में 1000 Employees होते तो हमें 2000 Variables Declare करने पडते। 1000 Variables में हर Employee Salary Store करने के लिए और 1000 Variables में Calculate होने वाले Bonus को Store करने के लिए।

यदि ऐसी स्थिति हो जाए तो ये सामान्य सी समस्या भी काफी जटिल हो जाएगी। Programmer को 2000 Variables को Manage करना होगा जो कि एक बहुत ही जटिल काम है। इस प्रकार की समस्या के समाधान के लिए "C" में Array का प्रयोग किया जा सकता है।

जब हमें एक ही Data Type के ढेर सारे Data के साथ प्रक्रिया करनी होती है, तब हम उन ढेर सारे Data को ठीक से Manage करने के लिए Array का प्रयोग कर सकते हैं। "C" Language में Array एक ऐसा Variable होता है, जो एक ही प्रकार के बहुत सारे Data को Memory में Store करके रख सकता है। यानी Array Create करके वास्तव में हम एक ही समय में एक ही Data Type के बहुत से Variables Create करते हैं। हम एक उदाहरण से इसी बात को समझने की कोशिश करते हैं।

एक कक्षा में अमित नाम के कई विधार्थी हो सकते हैं। इसलिए हर विधार्थी को पहचानने के लिए हर विधार्थी के नाम के साथ उसकी जाति का उल्लेख किया जाता है या फिर हर विधार्थी के नाम के साथ कोई Extension लगा कर उसकी अलग से पहचान बना दी जाती है। जैसे अमित1, अमित2 अमित3 आदि। यहां हम देख सकते हैं कि नाम तो एक ही है, लेकिन हर नाम की पहचान एक अलग विधार्थी के रूप में होती है।

यही प्रक्रिया हम Array के साथ प्रयोग करते हैं, जिसमें Array का नाम तो एक ही होता है, लेकिन Variables कई होते हैं। जब हम Variable को Array के रूप में Declare करते हैं, तब उस Array Variable के नाम के साथ एक अंक का प्रयोग "C" Compiler स्वयं ही कर लेता है। इस अंक को **Index Number** कहते हैं।

एक Array में हम समान प्रकार के एक से अधिक मानों को Store कर सकते हैं। किसी Array में जितने भी Data Store होते हैं, वे Data Array के **Element** कहलाते हैं और इन Elements की पहचान एक Logical Address द्वारा होती है, जिसे **Index Number** कहते हैं।

किसी भी Array में Store किया जाने वाला पहला मान हमेंशा Index Number 0 पर Store होता है और फिर क्रम से उसके आगे के Memory Locations पर मान Store होते हैं। Array का Index Number 0 किसी भी Array का **Base Address** होता है। एक Array में Data Store करने से पहले Array को Declare करना पडता है। Array को Declare करने के लिए निम्न Format का प्रयोग होता है:

```
Data_Type Array_Name [Array_Size]
```

Data_Type	जिस प्रकार के Data हमें इस Array Variable में Store करना हैं।
Array_Name	"C" Compiler जिस नाम से Array को पहचानता है, वह नाम।
[Array_Size]	इस कोष्ठक में Array की Size Define की जाती है जो "C" Compiler को यह बताता है कि Array में कितने Element Store होंगे या कितने Data Store होंगे। यह कोष्ठक बनाना जरूरी होता है।

जब हम कोई Array Declare करते हैं, तब हमें Array की Size भी Define करनी होती है। ये Size ही तय करता है कि हम Array में कितने Data Store करना चाहते हैं। मानलो कि हम 10 Employees के Bonus को Store करने के लिए Memory मे Space Create करना है, तो हमें Array का Declaration निम्नानुसार करना पडता है–

```
int Bonus[10] ;
```

ये Statement Memory में 20 Bytes की जगह Reserve करता है और हर दो Bytes को एक Index Number Assign कर देता है। यानी Memory में निम्नानुसार 10 Variables Create होते हैं:

```
Bonus[0]
Bonus[1]
Bonus[2]
Bonus[3]
Bonus[4]
Bonus[5]
Bonus[6]
Bonus[7]
Bonus[8]
Bonus[9]
```

सारांश में कहें तो Array एक ऐसा तरीका है, जिसमें हम एक ही समय में एक ही प्रकार के Data Type के कई मान Store करने के लिए, एक से अधिक Variables Create करते हैं। कोई Array कितने Variables Create करेगा यानी किसी Array में कितने मान Store किए जा सकते हैं, ये बात Array की Define की गई Size पर निर्भर करता है। हम यहां देख सकते हैं कि Create होने वाले सभी Variables का नाम तो समान है, लेकिन सभी को उनके Index Number के आधार पर अलग–अलग माना जा सकता है।

Array हमेंशा Memory में एक Continuous Memory Location पर Store होता है। यानी यदि किसी Integer प्रकार के Array की Size 5 है तो वह Memory में उसी स्थान पर Space Reserve करेगा जहां पर उसे 12 Bytes पूरे व Continuously प्राप्त होंगे। जैसे मानलो कि हमें 5

Student के Roll Number, Memory में Store करने हैं, तो हम निम्नानुसार एक Array, जिसकी Size 5 element की हो, Declare करते हैं:

```
int roll_num[5]
```

ये Declaration Memory में 5 Variable के लिए 5 Memory Locations बनाता है, जिसमें हरेक की Memory Space 2 Byte की होती है, क्योंकि हमने **int** प्रकार का Array Declare किया है। इस प्रकार एक ही नाम roll_num के 5 Memory Locations क्रमशः roll_num [0], roll_num [1], roll_num[2], roll_num[3], roll_num[4] बन जाते है, जिसमें हरेक में **int** प्रकार का एक Roll Number Store हो सकता है। इसे हम निम्नानुसार प्रदर्शित कर सकते हैं:

roll_num[0]	roll_num[1]	roll_num[2]	roll_num[3]	roll_num[4]
100	101	102	103	104

इस प्रकार पहला Roll No. Array के प्रथम Logical Address Index No 0 पर Store होगा। दूसरा Roll No. Array के Logical Address Index No. 1 पर Store होगा और यह क्रम इसी प्रकार से निम्नानुसार चलता रहेगा:

Array Name	Index Number
roll_number	[0]
roll_number	[1]
roll_number	[2]
roll_number	[3]
roll_number	[4]
roll_number	[5]

माना कि क्रम से 5 विधार्थीयों के Roll Number 100, 101,102, 103 व 104 हैं, तो ये Memory में निम्नानुसार Store होंगे:

roll_num[0]	roll_num[1]	roll_num[2]	roll_num[3]	roll_num[4]
100	101	102	103	104

इसे हम निम्नानुसार भी लिख सकते हैं:

Array Name	Index Number
roll_number[0]	100
roll_number[1]	101
roll_number[2]	102

roll_number[3]	103
roll_number[4]	104
roll_number[5]	105

Linear Arrays

Array एक ऐसा Data Structure होता है, जिसमें एक ही Data Type के **n** Data Items एक List के रूप में Store हो सकते हैं, जबकि **n** Array की Size को Define करता है। यदि किसी Array की Size n हो व n का मान 10 हो तो उस Array में हम केवल दस Data Items Store करके रख सकते हैं।

Array के हर Item को उसके Index Number से Access किया जाता है। किसी Array का प्रथम Item हमेंशा Index Number 0 पर Store होता है और Array का अन्तिम Item हमेंशा Index Number n-1 पर Store होता है। किसी Array के **Index Number 0** को Array का **Lower Bound** और **Index Number n-1** को Array का **Upper Bound** कहते हैं। किसी Array की Length या Size को हम निम्न सूत्र द्वारा प्राप्त कर सकते हैं:

```
SIZE = UB + 1
```

जहां

```
UB = Upper Bound
LB = Lower Bound
```

हम किसी भी Index Number को हमेंशा Bracket के बीच लिखते हैं। जैसे यदि हमें किसी Array के Index Number 4 के Data Item को Access करना हो तो हम Array[4] लिखते हैं।

Array भी तीन तरह के होते हैं:

1 **One - Dimensional Array**
2 **Two - Dimensional Array**
3 **Multi - Dimensional Array**

ऊपर जो प्रारूप समझाया गया है, वह One Dimensional Array का प्रारूप है। यदि किसी Array को Declare करते समय एक के बजाय दो Brackets में Array की Size को Define कर दिया जाए, तो यह एक **Two Dimensional Array** बन जाता है, और यदि Array की Size को दो से अधिक Brackets में दे दिया जाए तो यह **Multi Dimensional Array** Declare हो जाता है।

जब Array को Declare किया जाता है, तब वह Memory में वहीं जाकर Store होता है, जहां उसमें Define की गई पूरी जगह उसे क्रम से प्राप्त हो जाए। जब हमें कोई Table या सारणी Memory में Store करनी होती है, तब हम Two Dimensional Array का प्रयोग करते हैं। इसमें एक Dimension **Row** की व दूसरी Dimension **Column** की होती है और इसे हम अग्रानुार प्रदर्शित कर सकते हैं। माना तीन Student के **Hindi, English** व **Science** में क्रमशः निम्नानुसार मान प्राप्त हुए हैं:

Student/Sub	Hindi	English	Science
Student1	50	56	65
Student2	55	44	56
Student3	45	54	75

यदि हम इसे किसी Array में Store करना चाहें तो उस Array में ये मान निम्नानुसार विभिन्न Memory Locations पर Store होंगे व इन्हे निम्नानुसार Index Numbers प्राप्त होंगे जिनसे इनकी पहचान होगी–

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

इसमें प्रथम विधार्थी के Hindi के अंक 50 Memory Location के Index Number 0,0 पर Store होंगे। English के अंक 0,1 Location व Science के अंक 0,2 Location पर Store होंगे। इसी प्रकार दूसरे विधार्थी के Hindi के अंक Memory में 1,0 Index Number के Location पर, English के अंक 1,1 Location पर व Science के अंक 1,2 Location पर Store होंगे। इसे गणितीय रूप में हम निम्नानुसार लिख सकते हैं–

```
student[0,0]    50
student[0,1]    46
student[0,2]    65
student[1,0]    55
student[1,1]    44
student[1,2]    56
student[2,0]    45
student[2,1]    54
student[2,2]    75
```

हम इस Table को Row व Column में प्रदर्शित कर सकते हैं, लेकिन Memory में ये एक लगातार क्रम में Store होते हैं और इनकी पहचान इनके Index Number के आधार पर ही होती है। इस प्रकार यदि Index Number बदल जाता है, तो Memory Location भी बदल जाती है और

Store होने वाला मान कभी भी किसी एक Memory Location पर Store नहीं होता। यदि Two Dimensional Array को Declare करना हो तो हम निम्न Format का प्रयोग करते हैं –

```
Data_Type Array_Name [Row Size][Column Size]
```

इस प्रकार उपरोक्त उदाहरण का Array निम्नानुसार Declare होगा –

```
int student [3][3]
```

तीन Student के कारण Row की संख्या तीन है व तीन विषय के कारण Column की संख्या तीन है। इनका क्रम आपस में बदला भी जा सकता है, यानी Row की जगह Column व Column की जगह Row की Size भी दी जा सकती है। Two Dimensional Array में हम एक Array में Row व Column की संख्या के गुणनफल के बराबर संख्या में Elements Store कर सकते हैं। Multi Dimensional Array को भी इसी प्रकार से Declare किया जा सकता हैं और उसमें Store होने वाले Elements की संख्या उनकी Define की गई कुल Brackets की Size के गुणनफल के बराबर होती है। माना हम एक Multi Dimensional Array निम्नानुसार Declare करते हैं:

```
int x [2][2][3];
```

तो यह Statement Memory में कुल **2 X 2 X 3 = 12** Element Store कर सकेगा।

इस प्रकार हम Array को Use कर सकते हैं और एक Array को एक ही प्रकार के बहुत सारे Data को एक ही Variable में Store करने के लिए प्रयोग किया जा सकता है। Array एक User Defined Data Type है। हम विभिन्न प्रकार की सारी Arithmetical, Relational, Logical आदि क्रियाएं Array के साथ कर सकते हैं। जैसे उपर के एक उदाहरण में students के तीन विषयों में प्राप्त अंकों को हमें जोडना हो और हर विधार्थी के कुल अंक ज्ञात करने हों, तो हम निम्न प्रकार से उन्हें जोड सकते हैं:

Student 1 के कुल अंक

```
tot = student1[0,0] + student1[0,1] + student1[0,2];
```

इस Statement से प्रथम विधार्थी के अंक जो कि क्रमशः Index Number [0,0] , [0,1], [0,2] पर स्थित हैं, वे जुड कर Variable ***tot*** में प्राप्त हो जाते हैं।

ध्यान दें कि किसी भी Array में Index Number यह बताता है कि हमने जो मान किसी Array में दिया है, वह मान Memory में किसी Array की किस Location पर स्थित है। यह एक Address होता है ना कि कोई मान।

किसी भी Array में स्थित मान को हम केवल उसके Index Number के द्वारा ही Access कर सकते हैं। Index Number एक Address होता है, जो किसी Element की मेमोरी में स्थिति बताता है। इसीलिए यहां पर प्रथम विधार्थी के प्राप्तांकों को जोडने के लिए उनके Index Number के द्वारा उनका मान प्राप्त किया गया है ना कि सीधे ही प्राप्तांकों को जोड दिया गया है।

एक Array को जब हम Declare करते हैं तब वास्तव में हम एक ही बार में ढेर सारे Variables Declare कर रहे होते हैं। जैसे हम किसी Variable में पांच विधार्थीयों के Roll Number Input करना चाहते हैं। इस काम के लिए हम साधारण तौर पर पांच Variable निम्नानुसार Declare कर सकते हैं:

```
int roll_number0,
int roll_number1,
int roll_number2,
int roll_number3,
int roll_number4;
```

यही काम हम एक बार में एक Array द्वारा भी कर सकते हैं, यानीः

```
int roll_number[5]
```

यदि हम इस Array को एक दूसरे तरीके से लिखें, तो इसे निम्नानुसार भी लिख सकते हैं:

```
roll_number[0]
roll_number[1]
roll_number[2]
roll_number[3]
roll_number[4]
roll_number[5]
```

यदि दोनों प्रकार से Declare किये गए Variables की तुलना करें तो हम देखते हैं कि दोनों ही तरह से Variables समान ही Declare हुए हैं। यानी

Int	Array
roll_number0	roll_number[0]
roll_number1	roll_number[1]
roll_number2	roll_number[2]
roll_number3	roll_number[3]
roll_number4	roll_number[4]

यहां Array वही काम कर रहा है जो हम पांच Variables Declare कर के करने वाले हैं। इस प्रकार हम कह सकते हैं, कि "C" में Array एक ऐसी व्यवस्था है, जो हमें एक ही समय में एक ही प्रकार के बहुत सारे Variables Declare करने की सुविधा प्रदान करता है। यानी हम Declare तो एक ही Variable करते हैं लेकिन Index Number के कारण एक ही Variable की उतनी ही प्रतिलिपी बन जाती है जितनी हमने Array की Size दी होती है, और हर Variable के साथ स्वंय ही एक Index Number "C" Compiler लगा देता है, जो किसी भी प्रतिलिपी का Memory में Address उपलब्ध करवाता है या किसी भी Array Variable को Access करने का तरीका उपलब्ध करवाता है, जिससे हम उस Variable को Access कर सकते हैं।

Value Initialization

जिस प्रकार हम किसी भी अन्य Variable को प्रारम्भिक मान प्रदान कर सकते हैं, वैसे ही हम Array को भी प्रारम्भिक मान दे सकते हैं। **One Dimensional Array** को हम निम्नानुसार मान प्रदान कर सकते हैं:

```
static Data_Type   Array_Name[Size] = { List of Values };
```

```
int b[4] = { 12,22,22,1};
```

इस Array में एक **int** प्रकार का Variable b है, जिसका आकार 4 है। यानी यह Variable Memory में लगातार int प्रकार के चार मान Store हो सके ऐसी Location पर Store होगा और हर Location पर Store Elements का मान निम्नानुसार होगा:

```
b[0]    =   12
b[1]    =   22
b[2]    =   22
b[3]    =   1
```

यदि हम Array के कुछ मान Initialize करें व कुछ छोड दें तो शेष के मान स्वयं ही 0 Initialize हो जाते हैं लेकिन ये तभी होता है जब Array को **static** Storage Class में Declare किया गया हो। जैसे:

```
int b[4] = {1};
```

यह Memory में प्रथम Element को 1 Initialize करेगा शेष को 0 Initialize कर देगा। यानी:

```
b[0]    =   1
b[1]    =   0
b[2]    =   0
```

```
b[3]    =    0
```

इसी प्रकार Two Dimensional व Multi Dimensional Array को भी हम Initialize कर सकते हैं। जैसेः

```
int b[2][3] = {1, 2, 3, 4, 5, 6 };
```

इस Statement से प्रथम Row के तीन Column का मान 1, 2 व 3 हो जाएगा व दूसरे Row के तीनों Column का मान क्रमशः 4, 5, व 6 हो जाएगा। इसे अन्य तरीके से भी Initialize कर सकते हैं।

```
static int b[2][3] =  {1, 2, 3},{ 4, 5, 6 };          OR
static int b[2][3] = {{1, 2, 3}{ 4, 5, 6} };
```

यदि हम कहीं पर मान Assign ना करें तो वहां पर Automatically 0 Assign हो जाता है। जैसेः

```
static int b[2][3] =  {1, 2},{ 4 };                   OR
static int b[2][3] = {{1, 2}{ 4 } };
```

इन दोनों उदाहरणों में प्रथम Row के तीसरे Column व दूसरी Row के दूसरे व तीसरे Column का मान Assign नहीं किया है, इसलिए इनका मान स्वयं ही 0 Initialize हो जाएगा। जब हम चाहते हों कि किसी Array के सभी मान 0 हों तो हम इसे निम्नानुसार भी Assign कर सकते हैं।

```
static int b[2][3] =  {0},{0};              OR
static int b[2][3] = {{0}{0}};
```

इस प्रकार सभी Array के सभी Elements का मान 0 हो जाता है। यदि हमें Multi Dimensional Array को मान प्रदान करना हो तो भी यही तरीके अपनाते हैं।

किसी Array का Declaration करते समय हम Array की Size को किसी Constant Expression का प्रयोग करके भी Specify कर सकते हैं। जैसे

```
#define LEAP 1  /* in leap years */
int days[31+28+LEAP+31+30+31+30+31+31+30+31+30+31];
```

जब हम इस तरह से Array Declaration Statement लिखते हैं, तो इस Array की कुल Size **366** हो जाती है।

Example :

अब हम एक सामान्य से प्रोग्राम द्वारा Array का प्रयोग करना सीखते हैं। हम एक ऐसा प्रोग्राम बनाते हैं, जिसमें 1 से 10 तक की संख्या को एक Array में Store करना है और Array से उसी मान को प्राप्त करके पुनः Screen पर Print करना है।

प्रोग्राम विश्लेषण

हम जानते हैं कि Array में सभी मान Index Number के आधार पर विभिन्न Locations पर Store होते हैं, यानी यदि हम int num[10]; करते हैं तो ये Statement प्रोग्राम में निम्नानुसार 10 मान विभिन्न Locations पर Store करेगाः

```
num[0]
num[1]
num[2]
num[3]
num[4]
num[5]
num[6]
num[7]
num[8]
num[9]
```

हम देखते हैं कि ये Index Number क्रम से एक एक बढ रहे हैं। यदि हम एक Loop i चला कर इन Index Numbers को क्रम से बदलते रहें यानी num[i] कर दें और हर Index Number के बदलते ही उसमें मान Input कर दें, यानी जब i का मान 0 हो, तब num[0] Location प्राप्त होगी, जहां हम मान Input कर दें।

दूसरे Iteration में i का मान 1 हो जाए यानी num[1] हो तब हम दूसरा मान Input करें। इस प्रकार Loop को दस बार चला कर हम दस मान Input करें तो Array के सभी Locations पर मान को Input किया जा सकता है।

इसी तरह वापस Loop चला कर इन्ही Locations से पुनः मानों को प्राप्त भी किया जा सकता है। जब हम किसी Data Structure के विभिन्न Locations पर जाकर Data Structure के विभिन्न Locations के साथ प्रक्रिया करते हैं, तो इस प्रक्रिया को Traversing करना कहते हैं। यहां हम एक Array की Traversing कर रहे हैं। Array की Traversing करने के लिए हमें जो Steps Use करने पडते हैं, उसका Algorithm हम निम्नानुसार लिख सकते हैंः

```
START
DECLARE Array[size], I, UB, LB
SET I = LB                                  [Set Counter]
REPEATE FOR I = LB TO UB STEP SIZE 1
PROCESS Array[I]                            [Process Data Item]
```

```
6   SET I = I + 1                              [Increase Counter]
7   END
```

Program

```
#include<stdio.h>
main()
{
    int i, num[10];
    clrscr();

    //Traversing the ARRAY For Input 10 Array Elements
    for(i=0; i<10; i ++)
    {
        printf("\n Enter %d Element Of Array : ", i);
        scanf("%d", &num[i]);
    }

    //Traversing the ARRAY For Displaying Entered Elements
    for(i=0; i<10; i ++)
    {
        printf("\n %d Element Of Array:", num[i]);
    }
    getch();
}
```

इस प्रोग्राम में Loop चलाने के लिए एक **int** प्रकार का Variable i व num नाम का, int प्रकार का एक Array लिया है, जिसकी Size 10 Element की है। Data Input करने के लिए for Loop चलाया गया है। जब प्रथम बार Loop चलता है, तब प्रथम मान Input करने के लिए Message आता है। i का मान 0 होता है, जिससे num[i] Statement के कारण Input किया गया प्रथम मान num[0] Location पर Store हो जाता है। यहां मान Input करते ही Loop पुनः Execute होता है और हमसे दूसरा मान मांगता है। अब i का मान बढ कर 1 हो चुका होता है, इसलिए Input किया जाने वाला मान num[1] Location पर Store हो जाता है।

इस प्रकार क्रम से यह मान num[9] तक Input होता है। जैसाकि हमने पहले कहा, हम Array के Address द्वारा उसके Elements के साथ Arithmetical व विभिन्न प्रकार की Logical या Relational प्रक्रियाएं कर सकते है। हम ये काम एक उदाहरण द्वारा करते हैं। इस उदाहरण में एक Array में दस संख्याएं Input करनी है और हर संख्या का वर्ग उसी के समान Element Address पर एक दूसरे Array में Store करना है।

Program

```
#include<stdio.h>
main()
{
    int num[10], square[10], i;
    clrscr();

    // Inputting Array Elements
    for(i=0; i<10; i++)
    {
    printf ("Enter %d Element", i);
    scanf("%d", &num[i]);
    }

    /* Calculating Square of Every Element And
    Placing that in square Array */

    for(i=0; i<10; i++)
    {
        square[i] = num[i] * num[i];
    }

    //Displaying Both Array Elements
    for(i=0; i<10; i++)
    {
        printf("\n Square of %d is %d", num[i], square[i]);
    }
    getch();
}
```

इस प्रोग्राम में हमने दो Array लिए हैं। पहला Array num[i] प्रथम for Loop के दौरान मानों को Input करता है। दूसरा for Loop num[i] में Store मानों का वर्ग ज्ञात करते हुए दूसरे Array square[i] में रखता है और तीसरे Loop द्वारा दोनों Array में Store मानों को Output में Print कर दिया गया है।

हमने अभी Array को int प्रकार के Data Type के साथ प्रयोग किया। हम Array का प्रयोग विभिन्न प्रकार के सभी Data Types के साथ कर सकते हैं। यानी Array में Double प्रकार का मान Store करने के लिए Double Keyword का प्रयोग किया जाता है। यदि Array में Float प्रकार का मान Store करना हो, तो Array को Float प्रकार का Declare करते हैं। यदि हम

Array में String Store करना चाहें तो यह काम हम One-Dimensional Array का प्रयोग करके कर सकते हैं।

String के बारे में हम पहले ही बता चुके हैं, कि जब हमें प्रोग्राम में कोई String Store करनी होती है, तब हम उस Variable को One Dimensional Array के रूप में Declare करते हैं। क्योंकि एक Variable एक समय में केवल एक ही मान को Store करके रख सकता है। इसलिए जब हमें एक से अधिक मानों को एक समय में एक ही Variable में Store करके रखना होता है, तब हम Array का प्रयोग करते हैं। यही बात किसी Strings को किसी Variable में Store करने के सम्बंध में लागू होती है। एक String, Characters का एक पूरा समूह होता है। इसलिए एक String को किसी Variable मे Store करने के लिए उसे Array बनाना जरूरी होता है। ये Array हमेंशा char प्रकार के Data Type का होता है और इन्हें निम्नानुसार Declare करते हैं:

```
char Array Name[Size]
```

ध्यान दें कि एक char प्रकार के Data Type के Array का अंत हमेंशा NULL Character से होता है और यह NULL Character "C" Compiler स्वयं ही हर char प्रकार के Array के अंत में लगा देता है, इसलिए हमें जितने Characters एक Array में Store करने हों, Array की Size हमेंशा उससे एक अधिक लेनी चाहिये।

2-D Array

किसी 2-D Array में भी हम उसी प्रकार से Traversing कर सकते हैं जिस प्रकार से किसी 1-D Array में करते हैं। हालांकि Memory में सभी Data एक 1-D Array के रूप में ही Store होते हैं लेकिन किसी 2-D Array के Data को Logically हम निम्न Format में प्रदर्शित कर सकते हैं:

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

इस Array में तीन ROW व तीन ही COLUMNS हैं। DOS में Monitor की Screen भी Rows व Columns में विभाजित रहती है। यानी Screen पर कुल 25 Rows व 80 Columns होते हैं। हम जब भी कोई Matter Screen पर लिखते हैं तो सबसे पहले पहली Row में Characters Write होते हैं। जब पहली Row में 80 Characters Write हो जाते हैं तब वापस दूसरी Row में 80 Columns में Characters Write होते हैं। यानी हमारे Monitor की Screen भी एक 2-D Array या एक Table की तरह ही होती है।

जिस तरह Screen पर Characters Write होते हैं उसी तरह किसी 2-D Array में भी Characters Insert होते हैं। यानी यदि हमें किसी 2-D Array में क्रम से Data Input करने हों तो हमें सबसे पहले पहली Row के सभी Columns में Data Write करना होगा। जब पहली Row के सभी Columns में Data Write हो जाएगा तब हमें वापस दूसरी Row के सभी Columns में Data को Write करना होगा।

2-D Array के चित्र में हम देख सकते हैं कि पहली Row में Row का मान जब 0 होता है तब तीनों Columns का मान क्रम से 0, 1 व 2 होता है। इसी तरह दूसरी Row में Row का मान 1 होता है तब Columns का मान वापस 1, 2 व 3 होता है।

यानी यदि हम किसी 2-D Array में क्रम से Data Input करना चाहें तो हमें दो Loop चलाने होते हैं: पहला Loop Row के लिए व दूसरा Loop Column के लिए और Loop भी इस तरह से चलाने होते हैं कि जब तक पहली Row के सभी Columns में Data Feed ना हो जाएं तब तक दूसरी Row में Data Feed नहीं होना चाहिए। यानी हमें Nested Loop चलाना होता है।

Nested Loop में जब Outer Loop का मान 0 होता है तब Inner Loop का मान क्रम से 1, 2 व 3 होता है। फिर वापस Outer Loop का मान जब 1 होता है तब Inner Loop का मान 1, 2 व 3 होता है। इसी तरह हमें आगे भी Loop चलाने पडते हैं। इन सभी परिस्थितियों को ध्यान में रखते हुए तो किसी 2-D Array की Traversing करने का Algorithm हम निम्नानुसार लिख सकते हैं:

Algorithm

```
Here LARRAY is a Linear Array, LB is a LOWER BOUND of both the Loops
and UB is a UPPER BOUND of both the Loops.

1  START
2  REPEATE FOR Row = LB TO UB        [ Outer Loop ]
3  REPEATE FOR Columns = LB TO UB    [ Inner Loop ]
4  PROCESS LARRAY[Row][Columns]      [ Process Data Item ]
      [ End of Inner Loop ]
      [ End of Outer Loop ]
5  END
```

जिस तरह से दो Loop चला कर हम 2-D Array की Traversing कर सकते हैं उसी तरह यदि हमें किसी 3-D Array की Traversing करनी हो तो हमें तीन Nested Loop चलाने पडते हैं। किसी 3-D Array की Traversing करने का Algorithm निम्नानुसार लिखा जा सकता है–

Algorithm

```
Here LARRAY is a Linear Array, LB is a LOWER BOUND of All the Loops
and UB is a
```

```
UPPER BOUND of All the Loops.

1  START
2  REPEATE FOR Row = LB TO UB                    [ Outer Loop ]
3  REPEATE FOR Columns = LB TO UB                [ Inner Loop ]
4  REPEATE FOR InnerColumn = LB TO UB            [ Inner Loop ]
5  PROCESS LARRAY[Row][Columns][InnerColumn]     [ Process Data Item ]
   [ End of Inner Loop ]
   [ End of Inner Loop ]
   [ End of Outer Loop ]
6  END
```

Initializing Value of a Character Array (String)

हम एक char प्रकार के Array को भी प्रारम्भिक मान प्रदान कर सकते हैं। Array में Character Assign करने के दो तरीके हैं। पहले तरीके में Array के हर Character को Single Quote में लिखा जाता है। इसमें अन्तिम character को NULL करना जरूरी होता है। दूसरे तरीके में Array को Assign किये जाने वाले सभी Characters की पूरी String को Double Quote में एक साथ लिखा जाता है। इसमें String के अंत में NULL Character लिखना जरूरी नहीं होता है। देखे निम्न उदाहरणः

```
char name[6] = {'R', 'a', 'h', 'u', 'l', '\0'};      OR      char name[6] = {"Rahul "};
```

इस प्रकार से ये दोनों ही Statement सही हैं। हम इन में से किसी भी प्रकार के Statement को Use कर सकते हैं। ध्यान दें कि यदि Array की Size, Input किये गए String के बाद NULL Character के लिए नहीं बचता है तो भी "C" Compiler Array के अंत में NULL Character लगा देता है और हमारा Input किया गया अन्तिम Character हट जाता है।

जैसे ऊपर के ही उदाहरण को देखें। यदि हम इस Array में Size 6 के बजाय 5 कर दें तो Array में केवल Rahu ही Store होगा, क्योंकि Rahul के अन्तिम Character l के स्थान पर NULL Character प्रतिस्थापित हो जाएगा। एक अन्य तरीका भी है जिसमें हम Array की Size Define नहीं करते हैं। "C" Compiler स्वयं ही Size ले लेता है। इसी उदाहरण को वापस देखेंः

```
char name[ ] = {'R' 'a' 'h' 'u' 'l' '\0'};      OR  char name[ ] = {" Rahul "};
```

यदि हम इस प्रकार से Array के मान Initialize करें, तो "C" Compiler स्वयं ही Array की Size 6 मान लेता है। लेकिन इसका प्रयोग केवल तभी किया जा सकता है, जब हमें तुरन्त मान Initialize कर देना हो। यदि हमें String Run Time में Input करना हों, तो यह Statement काम नहीं करता है। अब हम KULDEEP MISHRA String को Program के Run Time में Array में Input करते हैं।

हमने पहले भी बताया था कि **scanf**() Function कुछ Special Characters जैसे कि Blank Space, Carriage Return, Form Feed, New Line, Tab Key के मिलते ही Terminate हो जाता है, इसलिए यदि हमें **scanf**() Function द्वारा String को Array में Store करना हो, तो हमें Loop का प्रयोग करना होगा।

क्योंकि हम जानते हैं कि Array में मान Index Number के आधार पर Input होते हैं। Loop का प्रयोग करने से Array में ये विशेष Characters भी एक सामान्य Character की तरह ही Input होंगे। इस प्रोग्राम में हमें एक–एक Characters को Loop की सहायता से Input करना है और विभिन्न Index Numbers के अनुसार विभिन्न Locations पर Input किये गए Characters को Store करना है। फिर जिस Array में विभिन्न Locations पर ये Characters Store हुए हैं, उस Array को Output में Print करने पर Input किया गया String ज्यों का त्यों हमें प्राप्त हो जाता है। इस समस्या के समाधान के लिए Loop को तब तक चलाया जाना चाहिये जब तक कि New Line ना मिल जाए। New Line तब प्राप्त होती है, जब हम Enter Key Press करते हैं।

Program

```
#include<stdio.h>

main()
{
    int x;
    char name[20], chara;
    clrscr();

    //Inputting String
    printf("Enter String and Press Enter");
    fflush(stdin);

    for(x=0; x!='\n'; x++)
    {
    scanf("%c", &chara);
    name[x] = chara;
    }

    //Printing Inputted Array Elements
    printf("\n %s", name);
    getch();
}
```

जब Program Execute होता है, तब scanf() Function द्वारा chara में प्रथम Character Input होता है। यहां x का मान 0 होने से Array के Index Number का मान भी 0 होता है, जिससे Input होने वाला प्रथम Character Array की name[0] Location पर Store हो जाता है। दूसरे Iteration में x = 1 हो जाता है और Input होने वाला अक्षर name[1] Location पर Store हो जाता है।

इस प्रकार ये क्रम तब तक चलता रहता है, जब तक कि हम Enter Press नहीं करते। जैसे ही हम Enter Press करते हैं, For Loop Terminate हो जाता है। name नाम के Array में Input किये गए सभी Characters Store रहते हैं। इसमें Store सभी Characters के समूह के कारण ये एक String हो जाता है इसलिए String को Print करने के लिए %s Control string का प्रयोग किया गया है।

इसी प्रकार से हम दो Strings को जोड सकते हैं और दो strings की तुलना कर सकते हैं। इन प्रक्रियाओं के लिए भी हमें एक–एक Character के साथ प्रक्रिया करनी होगी। उदाहरण के लिए मानलो कि हमारे पास दो String है, जिनमें क्रमशः Ram व Shyam Store हैं। यदि हमें इन्हे जोडना हो, तो एक अन्य Array लेना होगा और फिर एक Array के सभी Character को क्रम से इस Array में Store करना होगा फिर एक Space इस Array में जोडना होगा और उसके बाद अंत में दूसरे Array के Elements को Space के आगे से Input करना होगा।

Exercise:

1. Array कितने प्रकार के होते हैं? किसी Array में **n** तक की Fibonacci Series को Store करने का Program बनाते हुए Array को समझाईए, जबकि Series की Limit **n** को User Input करता है।
2. int, char, float व double प्रकार के चार Array बनाईए और इन्हें Appropriate Values Initialize कीजिए, जबकि Array की Size 10 है।
3. One-Dimensional व Two-Dimensional Array के Traversing का Algorithm बनाते हुए इस Algorithm को Implement करने का Program बनाईए तथा दोनों ही Programs के Flow को समझाईए।
4. एक Array में Stored Integer प्रकार के विभिन्न मानों की Sorting करने का Program Algorithm की मदद से बनाइए साथ ही Algorithm के Flow को भी समझाईए।
5. किसी Array में Stored विभिन्न प्रकार के Integer मानों में से **Smallest** व **Largest** Values को Output में Print कीजिए, जबकि Array में Stored विभिन्न प्रकार के मानों को User Input करता है।
6. किसी Array में Stored String Palindrome है अथवा नहीं, इस बात को Determine करने के लिए एक Program Develop कीजिए।

Chapter Level Exercise:

1. निम्न Format Output में Print करने के लिए तीनों प्रकार के Loops का प्रयोग करते हुए Program बनाईए।

Format 01

```
1
1 2
1 2 3
1 2 3 4
```

Format 02

```
1
2 2
3 3 3
4 4 4 4
```

Format 03

```
* * * * *
* * *
* *
*
* *
* * *
* * * * *
```

Format 04

```
1#
2 2#
3 3 3#
4 4 4 4#
3 3 3#
2 2#
1#
```

Format 05

```
1
2 3
4 5 6
7 8 9 10
```

Format 06

```
1
0 1
1 0 1
0 1 0 1
```

Format 07

```
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
```

Format 08

```
* * * * *
  * * * *
    * * *
      * *
        *
```

Format 09

```
     *
    ***
   *****
  *******
 *********
***********
 *********
  *******
   *****
    ***
     *
```

Format 10

```
***********
***** *****
****   ****
***     ***
**       **
*         *
**       **
***     ***
****   ****
***** *****
***********
```

Format 11

```
A B C D F G F D C B A
A B C D F   F D C B A
A B C D       D C B A
A B C           C B A
A B               B A
A                   A
```

Format 12

```
1
1 1
1 1 2
1 1 2 3
1 1 2 3 5
1 1 2 3 5 8
```

Format 13

```
----------------
|I             |
|IL            |
|ILI           |
|ILIK          |
|ILIKE         |
|ILIKEC        |
----------------
|ILIKEC        |
|ILIKE         |
|ILIK          |
|ILI           |
|IL            |
|I             |
----------------
```

2 n संख्याओं की Fibonacci Series को Output में Print करने का Program तीनों प्रकार के Loops का प्रयोग करते हुए बनाओ।

3 1 से 10 तक के पहाडे को Output में Print करने का Program तीनों Loops का प्रयोग करते हुए बनाओ।

4 एक ऐसा प्रोग्राम बनाओ जो Input किये गए अंकों का योग व उसका Reverse Format Print करे और साथ ही ये भी बताए कि संख्याएं Palindrome हैं या नहीं। यानी यदि 1234 Input किया जाए तो 1 + 2 + 3 + 4 + = 10 print करे तथा इसका Reverse यानी 4321 Print करे। अंत में एक Message आए जो ये बताए कि संख्या Palindrome नहीं है।

5 एक प्रोग्राम लिखो जिसमें User कोई String Input करे तो Output में उस String में कितने Vowels हैं, ये Print हो।

6 एक प्रोग्राम बनाओ जिसमें 10 **Student**s की लम्बाई व वजन Input किया जाए और Output में ये बताया जाए कि कितने **Student**s की लम्बाई 170 सेमी से कम व वजन 50 किलो से ज्यादा है।

7 एक ऐसा प्रोग्राम बनाओ जिसमें दो Array में मान Input किए जाए और समान Locations पर स्थित मानों का योग करके तीसरे Array में रखा जाए तथा Output में तीसरे Array के मानों को Print किया जाए।

8 Array द्वारा दो Strings को जोडने का प्रोग्राम बनाइये।

9 Input की गई Strings एक जैसी है या नहीं तथा Input String की Length कितनी है, ये ज्ञात करने का प्रोग्राम बनाइये।

10 Input की गई Strings में कितने शब्द हैं, ये ज्ञात करने का प्रोग्राम बनाइये।

11 Input की गई String Palindrome है या नहीं, ये ज्ञात करने का प्रोग्राम बनाइये।

12 निम्न Series के Results को Output में Print करने के Programs बनाईए, जबकि x व n का मान Keyboard से Input किया जा रहा होः

A $x = 1/!1 + 1/!2 + 1/!3 + . . . + 1/!n$

B $x = 1/!1 + 2/!2 + 3/!3 + . . . + n/!n$

C $x = 1^1/!1 + 2^2/!2 + 3^3/!3 + . . + n^n/!n$

D $x = 1^2/!1 + 2^2/!2 + 3^2/!3 + . . + n^2/!n$

E $x = 1^1+1/(!1)^1 + 2^2+2/(!2)^2 + 3^3+^3/(!3)^3 +...+ n^n+n/(!n)^n$

FUNCTIONS

Functions

जब हम बडे प्रोग्राम लिखते हैं, तब कई बार ये परेशानी आती है कि हमें एक ही काम को करने के लिए बार–बार कुछ Statements को लिखना पडता है। साधारण तौर पर इस बात को एक उदाहरण द्वारा समझते हैं। माना हमें पांच विधार्थीयों के कक्षा में प्राप्त कुल प्राप्तांकों का योग करना है, तो हर विधार्थी के अंकों के योग की गणना करने के लिए हमें पांच बार Program Codes लिखने होंगे।

इस प्रकार से प्रोग्राम लिखने पर प्रोग्राम की लम्बाई बढ जाएगी और Program बहुत जटिल हो जाएगा। इस समस्या से बचने के लिए हम अलग से एक प्रोग्राम लिख देते हैं, और जब भी हमें कोई गणना करनी होती है, तो हम उस Sub-Program को उपयोग में ले लेते हैं। इस प्रकार से किसी खास काम के Codes को प्रोग्राम में बार–बार ना लिख कर, उसे अलग से लिख लिया जाता है व आवश्यकता के अनुसार उपयोग में लिया जाता है। ये Sub-Program Function कहलाता है।

Function का प्रयोग करके किसी प्रोग्राम को कई छोटे–छोटे भागों में बांटा जा सकता है। यानी हम ये भी कह सकते हैं, कि एक Function, प्रोग्राम Codes का एक समूह होता है, जो एक विशेष काम के लिए बनाया जाता है। Function एक Block Box की तरह काम करता है। यह किसी भी अन्य Function से Data लेता है और व्यवस्था के अनुसार Value Return करता है। Function के अंदर लिखे गए Codes अदृश्य रहते हैं। main() Program में किसी Function में क्या प्रक्रिया हो रही ये किसी को पता नहीं चलता। Functions को आवश्यकतानुसार बनाने व Use करने से कई लाभ होते हैं जिनमें से कुछ निम्नानुसार हैं:

1. एक Function में बार–बार दोहराने वाले Statements का पूरा समूह लिख दिया जाता है और जब भी main() Program को उस Statement समूह की जरूरत होती है, तो उस Function को main() Program में Call कर लिया जाता है, जिससे main() Program की लम्बाई कम हो जाती है और गलतियों की सम्भावना कम हो जाती है।

2. Functions समझने में आसान होते हैं। यदि किसी Function में कोई गलती होती है, तो हमें पूरा प्रोग्राम Check नहीं करना होता बल्कि केवल उसी Function को Debug करना पडता है।

3. एक बार जो Function बना दिये जाते हैं उन Functions को एक अलग Source File में Save करके किसी भी अन्य प्रोग्राम में भी उपयोग में लाया जा सकता है। इस प्रकार किसी अन्य प्रोग्राम में हमें वापस उस Function को नहीं लिखना पडता। हम Functions की अपनी एक अलग Directory भी बना सकते हैं, जिसमें विभिन्न नए–नए Functions Store करके रख सकते हैं और उनका भविष्य में उपयोग कर सकते हैं।

Function दो प्रकार के होते हैं:

Library Functions

ये वे Function होते हैं जो "C" में पहले से बना कर रखे गए हैं। इन्हे लिखने की जरूरत नहीं होती है, बल्कि इन्हे सीधे ही उपयोग में लिया जा सकता है। जैसे printf(), scanf(), cos(), sin(), आदि।

User Defined Functions

ये वे Functions होते हैं, जो User अपनी आवश्यकता के अनुसार बनाता है और विभिन्न प्रोग्रामों में उपयोग में लेता है। किसी भी प्रोग्राम में main() एक अनिवार्य User Defined Function (UDF) होता है। हर "C" Program में Program Control सबसे पहले इसी main() Function को खोजता है और इसी के Statement Block का Execution करता है। main() Function किसी भी प्रोग्राम में सिर्फ एक बार ही लिखा जाता है और बाकी के अन्य Functions main() Function से बाहर लिखे जाते हैं। main() भी एक UDF है। एक User Defined Function किसी भी अन्य User Defined Function या Library Function को Call कर सकता है। हम main() Function को भी किसी भी अन्य User Defined Function में Call कर सकते हैं।

Calling Function and Called Function

जिस प्रोग्राम में किसी User Defined Function को उपयोग करने के लिए User Defined Function का नाम लिख कर उस User Defined Function के कोष्ठक में Arguments दिये जाते हैं और कोष्ठक के बाद सेमीकॉलन का प्रयोग किया जाता है, उस प्रोग्राम को Calling Function कहा जाता है, और जिस User Defined Function को उपयोग में लिया जाता है उस User Defined Function को Called Function कहा जाता है। किसी प्रोग्राम में किसी Function को Use करना, Function Call करना कहलाता है।

Function Definition

Function को निम्न Format में Define किया जाता है:

```
Return_Data_Type Function_Name ( Argument_List )
Argument Variables Declaration;
{
    local Variables;
    Statement 1;
    Statement 2;
    "   "   "
    Statement n;
    Return (Expression );
}
```

Return_Data_Type

यहां हमें ये Declare करना होता है कि UDF Function, Call कर रहे Function को जो मान Return करेगा, वह मान किस प्रकार के Data Type का होगा। Default रूप में एक Function **int** प्रकार का मान ही Return करता है। यदि हमें char प्रकार का मान Return करवाना हो, तो हमें यहां char लिखना होता है। यदि हम यहां पर कोई Data Type Declare ना करें, तो Function int प्रकार का मान Return करता है।

Function_Name

यहां हम अपने Function का नाम लिखते हैं कि हमारे Function को हम किस नाम से अन्य प्रोग्राम में Use करेंगे। यहां नाम देने में उन सभी नियमों का पालन करना पडता है, जिन नियमों का पालन हम किसी Variable को नाम देने में करते हैं। ध्यान दें कि कभी भी किन्हीं भी दो User Defined Functions का नाम एक समान नहीं होना चाहिये और किसी भी अन्य User Defined Function का नाम main() नहीं होना चाहिये क्योंकि एक प्रोग्राम में main() Function केवल एक ही हो सकता है। हम किसी Function को ऐसा नाम भी नहीं दे सकते हैं, जो कि पहले से ही Library में उपलब्ध हो। उदाहरण के लिए हम printf() या scanf() नाम का कोई User Defined Function Create नहीं कर सकते हैं, क्योंकि ये पहले से ही Library में Predefined हैं।

Argument List

जब हमें कोई मान किसी User Defined Function को दे कर उस पर कोई प्रक्रिया करवानी होती है, तब हम वे मान User Defined Function के कोष्ठक को देते हैं। ये मान User Defined Function के कोष्ठक में लिखे Variables में चले जाते हैं, उसके बाद ही कोई प्रक्रिया User Defined Function में होती है। ये Variables Argument List कहलाते हैं और जो मान User Defined Function को किसी Function से प्राप्त होते हैं, वे मान Argument Values कहलाते हैं।

Argument Variables Declaration

हम जो भी मान Calling Function से Argument के रूप में किसी User Defined Function में प्राप्त करते हैं, वे मान किस Data Type के हैं, ये Declare करना जरूरी होता है। Argument के रूप में प्राप्त मान किस प्रकार के हैं, इसका Declaration इस भाग के अंतर्गत किया जाता है।

Local Variables

यहां पर हम अपनी आवश्यकता के अनुसार अन्य Variables Declare करते हैं। ये Variables Function से एक निश्चित Output प्राप्त करने के उद्देश्य से Declare किये जाते हैं। इन Variables का किसी प्रकार का कोई असर Calling Program पर नहीं पडता। यहां Declare किये गए Variables केवल इसी Function के लिए उपयोगी होते हैं। जैसे ही Program Control इस

User Defined Function से बाहर निकलता है, यहां के सारे Variables नष्ट हो जाते हैं। Storage Class के अंतर्गत इस सम्बंध में काफी कुछ बताया गया है।

Return (Expression)

Function के इस भाग में किसी Calling Function को क्या मान Return करना है, ये यहां पर लिखा जाता है। Functions के बारे में एक खास बात ये है, कि एक User Defined Function एक समय में केवल एक ही मान Calling Function को Return करता है। यदि हमें एक से अधिक मान किसी Calling Function को Return करने हों, तो हमें जितने मान Return करने हैं, Function को उतनी ही बार Use करना पडता है।

Statement Block

किसी भी Function के सारे Executable Codes एक ही Statement Block में लिखे जाते हैं। ये Statement Block मंझले कोष्ठक का बना होता है।

Function Prototype

किसी Function को जब प्रोग्राम में उपयोग में लाया जाना होता है, तब उस Function को Header Files के बाद व main() Function से पहले Declare कर दिया जाता है। इसे Prototype Declaration कहा जाता है। ऐसा करने से Compiler को Program Execution के समय ये पता चल जाता है कि कोई Function प्रोग्राम में Use किया जा रहा है, जिसे प्रोग्राम में किसी अन्य स्थान पर या फिर किसी अन्य File में Define किया गया है। किसी भी Function को Define करते समय Function को प्राप्त होने वाले Arguments किस प्रकार के होंगे, ये निर्धारित करने के दो तरीके हो सकते हैं।

हम चाहें तो UDF को प्राप्त होने वाले Arguments के Data Type का Declaration Argument कोष्ठक के अन्दर भी कर सकते हैं, जिस कोष्ठक में Arguments प्राप्त होते हैं या फिर हम चाहें तो कोष्ठक में केवल Arguments प्राप्त कर लें और फिर अगले Statement में Arguments के Data Type का Declaration कर सकते हैं, जैसाकि पिछले Format में बताया गया है। यदि हम सीधे ही Argument कोष्ठक में Data Type का Declaration करना चाहें तो Function का Definition Format निम्नानुसार हो जाता हैः

```
Return-Data-Type Function-Name (Data-type Arg1, Data-type arg2, Data-type argn)
 {
      local Variables;
      Statement 1;
      Statement n;
```

```
        Return (Expression);
    }
```

इस प्रकार से हमने UDF के Definition के दो Format देखें। दोनों ही प्रकार के Definitions सही हैं। हम इनमें से किसी को भी Use कर सकते हैं। दूसरी प्रकार से Declare करने पर यह पूर्णतया निश्चित हो जाता है कि Calling Function से वही मान Argument के तौर पर कोष्ठक में लिखे गए Variables में आऐंगे जो सही होंगे। जैसे

```
    int max (int a , int b)
```

ये User Defined Function ये निश्चित कर देता है, कि Calling Function से प्राप्त होने वाले दोनों ही मान int प्रकार के होंगे अन्यथा इन Variables में Calling Function से मान नहीं आऐंगे, जबकि यदि हम निम्नानुसार Declaration करें:

```
    int max ( a, b )
    int a, b;
```

तो Calling Function से आने वाला मान a व b में जरूर आएगा फिर चाहे Calling Function में a व b को प्राप्त होने वाले मान Float या Double प्रकार के ही क्यों ना हों। इसलिए हम हमेंशा किसी भी प्रकार की परेशानी से बचने के लिए दूसरे Format के प्रकार से ही Arguments Receive करते हैं।

Types of Functions

मुख्यतः Function चार प्रकार के होते हैं:

Function Without Argument And Return Value

इस प्रकार के User Defined Function में Calling Function से किसी भी प्रकार का कोई मान Argument या Parameter के रूप में User Defined Function को Pass नहीं किया जाता है। ये User Defined Function, Calling Function को किसी प्रकार का कोई मान Return नहीं करते हैं।

इस प्रकार के User Defined Function केवल किसी एक खास काम के लिए बनाए जाते हैं और उनका काम पूरा होते ही इस प्रकार के Function का Calling Function से कोई सम्बंध नही रह जाता है।

इस प्रकार के बिना Argument व बिना Return value के Function का कोष्ठक खाली ही रखा जाता है और इस Function में Return Keyword का या तो किया ही नहीं जाता है या फिर Return (0) लिखा जाता है, जिसका मतलब होता है कि ये User Defined Function कोई मान Calling Function को Return नहीं कर रहा है। जैसे हम नीचे एक प्रोग्राम में इस प्रकार के Function का प्रयोग कर रहे हैं:

Program

```
#include<stdio.h>
main()
{
    int a, b, c;
    clrscr();
    printline();
    printf("\n Enter Values of A and B ");
    scanf("%d %d", &a, &b);
    c = a + b;
    printf("\n The Sum of A and B is %d", c);
    printline();
    getch();
}

//User Defined Function :
printline()
{
    int x;
    for(x=0; x<=40;x++)
    {
        printf("-");
    }
}
```

अब इस प्रोग्राम को Execute किया जाता है और दो संख्याएं 10 व 20 Input की जाती है। Output के रूप में हमें निम्न Output प्राप्त होता है:

Output

```
Enter Values of A and B 10  20
The Sum of A and B is 30
```

जब ये प्रोग्राम Run होता है तब Variables Declaration के बाद Program Control को सर्वप्रथम Statement printline() function प्राप्त होता है। Program Control सीधे ही इस Function में प्रवेश करता है। यहां Program Control, User Defined Function का एक Local Variable x Declare करता है।

फिर for Loop चलाया जाता है और for Loop के हर Iteration में - Print किया जाता है, जब तक कि Loop Terminate ना हो जाए। यहां - चिन्ह 41 बार Print होता है, जिससे एक Line बन जाती है।

Loop का Execution समाप्त होते ही Program Control इस User Defined Function से बाहर आ जाता है और वापस Calling Function main() मे पहुंच जाता है। यहां Variable A व B का मान प्राप्त करने का Message देता है। मान प्राप्त करके Program Control A व B के मान का योग करके Variable C में Store कर देता है।

फिर Program Control को वापस एक printf() Function प्राप्त होता है, जहां Variable C के मान को Print कर दिया जाता है। प्रोग्राम Control को वापस वही printline() Function प्राप्त होता है और Program Control वापस 41 characters की एक Line print कर देता है।

इस प्रकार इस प्रोग्राम में printline() एक User Defined Function बनाया गया है। ये Function, main() Function को कोई मान ना तो Return करता है ना ही कोई मान main() Function से प्राप्त करता है। इस प्रकार ये एक बिना Argument व बिना Return Value का User Defined Function है।

Void

जब हम किसी User Defined Function को कोई मान Argument या Parameter के रूप में प्रदान नहीं करते हैं, तब User Defined Function का कोष्ठक खाली रखा जाता है। इस खाली कोष्ठक में void Keyword का प्रयोग किया जा सकता है। void Keyword Program Control को बताता है कि इस User Defined Function को किसी प्रकार का कोई मान Calling Function से प्राप्त नहीं हो रहा है।

साथ ही हम void Keyword का प्रयोग Function Declaration के समय यदि Data Type से पहले कर दें, तो ये Statement Program Control को ये बताता है, कि वह अमुक User Defined Function, Calling Function को कोई मान Return नहीं कर रहा है। इसी बात को हम printline() Function के साथ प्रयोग करके बताते हैं।

```
//UDF :
void printline( void )
```

```
{
    int x;
    for(x=0; x<=40;x++)
    {
        printf("-");
    }
}
```

ये void printline(void) Function Program Control को ये बताता है कि इस Function से प्राप्त होने वाला Output void प्रकार का है यानी ये Function किसी प्रकार का कोई मान Calling Function को Return नहीं करेगा। कोष्ठक में void का प्रयोग Program Control को ये बताता है, कि Argument या Parameter के रूप में इस User Defined Function को Calling Function से किसी प्रकार का कोई मान प्राप्त नहीं हो रहा है, यानी ये कोष्ठक किसी प्रकार का कोई मान Calling Function से Receive नहीं कर रहा है और ये कोष्ठक खाली है।

Example :

चक्रवृद्धि ब्याज ज्ञात करते का प्रोग्राम लिखिए जिसमें printline Function *का प्रयोग किया जाए।*

Program

```
#include<stdio.h>
main()
{
    clrscr();
    printline();
    value();
    printline();
    getch();
}

printline()
{
    int i;
    for(i=1;i<=35;i++)
    printf("%c",'=');
    printf("\n");
}

value()
```

```
{
    int year, period;
    float inrate, sum, principal;

    printf("\n principal amount :");
    scanf("%f", &principal);
    printf("\n Interest rate?");
    scanf("%f", &inrate);
    printf("\n period?");
    scanf("%d", &period);

    sum=principal;
    year=1;

    while(year<=period)
    {
        sum*=(1+inrate);
        year+=1;
    }
    printf("\n%8.2f\n \n%5.2f \n%5d\n \n%12.2f\n",principal,inrate,period,sum);
}
```

Output

```
principal amount :1000
Interest rate?4
period?5
 1000.00
 4.00
 5
 3125000.00
```

Function With Argument But No Return Value

इस प्रकार के User Defined Function में Calling Function से Argument के रूप में Parameters Pass होते हैं, लेकिन Calling Function को किसी प्रकार का कोई मान Return नहीं होता है। जब main() Function के किसी Variable का मान, किसी User Defined Function द्वारा उपयोग में लाया जाना होता है, तो वह मान User Defined Function को Pass कर दिया जाता है।

जब main() Function के कई Variables या किसी अन्य Function के कई Variables के मान किसी User Defined Function के कोष्ठक में भेजे जाते हैं, तो इन मानों को Arguments कहते हैं। इन Arguments को Comma Separator द्वारा अगल रखा जाता है। जब हम किसी Calling Function के Variables के मान User Defined Function को Pass करते हैं, तो ये Arguments ACTUAL Parameters कहलाते हैं, क्योंकि Called Function को Calling Function के Variables के वास्तविक मान भेजे जा रहे होते हैं।

जब किसी Function को Call करते समय उसमें Arguments के रूप में ACTUAL Parameters Pass किये जाते हैं, तो इस प्रकार से Call किये गए Called Function को **Call By Value** Function कहा जाता है।

जब हम किसी User Defined Function को किसी अन्य Function से Argument Pass करते हैं, तो हमें User Defined Function के कोष्ठक में भी Arguments Declare करने पडते हैं, ताकि ACTUAL Parameters के रूप में प्राप्त होने वाले मानों को User Defined Function में किन्ही अन्य Variables में Store करके रखा जा सके और उनके साथ आवश्यकतानुसार प्रक्रिया करके वांछित परिणाम प्राप्त किये जा सकें।

User Defined Function में Declare किये गए ये Arguments FORMAL Arguments कहलाते हैं। Calling Function से प्राप्त ACTUAL Arguments के Variables के मान उसी क्रम में User Defined Function के कोष्ठक में Declare किये गए FORMAL Arguments के Variables को प्राप्त हो जाते हैं। इसे हम निम्न प्रोग्राम द्वारा समझाने की कोशिश करते हैं–

Program

```
#include<header File>
main()
{
    int x, y; //Variables Declaration
    float z;
    clrscr();
    sum (   x,      y,      z );            //ACTUAL Arguments
                 |       |       |
    .   .   .    |       |       |
                 |       |       |
    getch();     |       |       |
}                |       |       |
                 ↓       ↓       ↓
sum (       a,      b,      c )             //FORMAL Arguments
int a, b; //Arguments Declaration
float c;
```

```
{
    "   "   ";
    "   "   ";
}
```

इस प्रोग्राम में तीन Variables गए x, y, z लिए गए हैं। x व y int प्रकार के हैं, जबकि z Float प्रकार का है। main() से बाहर एक User Defined Function sum लिखा गया है। इस Function को main() Function में Call किया गया है। main() Function से Parameters के रूप में गए y व z को sum के कोष्ठक में लिख दिया गया है। इस प्रकार यदि हम मान लें कि x = 20, b = 10 है, तो x का मान a को व y का मान b को प्राप्त हो जाएगा। यानी x = a = 20 व y = b = 10 हो जाएगा। इस प्रकार User Defined Function में a का मान 20 व b का मान 10 हो जाएगा।

x, y व z का वास्तविक मान ही sum Function को दे दिया गया है, इसलिए ये ACTUAL Parameters हैं और इन Variables की एक Copy a, b व c को प्राप्त हो रही है, इसलिए a, b व c FORMAL Arguments हैं।

वास्तव में sum Function में गए y, z का जो मान क्रम से a, b व c को प्राप्त होता है, वह मात्र एक प्रतिलिपी होती है। यदि यहां हम b का मान बदल कर 34 कर दें तो भी ACTUAL Argument में y का मान 10 ही रहेगा।

यहां ये खास तरह से ध्यान रखना होता है कि User Defined Function में जितने Arguments Declare होते हैं, उतने ही Argument वह Calling Function से Accept करता है। यानी यदि हमने User Defined Function में केवल तीन Arguments के लिए Variables Declare किये हैं और Calling Function इस User Defined Function में चार मान Pass करता हैं तो क्रम से तीन Variables के मान तो User Defined Function में Declare Variables को प्राप्त हो जाएंगे लेकिन चौथा Parameter फालतू ही रहेगा। जैसे

Program

```
#include<header File>
main()
{
    int x, y; //Variables Declaration
    float z; clrscr();
    sum (   x,      y,      z ,     k );    //ACTUAL Arguments
    . . .
    . . .
    getch();
}
```

```
sum (        a,        b,        c )              //FORMAL Arguments
int a, b; //Arguments Declaration
float c;
{
    "    "    ";
    "    "    ";
}
```

इस प्रोग्राम में sum User Defined Function केवल तीन Argument ही Accept कर सकता है। इसलिए c=a, y=b व z=c हो जाएगा। लेकिन चौथा Argument k का यहां पर कोई उपयोग नहीं है।

क्योंकि ये Argument User Defined Function में pass ही नहीं होगा और फालतू में Memory में Space रोकेगा। इसी प्रकार यदि Actual Parameters, Formal Parameters से कम हो, तो User Defined Function में एक Variable खाली रहेगा और उसमें Garbage Value Store रहेगी। जैसे

Program

```
#include<header File>
main()
{
    int x, y; //Variables Declaration
    float z;
    clrscr();
    sum (    x,        y,        z );              //ACTUAL Arguments
    . . .
    . . .
    getch();
}

sum (        a,        b,        c,        d )     //FORMAL Arguments
int a, b; //Arguments Declaration
float c;
{
    //. . .
}
```

इस उदाहरण में Formal Parameters, Actual Parameters से अधिक हैं। इसलिए यहां User Defined Function में Variable d का कोई मतलब नहीं है और ये Garbage Value दिखाएगा। हम जिस क्रम में Actual Parameters लिखते हैं, उसी क्रम में वे User Defined Function के Variables को प्राप्त होते हैं।

यानी कि x का मान User Defined Function में a को ही प्राप्त होगा b को नहीं। y का मान b को ही प्राप्त होगा c को नहीं। इस प्रकार जिस क्रम में Arguments Pass किये जाते हैं, उसी क्रम में वे User Defined Function में Declare किये गए Arguments को प्राप्त होते हैं।

एक और खास बात यहां ध्यान रखने की होती है, जो कि पहले भी बताया गया है, कि Actual Arguments के रूप में प्राप्त होने वाले Variables का Data Type, Formal Arguments के Declared Data Type के समान होना चाहिये। जैसे निम्न प्रोग्राम के प्रारूप को देखें:

Program

```
#include<header File>
main()
{
    int x, y; float z; clrscr();
    sum (   x,      y,      z );            //ACTUAL Arguments
    . . .
    . . .
    getch();
}
sum (       a,      b,      c )             //FORMAL Arguments
int a, b; //Arguments Declaration
float c;
{
    "   "   ";
    "   "   ";
}
```

इस प्रोग्राम में x व y को int प्रकार का लिया गया है। इसलिए जब इन्हें Argument के रूप में sum Function को भेजा गया, तो ये मान क्रमशः a व b को प्राप्त हो रहे हैं। इसलिए ये जरूरी है कि a व b भी int प्रकार के हों साथ ही z का मान हमें Float में चाहिये, इसलिए z को User Defined Function में Argument के रूप में Pass करने पर z का मान c को प्राप्त होता है, जिससे ये जरूरी हो जाता है, कि c भी float प्रकार का हो।

इसीलिए Formal Argument के Declaration के बाद a व b को int व c को float प्रकार का Declare किया गया है। यहां Call किये जा रहे Function को main() Function द्वारा Actual Arguments Pass किये जा रहे हैं।

जब किसी Called Function में Actual values Pass किया जाता है, तो इस प्रकार के Function को **Call by Value** Function कहा जाता है। हम इस प्रकार के Argument without Return value Function को अच्छी तरह से समझने के लिए एक प्रोग्राम बनाते हैं। इस प्रोग्राम में दो संख्याओं के जोड व गुणा करने के लिए दो Function Use किये गए हैं।

Program

```
#include<stdio.h>
main()
{
    int j, k;
    clrscr();
    printf("\n Enter first and Second Value:");
    scanf("%d %d", &j , &k);
    sum (j, k);
    mul (j, k);
    getch();
}

//Function:
sum ( int x , int y)
{
    int z;
    z = x + y;
    printf("\n Sum of %d and %d is %d", x, y, z);
}

//Function:
mul (int l, int m )
{
    int n;
    n = l * m;
    printf("\n Multiplication of %d and %d is %d", l, m, n);
}
```

जब ये Program Execute होता है तो Program Control दो int प्रकार के Variable Declare करता है। फिर printf() Function से एक Message print करके Input लिया जाता है। ये मान क्रमशः j व k को प्राप्त होते हैं।

अब Program Control को एक User Defined Function sum प्राप्त होता है। यहां Actual Argument के रूप में j व k को Called Function sum को Pass किया जाता है। sum में ये मान Formal Argument के रूप में x व y को प्राप्त होते हैं। यहां Block में एक अन्य Variable z Declare किया है। x व y का योग करके प्राप्त मान को z में Store किया जाता है, और z का मान print करवा दिया जाता है।

Program Control ये मान Print करके पुनः main() Function में जाता है। यहां उसे एक और Function mul प्राप्त होता है और इसमें भी Actual Argument के तौर पर j व k का मान क्रमशः l व m को प्राप्त हो जाता है। Function Block में एक local Variable n Declare किया गया है। l व m के गुणनफल का मान इस n नाम के int प्रकार के Variable में Store कर दिया जाता है, जो कि केवल User Defined Function के लिए ही Use हो सकता है। यहां Output में n को Print कर दिया जाता है, जो कि j व k के मान का गुणनफल Print करता है।

Program Control वापस main() में जाता है। इस प्रकार इस प्रोग्राम में दो User Defined Function प्रयोग किये गए हैं, जिन्हे किसी भी अन्य Function या main() Function में प्रयोग करके किन्ही भी दो संख्याओं का योग या गुणनफल प्राप्त किया जा सकता है। इस प्रकार यहां दोनों ही Function, main() Function से Argument ले रहे हैं, लेकिन किसी भी प्रकार का कोई भी मान Return नहीं कर रहे हैं।

Example :

FUNCTION WITH ARGUMENTS AND NO RETURN VALUE

Program

```
#include<stdio.h>
main()
{
    int a, b;
    clrscr();
    printf("Enter the two integer value:");
    scanf("%d %d", &a, &b);
    sum();
    getch();
}
sum(int x, int y)
{
```

```
    int z;
  z=x/y;
  printf("z=%d", z);
}
```

Output

```
Enter the two integer value:9 3
z=3
```

Example :

चक्रवृद्धि ब्याज ज्ञात करने का प्रोग्राम फंक्शन के प्रयोग से लिखिए।

Program

```
#include<stdio.h>
void printline(char c);
void value(float, float, int);

main()
{
    float principal, inrate;
    int period;
    clrscr();

    printf("Enter principal amount interest:");
    printf("rate and period\n");
    scanf("%f %f %d", &principal, &inrate, &period);

    printline('z');

    value(principal, inrate, period);

    printline('c');

    getch();
}

//Function :
void printline(char ch)
```

```
{
      int i;
      for(i=1;i<=52;i++)
      printf("%c", ch);
      printf("\n");
}

//Function :
void value (float p, float r, int n)
{
       int year;
       float sum;
       sum=p;
       year=1;
       while(year<=n)
       {
         sum=sum*(1+r);
         year=year+1;
       }
       printf("%f \t %f \t %d \t %f \n", p, r, n, sum);
 }
```

Output

```
Enter principal amount interest: rate and period
1000 2.5 6
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
1000.000000    2.500000      6      1838265.625000
cccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Function With Argument And Return Value

इस प्रकार के User Defined Function में Calling Function से User Defined Function को Argument भी Pass किये जाते हैं और User Defined Function से किसी प्रकार की प्रक्रिया करवा कर वापस मान भी प्राप्त किये जाते हैं। हम ऊपर के ही प्रोग्राम में थोडा सा बदलाव करके इसे Function with Argument and Return Value में बदल सकते हैं। देखें निम्न प्रोग्राम

Program

```
#include<stdio.h>
```

```
main()
{
    int j, k, c, d;
    clrscr();

    printf("\n Enter first and Second Value:");
    scanf("%d %d", &j , &k);

    c = sum (j, k);
    printf("\n Sum of %d and %d is %d", j, k, c);

    d = mul (j, k);
    printf("\n Multiplication of %d and %d is %d", j, k, d);

    getch();
}

sum ( int x , int y)
{
    int z;
    z = x + y;
    return (z );
}

mul (int l, int m )
{
    int n;
    n = l * m;
    return ( n );
}
```

इस प्रोग्राम में दो अन्य Variable c व d को main() Function में Declare किया गया है। सारी प्रक्रिया पहले प्रोग्राम की तरह ही रहती है, लेकिन जब Program Control, User Defined Function sum में पहुंचता है, तब j व k का Formal Argument x व y में जाता है और x व y के मानों का योग z में Store हो जाता है। Program Control को यहां Return Statement मिलता है, जो z का मान main() को Return कर देता है। इससे z में Store मान main() Function में Declared Variable c को Assign हो जाता है, क्योंकि c = sum (j, k) लिखा है। इससे मान Return होने के बाद c = z हो जाता है और Output में c को Print करवा दिया जाता है।

फिर Program Control को d = mul (j, k); Statement प्राप्त होता है। Program Control, mul User Defined Function में जाता है। यहां main() Function के j व k का मान l व m को प्राप्त हो जाता है। इस Function में l व m के गुणनफल को n में Store किया जाता है और ये मान main() Function को Return कर दिया जाता है, जिससे d का मान n के मान के बराबर हो जाता है। अब Output मे d का मान Print कर दिया जाता है, जो कि j व k के Multiplication के बराबर है। इस प्रकार से हम किसी भी Function से User Defined Function में Argument प्राप्त करके पुनः Calling Function को मान Return कर सकते हैं।

Example :

ARGUMENT WITH RETURN VALUE

Program

```
// Prototype Declaration
void printline(char ch, int len);
value(float, float, int);
#include<stdio.h>

main()
{
    float principal, inrate, amount;
    int period;
    clrscr();

    printf("Enter the principal amount interest:");
    printf("rate and period:\n");
    scanf("%f %f %d", &principal, &inrate, &period);
    printline('*',52);
    amount=value(principal, inrate, period);

    printf("\n%f\t%f\t%d\t%f\n\n", principal, inrate, period, amount);
    printline('=',52);
    getch();
}

//Function :
 void printline(char ch, int len)
 {
     int i;
```

```
        for(i=1;i<= len; i++)
        printf("%c", ch);
        printf("\n");
    }

    //Function :
    value(float p, float r, int n)
    {
        int year;
        float sum;
        sum=p;
        year=1;

        while(year<=n)
        {
            sum=sum*(1+r);
            year=year+1;
        }
        return(sum);
    }
```

Output

```
Enter the principal amount interest: rate and period:
4500
3
5
*****************************************************
4500.000000    3.000000      5      20480.000000
=====================================================
```

Example :

ARGUMENT WITH RETURN VALUE

Program

```
#include<stdio.h>

main()
{
```

```
    int x, y; /* input Data */
    double power (int, int); /* prototype declaration */
    clrscr();

    printf("Enter the x, y:");
    scanf("%d%d", &x, &y);

    printf("%d to power %d is %f\n", x, y, power(x, y));
    getch();
}


//Function :
double power (int x, int y)
{
    double p;
    p=1.0;  /* x to power zero */
    if(y>=0)
        while(y--) /*computes positive powers */
        p=p*x;
    else
        while(y++) /* computes negative powers */
            p=p/x;
    return(p); /* Return double types */
}
```

Output

```
Enter the x, y:25 5
25 to power 5 is 9765625.000000
```

Function Without Argument But Return Value

इस प्रकार के Function में हम Calling Function से कोई मान, User Defined Function में प्रदान नहीं करते हैं, लेकिन Calling Function को मान Return किया जाता है। इस तरह के Function व्यवहारिक तौर पर बहुत ही कम प्रयोग में आते हैं।

Example :

Factorial ज्ञात करने का प्रोग्राम *Function* के प्रयोग से बनाइये।

Program

```
#include<stdio.h>

main()
{
    int a, b;
    clrscr();

    printf("enter the factorial value:");
  scanf("%d", &a);

    b=fact(a);

    printf("fact=%d", b);
  getch();
}

fact(int x)
{
    int i, sum=1;
    for(i=1;i<=x; i++)
    sum*=i;
    return(sum);
}
```

Output

```
enter the factorial value:5
fact=120
```

Exercise:

1. Function बनाने के उद्देश्य को समझाते हुए इसके लाभ लिखिए।
2. Function कितने प्रकार के होते हैं? एक Function का Block Structure बनाते हुए Function के विभिन्न अवयवों का वर्णन कीजिए।
3. Function Prototypes को समझाईए।
4. Function कितने प्रकार के होते हैं? सभी प्रकार के Functions को एक उचित उदाहरण Program बनाते हुए समझाईए।
5. दो Strings को आपस में जोडने के लिए एक Function बनाईए और इस Function को एक main() Function में Call कीजिए।
6. एक Power Function power(value, exponent) बनाईए जो Argument के रूप में आने वाले मान का Power Return करने का काम करे।
7. Keyboard से Input किया गया Year Leap Year है अथवा नहीं, इस बात का पता लगाने के लिए एक Function बनाईए।
8. Keyboard से Input किए गए किसी Positive Integer के Prime Factors (गुणनखण्ड) ज्ञात करने का Function Create कीजिए। उदाहरण के लिए 24 का Prime Faction 24 = 2 * 2 * 2 * 4 होता है। इसी तरह 70 का Prime Factor 70 = 2 * 5 * 7 होता है।

Recursion and Recursive Function

हम जिस प्रकार से User Defined Function को main() Function में Call करते हैं, उसी प्रकार से किसी भी अन्य Function को किसी भी User Defined Function में Call कर सकते हैं। यहां तक कि main() Function को भी किसी User Defined Function में Call कर सकते हैं, क्योंकि main() Function भी एक User Defined Function ही है, जिसे User अपनी आवश्यकता के अनुसार लिखता है। हम किसी Function को खुद उसी Function में Call कर सकते हैं। जब हम किसी Function को वापस उसी Function में Call करते हैं, तो ये एक प्रकार की Looping हो जाती है।

इस प्रक्रिया को "C" में Recursion कहा जाता है और ऐसे Function को Recursive Function कहते हैं। इस प्रक्रिया में वह Function तब तक Execute होता रहता है, जब तक कि किसी Condition द्वारा उसे Terminate ना कर दिया जाए। Factorial ज्ञात करने में Recursive Function को Use किया जा सकता है। ये Recursive Function को समझने का एक अच्छा उदाहरण है।

Recursive Function :

```
Factorial ( n )
int n;
{
    int fact;
    if ( n = = 1)
        return ( 1 );
    else
        fact = n * Factorial ( n –1 );
    return ( fact );
}
```

मानलो कि हमें संख्या 5 का Factorial ज्ञात करना है। मान 5 का Factorial ज्ञात करने के लिए हमें मान 5 को Factorial Function में Argument के रूप में भेजना होता है। ये Function स्वयं को 5 बार निम्नानुसार Recursively Call करता है:

```
Factorial(n) =  Factorial(5)
             =  5 * Factorial(4)
             =  5 * ( 4 * Factorial(3))
             =  5 * ( 4 * ( 3 * Factorial(2)))
             =  5 * ( 4 * ( 3 * ( 2 * Factorial(1))))
             =  5 * ( 4 * ( 3 * ( 2 * ( 1 ))))
             =  120
```

जब हम इस Function में संख्या 5 Argument के रूप में देते हैं तो सबसे पहले if Condition Check होती है कि क्या n का मान 1 है या नहीं। चूंकि अभी n का मान 5 है इसलिए if Condition False Return करता है और Program Control else Statement Block में जाता है।

यहां पर वापस Factorial Function मिलता है लेकिन इस बार Factorial Function में Argument के रूप में 5-1 = 4 जाता है। वापस से Factorial Function Call होता है और if Condition में Check किया जाता है कि n का मान 1 है या नहीं।

यहां पर वापस n का मान 4 होने से Condition False हो जाती है और else Statement Block Execute होता है। वापस से Factorial Function Call होता है और Argument के रूप में इस बार Function में 4-1 = 3 जाता है।

एक बार फिर से Factorial Function में n का मान Check होता है। n का मान इस बार 3 है इसलिए वापस से else Statement Block Execute होता है जहां फिर से Factorial Function Call होता है और इस बार इस Function में Argument के रूप में 3-1 = 2 जाता है।

फिर से if Condition False हो जाती है और else Statement Block Execute होता है और एक बार फिर से Factorial Function Call होता है। इस बार Argument के रूप में 2-1 = 1 जाता है।

इस बार n का मान 1 होता है इसलिए if Condition **True** हो जाती है और Program Control इस Factorial Function से बाहर आ जाता है तथा Return Value के रूप में 1 Return करता है। Program Control अन्तिम Factorial से बाहर आते ही Second Last Factorial में पहुंचता है। यहां मान 1 का मान 2 से गुणा होता है। और परिणाम fact नाम के Variable में Store हो जाता है।

अब ये Factorial Function fact के मान 2 को Return करता है। इस मान 2 का गुणा पिछले Factorial के मान 3 से होता है और वापस से परिणाम fact नाम के Variable में Store हो जाता है। ये मान वापस पिछले Factorial में Return होता है जहां पर Return होने वाले मान 6 का गुणा मान 4 से होकर Variable fact में Store हो जाता है।

ये मान अन्तिम Factorial में Return होता है जहां मान 5 से इस Return होने वाले मान 24 का गुणा होता है। अब ये अन्तिम Factorial Function 120 Return करता है जो कि उस main() Function या Calling Function में Return होता है जिसमें इस Factorial Function को Call किया गया था और Argument के रूप में मान 5 भेजा गया था।

साधारण तरीके से देखें तो निम्न प्रोग्राम में main() Function को पुनः main() Function में Call किया गया है, जिससे Program Infinite हो जाता है। क्योंकि जैसे ही Program Control, Block के Statement को Execute करता है, उसे वापस main() Function मिल जाता है और

Program को वापस main() को Execute करना पडता है। ये क्रम अनन्त तक चलता रहता है। जब एक Function में किसी अन्य Function को प्रयोग किया जाता है, तो इसे Function की Nesting करना कहते हैं।

Program

```
#include<stdio.h>

main()
{
    printf("\t Govinda ");
    main();
}
```

हमारा CPU हर समय किसी ना किसी Program के किसी ना किसी Statement व Data पर Processing कर रहा होता है। जब हम किसी Function को Call करते हैं तो हमारा CPU अपने पुराने काम को छोड कर नए काम को करता है।

उदाहरण के लिए जब पहली बार Factorial Function Call होता है, तो Factorial Function के Call होने से पहले हमारा Program Main Function के Statements का Execution कर रहा होता है।

Factorial Function के Call होते ही हमारा CPU जिस Data की Processing कर रहा होता है उस Data को और उस Data पर जिस प्रकार की Processing करनी है, उस Processing की Instructions को एक विशेष प्रकार की Memory में रख देता है, जिसे Stack कहते हैं। Stack एक ऐसी Memory होती है जिसका प्रयोग हमारा CPU अपने Data व Programs Instructions को कुछ समय के लिए Temporarily Store करने के लिए करता है।

मानलो कि हमने Factorial Function को main() Function में Call किया है। तो CPU Factorial Function को Call करने से पहले main() Function के Data को Stack में रख देता है। फिर Factorial Function को Call करता है और Argument के रूप में Calling Function से आने वाले मान को Hold करने के लिए Variable n Create करता है। जब Program Control आगे बढता है तो उसे वापस Factorial Function प्राप्त होता है।

इस Factorial Function को Call करने से पहले Program Control अपने Current Data 5 को जो कि Variable n में पडा है, को Stack पर रखता है और फिर से Recursively Factorial Function को Call करता है साथ ही इस बार इसी Function में Argument के रूप में एक नया मान 4 प्रदान करता है।

फिर से मान 4 को Hold करने के लिए एक Variable n Create करता है। Program आगे बढता है तो वापस उसे एक और Factorial Function मिलता है।

इस Factorial को तीसरी बार Call करने से पहले वापस मान 4 को Stack पर रखता है। अब Stack में दो मान 5 व 4 होते हैं। फिर से Factorial Call होता है और Argument के रूप में आने वाले मान 3 को Hold करने के लिए एक Variable n Create करता है।

Program फिर आगे बढता है और फिर से Factorial को Call करने से पहले मान 3 को Stack में रखता है। इस तरह अब Stack में क्रमशः 5, 4, व 3 Stored रहते हैं।

इसी तरह से दो बार और Factorial Function Call होता है और Stack में क्रमशः 5, 4, 3, 2 व 1 Store हो जाता है। अब Program Control अन्तिम Factorial से 1 Return करता है। इस 1 का गुणा Stack पर पडे हुए 1 से होता है।

Program Control अब चौथे Factorial से 1*2 = 2 Return करता है जिसका गुणा Stack पर पडे हुए मान 3 से होता है।

अब तीसरा Factorial Function 2*3 = 6 Return करता है जिसका गुणा Stack पर पडे हुए दूसरे Factorial के मान 4 से होता है।

ये दूसरा Function पहले Factorial को 24 Return करता है जिसका गुणा पहले Factorial के Stack में पडे हुए मान 5 से होता है।

अन्त में ये पहला Factorial Function Calling Function को मान 120 Return करता है। इस तरह से एक Recursive Function Call होता है और Recursively Result प्रदान करता है।

Recursive Function एक प्रकार की Looping प्रदान करता है। यानी हम जो भी Program Looping का प्रयोग करके बनाते हैं वे सभी Program Recursive Functions का प्रयोग करके बनाए जा सकते हैं। Recursive Function को Use करने के कुछ मुख्य फायदे निम्नानुसार हैं:

1 Recursive Functions Non – Recursive Functions की तुलना में लिखने व समझने में सरल, Clear व छोटे होते हैं।
2 Program Directly किसी भी Problem के Solution का सिद्धांत Reflect कर देता है, कि किस प्रकार से किसी समस्या को Solve किया गया है।
3 इस प्रकार से Create किए गए Software को Maintain करना सरल होता है।
4 Recursive Functions का प्रयोग Non – Linear Data Structures को Maintain व Handle करने के लिए बहुत ही उपयोगी होता है।

किसी Non – Linear Data Structure को Handle करने के लिए हम किसी भी अन्य प्रकार के Statements का प्रयोग नहीं कर सकते हैं। वहां केवल Recursive Functions ही पूरी सक्षमता व पूर्णता के साथ उपयोग में लाया जा सकता है। हालांकि Recursive Functions लिखना व Maintain करना काफी सरल होता है लेकिन यदि थोडी सी असावधानी हो जाए और इस Function में कोई Bug हो जाए, तो उसे Debug करना बहुत ही भयानक काम बन जाता है। किसी भी प्रोग्राम में Function दो प्रकार से Call किये जा सकते हैं:

1 Call By Value

जब Calling Function से Variable के मान की एक Copy Parameter के रूप में User Defined Function को प्रदान की जाती है, तब इस प्रकार के Function को **Call By Value** Function कहा जाता है।

2 Call By Address

जब Calling Function से Variable के मान के स्थान पर Called Function में Variable का Address Pass किया जाता है, तो ऐसा Called Function, Call By Reference Function कहलाता है। ध्यान रहे कि Address हमेंशा Pointer Variable द्वारा भेजा जाता है।

Storage Classes

जब कोई Variable मान प्राप्त करता है तो वह मान Memory में Store होता है। "C" में Variable की Value Store होने की दो Locations हो सकती हैं:

1 Main Memory
2 C. P. U. Resister

Variable के मान को हम हमारी जरूरत के अनुसार इन दोनों में से किसी भी मनचाही जगह पर रख सकते हैं। Variable के मान को Memory में अलग–अलग जगह Store करने की Locations को ही "C" की Storage Class कहते हैं। "C" Language में मुख्य रूप से चार Storage Class हैं।

प्रोग्राम के Variable का जो भाग वास्तव में Active रूप से प्रोग्राम में Use हो रहा होता है, उसे Variable का Scope कहा जाता है और किसी Variable में Store मान कब तक Accessible रहेगा इसे Variable की Longevity कहते हैं। Variable को भी हम तीन भागों में बांट सकते हैं। ये भाग Variable के Declaration पर निर्भर करते हैं।

Type of Variables In Program

किसी प्रोग्राम में Variables को मुख्यतः तीन प्रकार से Declare किया जा सकता है और इन तीनों प्रकार के Variables का अपना अलग प्रकार का काम होता है।

Internal or Local or Private Variables

वे Variables जो किसी भी Function के Statement Block में अंदर (यानी मंझले कोष्ठक के अन्दर) Declare किये जाते हैं, Local Variables कहलाते हैं। ये जिस Function में उपयोग के लिए Declare किये जाते हैं, उस Function का काम पूरा होते ही या Program Control के उस Function के Statements का Execution करके Statement Block से बाहर निकलते ही, ये Variables व इनके मान नष्ट हो जाते हैं। इन Variables का मान किसी भी प्रकार से किसी अन्य Function को असर नहीं पहुंचाता है। अभी तक हमने जितने भी प्रोग्राम बनाए है, उन सभी में Local Variables का प्रयोग किया गया है।

Formal Variables

जब Calling Function से कोई मान किसी User Defined Function में Pass किया जाता है, तो वह Actual Argument के रूप में User Defined Function को Variable का वास्तविक मान भेजता है। User Defined Function में Calling Function से आने वाले मान को Receive करने के लिए जिन Variables को Define किया जाता है, वे Variables, Formal Variables कहलाते हैं। क्योंकि ये केवल औपचारिक Variables होते हैं, जिनमें Calling Function से मान आता है। Formal Variables किसी भी Function के Statement Block से (यानी मंझले कोष्ठक से पहले) बाहर की तरफ Declare किये जाते हैं।

External or Global or Public Variables

इस प्रकार के Variables main() Function के Statement Block से बाहर Declare किये जाते हैं। इस प्रकार के Variables की विशेषता ये होती है, कि इस प्रकार के Variables को प्रोग्राम में कोई भी Function या User Defined Function Use कर सकता है। इन्हे एक बार ही Define व Declare किया जाता है और फिर इन्हे कहीं भी प्रयोग में लाया जा सकता है। इन्हे वापस Declare करने की जरूरत नहीं रहती है। जैसे निम्न प्रोग्राम देखें–

Program

```
#include<stdio.h>
int x = 6, y = 8, z = 4;
main()
{
    int b, c ;
    c = x + y;
```

```
        printf("\n C is %d ", c);
        b = y – z;
        printf("\n B is %d ", b);
        mul(b);
        getch();
    }

    mul(int l)
    {
        int m;
        m = l * y;
        printf("\n Multiplication of B and Y is %d ", m);
    }
```

Output

```
C is 14
B is 4
Multiplication of B and Y is 32
```

आइये, समझते हैं कि ये Output किस प्रकार प्राप्त हुआ। जैसा कि हमने पहले कहा कि Global Variables का उपयोग कोई भी Function कर सकता है, इसलिए यहां c = x + y; Expression से Global Variable x का मान 6 व Global Variable y का मान 8 जुड कर Local Variable c को 14 प्राप्त हो गया है। इसी प्रकार b = y – z; से Global Variable y के मान 8 में से Global Variable z का मान 4 घट कर, प्राप्त मान Local Variable b को प्राप्त हो गया है। इस प्रकार b का मान Output में 4 print हुआ है।

ध्यान दें कि इस प्रोग्राम में mul नाम के User Defined Function को केवल एक ही मान (Local Variable b) को Actual Argument के रूप में भेजा गया है और इसके Formal Variable को Prototype के रूप में int प्रकार का Declare किया गया है। mul, User Defined Function में एक local Variable m Declare किया गया है।

यहां Global Variable y के मान को Declare नहीं किया गया है बल्कि सीधे ही mul Function में उपयोग में ले लिया गया है। इस प्रकार Global Variable को प्रोग्राम में केवल एक ही बार Declare करना होता है, फिर Global Variable के मान को Program में कहीं भी Use किया जा सकता है।

Automatic Storage Class

जब हम किसी Variable के Declaration से पूर्व Storage Class के रूप में auto Keyword का प्रयोग करते हैं, तो वह Variable Automatic Storage Class में Store होता है। Automatic Storage Class के Variables, Local Variables होते हैं। इनका उपयोग वहीं किया जा सकता है, जिस Function में इनको Declare किया गया है।

जैसे ही Program Control उस Function से बाहर निकलता है, जिसमें ये Variable Declare किया गया है, तो Program Control के उस Function से बाहर निकलते ही उस Variable का मान नष्ट हो जाता है। जब हम किसी Function में Variable Declare करते हैं, और उसके साथ किसी भी Storage Class का प्रयोग नहीं करते हैं, तो "C" Compiler उसे Default रूप से Automatic Storage Class का मान लेता है। जैसे

auto int num=0 व int num=0

इन दोनों ही Declaration का अर्थ समान ही है। Automatic Class की एक खास बात ये भी है कि यदि एक ही नाम के कई Variables एक ही प्रोग्राम में विभिन्न Function में Declare किये जाएं और हर Function में Variable को भिन्न मान प्रदान किया जाए, तो भी Program Control इसे मान्य करता है।

क्योंकि एक main() Function के अंदर लिखे गए विभिन्न User Defined Function में Variable का नाम समान होने पर भी, यदि UDF में उस समान नाम वाले Variable का मान परिवर्तन हो रहा हो तो भी main() Function में उस परिवर्तन का कोई फर्क नहीं पडता है। हम यही बात एक प्रोग्राम द्वारा समझाने की कोशिश करते हैं।

Program

```
#include<stdio.h>
main()
{
    auto int num = 6;
    clrscr();
    {
        {
            printf("%d", num);
        }
        printf("%d", num);
    }
    printf("%d", num);
    getch();
}
```

Output

```
666
```

इस प्रोग्राम के Output में 666 प्राप्त हुआ। इस प्रोग्राम में हर Block में एक printf() Function है, जो num के मान को Print कर रहा है। जब Program Control प्रथम Statement Block में प्रवेश करता है, तो वहां उसे दूसरा Statement Block प्राप्त होता है। यहां num नाम के Variable में Store मान को Print करने का Statement लिखा है, जो main() Function में लिखे num के मान 6 को Print कर देता है।

फिर Program Control इस Statement Block से बाहर आता हैं। यहां वापस उसे एक printf() Function मिलता है और वापस num नाम के Variable का मान Print किया जाता है।

Variable में जब Program Control इस Statement Block से भी बाहर निकलता है, तब उसे main() Function में लिखा printf() Function प्राप्त होता है। वापस num का मान 6 Print हो जाता है।

इस प्रकार तीनों Block में num का मान 6 ही Print हुआ क्योंकि हर Statement Block main() Function में लिखे num के मान को ही Print कर रहा है। जबकि नीचे लिखे इसी प्रोग्राम में थोडा सा बदलाव करने से Output में हमें भिन्न–भिन्न संख्याएं प्राप्त हो रही हैं।

Program

```
#include<stdio.h>
main()
{
    auto int num = 6;
    clrscr();
    {
        auto int num =8;
        {
            auto int num = 1;
            printf("%d", num);
        }
        printf("%d", num);
    }
    printf("%d", num);
    getch();
}
```

Output

```
186
```

इस प्रोग्राम को Execute करने पर हमें 186 Output में प्राप्त होता है। आइये समझने की कोशिश करते हैं, कि ऐसा क्यों हुआ। जब प्रोग्राम Control main() Function में पहुंचता है, तो वहां num नाम के एक Variable को Declare करता है और उसका मान 6 Initialize करता है। फिर Program Control आगे बढता है, तब उसे एक Statement Block प्राप्त होता है।

इस Statement Block में पहुंचते ही Program Control को वापस एक auto Storage Class का Variable num प्राप्त होता है। "C" Compiler इसे भी Memory में जगह देता है और इसका प्रारम्भिक मान 8 Initialize करता है। Program Control इससे आगे बढता है, तो वापस उसे एक Statement Block प्राप्त होता है, और यहां वापस एक auto Storage Class का Variable num Declare करता है और इसे मान 1 Assign करता है।

इसी Statement Block में अगला Statement है, जिसमें printf() Function द्वारा num के मान को Print किया गया है। num का मान तो तीनों Blocks में भिन्न है, तो फिर कौनसा मान Output में Print होगा?

यहां अंक 1 Output में Print हो रहा है। ऐसा इसलिए हो रहा है, क्योंकि किसी भी प्रोग्राम के किसी भी Function में जब किसी Variable के मान को Print करवाया जाता है, तो "C" Compiler सर्वप्रथम दिये गए Variable के नाम को Local Variables में ढूंढता है और यदि Local Variables में उस नाम का Variable नहीं मिलता, तब Program Control उसी प्रोग्राम में कहीं और यानी उस Statement Block या Function से बाहर उस नाम के Variable को ढूंढता है और जहां उस नाम का Variable प्राप्त हो जाता है, वहीं का मान Output में Print कर देता है।

ध्यान दें कि यह Variable ढूंढने का क्रम Inner Block से Outer Block की ओर चलता है। यानी यदि हमारे Program में हम किसी Statement Block के चौथे Inner Block में हैं और वहां किसी Variable का मान Print करवा रहे हैं, तो सर्वप्रथम Program Control चौथे Statement Block में उस Variable को ढूंढेगा। यदि Program Control को चौथे Statement Block में वह Variable प्राप्त नहीं होता, तो Program Control तीसरे Statement Block में उस Variable को खोजेगा।

यदि तीसरे में भी वह Variable प्राप्त ना हो तो दूसरे Statement Block में उस Variable को खोजेगा और यदि यहां भी उसे वह Variable प्राप्त ना हो तो अंत में Program Control main() Function में उस Variable को खोजेगा। इस क्रम में जहां भी पहले वह Variable प्राप्त हो जाएगा Program Control वहीं के मान को उस चौथे Inner Block द्वारा Output में Print कर देगा।

इसी कारण से ऊपर बताए गए प्रथम प्रोग्राम में तीनों Statement Blocks में num का मान 6 ही Print हुआ है। लेकिन इस प्रोग्राम में ऐसा नहीं हुआ है। क्योंकि इस प्रोग्राम में हर Statement

Block में num नाम का एक Variable Declare किया गया है, जो कि उस Statement Block के लिए एक Local Variable है और जैसा कि हमने अभी कहा कि जब Program Control को किसी Variable के मान के साथ प्रक्रिया करनी होती है, तो सर्वप्रथम Program Control उस Variable को Local Variables में खोजता है।

यहां दूसरे Statement Block में num का मान 1 Assign किया गया है और इसी Block में num को Print किया गया है, इसलिए यहां num का मान 1 ही Output में Print हो रहा है। इस Statement Block के अलावा जो दो अन्य Variables हैं, जिनका नाम भी num ही है। इस Statement Block से बाहर होने के कारण इस Statement Block के लिए वे num Global Variable हो गए हैं। यदि हम इस तीसरे Statement Block में num को Declare नहीं करते और num को Print करते तो Program Control Outer Statement Block के num के मान 8 को Output में Print कर देता और हमारा Output 886 प्राप्त होता।

इसी प्रकार जब Program Control इस Statement Block से बाहर आता है, तब उसे यही print() Function प्राप्त होता है और यहां Output में num का मान 8 print होता है, क्योंकि इस Statement Block में भी num नाम का एक Local Variable Declare किया गया है, जिसका मान 8 दिया गया है। यदि यहां पर ये Variable Declare नहीं करते तो Output में यहां 8 की बजाय 6 print होता और हमारा Output 166 होता।

Extern Storage Class

इस Class में वे Variables होते हैं, जिन्हें कोई भी अन्य Function Use कर सकता है। ये Global Variables होते हैं और इन्हे main() Function से बाहर Declare किया जाता है। ये Variables main() Function से पहले लिखे जाते हैं, और इसमें प्रोग्राम के अंत तक मान Store रहता है।

जब किसी प्रोग्राम में लगभग हर Function में किसी Variable का प्रयोग हो रहा हो, तो उस Variable को extern Keyword का प्रयोग करके Global Declare कर दिया जाता है, ताकि Memory Space बच सके और प्रोग्राम लम्बा व जटिल ना हो जाए। जैसे extern int num = 10; Global Variable की ये विशेषता होती है कि इसका प्रयोग कोई भी Block या Function कर सकता है और अपनी जरूरत के अनुसार इसका मान बदल सकता है।

अभी तक हमने देखा कि हम main() Function व User Defined Function सभी एक ही Program File में लिखते रहे हैं, लेकिन वास्तविक जीवन में जब हम बडे–बडे प्रोग्राम लिखते हैं, तब सारे के सारे Program Codes एक ही Program File में ना लिख कर कई Files में, User Defined Function के रूप में लिख देते हैं और उन्हें Compile कर लेते हैं। फिर हम उस Function फाईल को main() Program File में Link कर लेते हैं। जब हमें किसी Function की जरूरत होती है, तब उस File से Required Function को Call कर लेते हैं।

ये एक महत्वपूर्ण तरीका है, क्योंकि इस तरीके में किसी एक File में हुए Changes का किसी अन्य File पर कोई फर्क नहीं पडता है। जब हम एक से अधिक Source Files का प्रयोग करते हैं, तब कुछ Global Variables होते हैं, जिन्हे किसी भी अन्य User Defined Function में Use करना होता है।

किसी भी User Defined Function में किसी अन्य Source File में लिखे Global Variables को Use करने की सुविधा हम extern Keyword का प्रयोग करके प्राप्त कर सकते हैं। जब हम किसी Global Variable को किसी अन्य Source File में Use करना चाहते हैं, तब उस Global Variable को extern Keyword का प्रयोग करके लिखते हैं।

extern Keyword "C" Compiler को ये बताता है कि वह अमुक Global Variable किसी अन्य Source Code File में Declare किया गया है। extern Variable को ठीक से समझने के लिए हम यहां पूर्व में बनाए गए Function को ही दूसरे तरीके से बना रहे हैं। सबसे पहले नीचे लिखे mul Function को एक Source File में Save करते हैं और इस File का नाम extern.c देते है–

```
//Function :
mul(int l)
{
    extern y;
    int m;
    m = l * y;
    printf("\n Multiplication of B and Y is %d ", m);
}
```

अब निम्न प्रोग्राम को एक अलग Source File में लिख कर उसे extern.c नाम से save करते हैं।

Program

```
#include<stdio.h>
#include "extern.c"

int x = 6, y = 8, z = 4;

main()
{
    int b, c ;
    c = x + y;
    printf("\n C is %d ", c);
    b = y – z;
```

```
        printf("\n B is %d ", b);
        mul(b);
    }
```

इस प्रोग्राम को Execute करने पर भी वही Output प्राप्त होता है, जो पिछली बार प्राप्त हुआ था। लेकिन इस प्रोग्राम में हमने दो Source Files को Use किया है। mul Function को एक अन्य Source File extern.c में लिखा है। इस Function को main() Program में Use करने के लिए main() Program में #include" extern.c" Statement से mul Function की Program File को Link किया गया है।

"C" में किसी पहले से बनी Source File को दो तरीकों से main() Program File में Link किया जा सकता है। जब हमें जो File हमारी main() Program File में Use करनी है, वह यदि "C" की Standard Library में होती है, तो उसे < > चिन्ह के बीच में लिखते हैं।

लेकिन जब हमारी Source File Current Directory जिसमें हम Currently काम कर रहे हैं, उसमें हो, तब हम उस Function File को main() Program File से link करने के लिए उसे #include Keyword के साथ " " के अन्दर लिखते हैं। जैसा कि इस प्रोग्राम में किया गया है। हम हमारे द्वारा बनाए गए Functions को एक अलग Directory में भी Store करके रख सकते हैं।

जब हम हमारे Functions किसी अलग Directory में Store करके रखते हैं, तब हमें main() Program से उस Function File को Link करने के लिए उस File का पूरा path < > या " " चिन्ह के बीच लिखना पडता है।

माना हमने C: Drive में एक My_Functions नाम का Folder बनाया है, और हमारे द्वारा बनाए गए सारे Functions हम इसी Folder में रख रहे हैं, तो इस Folder से किसी भी Function को main() Program में Link करने के लिए हमें निम्नानुसार Statement लिख कर हमारी जरूरत वाली File को main() Program में जोडना होगा।

```
#include <C:\my_Functions\ File name>      OR      #include "C:\my_Functions\ File name"
```

ध्यान रखें कि ये Code इसी प्रकार से लिखे जाते हैं। इनके बीच Space मान्य नहीं है। अब देखें कि mul Function में y को Declare नहीं किया गया है बल्कि extern y; लिखा गया है। ये Statement "C" Compiler को बताता है कि जो Variable y यहां Use हो रहा है वह Global Variable है और उसे किसी अन्य Source File में Declare किया गया है। जब इस File को main() Program File से Link किया जाता है, तब Program Control स्वयं ही y का मान main() Program File से प्राप्त कर लेता है और यहां पर Calculation की जाती है।

इस प्रकार से extern Storage Class में Variable को Use किया जाता है। ध्यान रखें कि extern का प्रयोग हमेंशा Secondary Reference बनाने में किया जाता है। इसलिए किसी भी Global Variable का नाम एक प्रोग्राम में केवल एक ही बार Use करना चाहिये अन्यथा "C" Compiler Confuse हो जाएगा कि किस Variable का मान वह Access करे।

Static Storage Class

हम जिन Variables को static Class में रखते हैं उन Variables मे Store किये गए मान प्रोग्राम के अंत तक Memory में रहते हैं। किसी Variable को static Declare करने के लिए हम "C" के static Keyword का प्रयोग करते हैं। जैसे static int value; एक static Class में Declare Variable, Local भी हो सकता है और Global भी।

हम किसी भी Variable को static Declare कर सकते हैं। एक Local Variable के रूप में एक static Variable, Automatic Class के Variable की तरह ही काम करता है। लेकिन फिर भी auto व static प्रकार के Variables में कुछ अन्तर होता है। static प्रकार के Variable की प्रकृति को अच्छी तरह से समझने के लिए हम नीचे दो प्रोग्राम बना रहे हैं।

Program

```
#include<stdio.h>
main()
{
    int j;
    auto int k = 0;
    clrscr();

    for ( j = 0; j < 3; j++ )
    {
        k = k + 1;
        printf("\n %d", k);
    }
    getch();
}
```

Output

```
1
1
1
```

```
#include<stdio.h>
main()
{
    int j;
    static int k = 0;
    clrscr();

    for ( j = 0; j < 3; j++ )
    {
        k = k + 1;
        printf("\n %d", k);
    }
    getch();
}
```

Output

```
1
2
3
```

हमने हमेंशा देखा है कि जब भी किसी Loop के प्रथम Iteration के बाद पुनः Program Control दूसरे Iteration के लिए उस Loop में प्रवेश करता है तो Loop के अंदर प्रयोग किये गए किसी भी Variable का मान पुनः वही हो जाता है जो Initialize कर दिया जाता है।

जैसा कि पहले प्रोग्राम में हुआ है। इस पहले प्रोग्राम में k को auto प्रकार का लिया है, इसलिए जैसे ही Program Control इस Loop के Statement Block से बाहर जाता है, k का मान वापस 0 Initialize हो जाता है। लेकिन जो Variable **static** प्रकार की Storage Class में Declare किये जाते हैं उन Variables में मान Program के अन्त तक रहते है।

यही कारण है कि जहां प्रथम प्रोग्राम के Output में k का मान Loop के तीनों Iteration में समान आ रहा है वहीं दूसरे Program में Loop के हर Iteration में k का मान बदल रहा है। क्योंकि Static Variable में Store मान Program Control के Loop से बाहर जाने के बावजूद भी नहीं मिटता है। Program Execution के दौरान सर्वप्रथम k का Declare होता है साथ ही k को प्रारम्भिक मान 0 Initialize किया जाता है।

Program Control जब for Loop में प्रवेश करता है तो k = k + 1; **Expression** से k का मान Increment होकर 1 हो जाता है। Program Control k के मान को Print करता है और Statement Block से बाहर आ जाता है।

वापस जब दूसरे Iteration के लिए Program Control इस Statement Block में प्रवेश करता है तो वापस उसे k = k + 1 **Expression** मिलती है। Program Control वापस k का मान एक और बढाता है।

लेकिन हमने k को इस प्रोग्राम में Static Storage Class के अंतर्गत Declare किया है इसलिए जब Program Control वापस इस Statement Block में प्रवेश करता है तो k का पुराना मान नष्ट नहीं होता बल्कि स्थिर रहता है। इस Statement से वापस 1 अंक Increment हो कर k का मान अब 2 हो जाता है और ये क्रम तब तक चलता रहता है जब तक कि for Loop की Condition false ना हो जाए यानी Loop Terminate ना हो जाए।

Register Storage Class

हम जिन Variables को Register Storage Class में रखते हैं, उन Variables मे Store मान Memory में Store ना हो कर C. P. U. Register में Store होते हैं। C. P. U. Resister में Store होने वाले Variable की Access Rate या Access गति Memory की तुलना में बहुत अधिक होती है। ये केवल Local Variable के साथ ही प्रयोग होते हैं और किसी भी प्रोग्राम में Resister Variable की संख्या 14 से अधिक नहीं हो सकती है।

यदि हम 14 से अधिक Variable को Register प्रकार का Declare करते हैं तो Program Control स्वयं ही पिछले Variables को **auto** प्रकार के Variables में Convert कर देता है। यानी यदि हम किसी प्रोग्राम में 15 Variable को Register प्रकार का Declare करेंगे तो सबसे पहला Variable auto प्रकार में Convert हो जाएगा।

क्योंकि जैसे–जैसे हम नए–नए Variables Register प्रकार के Declare करते जाते हैं, हमारे पुराने Variables auto प्रकार के होते जाते हैं। किसी Variable को Register प्रकार का Declare करने के लिए हम register Keyword का निम्नानुसार प्रयोग कर सकते हैं:

```
register int num;
```

Exercise:

1. Recursion से आप क्या समझते हैं? Recursive Function का प्रयोग करते हुए 1 से 10 तक की गिनती Print करने का Function लिखिए साथ ही Function के Flow को भी समझाईए।
2. पांच अंकों की एक संख्या Input कीजिए और उस संख्या के सभी अंको का योग Recursive Function का प्रयोग करके ज्ञात करने का Function लिखिए।
3. Recursive Function का प्रयोग करते हुए **n** तक की Fibonacci Series Output में Print कीजिए, जबकि **n** User Input करता है।
4. Keyboard से Input की गई संख्या का Binary Equivalent ज्ञात करने का Function Create कीजिए।
5. किसी Function में Arguments Pass करने के कितने तरीके होते हैं? नाम लिखते हुए संक्षेप में समझाईए।
6. Storage Class किसे कहते हैं? ये कितने प्रकार के होते हैं?
7. Variables कितने प्रकार के होते हैं? एक उचित Program के आधार पर समझाईए।
8. विभिन्न प्रकार की Storage Class को उचित उदाहरणों के आधार पर समझाईए।

POINTERS

Pointers

जैसाकि नाम से ही स्पष्ठ हैं, Pointer किसी को Point करने का काम करता है। "C" में भी Pointer यही काम करता है। Pointer एक ऐसा Variable होता है जो किसी Data Item के मान को नहीं, बल्कि उस Data Item के Memory में Store होने की Memory Location या Memory Address को Store करता है। आइये इसे समझने की कोशिश करते हैं।

Computer प्रोग्राम की Instructions का उपयोग करने के लिए Memory का प्रयोग करता है। यानी हम चाहे Key Board की एक Key ही Press करें या पूरा का पूरा प्रोग्राम Execute कर दें, Computer सर्वप्रथम उस Instruction को Memory में Store करता है। आइये समझते हैं कि मेमोरी क्या होती है?

Computer एक Electronic Machine है। हम इसमें जो भी काम करते हैं, वे सभी Electrical Signals के उतार चढाव के रूप में Computer में Input होते हैं। इन Signals को जब किसी प्रारूप में दर्शाया जाता है, तो उस प्रारूप को Binary Digits कहते हैं। Computer Binary बीजगणित के अनुसार ही काम करता है। Binary गणित में केवल दो ही अंक 0 व 1 होते हैं। 0 का अर्थ है Negative या Signal नहीं है और 1 का अर्थ है Positive या Signal उपस्थित है। इन 0 व 1 को विभिन्न समूह में व्यवस्थित करके विभिन्न अंक अक्षर आदि बनाए जा सकते हैं।

चूंकि हम जितने भी अंक अक्षर व Special Symbols उपयोग में लेते हैं, उनकी कुल संख्या 256 है। (यानी A to Z तक के 26 Capital Letters, 26 Small Letters, 0 से 9 तक के कुल 10 अंक, कुछ Special Symbols जैसे कि # $ % @ & ^ * ( ) _ + आदि, व Alt के साथ प्रयोग होने वाले विभिन्न चिन्ह मिला कर कुल 256 चिन्ह) इन 256 अक्षरों के समूह में सभी आवश्यक अंक अक्षर व Special Symbols आ जाते हैं।

चूंकि Computer इन अंको, अक्षरो व Special Symbols को ज्यों का त्यों Accept नहीं कर सकता। क्योंकि कम्प्यूटर केवल Signals के उतार चढाव को ही समझता है। Signals के उतार चढाव को केवल Binary रूप में ही Computer को समझाया जा सकता है और Binary अंक तो मात्र दो ही होते हैं। फिर क्या किया जाए? हम देखते हैं कि इन दो Binary अंकों का समूह बनाया जाए तो निम्न समूह बनते हैं:

```
00    10    01    11
```

यानी दो अंको को विभिन्न क्रम में रख कर हम केवल (2^2) चार समूह प्राप्त कर सकते हैं, जबकि हमें 256 चिन्हों के लिए Binary Digits के समूह की जरूरत है। इसलिए हम दो की बजाय तीन के समूह में इन Binary अंकों को व्यवस्थित करके देखते हैं।

```
000   001   011   010   100   110   111   101
```

अब दर्शाए अनुसार 8 समूह (2^3) बनते हैं जो कि हमारे वांछित चिन्हों 256 से काफी कम हैं। इसीलिए इसी क्रम को और आगे बढाया जाता है और इन्हीं दो Binary Digits को चार–चार के समूह में रखा जाए तो कुल (2^4) 16 समूह बनते हैं।

यदि हमं इन Binary Digits को आठ–आठ के समूह में रखें तो (2^8) हमारे 256 चिन्हों के लिए 256 Binary Digits के भिन्न–भिन्न समूह प्राप्त हो जाते हैं। यानी यदि क्रम से 0 व 1 को आठ बार भिन्न–भिन्न प्रकार से लिखा जाए और आठ Digits के हरेक समूह को एक चिन्ह प्रदान कर दिया जाए तो हम हमारे चिन्हों, अंको व अक्षरों को Computer में Binary Digits के रूप में भेज सकते हैं।

इन Binary Digits को Computer भाषा में Bits कहते हैं और चूंकि आठ भिन्न प्रकार की Bits से विभिन्न 256 चिन्ह प्राप्त किये जाते हैं, इसलिए Memory में भी इस प्रकार की व्यवस्था की गई है कि Input के रूप में आने वाला हर Binary Digits का समूह, आठ–आठ के समूह में ही Store हो। इन्हीं आठ–आठ Digits के समूह को Byte कहा जाता है।

यानी एक Byte में आठ Bit होते हैं। आठ Bit के होने का मतलब है कि भिन्न प्रकार के 0 व 1 का समूह Memory में Input हुआ है या आठ Signals का एक समूह Input हुआ है, जिसमें कुछ Signal में Voltage है व कुछ Signals में Voltage नहीं है।

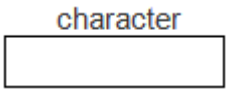
Compute की Memory **'Storage Cells'** का एक क्रमिक समूह होता है और हर Cell एक Byte को इंगित करता है। Memory के हर Storage Cell का एक **Unique Number** होता है, जिसे उस Memory या Storage Cell का **Address** कहते हैं। इस प्रकार से Memory के हर Storage Cell या हर Byte का एक Unique Address होता है और इस Address की कोई इकाई (Unit) नहीं होती है। Pointer द्वारा हम इसी Address के साथ प्रक्रिया करते हैं।

Understanding Pointers

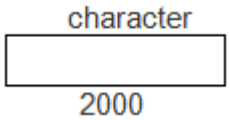
जब भी हम किसी Variable को Declare करते हैं तो वह Variable Memory में किसी Storage Cell में जा कर Store हो जाता है। वह Variable Memory में जिस Storage Cell पर जा कर Store होता है, उस Storage Cell का एक Unique Address होता है। Pointer द्वारा हम उस Storage Cell के Address को Access करते हैं।

इस बात को हम एक उदाहरण द्वारा समझते हैं। माना हमने एक char प्रकार का Variable character Declare किया, तो वह Memory में निम्नानुसार एक Byte की Space Reserve करेगाः

```
char character;
```

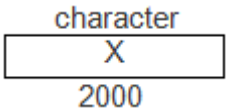


माना इस Storage Space की Storage Cell का Address 2000 है। इसे हम निम्नानुसार प्रदर्शित कर सकते हैं।



अब यदि हम इस character में कोई अक्षर Input करें तो वह अक्षर निम्नानुसार Store होगा–

```
character = 'x';
```



यदि हमें इस Variable (character) को Access करना हो तो उसे हम दो तरीकों से इसे Access कर सकते हैं।

1 हम इस Variable (character) का नाम Use करके character में Store अक्षर को Access कर सकते हैं।
2 हम इस Variable (character) के Storage Address को Use करके Character में Store अक्षर को Access कर सकते हैं।

यदि सामान्य तरीके से Variable (character) में Stored अक्षर x को प्रिंट करना हो तो हम निम्न Statement लिखते हैं:

```
printf("%c", character );
```

लेकिन यदि इस Variable (character) में Store अक्षर x को इसके Storage Cell की Address द्वारा Output में Print करना हो, तो ये काम हम सामान्य तरीके से नहीं कर सकते। इसके लिए हमें एक ऐसा Variable Declare करना होगा जो किसी Variable की Storage Cell का Address अपने में Store करके रखता हो। यानी उस Variable में (character Variable) की

तरह कोई character Store ना हो कर इस Variable (character) के Storage Cell का Address Store हो।

जब हमें ऐसा Variable Declare करना होता है, जो value के रूप में किसी अन्य Variable की Storage Cell का Address Memory में Store करता है, तो इस प्रकार के Variable को **Pointer Variable** कहते हैं या फिर हम ये भी कह सकते हैं, कि Pointer Variable एक ऐसा Variable होता है, जो Value के रूप में किसी अन्य Variable द्वारा Reserve की गई Space की Storage Cell का Address ग्रहण करता है।

Defining Pointers

जब हमें किसी Variable में किसी अन्य Variable की Storage Cell का Address Store करना होता है, तब हम इस Variable को Pointer प्रकार का Declare करते हैं। Pointer Variable भी उसी प्रकार से Declare किये जाते हैं, जिस प्रकार से अन्य सामान्य Variables Declare किये जाते हैं। किसी Variable को Pointer प्रकार का Declare करने के लिए उसे * चिन्ह के साथ Declare करते हैं। इसके Declaration का Syntax निम्नानुसार होता है–

```
Data Type *Pointer_Variable_name ;
```

ऊपर बताए उदाहरण में यदि हमें Variable (character) की Storage Cell का Address 2000 किसी Variable में Store करना हो, तो एक अन्य Variable **ptr** Declare करना होगा, जो कि Pointer प्रकार का होगा। इसे हम निम्नानुसार करेंगेः

```
char *ptr;
```

किसी भी Variable के Storage Cell के Address द्वारा उस Variable के मान को Access करने के लिए उस Variable के Storage Cell के Address को किसी अन्य Pointer प्रकार के Variable में Store करना इसलिए जरूरी होता है, क्योंकि हम किसी भी Memory Storage Cell को बिना किसी Variable में Store किये Access नहीं कर सकते।

यहां ध्यान से इस Pointer Variable के Declaration को देखें तो सवाल दिमाग में आ सकता है कि एक Pointer Variable में केवल किसी अन्य Variable का Address ही Store हो सकता है और कोई भी Address एक Unsigned Integer ही होता है, तो फिर यहां इस Pointer ptr को char प्रकार का क्यों Declare किया गया है, int प्रकार का क्यों नहीं किया गया?

ऐसा इसलिए किया गया है क्योंकि ये Data Type "C" Compiler को बताता है, कि इस Pointer Variable में जिस Variable की Storage Cell का Address Store होगा, वह char प्रकार का

ही हो सकता है। यानी हम एक int प्रकार के Variable के Address को char प्रकार के Pointer Variable में Store नहीं कर सकते हैं।

char प्रकार का ये Pointer Variable "C" Compiler को ये बताता है कि हम जिस Variable के Address को Access कर रहे हैं, वो char प्रकार का है और char प्रकार का ये Variable, Memory में एक Byte की Space Reserve करता है।

इसी प्रकार यदि हमें किसी int प्रकार के Variable का Address किसी Pointer Variable में Store करना हो तो Pointer Variable को भी int प्रकार का ही Declare करना होगा। तभी "C" Compiler समझ पाएगा कि इस Pointer Variable में जिस Variable का Address Store है वह int प्रकार का मान Store करता है और Memory में दो Byte की Space Use करता है।

Accessing the Address of the Variable

Pointer Variable Declare करने के बाद हमें उस Pointer Variable में उस Variable का Address Store करना पडता है, जिसे Pointer द्वारा Access करना है। यहां हम char प्रकार के Variable (character) में Stored अक्षर को Output में Print करना चाहते हैं।

इसलिए Pointer Variable, ptr में हमें Variable (character) का Address Store करना होगा। हम जानते हैं कि **&** एक **Address Operator** है। इसे सामान्य तौर पर **Address of the Variable** भी कहा जा सकता है। यानी जब किसी Variable में कोई मान Store करना होता है, तब हम & के बाद उस Variable का नाम लिखते हैं, जिसमें कोई मान Store करना है। जैसे निम्न Statements देखें

```
int num;
```

यदि हमें इस num में कोई मान Store करना हो तो हम

```
scanf("%d", &num);
```

लिखते हैं। ये Statement "C" Compiler को बताता है कि Key Board से Input होने वाले मान को Memory के उस Storage Cell में ले जाकर Store कर दो जिसका नाम num Declare किया गया है। यानी & Input होने वाले मान के Store होने की Location या Address Compiler को बताता है।

माना num नाम का Variable Memory में जिस Storage Cell में Store होता है उस Storage Cell का Address 2002 है, तो & Operator "C" Compiler को Inputted मान को Storage Cell 2002 का Address प्रदान करता है और Inputted मान इस Memory Location पर

Store हो जाता है, जिसका नाम num Declare किया गया है। इसी & Operator का प्रयोग करके हम एक Pointer Variable में किसी Variable की Storage Cell का Address Store कर सकते हैं।

यही प्रक्रिया यहां char प्रकार के Variable (character) की Storage Cell का Address Pointer Variable ptr में Store करने के लिए अपनाई गई है। यानी

```
ptr = &character;
```

इस Statement से Variable (character) का Address Pointer Variable ptr में Store हो जाता है। ये बात हमेंशा ध्यान रखें कि हम किसी भी Variable का Address उसी Variable में Store कर सकते हैं जिसे Pointer प्रकार का Variable Declare किया गया हो। इसलिए Address Initialization से पूर्व Variable का Pointer प्रकार का Declare होना जरूरी होता है।

यदि हम कोई Variable Declare करने के बाद केवल ये जानना चाहते हैं, कि वह Variable किस Address पर Store हुआ है और उस Address को किसी Pointer Variable में Store करना नहीं चाहते, तो हमें **%u** Control String को Use करना पडता है और उस Variable का Address पता करने के लिए & Operator का प्रयोग करना पडता है।

जैसे हम ये जानना चाहते हैं कि हमने char प्रकार का जो Variable character Declare किया है, वह किस Memory Location या किस Address पर Store हुआ है तो हमें निम्न Statement देना होगा–

```
printf(" The Address Of Variable Character is %u ", &character);
```

इस Statement में %u का प्रयोग इसलिए किया जाता है, क्योंकि कोई भी Memory Location हमेंशा एक बिना चिन्ह वाली संख्या होती है और हम किसी Storage Cell के साथ कोई Calculation नहीं कर सकते हैं। ये Statement Output के रूप में उस Storage Cell का Address Print करता है, जिस Storage Cell में character नाम का Variable Store होता है।

Accessing a Address Through It's Pointer

Pointer Variable Declare करके उस Pointer Variable को वांछित Variable का Address Initialize करने के बाद, जब हमें Pointer Variable को Access करना होता है, तब हमें * **Indirection Operator** को Use करना पडता है। इसे सामान्य तौर पर **Value at the Address** Operator भी कहा जाता है।

यानी किसी Pointer Variable में Stored Address जिस Variable का है, उस Variable का मान प्राप्त करने के लिए इस Operator का प्रयोग किया जाता है। जैसे ptr में character नाम के Variable का Address Stored है और हम ptr द्वारा character में Store अक्षर x को Output मे Print करना चाहते हैं तो हमें निम्न Statement देना होगा–

printf("\n Value At the Address %u of Variable character is %c", ptr, *ptr);

यहां **ptr** Output मे character की Storage Cell का Address Print करता है और ***ptr**, Output में अक्षर character नाम के Variable में Store अक्षर x को Print करता है। *ptr "C" Compiler को ये बताता है कि Pointer Variable ptr में जिस Variable के Storage Cell का Address Store है (यहां Pointer Variable में character के Storage Cell का Address Store है।) उस Address पर जो मान स्थित है (यहां अक्षर x Stored है) उसे Output में Screen पर Print कर दो।

इस प्रकार से किसी Pointer Variable में स्थित Address जिस Variable का हो उस Variable का मान Output में यदि Print करना हो, तो हमें * Operator के साथ उस Pointer Variable को Use करना पडता है। अब हम एक उदाहरण देखते हैं जिसमें Address Operator & व Indirection Operator * को साथ में प्रयोग करके कुछ Assignment किये गए हैं।

Program

```
#include<stdio.h>
main()
{
    char character = 'x';
    char *ptr;
    char new1;
    ptr = &character;
    clrscr();
    printf("\n Address of character is %u ", &character);
    printf("\n Address in Pointer Variable ptr is %u ", ptr);
    printf("\n Value in the character is %c ", character);
    printf("\n Value at the Address %u Stored is %c ", ptr,*ptr);
    new1 = *ptr;
    printf("\n Value in the new1 is %c ", new1);
}
```

Output

```
Address of character is 65535
Address in Pointer Variable ptr is 65535
```

```
Value in the character is x
Value at the Address 65535 Stored is x
Value in the new1 is x
```

इस Output से स्पष्ट है कि Variable character का Output 65535 है और & character **Expression** से Pointer Variable ptr में character का Address Store हो गया है। इसीलिए दूसरे Output में ptr में स्थित Address का मान भी 65535 है। तीसरे Output में character Variable में Stored अक्षर x को सामान्य तरीके से Print किया गया है।

चौथे Output में ये बताया गया है कि Pointer Variable ptr में 65535 Address Stored है। (हमें प्रथम Output से पता है कि ये character नाम के Variable की Storage Cell का पता है।) इस Address पर अक्षर x Stored है। इस Output को *ptr Argument द्वारा Output में Print किया गया है, जो कि वास्तव में character का ही मान Print कर रहा है।

new1 = *ptr; Statement द्वारा new1 में Pointer Variable ptr में Stored Address के Variable का मान Assign किया गया है जो कि character x है और इसे ही पांचवे Output में Print किया गया है।

इस प्रकार से & जहां **Address of the Variable** बताता है यानी कि कोई मान किस Address पर Stored है। वहीं * **Value at the Address** बताता है यानी कि किसी Address पर क्या मान Stored है।

Pointer Expressions

यह बात हमेंशा ध्यान रखें कि कभी भी Pointers की आपस में कोई Calculations नहीं होती है बल्कि इन Pointers में जिन Variables के Address होते हैं, उनके मानों की आपस में गणना होती है। जैसे

```
int b, c, d;
int *bp, *cp, *dp;
bp = b;
cp = c;
dp = d;
b = 6;
c = 8;
```

अब निम्न गणनाएं देखें

```
d = *bp + *cp;
```

इस **Expression** से d का मान d = 6 + 8 यानी 14 होगा।

```
c = c + *cp;
```

इस **Expression** से c का मान c = 8 + 8 यानी 16 हो जाएगा।

```
*bp = c + b;
```

इस **Expression** से b का मान b = 8 + 6 यानी 14 हो जाएगा।

```
(float)*dp = (float)c / *bp;
```

इस **Expression** से d का मान float में प्राप्त होगा और c का मान float में बदल जाएगा । उसके बाद b के मान 6 का c के मान 8 में भाग दिया जाएगा। प्राप्त मान d में Store हो जाएगा। इस प्रकार से Pointer Variable की भी Calculations होती है लेकिन उन Calculations का असर Pointer में Store Address पर नहीं होता बल्कि जिन Variables के Address Pointer में Store हैं उन Variables के मानों पर होता है।

कोष्ठक में लिखी गई **Expressions** पहले Calculate होती हैं क्योंकि कोष्ठक को प्राथमिकता क्रम में सर्वप्रथम रखा गया है। Pointer के साथ भाग की क्रिया करते समय विशेष ध्यान रखना चाहिये क्योंकि यदि * व / के बीच Space ना रखा जाए तो ये Comment देने वाले चिन्ह /* में बदल जाता है और Program सही होते हुए भी Execute नहीं होता है।

Addition and Subtraction A Number to a Pointer

इसे समझने के लिए एक उदाहरण देखते हैं–

Program

```
#include<stdio.h>
main()
{
    int j = 10, *k;
    k = &j;
    printf("\n The Address of k is %u" , k);
    printf("\n The Value of j is %d" , j);
    k = k + 3;
    printf("\n Now The Address of k is %u" , k);
```

```
        k = k – 3;
        printf("\n Now The Address of k is %u" , k);
        getch();
    }
```

Output

```
The Address of k is 65524
The Value of j is 10
Now The Address of k is 65530
Now The Address of k is 65524
```

इस प्रोग्राम में हमने j का मान 10 Assign किया है और j का Address k में Store किया है। k में Stored Address 65524 है। जब हमने k में 3 जोडा और k का मान Print किया तो k का मान 65530 हो गया जबकि ये मान 65527 ही होना चाहिये था। ऐसा इसलिए होता है क्योंकि यहां Pointer अपने **Scale Factor** के अनुसार बढ रहा है।

चूंकि k में Stored Address int प्रकार का है इसलिए k का मान 3 अंक बढाने का मतलब है, k के Address को 3 अंक बढाना और k में Stored Address int प्रकार का होने से ये मान तीन के बजाय 6 बढा है, क्योंकि Pointer Scale Factor के कारण Address Data Type के अनुसार ही Increase या Decrease होता है।

इसी प्रकार से जब k में से 3 घटाया जाता है, तब वास्तव में ये Statement "C" Compiler को बताता है कि k में Stored Address से Data Type के अनुसार तीन Address पीछे जाना है। यहां int प्रकार का Data Type है। इसलिए Scale Factor के अनुसार k का मान वापस 65530 हो जाता है। इस प्रकार से किसी Pointer में Stored Address का मान Scale Factor के अनुसार घटता या बढता है।

Pointer Increment and Scale Factor

जब Pointer को Increase या Decrease करना होता है, तो ये सामान्य Variables की तरह Increase या Decrease नहीं होते हैं। जब Pointer को Increase या Decrease किया जाता है तो इनका Increment या Decrement इस बात पर निर्भर करता है कि Pointer Variable किस प्रकार के Data Type के Variable का Address ग्रहण करेगा।

जैसे int प्रकार का Pointer दो byte की Space Reserve करता है इसलिए यदि इस Pointer Variable को Increase या Decrease किया जाए, तो Pointer में स्थित Address दो–दो के क्रम में बढेगा या घटेगा।

इसी प्रकार यदि char प्रकार का Pointer हो तो वह एक–एक के क्रम में बढेगा या घटेगा। यदि Pointer Variable में किसी double प्रकार के Data Type का Address Store है, तो इस Variable को Increase या Decrease करने पर Pointer में Store Address आठ–आठ Byte के क्रम में Increase या Decrease होगा। इसे Pointer का **Scale Factor** कहते हैं।

Scale Factor को हम निम्नानुसार एक प्रोग्राम द्वारा समझने की कोशिश करते हैं। इस प्रोग्राम के Output में आप देख सकते हैं कि ch एक char प्रकार का Variable है और इसका Address Increase करने पर ये एक–एक के क्रम में बढता है क्योंकि char प्रकार का Variable Memory में एक Byte लेता है।

जबकि int प्रकार के Variable का Address दो–दो के क्रम में Increase हो रहा है और float प्रकार के Data type के Variable का Address चार Byte के क्रम में Increase हो रहा है क्योंकि float प्रकार का Data Memory में चार Byte की Space लेता है।

Program

```
#include<stdio.h>
main()
{
    char ch, *chp = &ch;
    int num, *nump = &num;
    float digit, *digitp = &digit;
    clrscr();
    printf("\n Address of ch is %u", chp);
    chp++;
    printf("\n After Increasing the Address of ch is %u", chp);
    chp++;
    printf("\n After Increasing the Address of ch is %u", chp);
    printf("\n \n Address of integer num is %u", nump);
    nump++;
    printf("\n After Increasing the Address of integer num is %u", nump);
    nump++;
    printf("\n After Increasing again Address of integer num is %u", nump);
    printf("\n \n Address of float digit is %u", digitp);
    digitp++;
    printf("\n After Increasing the Address of float digit : %u", digitp);
    digitp++;
    printf("\n After Increasing again Address of float digit :%u", digitp);
    getch();
}
```

Output

```
Address of ch is 65525
After Increasing the Address of ch is 65526
After Increasing the Address of ch is 65527

Address of integer num is 65522
After Increasing the Address of integer num is 65524
After Increasing again Address of integer num is 65526

Address of float digit is 65518
After Increasing the Address of float digit is 65522
After Increasing again Address of float digit is 65526
```

Exercise:

1 Pointers को समझाते हुए एक Program द्वारा समझाईए कि हम किसी Variable को उसके Address द्वारा किस प्रकार से Access कर सकते हैं।

2 Pointer Expression को समझाईए।

3 Scale Factor से आप क्या समझते हैं? एक उदाहरण द्वारा Pointer Increment को समझाईए।

Function with Arrays

जिस प्रकार एक Function में किसी साधारण Variable का मान Argument के तौर पर Pass करते हैं, उसी तरह से एक Array को भी किसी Function में Argument के रूप में Pass कर सकते हैं। Argument के रूप में Array का नाम व Array की Size Actual Argument के रूप में User Defined Function को Pass की जाती है। जब हमें किसी User Defined Function में Array का मान प्राप्त करके उस पर प्रक्रिया करनी होती है, तब हम User Defined Function में Array का नाम व Array की Size Receive करते हैं। इसलिए User Defined Function को निम्न Format में Define करना होता है।

```
Return_Data_Type Function_Name ( Array [ ] , size);
Data_Type Array [ ] , size;
{
    Statement Block ;
    return ( Expression ) ;
}
```

इस प्रारूप के अनुसार हम एक User Defined Function लिखते हैं, जिसमें Array [ ] में प्राप्त हुए सभी मानों में से सबसे बडा मान Calling Function को Return करना है। Function निम्नानुसार है:

```
float  largest ( float Array [ ] , int size)
//float Array [ ] ;
//int size;
{
    int j;
    float max;

    max = Array [0] ;

    for (j = 1; j < size; j++)
        if (max < Array [i])
            max = Array [i];

    return (max) ;
}
```

अब एक प्रोग्राम द्वारा इस Function को Use करते हैं। हम एक प्रोग्राम लिखते हैं, जिसमें किसी Array में 10 मान Input करने हैं और उन 10 मानों में से सबसे बडे मान को Output में Print करवाना है। सबसे बडा मान ज्ञात करने के लिए इस Function को Use किया गया है।

Program

```
#include<stdio.h>

main()
{
    float value[10], large;
    int j;

    for ( j = 0; j < 10; j++)
    {
        printf("\n Enter %d Value ", j+1);
        scanf("%f", &value[j]);
    }

    large = largest (  value,   10 );
    printf("\n Largest Value is %f", large);
    getch();
}

//User Defined Function
float  largest ( float Array [ ] , int size)
{
    int j;
    float max;

    max = Array [0] ;

    for (j = 0; j < size; j++)
        if (max < Array [j])
            max = Array [j];

    return (max) ;
}
```

इस प्रोग्राम में main() Function में value नाम के Array में दस मान Input किये जाते हैं। फिर largest Function को Call किया जाता है। यहां largest Function में value नाम के Array मे Input किये गए सभी Elements, largest Function के Array नाम के Variable में चले जाते हैं

और Array की Size 10 largest Function के size नाम के Variable में चली जाती है। यानी Array = value व size = size हो जाता है।

Program Control जब User Defined Function में आता है, तब Array में value के सभी Element आ जाते हैं व Array की size 10 हो जाती है, जो कि main() Function से प्राप्त हुई है। अब इस User Defined Function में दसों Elements में से Largest मान Calculate होता है, और वह मान max नाम के Variable में आ जाता है।

max नाम के Variable के मान को पुनः main() Function को Return किया जाता है, जिससे main() Function में max का मान large नाम के Variable को प्राप्त हो जाता है। फिर large नाम के Variable के मान को Output में Print करवा दिया जाता है, जो कि दसों Elements में से सबसे बडा मान होता है।

जब main() Function के Actual Arguments, User Defined Function के Variables को Formal Arguments के रूप में प्राप्त होते हैं, तब User Defined Function में प्राप्त हुए Argument को Define करना जरूरी होता है, कि कौनसा Variable क्या है।

यानी जब value नाम के Array Variable से सभी Elements, User Defined Function के Array नाम के Variable को प्राप्त होते हैं, तब User Defined Function में ये Define करना जरूरी होता है, कि यहां Array नाम के Variable में किसी Array से Elements प्राप्त हुए हैं। इसलिए Array को भी हमें Array प्रकार का ही घोषित करना होता है। Array नाम के इस Variable, जो कि User Defined Function में Arguments लेने के लिए लिखा गया है, निम्न प्रकार से Declare किया जाता है।

```
float Array [ ] ;
```

जब Program Control इस User Defined Function में प्रवेश करता है, तो उसे पता चल जाता है, कि Array नाम के Variable में जो मान Calling Function से प्राप्त हुआ है, वह Array प्रकार का मान है।

यानी main() Function से value नाम के Array में Store किये गए सभी मान, इस User Defined Function के Array नाम के Variable में Copy हो गए हैं, और Array नाम का ये Variable main() Function में Declare Value नाम के Array Variable के Elements को Store किये हुए है।

इस Declaration में Array के Bracket को Empty रखा जाना जरूरी होता है, क्योंकि Array की Size एक अन्य size नाम के Variable में प्राप्त होती और ये size भी Argument के रूप में Calling Function से प्राप्त होती है। इस size के Data Type को भी Define करना जरूरी होता

है। Array की size हमेंशा पूर्णांक में होती है। इसलिए इस size को हमेंशा int प्रकार के Data Type के रूप में Declare किया जाता है।

"C" Language में Strings के साथ विभिन्न प्रकार की प्रक्रियाओं को करने के लिए भी कुछ Standard Functions बनाए गए हैं और इन Functions को Library के रूप में हमें प्रदान किया गया है। सामान्यतया सर्वाधिक काम आने वाले चार Functions को हम यहां पर समझाने की कोशिश कर रहे हैं। ये Functions निम्नानुसार हैं:

strcat() Function

इस Function द्वारा हम दो Strings को आपस में जोड सकते हैं। इसका Syntax निम्नानुसार होता है:

```
strcat(str1, str2);

str1First String (Source String)
str2      Second String (Target String )
```

इस Function द्वारा str1 में str2 का String Add हो जाएगा जबकि str2 में कोई बदलाव नहीं होगा। हम Source String में सीधे ही String भी Store करवा सकते हैं। जैसे

```
strcat( remark, "GOOD");
```

साथ ही हम Strings की Nesting भी कर सकते हैं। जैसे

```
name1[] = {"Madhav"};
name2[] = {"Raghav"};
name3[] = {"Gopal"};
strcat( strcat( name1, name2 ), name2);
```

Output

```
MadhavRaghavGopal
```

इस Function को Computer की Library में निम्नानुसार Define किया गया है:

```
// strcat:  concatenate source to end of target;
// target must be big enough

  void strcat(char target[], char source[])
```

```
{
    int i, j;

    i = j = 0;
    while (target[i] != '\0') /* find end of target */
        i++;
    while ((target[i++] = source[j++]) != '\0')
      /* copy to target */
        ;
}
```

इस Function के काम करने का Logic ये है कि हम जिस Source Strings को Target String के अन्त में जोडना चाहते हैं, सबसे पहले हमें उस Target String के अन्त तक पहुंचना होता है। Target String के अन्त पर पहुंचने के लिए हमें एक Loop चलाना होता है, जो तब तक चलता है, जब तक कि Target String का अन्त यानी NULL प्राप्त नहीं हो जाता।

जब Control Target String के अन्त पर पहुंच जाता है, तब एक और Loop चलाया जाता है और इस Loop द्वारा Source String से एक बार में एक Character को Read किया जाता है और Target String में Store कर दिया जाता है।

ये प्रक्रिया तब तक दोहराई जाती है, जब तक कि Source String से Target String में Copy किया जाने वाला Character '\0' यानी NULL Character नहीं होता। जैसे ही इस दूसरे while Loop को NULL Character प्राप्त होता है, while Loop Terminate हो जाता है, क्योंकि NULL Character के मिलने का मतलब ही यही है, कि Copy की जा रही String Target String में Copy हो चुकी है और Copy होने के लिए Source String में एक भी Character नहीं है।

strcpy() Function

इस Function द्वारा हम दूसरे Strings को प्रथम String में Copy कर सकते हैं। इसका Syntax निम्नानुसार होता है:

```
strcpy(str1, str2);
str1      First String (Source String)
str2          Second String (Target String )
```

इस Function द्वारा str1 में str2 का String Copy हो जाएगा जबकि str2 में कोई बदलाव नहीं होगा। हम Source String में सीधे ही String भी Store करवा सकते हैं। जैसे

```
strcpy( remark, ”GOOD”);
```

या किसी अन्य Array के String को भी Copy कर सकते हैं। जैसे–

```
name1[] = {“kuldeep”};
name2[10];
strcpy( name2, name1);
```

इस Statement से Array name1 का String Array name2 में Copy हो जाएगा जबकि Array name1 में कोई परिवर्तन नहीं होगा।

strlen() Function

इस Function द्वारा हम किसी Strings की Length ज्ञात कर सकते हैं कि उस String में या उस Variable ने कितने Byte Memory में लिए है। इसका Syntax निम्नानुसार होता है–

```
n = strlen(str1 or Identifier);
```

यहां n एक int प्रकार का Identifier होता है। जैसे–

```
char name[] = {“Dev”};
int n;
n = strlen(name);
```

यदि यहां n को Print करवाया जाए तो n का मान 4 प्राप्त होगा जो कि Array द्वारा Reserve की गई Memory बताता है।

strcom() Function

इस Function द्वारा हम दो Strings की आपस में तुलना कर सकते हैं। इसका Syntax निम्नानुसार होता है:

```
strcom(str1, str2);
```

```
str1 First String (Source String)
str2     Second String (Target String )
```

इस Function द्वारा str1 में str2 का String Comparison होगा और यदि तुलना में दोनों Strings के मान हर Elements पर समान हों तो ये Function 0 return करता है। यदि दोनों Strings के मान हर Location पर समान नहीं होंगे तो जिस Location पर इनका मान Change होता है, उन दोनों Locations के मानों का अन्तर Output में प्राप्त होता है। जैसेः

```
n = strcom("there", "their");
printf("\n %d", n);
```

हम characters के साथ गणितीय गणनाएं भी कर सकते हैं, क्योंकि Characters Memory में ASCII Numbers के अनुसार ही स्टोर होता है। जैसे एक उदाहरण से इस बात को समझते हैं।

Program

```
main()
{
    char a, b;
    a = 'x';
    printf("\n A is %c", a);
    printf("\n A is %d", a);
    a = a – 1;
    printf("\n Now a is %c", a);
    printf("\n Now a is %d", a);
    getch();
}
```

Output

```
A is x
A is 120
Now a is w
Now a is 119
```

120 व 119 x व w की ASCII Value है, जिनका हम अपनी जरूरत के अनुसार विभिन्न उपयोग कर सकते हैं। कई बार हमें String के मान को Integer में बदलना पडता है। जैसे हमने किसी साल को string में Store कर रखा है और हमें उस साल के साथ गणितीय प्रक्रिया करनी हो तो हमें उस String को Integer में Convert करना पडता है। इस काम के लिए हम atoi() Function का प्रयोग करते हैं। जैसे –

```
char year[] = {"1998"};
```

अब यदि हमें Year में से 10 साल घटाना हो तो हम सीधे ही इसमें से 10 साल नहीं घटा सकते, क्योंकि Year को String के रूप में Store किया गया है। इसलिए इस String को पहले **int** में बदलना होगा फिर 10 साल घटाना होगा। ये काम हम निम्नानुसार कर सकते हैं–

```
char year[] = {"1998"};
int n;
n = atoi(year);
n = n-10;
```

अब यदि n को Print किया जाए तो हमें n का मान 1988 प्राप्त होगा। इस प्रकार के ढेर सारे Functions "C" में उपलब्ध हैं। चूंकि Strings भी एक 1-D Array में ही Store होता है इसलिए हम किसी Array की Traversing भी उसी प्रकार से कर सकते हैं जिस प्रकार से किसी Integer प्रकार के Array के विभिन्न मानों की Traversing करते हैं।

Working with Binary Digits

कई बार हमें किसी Integer मान की Binary Values के साथ प्रक्रिया करने की जरूरत पड जाती है। इस स्थिति में हमें किसी मान के Binary Equivalent को Screen पर Display करने की जरूरत पडती है। लेकिन "C" Language में कोई ऐसा Function नहीं है, जो किसी Integer मान की Binary को Screen पर Display करता हो। फिर भी "C" Language में उपलब्ध Bitwise Operators का प्रयोग करके हम एक ऐसा Function Create कर सकते हैं, जो किसी Integer संख्या की Binary Value को Screen पर Display कर सकता है।

The binary( ) Function

```
binary( int n )
{
    int  j, k, andMask;

    for ( j=15; j >= 0; j--)
    {
        andMask = 1 << j;
        k = n & andMask;
        k == 0 ? printf("0") : printf("1");
    }
}
```

इस Function में Formal Argument Accept करने के लिए Formal Variable n लिया है। किसी भी Operand के Bit-Pattern को 16 Digits में show किया जाता है और इस 16 Digit के Bit-Pattern में Bits हमेंशा Right Side से fill होना शुरू होते हैं और अन्तिम Bit हमेंशा Left Side में fill होता है। जबकि जिस क्रम में ये Bits fill होते हैं, उसी क्रम में यदि इन्हे Screen पर Print कर दिया जाए, तो Print होने वाला Bit-Pattern, Real Bit-Pattern से बिल्कुल विपरीत होगा।

इसलिए ये जरूरी है कि किसी भी Operand के Bit-Pattern की अन्तिम Location के Bit को Screen पर सबसे पहले Print किया जाए व प्रथम Bit को सबसे अंत में, ताकि Print होने वाला Bit.Pattern वास्तविक Bit.Pattern हो।

इसी वजह से हमने इस Function में एक for Loop चलाया है और इस Loop को Increment ना करके Decrement किया है ताकि सर्वप्रथम 15th Bit की ON या OFF स्थिति Print हो और Bit. Pattern के प्रथम Bit की स्थिति सबसे अंत में Print हो। इस for Loop के Variable का प्रारम्भिक मान 15 दिया है, ताकि Operand के Bit-Pattern की 15th Location का Bit सर्वप्रथम Print हो।

इस Loop के Variable को क्रम से Decrement किया गया है, ताकि जब 15th Location का Bit Print हो जाने के बाद 14th Location का Bit Print हो। इस प्रकार से ये क्रम 0 तक चलाया गया है, ताकि किसी भी Operand के Bit.Pattern की Bits 16 Digits तक Print हो।

यदि किसी Bit-Pattern को एक Bit Left में Shift किया जाए, तो Bit-Pattern के Operand का मान दुगुना हो जाता है। यदि इसी Concept का उपयोग किया जाए तो हम कह सकते हैं कि हमें एक ऐसे Bit-Pattern की जरूरत होगी जिसका मान Loop के प्रथम Iteration पर 1000000000000000 हो ताकि हम 15th Bit की ON या OFF स्थिति को Print कर सकें।

फिर Loop के दूसरे Iteration में इस Mask का Bit-Pattern 0100000000000000 हो ताकि हम 14th Bit की स्थिति को Print कर सकें और इसी क्रम में Loop के अन्तिम Iteration पर इस Mask का Bit-Pattern 0000000000000001 हो जाए, ताकि हम Operand के प्रथम Bit की स्थिति को Print कर सकें।

यदि हम प्रथम Bit के Pattern को देखें तो ये अंक 1 का Bit-Pattern है। यानी 000000000000000**1** Bit-Pattern को यदि Decimal में बदला जाए तो इसका मान 1 होगा। यदि इस अंक के Bit-Pattern को एक अंक Left Shift किया जाए तो ये मान 00000000000000**1**0 होगा, जो कि किसी Operand के दूसरे Bit को दर्शाएगा। इस कारण से हमने **`andMask = 1 << i;`** Statement प्रयोग किया है।

Loop के प्रथम Iteration में **j** का मान 15 होता है। इस वजह से ये Statement **`andMask = 1 << 15;`** हो जाता है। ये Statement अंक एक के Bit-Pattern का मान 15 Bit Left में Shift कर देता है, जिससे Bit-Pattern **1**00000000000000 हो जाता है। यदि इस संख्या को Decimal में Convert करें, तो इस Bit-Pattern का मान **32768** होता है, जो कि 15th Location का मान होता है। इस प्रकार से andMask का मान 32768 हो जाता है।

अब दूसरा Statement Execute होता है। **`k = n & andMask;`** Statement को यदि Expand करें, तो ये **`k = n & 32768;`** हो जाता है। अब Called Function से n का जो मान प्राप्त होता है उस मान के 15th Bit को Bitwise Operator & द्वारा Check किया जाता है कि 15th Bit की स्थिति क्या है ? **`k==0 ? printf("0") : printf("1");`** Statement द्वारा यदि इस 15th Location की Bit **OFF** होती है, तो Output में k का मान **0** Print हो जाता है अन्यथा k का मान **1** Print होता है।

दूसरे Iteration में for Loop के Variable का मान जब 14 हो जाता है, तब andMask = 1 << j; Statement द्वारा andMask का मान 16384 हो जाता है। **`k = n & andMask;`** Statement द्वारा Expand होता है और **`k = n & 16384;`** हो जाता है। अब Called Function से प्राप्त Variable n के मान की 14th Location की Bit Check होती है और Bit की स्थिति को Print कर दिया जाता है।

इस प्रकार से 13th , 12th 11th को Check करते हुए अंत में 0th Location पर के Bit को Check किया जाता है। यदि Bit On होता है तो 1 Print होता है और यदि Bit OFF होता है तो 0 print होता है। इस प्रकार से किसी भी Operand के Bit-Pattern को इस Function द्वारा Print किया जा सकता है।

किसी संख्या की Binary Value ज्ञात करने के लिए इस Function को हम निम्नानुसार Use कर सकते हैं:

Program

```
#include <stdio.h>
#include <conio.h>
void binary(int);

main()
{
    int bin;

    printf("Enter an Integer to get that BINARY equivalent : ");
    scanf("%d", &bin);
    printf("\n Binary of the %d is : ");
```

```
        binary(bin);
        getch();
    }

    //UDF
    void binary( int n )
    {
        int  j, k, andMask;

        for ( j=15; j >= 0; j--)
        {
            andMask = 1 << j;
            k = n & andMask;
            k == 0 ? printf("0") : printf("1");
        }
    }
```

Output

```
Enter an Integer to get that BINARY equivalent : 45
Binary of the 45 is : 0000000000101101
```

इस **binary**() को Modify करके हम इसे किसी बडी संख्या की Binary ज्ञात करने के लिए भी Use कर सकते हैं। Modified Function निम्नानुसार बनाया जा सकता है:

```
void binary( long n )
{
    long  j, k, andMask;

    for ( j=31; j >= 0; j--)
    {
        andMask = 1 << j;
        k = n & andMask;
        k == 0 ? printf("0") : printf("1");
    }
}
```

Subtraction One Pointer to another Pointer

किसी Array के दो Elements का Address यदि दो अलग–अगल Pointer Variables में Stored हो तो हम इन्हें आपस में घटा सकते हैं। एक Pointer में से दूसरे Pointer को घटाने पर प्राप्त होने वाला मान प्रथम Element से दूसरे Element के बीच की दूरी Bytes में बताता है। इसे समझने के लिए निम्न उदाहरण देखिये:

Program

```
#include<stdio.h>
main()
{
    int j[4], *k , *l;
    k = &j[1];
    l = &j[3];
    clrscr();
    printf("\n Address of j[1] is %u ", k);
    printf("\n Address of j[3] is %u ", k);
    printf("\n j[3] – j[1] =", j-i);
    getch();
}
```

Output

```
Address of j[1] is 65520
Address of j[3] is 65524
j[3] - j[1] = 2
```

इस उदाहरण में हम देखते हैं कि j[1] का Address 65520 व j[3] का Address 65524 है। j[3] - j[1] करने पर 65524 - 65520 होना चाहिये। लेकिन ऐसा नहीं होता, और इसका मान 2 प्राप्त होता है। ऐसा इसलिए होता है, क्योंकि किसी Pointer में से जब उसी Array के किसी अन्य Element के Pointer को घटाया जाता है, तब प्राप्त होने वाला मान Addresses की आपस की गणना का मान नहीं होता है, बल्कि ये मान उन दोनो Addresses के बीच की दूरी बताता है, कि दूसरा Element प्रथम Element से कितना दूर या कितनी Byte दूर स्थित है। यहां j[3], j[1] से दो Byte की दूरी पर स्थित है। क्योंकि ये int प्रकार का Array है और int Memory में दो Byte की Space लेता है।

Comparison of two Pointers

यदि किसी Array के एक ही Elements के दो Pointers हों तो उनका आपस में Comparison भी किया जा सकता है कि उन दोनों Pointers के Address समान हैं या नहीं। जैसे

Program

```
#include<stdio.h>
main()
{
    int j[4], *k, *l;
    k = &j[3];
    l = &j[j + 3];
    clrscr();

    if( k = = l)
        printf("\n Both Pointers  Are  Pointing the same Location");
    else
        printf("\n Both Pointers  Are  Not Pointing the same Location");
    getch();
}
```

ये बात हमेंशा ध्यान रखें कि कभी भी दो Pointers में Stored Addresses को

- आपस में जोड कर नया Address प्राप्त नहीं किया जा सकता।
- आपस में किसी स्थिरांक से भाग नहीं दिया जा सकता। और
- आपस में किसी स्थिरांक से गुणा नहीं किया जा सकता।

Array in Function through Pointer

Function को पढते समय हमने बताया था कि किस प्रकार से एक Array को Function के साथ उपयोग में लाया जा सकता है। यहां हम ये जानेंगे कि किसी Array को Pointer द्वारा किस प्रकार से किसी Function में Argument के रूप में भेजा जा सकता है। Pointer की ये विशेषता है कि यदि किसी Variable को Pointer द्वारा Function में भेजा जाता है, तो Argument के रूप में Array का Base Address ही Function को Pass होता है।

यदि एक सामान्य Formal Variable में इस Argument को Accept किया जाए तो ये Array का सामान्य तरह से उपयोग करना होता है, लेकिन यदि उस Base Address को किसी ऐसे Formal Variable में Accept किया जाए जो खुद एक Pointer Variable हो तो ये Array का Pointer हो जाता है। इसे समझने के लिए निम्न प्रोग्राम देखते हैं।

Program

```
#include<stdio.h>
main()
{
    int j[] = { 11,23,33,22,44,55,66};
    clrscr();
    display ( &j[0], 6 );
}

display ( int *m, int n )
{
    int k;
    for( k = 0; k < 5; k++ )
    {
        printf("\t Element = %d ", *m );
        m++;        //Increment Pointer to Point next Location
    }
}
```

Output

```
Element = 11
Element = 23
Element = 33
Element = 22
Element = 44
Element = 55
```

इस प्रोग्राम में Array j का Base Address व इसकी Size को Argument के रूप display() नाम के Function में भेजा गया है। जो कि क्रमशः *m व n नाम के Formal Arguments को प्राप्त हो गए हैं जिन्हे User Defined Function में main() Function से आ रहे Arguments को Accept करने के लिए Declare किया गया है।

ध्यान दें कि Variable m को Pointer Variable Declare किया गया है, क्योंकि Argument के रूप में Array का Base Address आ रहा है और Address को केवल Pointer Variable ही ग्रहण कर सकता है। जब display() Function का for Loop चलता है तब प्रथम Iteration में Variable m में प्राप्त Base Address के मान को Print कर देता है। फिर m का मान Increment किया गया है।

इससे m का Address दूसरे Element के Address पर पहुँच जाता है और दूसरे मान को Output में Print कर देता है। ये क्रम 6 बार चलता है, क्योंकि Variable n में Array की Size 6 Argument के रूप में main() से प्राप्त होती है और for Loop को तब तक चलाया गया है जब तक कि n का मान 6 ना हो जाए। इस प्रकार से एक Array को Pointer के साथ User Defined Function में भेजा जा सकता है व आवश्यकतानुसार उपयोग में लाया जा सकता है।

एक Variable की तरह ही एक Function का भी Memory में एक Location Address होता है। यानी एक Variable की तरह ही Function भी Memory में किसी Storage Cell में जा कर Store होता है और उस Storage Cell का कोई Address होता है।

इस कारण से एक Pointer को किसी Function का Address प्रदान करके Function Pointer की तरह भी Declare किया जा सकता है, जिसे आवश्यकता के अनुसार बाद में किसी अन्य Function में Argument के रूप में Use किया जा सकता है। जब एक Pointer को, किसी Function को Point करना होता है, तब उस Function Pointer को निम्नानुसार तरीके से Declare करना पडता है।

```
Data Type (*Function_Pointer_Name ) ( );
```

ये Declaration "C" Compiler को बताता है कि Function_Pointer_Name एक Pointer का नाम है जिसमें किसी Function का Address Store किया जा सकता है और ये Function_Pointer_Name Data Type प्रकार का मान Return करेगा।

यहां ये बात हमेंशा ध्यान रखें कि किसी Function को Point करने के लिए जो Pointer Declare किया जाता है, उसे हमेंशा कोष्ठक के अन्दर ही लिखना जरूरी होता है। यानी Function Pointer का नाम हमेंशा कोष्ठक में लिखा जाना चाहिये। माना हमने Function Pointer का Declaration निम्नानुसार किया:

```
Data Type *Function_Pointer_Name ( );
```

ये Declaration "C" Compiler को ये बताएगा कि Function_Pointer_Name एक Function है, जो कि Data Type प्रकार का एक Pointer Return करेगा। यानी ये Declaration किसी Function का Pointer नहीं होगा बल्कि Function_Pointer_Name नाम का एक Function होगा जो कि Data Type प्रकार का एक Pointer Return करेगा। Function Pointer को Declare करने का एक उदाहरण देखते हैं।

हमने Function Chapter के अंतर्गत sum() Function का प्रयोग किया है। आइये इस sum() Function को एक Pointer Function द्वारा Use करते हैं।

```
int (*sumptr)(), sum();
```

```
sumptr = sum;
```

यहां sum() एक Function है और *sumptr एक Function का Pointer है। दूसरे Statement में इस Function Pointer sumptr() को sum() नाम के Function का Address प्रदान किया गया है। sum int प्रकार का मान Return करेगा इसलिए sumptr को भी int प्रकार का Declare किया गया है, क्योंकि हम जानते हैं कि एक Pointer को उसी Data Type का Declare किया जाना जरूरी होता है, जिस Data Type का हम उसके Variable में मान Store करते हैं।

इसी वजह से यहां दोनों को int प्रकार का Declare किया गया है। अब हम sum() Function को उसके नाम sum() के बजाय उसके Pointer sumptr द्वारा भी Access कर सकते हैं। यानी अब हम Function sum() को Call करने के लिए इसके Pointer sumptr को Argument List के साथ Use कर सकते हैं। जैसे

```
( *sumptr ) ( x, y )
```

इसकी जगह हम निम्न Statement का भी प्रयोग कर सकते हैं।

```
sum( x , y );
```

ये दोनों ही Statements समान Output प्रदान करेंगे। केवल इनके काम करने के तरीके में अन्तर होगा। ये सवाल हमारे दिमाग में आ सकता है, कि जब हम एक साधारण तरीके से किसी Function को प्रयोग कर सकते हैं, तो फिर ये तरीका क्यों Use किया जाए। इसका जवाब ये है कि Functions को इस प्रकार से Use करके हम Memory Resident Programs लिख सकते हैं और Computer Virus व Virus Vaccines बनाया जा सकता है।

Function Returning Pointers

जिस प्रकार से हम int, float, double, char प्रकार के मान User Defined Function से प्राप्त करते हैं वैसे ही एक User Defined Function द्वारा Pointer भी Return करवाया जा सकता है। इसके लिए हमें Calling Function व Called Function दोनों में ही Function definition करना पडता है। निम्न प्रोग्राम में इसे समझाने की कोशिश की जा रही है।

Program

```
#include<stdio.h>
main()
{
    int *Pointer;
    int *Function();
```

```
        Pointer = Function();
        printf("\n %u \n %u ", Function(), Pointer);
    }

    int *Function()
    {
        int i = 20;
        return ( &i );
    }
```

Output

```
66520
66520
```

इस Program से पता चलता है कि Function एक Pointer return कर रहा है। इस Pointer में User Defined Function के local Variable i का Address Stored है और ये Address Pointer नाम के Variable को प्राप्त हो रहा है। इसी प्रकार से हम किसी भी Function से किसी भी प्रकार के Data type का Pointer Return value में प्राप्त कर सकते हैं।

One – Dimensional Array with Pointer

जैसाकि हम जानते हैं कि जब कोई Array Declare किया जाता है तो Array अपनी Size व Data Type के अनुसार एक ऐसी जगह पर जा कर Store होता है, जहां उस Array की Size के अनुसार सभी Array Element Memory में Store हो सकें। जैसे int b[10] करने पर Array Memory में 20 Byte की Space लेता है और Memory में ऐसी जगह पर जा कर Store होता है जहां सभी Elements एक क्रम में Memory में Store हो सकें। जब हम कोई Array Declare करते हैं तो उस Array के प्रथम Element के Address को Base Address कहते हैं। जैसे

```
int b[10];
```

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
1000	1002	1004	1006	1008	1010	1012	1014	1016	1018

इस Declaration में Array का Base Address 1000 है। किसी Pointer में हम किसी भी Array का केवल Base Address ही Pass करते हैं। माना एक Pointer Argument int *bp; है। इस Pointer Variable को इस Array का Base Address निम्नानुसार दिया जा सकता है।

```
bp = &b[0];          otherwise          bp = b;
```

यहां दिया गया दूसरा Statement भी bp में Base Address Store करता है, क्योंकि किसी भी Array में प्रथम Element हमेंशा Base Address से ही Memory में Store होना शुरू होता है। यदि किसी भी Array का नाम किसी Pointer Variable को Assign किया जाए तो वास्तव में उस Pointer Variable में उस Array का Base Address ही Store होता है। जब हमें क्रम से किसी Array में मान Store करना होता है या फिर किसी Array में Stored मान को क्रम से प्राप्त करना होता है, तब हमें & Operator का प्रयोग करने की जरूरत नहीं होती है।

लेकिन जब हमें किसी Array के किसी बीच के Element को Access करना होता है, तब हमें & Address Operator का प्रयोग करना जरूरी हो जाता है। क्योंकि Pointer Variable तो हमेंशा Scale Factor के क्रम में ही Increase या Decrease होता है।

जब किसी Pointer को Increase किया जाता है, तब ये Pointer तुरन्त बाद की Location को, उसके Scale factor के अनुसार Point करता है। हम किसी Array को उसके Index Number द्वारा Access करना सीख चुके हैं। अब हम एक प्रोग्राम द्वारा किसी Array के मान को Pointer द्वारा Access करना सीखेंगे।

Program

```
#include<stdio.h>

main()
{
    int j[] = { 11,23,33,22,44,55,66};
    int k;
    clrscr();
    for( k = 0; k < 5; k++ )
    {
        printf("\n address = %u ", &j[k] );
        printf("\t Element = %d ", j[k] );
    }
    getch();
}
```

Output

```
address = 65512     Element = 11
address = 65514     Element = 23
address = 65516     Element = 33
```

```
address = 65518        Element = 22
address = 65520        Element = 44
address = 65522        Element = 55
```

अब इसी उदाहरण को हम Pointer का उपयोग करके Array के मान व Address प्राप्त करते हैं। इसके लिए निम्न प्रोग्राम होगा। इस प्रोग्राम में हमने Base Address प्राप्त करने के लिए (Address of the 0th Element) निम्न Statement Use किया है:

```
l = &j[0];          //Assign Address 65512 to l
```

जब हम Loop में पहली बार पहुंचते हैं तो l का Address 65512 होता है और मान 11 होता है। ये निम्न Statements द्वारा Output में Print होता है।

```
printf("\n address = %u ", l );
printf("\t Element = %d ", *l );
```

जब l का मान Increment होता है तब l का मान बढ कर 65514 हो जाता है। इस Address पर 23 स्थित है। ये दोनों मान पुनः इसी Printf() Function द्वारा Output में Print हो जाते है। ये क्रम Array के अंत तक चलता रहता है। इस प्रकार से हम Pointer द्वारा One Dimensional Array के मानों को Access कर सकते हैं।

Program

```
#include<stdio.h>

main()
{
    int j[] = { 11,23,33,22,44,55,66};
    int k, *l;
    l = &j[0]; //Assign Address of Zeroth Element
    clrscr();
    for( k = 0; k < 5; k++ )
    {   printf("\n address = %u ", l );
        printf("\t Element = %d ", *l );
        l++; //Increment Pointer to Point next Location
    }
    getch();
}
```

Output

```
address = 65512      Element = 11
address = 65514      Element = 23
address = 65516      Element = 33
address = 65518      Element = 22
address = 65520      Element = 44
address = 65522      Element = 55
```

जैसा कि उदाहरण में बताया कि किसी Pointer को Array का Base Address देने के लिए l = &j[0]; Statement दिया गया है। यदि Pointer l को Increment करते हैं तो इसका मतलब है कि हम Array के रूप में इसे निम्नानुसार भी लिख सकते हैं।

```
l = &j[0];
```

दूसरे Element के Address के लिए

```
l +1 = &j[1];
```

इसी प्रकार तीसरे Element के Address के लिए

```
l = &j[0]; +2 = &j[2];
```

इस प्रकार हम Pointer द्वारा भी किसी Array के किसी भी Element पर पहुंच सकते हैं। क्योंकि जब किसी Pointer को Increment या Decrement किया जाता है, तब वह Pointer अपने Scale Factor के अनुसार दूसरे Element के Location को Point करता है। इसलिए हम किसी भी Array के किसी भी Element को Pointer द्वारा प्राप्त कर सकते हैं। जैसे

```
int j[] = {11,22,33,44,55,66};
int *k = j;
```

इस Statement से k में Array j का Base address आ जाएगा। यदि Pointer द्वारा इस Base Address के मान को Print करना हो तो हमें निम्न Statement देना होगा।

```
printf("\n Value of the First Element is " , *k );
```

यदि हम चाहें तो इसे इस प्रकार से भी लिख सकते हैं।

```
printf("\n Value of the First Element is " , *(k) );
```

अब यदि हमें दूसरे Element का मान प्राप्त करना हो तो निम्नानुसार प्राप्त कर सकते हैं।

```
printf("\n Value of the Second is " , k +1 );
```

इसी मान को हम निम्न Statement Use करके प्राप्त कर सकते हैं।

```
printf("\n Value of the Base Address is " , *(k+1) );
```

इस Statement से Pointer k Array के दूसरे Element को Point करेगा इसलिए दूसरा मान Output में Print होगा। इस तरह हम किसी Array की विभिन्न Locations पर Stored मान को निम्न सुत्र में लिख कर प्राप्त कर सकते हैं।

```
* ( Base Address + Index Number of Row );
```

जब हम j[i] लिखते हैं तो Compiler स्वयं ही इसे *(j + i) में बदल देता है। इसलिए हम नीचे लिखे गए चारों Statement को एक दूसरे के स्थान पर प्रतिस्थापित कर सकते हैं।

```
( 1 )   *( j + i )        ( 2 )   *( i + j )
( 3 )   j [ i ]           ( 4 )   i [ j ]
```

निम्न प्रोग्राम द्वारा एक ही परिणाम को इन चार तरीकों से प्राप्त किया गया हैं।

Program

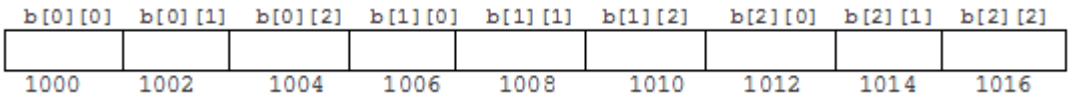
```
#include<stdio.h>
main()
{
    int j = {11, 22, 33, 44, 55, 66};
    int i;
    clrscr();
    for ( i = 0; i <= 5; i++ )
    {
    printf("\n Address Of %d Element is %u ", i+1, &j[i] );
    printf(" Element with Statement j [ i ] = ", j[i] );
    printf(" Element with Statement *( j + i ] = ", *(j+i));
    printf(" Element with Statement *( i + j ) = ", *( i+j));
    printf(" Element with Statement  i [ j ] = ", i[j]);
    }
    getch();
}
```

Output

```
Address Of 2 Element is 65516
Element with Statement j[ i ] = 22
Element with Statement *( j + i ) = 22
Element with Statement *( i + j ) = 22
Element with Statement  i[ j ] = 22
Address Of 3 Element is 65518
Element with Statement j[ i ] = 33
Element with Statement *( j + i ) = 33
Element with Statement *( i + j ) = 33
Element with Statement  i[ j ] = 33
Address Of 4 Element is 65520
Element with Statement j[ i ] = 44
Element with Statement *( j + i ) = 44
Element with Statement *( i + j ) = 44
Element with Statement  i[ j ] = 44
Address Of 5 Element is 65522
Element with Statement j[ i ] = 55
Element with Statement *( j + i ) = 55
Element with Statement *( i + j ) = 55
Element with Statement  i[ j ] = 55
Address Of 6 Element is 65524
Element with Statement j[ i ] = 66
Element with Statement *( j + i ) = 66
Element with Statement *( i + j ) = 66
Element with Statement  i[j] = 66
```

Pointer with 2-Dimensional Array

एक बात हमेंशा ध्यान रखें कि Memory में ना तो कोई Row होती है ना ही कोई Column, Memory में सभी Data एक क्रम में Store होते हैं। इसलिए ये माना जा सकता है कि किसी Array के सभी Elements एक Row में ही Store होते हैं। जो भी Index Number होते हैं, वे सभी Logical होते हैं। Physical तो मात्र Address होते हैं। एक Two Dimensional Array के सभी Elements निम्नानुसार Store होते हैं। जैसे int j[3][3]; ये Memory में निम्नानुसार क्रम से Store होते हैं–



इस प्रकार से हम कह सकते हैं कि एक 2-Dimensional array में कई One Dimensional Array होते हैं, जो एक श्रृंखला में व्यवस्थित रहते हैं। जैसे कि

```
int s[5][2];
```

इस Declaration को ये कह सकते हैं कि ये 5 One Dimensional Array हैं और पांचों में ही दो–दो Byte के पांच Elements है। हम एक Single One Dimensional Array को एक Index Number द्वारा संकेत कर सकते हैं। इसी तरह से यदि हम मान लें कि s एक One Dimensional Array है तो इसके प्रथम Element को s[0] मान सकते हैं। इसी तरह दूसरे Element को s[1] व तीसरे Element को s[2] मान सकते हैं। खास तौर से हम ये कह सकते हैं कि s[0] प्रथम One Dimensional Array का Base Address देता है, s[1] दूसरे One Dimensional Array का Base Address दे रहा है। और इसी प्रकार से आगे भी माना जा सकता है। इस बात को निम्न उदाहरण द्वारा अधिक अच्छी तरह से समझा जा सकता है।

Program

```
#include<stdio.h>
main()
{
    int s[5][2] = { {11,22},{33,44},{55,66},{77,88},{99,00}};
    int i, j;
    clrscr();
    for( i = 0; i <= 3; i++ )
    {
    printf("\n Address Of %d One-Dimensional Array = %u ", i+1,  s[i] );
    }
    getch();
}
```

Output

```
Address Of 1 One-Dimensional Array = 65506
Address Of 2 One-Dimensional Array = 65510
Address Of 3 One-Dimensional Array = 65514
Address Of 4 One-Dimensional Array = 65518
```

आइये समझने की कोशिश करते हैं कि Program कैसे काम करता है। "C" Compiler जानता है कि s एक int प्रकार का Array हैं, इसलिए Array का हर Element 2 Byte की Space लेता है। यहां हर Row में दो Element हैं, इसलिए हर Row 4 Byte लेगा।

चूंकि ये माना जा सकता है कि Memory में किसी भी 2-Dimensional array को उस Array की दूसरी size के अनुसार उतने ही One Dimensional array माने जा सकते हैं, क्योंकि 2-Dimensional array में भी सभी Elements एक ही Row में Memory में Store होते हैं।

इस प्रकार से यहां 2–2 Elements के 5 one-Dimensional Array माने जा सकते हैं और हर One-Dimensional Array Memory में 4 Byte की Space ले रहा है, क्योंकि हर One-Dimensional Array में दो Element हैं व हर Element int प्रकार का है।

इस कारण से हर one-Dimensional Array Memory में 4 Byte ले रहा है। इस प्रकार से एक 2-Dimensional Array की दूसरी Row को दूसरा One-Dimensional Array, तीसरी Row को तीसरा One-Dimensional Array, चौथी Row को चौथा One-Dimensional Array कहा जा सकता है और यही क्रम आगे भी माना जा सकता है। यदि इस 2-Dimensional Array का Memory map देखा जाए तो वह निम्नानुसार होगाः

s[0][0]	s[0][1]	s[1][0]	s[1][1]	s[2][0]	s[2][1]	s[3][0]	s[3][1]	s[4][0]	s[4][1]
65506	65508	65510	65512	65514	65516	65518	65520	65522	65524

जब Array s को Declare किया जाता है तभी "C" Compiler जान जाता है कि s में कितने Column हैं। इसलिए हम One-Dimensional Array के अंतर्गत बताए अनुसार s[0] को (s+0) व s[1] को (s+1) कह सकते है।

चूंकि (s+0) का Address 65506 है इसलिए (s+1) का Address 65510 होगा। (s+2) का Address 65514 होगा और ये क्रम Array के अन्तिम मान तक चलता रहेगा। इस प्रकार से अब हम किसी भी अमुक Row में पहुंच सकते हैं।

माना हम s[2][1] Element पर Pointer द्वारा पहुंचना चाहते हैं। हम उपरोक्त Declaration से जानते हैं कि s[2] का Address 65514 है जो कि Second one-Dimensional Array का Address है।

स्पष्ट रूप से यदि हम (65514+1) करें, तो हमें 65516 (Pointer के Scale Factor के कारण) Address प्राप्त होगा और यहां पर स्थित मान को *(Value at Address Operator) द्वारा प्राप्त किया जा सकता है यानी *(s[2] +1) लेकिन हमने पहले पढा कि किसी j[i] व *(j+i) समान हैं यानी दोनों ही Expression से समान Output प्राप्त होता है।

इसी प्रकार से *(s[2]+1) भी *(*(s+2)+1) के समान है। उदाहरण के लिए निम्नलिखित सभी Expressions किसी समान Element Location का संकेत देते हैं:

```
s[2][1]
*(s[2] + 1)
*( *(s + 2 ) + 1 )
```

इसे सुत्र के रूप में इस प्रकार लिख सकते हैं।

```
*( *( Base Address + Index Number of Row) + Index Number of Column );
```

इस प्रकार से इसी Concept पर हम ऊपर दिये गए उदाहरण के सभी Elements का मान Pointer द्वारा Output में Print कर सकते हैं। Program निम्नानुसार होगा–

Program

```
#include<stdio.h>
main(){
    int s[5][2] = { {11,22},{33,44},{55,66},{77,88},{99,00}};
    int i, j;
    clrscr();
    for( i = 0; i <5; i++ ){
        for( j = 0; j <= 1; j++ )
        printf("\n Value Of s[%d][%d]th  Element is = ", *( *( s + i ) + j ) );
        printf("\n");
    }
    getch();
}
```

Output

```
Value Of s[1][1]th  Element is = 11
Value Of s[1][2]th  Element is = 22
Value Of s[2][1]th  Element is = 33
Value Of s[2][2]th  Element is = 44
Value Of s[3][1]th  Element is = 55
Value Of s[3][2]th  Element is = 66
Value Of s[4][1]th  Element is = 77
Value Of s[4][2]th  Element is = 88
Value Of s[5][1]th  Element is = 99
Value Of s[5][2]th  Element is = 0
```

Array of Pointers

जिस प्रकार से int प्रकार के ढेर सारे Data का एक Array हो सकता है, char प्रकार के ढेर सारे Data का एक Array हो सकता है उसी प्रकार से एक ऐसा भी Array बनाया जा सकता है जिसमें समान प्रकार के विभिन्न Pointers को Store किया जा सकता है। इसे समझने के लिए निम्न उदाहरण देखें:

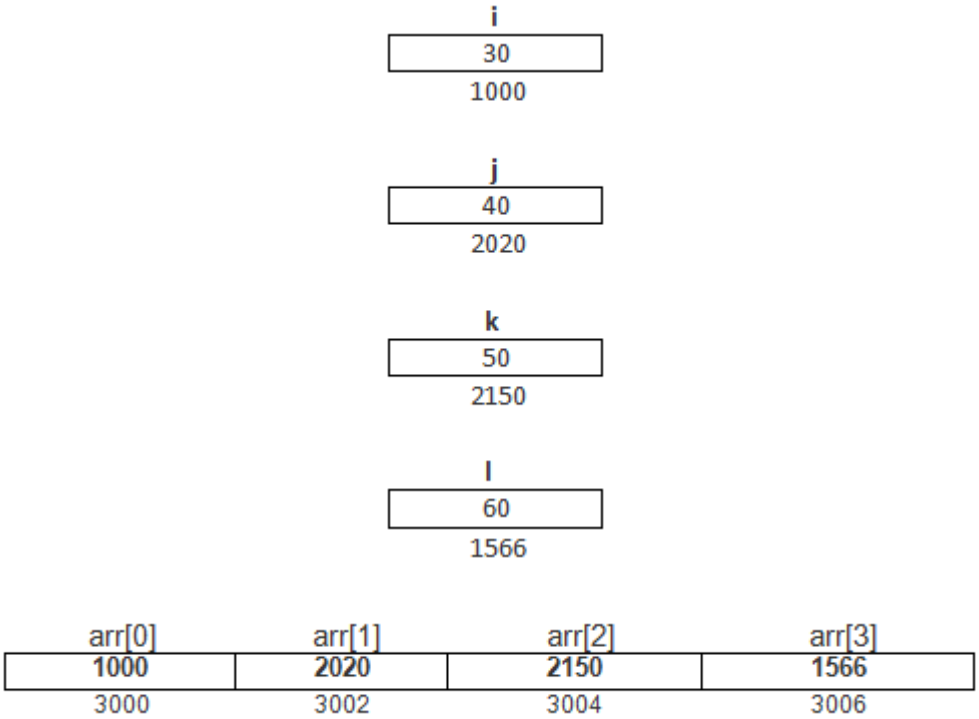
Program

```
#include<stdio.h>
main()
{
    int *arr[4]; //Array of Integer Pointers
    int i = 30, j = 40, k =50, l = 60;
    clrscr();
    arr[0] = &i;
    arr[1] = &j;
    arr[2] = &k;
    arr[3] = &l;
    for( m = 0; m < 4; m++ )
        printf("\n Value of All Pointers is %d ", *(arr[m]));

    getch();
}
```

Output

```
Value of Pointers is 30      //Which is the value of Variable i
Value of Pointers is 40      //Which is the value of Variable j
Value of Pointers is 50      //Which is the value of Variable k
Value of Pointers is 60      //Which is the value of Variable l
```



Pointers and Strings

Characters के समूह को Computer Science में String कहा जाता है। Strings के बिना कोई भी Application Software नहीं बन सकता। यानी सभी Software में String का प्रयोग जरूर होता है। Characters के समूह को हम सामान्यतया एक One-Dimensional Array में Store करते हैं।

हम जानते हैं कि Strings को एक Array के रूप में Memory में Store किया जाता है और ये String तब Terminate होती है जब "C" Compiler को Null Character '\0' मिलता है। जिस प्रकार से हम किसी int प्रकार के Array का Base Address किसी Pointer को देकर उस Array के हर Element को Access कर सकते हैं, उसी प्रकार से किसी String के Array का भी Base Address किसी Pointer को दिया जा सकता है और उस Pointer के प्रयोग द्वारा String के सभी Characters को भी उसी तरह से Access किया जा सकता है जिस तरह से हम किसी Integer प्रकार के मानों वाले Array के सभी Elements को Access कर सकते हैं। इस Concept को समझने के लिए निम्न Program देखिए:

Program

```
#include<stdio.h>

main()
{
```

```
    char *cp, name[] = {"RAM"};
    cp = &name;
    clrscr();

    while(*cp != '\0' )
    {
        printf("\n Character %c Stored At Address %u ", *cp, cp);
        cp++;
    }
    getch();
}
```

Output

```
Character R Stored At Address 65522
Character A Stored At Address 65523
Character M Stored At Address 65524
```

इस प्रोग्राम में *cp एक Character Pointer है व name नाम का एक Character Array है जिसमें RAM Stored है। cp = &name द्वारा Array का Base Address Pointer cp को दिया गया है। फिर एक While Loop चलाया गया है। ये Loop तब तक चलता है जब तक कि Program Control को Null Character नहीं मिल जाता।

printf() Function इस cp के Address पर Stored Character को Output में Print कर देता है। पहली बार ये Address 65522 होता है। cp को Increment करने पर ये Address 65523 हो जाता है। इस Address पर Character A Stored है इसलिए Output में A print हो जाता है। वापस cp Increment हो कर 66524 हो जाता है। इस Location पर M Stored है इसलिए Output में M Print हो जाता है। cp एक char प्रकार का Pointer है इसलिए cp का Scale Factor के अनुसार एक–एक Byte का Increment होता है जो कि Output में देखा जा सकता है।

हम जानते हैं कि यदि हमें किसी नाम को Computer में Store करना व Access करना हो तो हमें एक One – Dimensional Array में Name की Size को Define करना होता है कि हम अधिकतम कितने Characters तक का नाम Store करना चाहते हैं।

कई बार हम हमारी आवश्यकता से कम Size लेते हैं तो कई बार हम हमारी आवश्यकता से अधिक Size ले लेते हैं। इन दोनों ही स्थितियों में एक समस्या है। यदि हम आवश्यकता से कम Size का Array Declare करते हैं तो Program के Crash होने की सम्भावना रहती है। जबकि यदि हम आवश्यकता से अधिक Size का Array लेते हैं तो बाकी बची हुई Memory भी Array के लिए

Reserved रहती है, जिसे कोई भी अन्य Program तब तक Use नहीं कर सकता जब तक कि उस Program को Terminate ना कर दिया जाए।

इस समस्या से बचने के लिए हम एक Character प्रकार के Pointer Array का प्रयोग कर सकते हैं। जब हम किसी नाम को किसी Pointer द्वारा Memory में Store करते हैं, तो वह नाम Memory में उतनी ही जगह लेता है जितनी उसे जरूरत होती है। किसी String के लिए इस प्रकार का Memory Allocation Dynamic Memory Allocation कहलाता है।

Array of Pointers To String

एक String को हमेंशा एक One-Dimensional Array के रूप में Memory में Store होता है। इस Array को यदि Pointers के प्रयोग से उपयोग में लेना हो तो हम एक Pointer Variable Declare करते हैं और उस Pointer Variable में उस String का Base Address दे देते हैं।

इस Pointer को Increment करके हम उस Array में अलग–अलग Locations पर स्थित characters पर pointer द्वारा Move कर सकते हैं। इस तरह से हम कह सकते हैं कि एक ऐसा One-Dimensional Array जिसमें String Store हो, को एक ही pointer द्वारा Access किया जा सकता है।

इसी तर्क पर यदि हम आगे बढें तो ये भी कह सकते हैं कि एक string प्रकार के One-Dimensional Array का Base Address प्राप्त करके यदि हम उस string के हर character के साथ प्रक्रिया कर सकते हैं तो फिर एक ऐसा Two-Dimensional Array जिसमें कई strings हों, को भी एक Pointer द्वारा Access किया जा सकता है। कैसे ? आइये समझने की कोशिश करते हैं।

हमने Pointers के बारे में पढते समय ये बताया था कि एक Two-Dimensional Array को उस Array की संख्या के अनुसार कई One-Dimensional Array के रूप में मान सकते हैं। यदि एक ऐसा pointer प्रकार का Array Declare किया जाए, जिसमे हर One-Dimensional Array या Row का Base Address इस Array में Store कर दिया जाए तो हम इस Pointer Array द्वारा एक Two-Dimensional string Array को Handle कर सकते हैं।

हम एक उदाहरण द्वारा इस बात को समझते हैं। माना यदि हमें किसी Variable में एक नाम Store करना जिसमें अधिकतम 10 अक्षरों का नाम Store हो सके तो हम निम्नानुसार एक One-Dimensional Array Declare कर सकते हैं–

```
char name[10];
```

इस Array मे हम केवल एक ही नाम Store कर सकते हैं। यदि हमे ये जरूरत हो कि हम 10 अक्षरों के 10 नाम Memory में Store करना चाहें तो या तो हमें इसी प्रकार के 10 अन्य Variables Declare करने होंगे जिसमें 10 अलग–अलग नाम Store किये जा सकें या फिर हम इस Array को ही ऐसा बना दें कि यही Array 10 नाम Accept कर सके।

यदि हम इसी Array द्वारा 10 नाम Memory में Store करना चाहते हैं, तो हमें इस Array को निम्नानुसार Two-Dimensional Array में परिवर्तित करना पडेगाः

```
char name[10][10];
```

अब यदि इस Array में हम 10 अक्षरों तक के 10 नाम Store करें तो ये Memory में निम्नानुसार Store होंगेः

Base Address	0	1	2	3	4	5	6	7	8	9
1500	K	U	L	D	E	E	P	\0		
1510	D	E	V	D	A	A	S	\0		
1520	R	A	J	A	\0					
1530	K	U	L	D	E	E	P	\0		
1540	H	I	M	A	N	S	U	\0		
1550	N	A	V	N	E	E	T	\0		
1560	N	A	M	E	E	T	A	\0		
1570	R	A	J	E	N	D	R	A	\0	
1580	R	O	H	A	N	\0				
1590	M	O	H	A	N	\0				

इस Table में देख सकते हैं कि काफी Space Array द्वारा फालतू Reserve रहता है। यदि इसके स्थान पर एक Pointer Array Declare करें तो हम इस Space को बचा सकते हैं। हम एक Pointer Array में नाम Store करने के लिए निम्नानुसार Declaration कर सकते हैं–

```
char *name[] = {"Kuldeep"};
```

इस Declaration का मतलब है कि name नाम के Variable में String "Kuldeep" का Base Address है। यदि हम इस Pointer Array में Array Elements के Address Store ना करें यानी इसे एक सामान्य Array ही रहने दें तो इसे हम निम्नानुसार भी लिख सकते हैं–

```
char name[] = "Kuldeep";
```

इस प्रकार से हम ये भी कह सकते हैं कि Array में “Kuldeep” नाम का String Index Number 0 पर Store है। क्योंकि किसी Array में कोई भी पहला मान हमेंशा Index Number 0 पर Store होता है। यानी हम इसे name[0] = “Kuldeep” भी लिख सकते हैं।

इसका मतलब ये हुआ कि यदि Array को हम Pointer Array बनाते हैं तो इस Array में Index Number 0 पर String “Kuldeep” का Address Store होता है (पूरा String नहीं) और यदि हम इस Array को Pointer Array नहीं बनाते हैं तो यह एक char प्रकार का One-Dimensional Array ही होता है, जिसमें characters का एक समूह जिसे string कहते हैं, Store होता है।

माना कि string ”Kuldeep” Memory में 1500 Storage cell में जाकर Store होता है। तब name के Index Number 0 का Address 1500 होगा और ये string Access करने के लिए हमें Index Number 0 को name[0] Statement द्वारा Use करना पडेगा।

यदि हम इस String को Pointer द्वारा Access करना चाहें तो हमें Pointer को Increment करना होगा, जिससे One By One Character Read होंगे। इसी Array को यदि Pointer प्रकार का Declare कर दिया जाए यानी *name कर दिया जाए तो इस Array में Index Number 0 पर Stored String “Kuldeep” का Base Address 1500 Store हो जाएगा। इस Pointer Array द्वारा भी यदि हम Array के Element को Access करना चाहें तो हमें name[0] Statement को ही Use करना होगा।

यदि हम इस Array के मानों को Pointer द्वारा Read करना चाहें तो हम Character By Character इसे Read नहीं कर सकते हैं क्योंकि ये एक Pointer Array है और pointer में हमेंशा किसी अन्य Variable का Address Stored होता है।

इसलिए इस Array में केवल एक मान 1500 यानी string “Kuldeep” का Base Address Stored है ना कि String, जिसे Pointer को Increment करके One By One Characters को Read किया जा सके।

किसी Pointer array में किसी One-Dimensional Array का Base Address Store होना ये बताता है कि यदि इस Address के मान को Print किया जाए तो हमें पूरा String प्राप्त हो जाएगा। इन दोनों के अन्तर को निम्न चित्र द्वारा बताया जा रहा है।

char name [ ] = {“Kuldeep”};

K	U	L	D	E	E	P	\0
1500	1501	1502	1503	1504	1505	1506	507

char *name = { “Kuldeep” };

1500	\0
1500	

char *name = { "Kuldeep" }; को हम ये भी कह सकते हैं कि Array Name के Index Number 0 पर Element के रूप में एक One-Dimensional Array का Base Address Stored है। यानी name[0] = {"Kuldeep"};

चूंकि Pointer Array *name एक One-Dimensional Array है और इसमें केवल एक ही Address Stored है। इसलिए हम चाहें तो इसमें आवश्यकतानुसार और भी कई One-Dimensional Arrays के Address Store कर सकते हैं। ऐसा करने के लिए हम निम्नानुसार Initialization कर सकते हैं–

char *name[ ] = { "Kuldeep", "Devdaas" };

इस Declaration से String "Devdaas" नाम का एक और One-Dimensional Array बन जाएगा और इस Array का Base Address name[1] Location पर Store होगा। इसे भी चित्र में दिखाया गया है।

char name [ ] = {"Devdaas"};

D	E	V	D	A	A	S	\0
1508	1509	1510	1511	1512	1513	1514	1516

char *name = { "Kuldeep", "Devdaas" };

1500	1508	\0
1601	1602	1603

इस प्रकार से हम देखते हैं कि किस तरह से हम एक One-Dimensional Pointer Array से Two Dimensional String Array को Handle कर सकते हैं। Pointer का इस प्रकार से प्रयोग करके हम काफी Memory बचा सकते हैं क्योंकि Pointer को जब इस प्रकार से Use किया जाता है, तब हमें Array की Size Declare नहीं करनी होती है। ये Matter अच्छी तरह से समझ मे आ जाए इसके लिए यहां एक उदाहरण द्वारा इसे बताया जा रहा है।

Program

```
#include<stdio.h>

main()
{
    char *n[4];
    int j;
    clrscr();

    for(j=0;j<4;j++)
```

```
        {
            printf("\n Enter String %d", j+1);
            gets(n[j]);
        }

        printf("\nBase Address of Row1 is %u ",n[0]);
        printf("\nBase Address of Row2 is %u ",n[1]);
        printf("\nBase Address of Row3 is %u ",n[2]);
        printf("\nBase Address of Row4 is %u ",n[3]);

        printf("\nValue of Row1 is %s ",n[0]);
        printf("\nValue of Row2 is %s ",n[1]);
        printf("\nValue of Row3 is %s ",n[2]);
        printf("\nValue of Row4 is %s ",n[3]);
        getch();
    }
```

Output

```
Enter String 1 kuldeep
Enter String 2 harish
Enter String 3 raja
Enter String 4 rajesh
Base Address of Row1 is 3432
Base Address of Row2 is 36002
Base Address of Row3 is 12803
Base Address of Row4 is 36518
Value of Row1 is kuldeep
Value of Row2 is harish
Value of Row3 is raja
Value of Row4 is rajesh
```

इस Program में हम जब भी कोई String Input करते हैं, तो वह String Memory में किसी Location पर जा कर एक One-Dimensional Array के रूप में Store हो जाती है।

चूंकि हमने यहां जो Array लिया है वो Pointer प्रकार का है और इस Array में केवल किसी भी One-Dimensional Array का Address ही Store हो सकता है। इसलिए हमारे द्वारा Input किया गया String Memory मे इस Array में Store ना हो कर किसी भी अन्य Location पर Store हो जाता है।

यहां हमने string Input करने के लिए एक Pointer प्रकार के Variable को माध्यम बनाया है इसलिए इस pointer array में उस string के प्रथम अक्षर का Address या Base Address उस Index Number के स्थान पर Element के रूप में Store हो जाता है जिस Index Number के प्रयोग से हम हमारा String Input करते हैं।

इस प्रकार से हम एक One-Dimensional Pointer Array द्वारा एक Two-Dimensional String Array को प्रयोग कर सकते हैं। इसी Array of Pointers To String का प्रयोग Command Line Arguments को Accept करने के लिए किया जाता है।

कई बार हमें ऐसी जरूरतें पडती हैं, जिसमें किसी मान को Store तो एक Character Array के रूप में किया गया होता है, लेकिन Access करते समय उस Character Array में Stored मान को Integer या Float प्रकार के मान के रूप में Use करना होता है।

इस स्थिति में हमें उस Array में Stored मान को Integer या Float प्रकार के मान में Convert करने की जरूरत पडती है। चलिए, हम इसी प्रकार की एक समस्या का समाधान प्राप्त करने की कोशिश करते हैं। निम्न Example Program में हम एक Character Array में Stored Integer प्रकार के मान को Integer प्रकार के मान में Convert करने के लिए एक Function Create कर रहे हैं।

Program

```
#include <ctype.h>
/* atoi:  convert String Integer to Numerical Integer */

int atoi(char strInteger[])
{
    int i, n, sign;

    for (i = 0; isspace(strInteger[i]); i++) /* skip white space */
            ;

    sign = (strInteger[i] == '-') ? -1 : 1;

    if (strInteger[i] == '+' || strInteger[i] == '-')/* skip sign */
            i++;

    for (n = 0; isdigit(strInteger[i]); i++)
        n = 10 * n + (strInteger[i] - '0');

    return sign * n;
```

```
}
```

इस Function में हमने **ctype.h** नाम की Header File को Include किया है। इस Header File में Character Manipulation से सम्बंधित कई Functions हैं, जिन्हें हम हमारी जरूरत के आधार पर Use कर सकते हैं। जब हमें किसी Character Array में String Format में Stored *String Integer* को *Numerical Integer* में Convert करना होता है, तब हम उस Character Array को इस Function में Formal Argument के रूप में Pass करते हैं।

ये Function Argument के रूप में Calling Function से String Integer को प्राप्त करता है और उस String Integer को Numerical Integer मान में Convert करके फिर से Calling Function में Return कर देता है।

जब ये Function Call होता है, तब Argument के रूप में इसमें उस String Integer Array को भेजा जाता है, जिसमें String Format में Integer मान Stored होता है। ये Function उस मान को strInteger नाम के Array Variable में प्राप्त करता है।

किसी Character Array में जो Integer मान Store होता है, उस मान को Store करते समय String Integer मान से पहले Space का प्रयोग किया गया हो सकता है, जबकि एक Integer प्रकार के मान में कोई Space नहीं होता है। इसलिए सबसे पहले हमें किसी String Integer में स्थित Spaces को Remove करना होता है।

किसी Character Array में से Space को खोजने का काम करने के लिए हम **ctype.h** नाम की Header File में Define किए गए ***isspace***() Function को Use करते हैं। ये Function उस स्थिति में True Return करता है, जब इसे किसी Character Array में Space, Tab, New Line Character या कोई अन्य Blank Space Character प्राप्त होता है। इस Function को निम्नानुसार एक for Loop में Use किया गया है:

```
for (i = 0; isspace(strInteger[i]); i++)            /* skip white space */
                    ;
```

इस Loop की Body नहीं है, क्योंकि इस Loop में हमें कोई Extra काम नहीं करवाना है। चूंकि इस Function में आने वाला Argument एक One-Dimensional Character Array है और किसी 1-D Array में स्थित सभी Characters को एक Index Number द्वारा Access किया जा सकता है।

इस Loop से हमारी Requirement भी यही है कि हम इस Character Array के हर Element पर जा कर ये Check करें, कि उस Index Number की Position पर कोई Space Store है या नहीं। ये Loop तब तक चलता है, जब तक Computer को Character Array में Blank Space प्राप्त होता रहता है, जैसे ही **isspace**() Function को Blank Space के अलावा कोई Character प्राप्त होता है, ये Function False Return करता है, जिससे Loop Terminate हो जाता है।

उदाहरण के लिए मानलो कि किसी Character Array में Integer मानों से पहले चार Space हैं, तो ये Loop चार बार चलता, जिससे Variable **i** का मान 3 हो जाता, जो इस बात का Signal है कि Actual Integer मान Character Array के Index Number 3 से शुरू हो रहा है।

किसी Character Array में किसी मान को Store करते समय उसके साथ Sign का चिन्ह भी Store किया गया हो सकता है। इस स्थिति में अब हमें ये Check करना होता है, कि Character Array में '+' या '-' जैसा कोई चिन्ह है या नहीं। इस बात को Check करने के लिए Program में अगले Statement के रूप में निम्नानुसार एक ***Ternary Operator*** Use किया गया है:

```
sign = (strInteger[i] == '-') ? -1 : 1;
```

मानलो कि हमारे Character Array में चार Space थे जिसे पिछले Loop का प्रयोग करके हमने Skip किया। Skip करने पर **i** का मान 3 हो गया, जो कि Character Array के Index Number **3** को Specify करता है। चूंकि अब Space नहीं है इसका मतलब ये है कि Space के अलावा कोई Character है और वह Character Minus (–) है या नहीं इस बात की जांच करने के लिए हमने इस Statement को लिखा है।

ये Statement Character Array के Index Number **3** को Minus Sign के लिए Check करता है और यदि Computer को इस Index Number पर Minus ('-') का Character मिलता है, तो Computer **sign** नाम के Variable में **-1** Store कर देता है, जो इस बात का संकेत होता है, कि Character Array में एक Minus Sign वाली संख्या Stored है। चिन्ह का पता लगाने के बाद फिर से एक Loop चलाया जाता है और ये Loop अब Character Array में Stored सभी Digits को Integer में Convert करने का काम करता है।

```
for (n = 0; isdigit(strInteger[i]); i++)
   n = 10 * n + (strInteger[i] - '0');
```

इस Loop में हमने एक Variable **n** को **0** Assign किया है। फिर **ctype.h** नाम की Header File के **isdigit**() Function को Use करके Index Number **i** की Position पर Stored Character को इस बात के लिए Check किया है, कि वहां पर कोई Digit Stored है या नहीं।

यदि Index Number **i** की Position पर कोई Digit होता है, तो ये Program True Return करता है। जब ये Program True Return करता है, तब Computer **for** Loop के अगले Statement को Execute करता है। इस Statement में (strInteger[i] - '0') Code Use किया गया है। इस Code को Use करने का कारण ये है, कि Character Array में हर Position पर किसी भी Character की ASCII Value Stored होती है।

मानलो कि Character Array में 23 Stored है, जिसे Integer में Convert करना है। इस स्थिति में वास्तव में Character Array में **2** व **3** Stored नहीं है बल्कि इनकी ASCII Value **50** व **51** Stored है। इसलिए यदि हम Directly इसे ज्यों का त्यों Use करें यानी (strInteger[i] - '0') Code के स्थान पर (strInteger[i]) Code को Use करें, तो 50 को गुणा **n** में Stored मान **0** से होने पर **0** जो कि हमारा Required Result नहीं है।

इस स्थिति में जब हम ***strInteger*** Array के Index Number **i** की Location पर Character Format में Stored Digit की ASCII Value में से Integer मान 0 की ASCII Value को घटाते हैं, तो Resultant रूप में हमें Integer मान ही प्राप्त होता है।

उदाहरण के लिए मान यदि हम मान 23 के 2 की बात करें, तो 2 के ASCII Code **50** में से 0 के ASCII Code **48** के घटाने पर 2 ही प्राप्त होता है, लेकिन ये 2 एक Integer मान होता है, ना कि Character मान।

Character Array में Stored String Integer मान को Numerical Integer मान में Convert करने के बाद यदि संख्या Minus वाली होती है, तो मान **-1** Sign नाम के Variable में Store हो जाता है, जिसका प्रयोग Function के अन्त में Return होने वाले Integer मान के Sign को Change करने के लिए किया जाता है।

जब **atoi**() Function से किसी Minus Sign के मान को Return करना होता है, तब **sign** Variable में Stored मान -1 को Return किए जाने वाले मान **n** से गुणा करके Resultant मान को Return कर दिया जाता है, जो कि एक Negative मान होता है।

इसी तरह से यदि हमें किसी Character Array में Stored String Formatted Float, Double, Long या किसी अन्य प्रकार के मान को Numerical Form में Convert करना हो, तो हम इसी तरह के Process को Use करके ये काम कर सकते हैं। वैसे इस तरह के Conversion Functions को पहले से ही Develop करके Library के रूप में हमें प्रदान कर दिया गया है, जिन्हें हम Directly Use कर सकते हैं।

किसी Character Array में Stored String को हम Reverse में Convert करने के लिए भी Function बना सकते हैं। इस Function की जरूरत उस स्थिति में पडती है, जब हम किसी मान को किसी Character Array में Store करते हैं और वह मान उस Character Array में Reverse Format में Store हो जाता है।

```
/* reverse:  reverse string s in place */
#include <string.h>
void reverse(char str[])
{
    int c, i, j;
```

```
    for (i = 0, j = strlen(str)-1; i < j; i++, j--)
      {
        c = str[i];
        str[i] = str[j];
        str[j] = c;
    }
}
```

जब इस Function में किसी Character Array को भेजा जाता है, तब वह Character Array **str** नाम के Argument में आकर Store हो जाता है। इस Function में एक **for** Loop चलाया गया है और इस for Loop में एक ही बार में एक Variable **i** को Increment किया गया है साथ ही दूसरे Variable **j** के मान को Decrement किया गया है।

Loop को इस तरह चलाया गया है कि Character Array के **First Index Number** को **Last Index Number** से **Swap** करता है। फिर Second Index Number के Character को Second Last Character से Swap करता है और ये प्रक्रिया तब तक चलती है, जब तक कि **i** का मान **j** के मान से ज्यादा नहीं हो जाता है। ये Loop तभी Terminate होता है, जब Character Array में Stored पूरी String Reverse हो जाती है। इस Function को हम निम्नानुसार Use कर सकते हैं:

UDF

```
/* reverse:  reverse string s in place */

#include <string.h>

void reverse(char str[])
{
    int c, i, j;

    for (i = 0, j = strlen(str)-1; i < j; i++, j--)
      {
        c = str[i];
        str[i] = str[j];
        str[j] = c;
    }
}
```

Program

```
#include <stdio.h>
#include <conio.h>

main()
{
    char name[] = "Manbhavan";

  reverse(name);
  puts(name);
  getch();
}
```

Output

```
navahbnaM
```

जिस तरह से किसी Character Array में String Format में Stored Numerical मान को Calculation में उपयोग में लेने के लिए उसे Integer, Float जैसे किसी Format में बदलने की जरूरत पडती है, उसी तरह से कई बार हमें किसी Basic Data Type के Variable में Stored मान को Character Array में Store करने की भी जरूरत पडती है। इस प्रकार की जरूरत को पूरा करने के लिए हम पिछले प्रकार के **atoi**() Function का Reverse Function **itoa**() Create कर सकते हैं।

```
/*itoa:  convert Numerical Integer to Characters Array Integer*/

  void itoa(int n, char str[])
  {
      int i = 0, sign;

      if ((sign = n) < 0)  /* record sign */
          n = -n;        /* make n positive */

        do             /* generate digits in reverse order */
        {
          str[i++] = n % 10 + '0';       /* get next digit */
      }while ((n /= 10) > 0);                    /* delete it */

      if (sign < 0)
          str[i++] = '-';
```

```
        str[i] = '\0';

        reverse(str);
    }
```

ये Function दो Arguments लेता है। पहला Argument वह Integer मान होता है, जिसे String str में Store करना है और दूसरा Argument एक Character Array होता है, जिसमें पहले Argument के Numerical Integer मान को Character Array में String के रूप में Store करना होता है।

जब ये Function Call किया जाता है, तब सबसे पहले ये Check किया जाता है कि ये संख्या Positive है या नहीं। Sign Check करने के लिए एक if Statement में ये Check करवाया जाता है कि संख्या 0 से छोटी होती है या नहीं। यदि संख्या 0 से छोटी हो तो **sign** नाम के Variable में Sign को Store किया जाता है और Negative Sign वाली संख्या को निम्न Statement द्वारा Positive संख्या में Convert कर लिया जाता है:

```
if ((sign = n) < 0)  /* record sign */
   n = -n;        /* make n positive */
```

Character Array में Store किए जाने वाले मान को Positive मान में Convert करने के बाद अब उस मान से एक–एक Digit को Character में Convert करने के लिए निम्नानुसार एक **do ... while** Loop का प्रयोग किया जाता है:

```
do                          /* generate digits in reverse order */
{
    str[i++] = n % 10 + '0';   /* get next digit */
}while ((n /= 10) > 0);     /* delete it */
```

इस Loop में सबसे पहले 10 का भाग देकर शेषफल प्राप्त किया जाता है। ये शेषफल Store किए जाने वाले Integer का अन्तिम Digit होता है। चूंकि अभी भी ये मान एक Digit है, जबकि Character Array में Store करने से पहले इसे Character में Convert करना जरूरी है, इसलिए इसमें मान Zero की ASCII Value को जोड कर इस मान को Digit से Character में Convert किया गया है।

Digit से Character में Convert होने के बाद इस Character को Character Array str के Index Number 0 पर Store कर दिया जाता है और उसके बाद **i** के मान को Increment किया जाता है।

इसके बाद while Condition Brace में Store किए जाने वाले Integer मान में 10 का भाग देकर Integer मान की अन्तिम Digit को Delete किया जाता है। अन्तिम Digit को Delete करने के बाद Check किया जाता है कि क्या Integer मान अभी भी 0 से बडा है या नहीं।

यदि Integer का मान 0 से बडा है, तो इसका मतलब ये होता है कि Integer में अभी भी कोई Digit है जिसे Character में Convert करना बाकी है, इसलिए ये Loop फिर से Iterate होता है और फिर से इस Integer संख्या में 10 का भाग देकर Reminder के रूप में Integer के अन्तिम Digit को प्राप्त किया जाता है। ये प्रक्रिया तब तक चलती रहती है, जब तक कि Integer की सभी Digits Character Array में Store नहीं हो जाती हैं।

एक बार Character Array str में Integer मान के Store हो जाने के बाद अब उस Integer के Sign को Character Array में Store करने की जरूरत पडती है। इसके लिए निम्न Statements का प्रयोग किया जाता है:

```
if (sign < 0)
   str[i++] = '-';

str[i] = '\0';
```

ये Statement Check करता है कि **sign** का मान **0** से छोटा है या नहीं। यदि जो Integer हमने Character Array str में Store किया है, वह Negative यानी 0 से छोटा होता है, तो इस Statement की **if** Condition **True** हो जाती है और Character Array में Index Number **i** की Position पर Minus के Sign को Store कर दिया जाता है। अन्त में String के अन्त को Specify करने के लिए Character Array में अन्तिम Character के रूप में **NULL** को Store किया जाता है, जो String के अन्त को Specify करता है।

चूंकि हमने Character Array में जिस Integer को Store किया है, वह Integer Reverse Order में Store हुआ है। इसलिए Reverse Order में Stored Integer मान को Forward Order में Convert करने के लिए हमें **reverse**() नाम के Function को Use करना होता है। ये Function हमने इस Function से पहले Develop किया है। ये Function Character Array में Stored Integer के String Representation को Reverse Order में Convert कर देता है।

जिस तरह से हमने एक Integer प्रकार के मान को एक Character Array में String के रूप में Store किया है, ठीक इसी तरह से हम अन्य प्रकार के मानों को भी एक String Representation के रूप में किसी Character Array में Store कर सकते हैं। इस Function को Practically Use करने के लिए हम निम्नानुसार एक **main**() Function Develop कर सकते हैं:

Program

```
#include <stdio.h>
```

```
#include <conio.h>

main()
{
    char str[15];
  int Integer = 30555;

  itoa(Integer, str);
  reverse(str);
  puts(str);
  getch();
}
```

Exercise:

1 एक Array को किसी Function में Argument के रूप में किस प्रकार से Pass किया जा सकता है? एक उचित उदाहरण Program द्वारा समझाईए।

2 एक String में से Sub-String Search करने का Function बनाईए।

3 **strcat**() Function व **strcpy**() Function के बीच के अन्तर को एक उदाहरण Program द्वारा समझाईए।

4 किसी String की Length ज्ञात करने का Function Create कीजिए जिसे strlen() Function के स्थान पर Use किया जा सके।

5 दो Pointers के Subtraction व Comparison को एक उदाहरण Program द्वारा समझाईए। Pointers Arithmetic की सीमाएं बताईए।

6 किसी Array के Address को किसी Function में Pass करने व उस Array को Access करने के तरीके को एक उदाहरण Program द्वारा समझाईए।

7 किसी Function का Pointer किस तरह से Create किया जा सकता है ? किसी Function के Pointer का प्रयोग करके उस Function को Call करने के तरीके को विस्तार से समझाईए।

8 Function का Pointer व Function Returning Pointer के अन्तर को एक उदाहरण Program द्वारा समझाईए।

9 विभिन्न प्रकार के Array व Pointers के बीच के आपसी सम्बंध को विस्तार से समझाईए।

10 Array का Pointer व Pointers का Array इन दोनों के अन्तर को एक उचित उदाहरण Program द्वारा समझाईए।

11 किसी Float Type के मान को एक Character Array में String Format में Store करने के लिए **atof**() Function Create कीजिए।

12 एक Character Array में Stored Float Type के String Format मान को Float Type के मान में Convert करने के लिए **ftoa**() Function बनाईए।

PREPROCESSOR

C Preprocessor

हम हमारे Source Program में जो Source Codes लिखते हैं, वे तब Process होते हैं, जब हम हमारी Source File को Compile करते हैं। लेकिन "C" में कुछ Preprocessors की व्यवस्था भी की गई है। Preprocessor के रूप में हम हमारी Source File में जो भी Statements लिखते हैं, वे Compilation से पहले Process होते हैं। इन Statements को सामान्यतया **Directives** कहा जाता है।

हम हमारे Program में C Preprocessors का प्रयोग किए बिना भी Program लिख सकते हैं। लेकिन Preprocessors "C" की एक बहुत ही अच्छी Facility है, जिसे Use करके हम हमारे Program को अधिक Manageable तरीके से Develop कर सकते हैं।

सामान्यतया इन Preprocessors का प्रयोग Header Files में किया जाता है, ताकि Header Files को कई प्रकार से Use किया जा सके। इन Preprocessors का प्रयोग Header File को Custom तरीके से Source File में Expand करने के लिए किया जाता है।

किसी "C" Program को लिखने व Execute करने के बीच कई Steps होते हैं। इन विभिन्न प्रकार के Steps के समूह को **"Build Process"** कहा जाता है। जब हम किसी Source Program को Compile करते हैं तो Program Compile होने से पहले एक अन्य Program में Pass होता है जिसे Preprocessor कहते हैं।

"C" में लिखे गए Program को हम सामान्यतया **"Source Code"** कहते हैं। Preprocessor Source Codes पर काम करता है और Expanded Source Codes Generate करता है। यदि किसी Source Code File का नाम Program1.C है तो Expanded Source Code File का नाम Program1.I हो जाता है।

Directives

Preprocessors कई Features प्रदान करता है जिसे Directives कहते हैं। Preprocessor Directives को लिखने का तरीका "C" के Statements लिखने के तरीके से अलग होता है। हर Preprocessor Directive **#** Sign से शुरू होता है और Preprocessor Statement के बाद Semi Colon का प्रयोग नहीं किया होता है।

हमने इन Directives का प्रयोग **#define** व **#include** के रूप में पिछले Programs में किया है। इन Directives को Program में कहीं भी Place किया जा सकता है लेकिन सामान्यतया इन्हें किसी भी Function Definition के पहले लिखते हैं चाहे वह main() Function ही क्यों ना हो। सामान्यतया Preprocessors को **Macro** भी कहते हैं। हम निम्न Directives को सर्वाधिक उपयोग में लेते हैं:

Directive	Function
#define	Defines a **Macro** Substitution
#undef	Undefines a **Macro**
#include	Includes a File in the Source Program
#ifdef	Tests for a **Macro** Definition
#endif	Specifies the end of #if
#ifndef	Checks whether a **Macro** is defined or not
#if	Checks a Compile Time Condition
#else	Specifies alternatives when #if Test Fails

इन Directives को तीन भागों में बांटा जा सकता है:

Macro Substitution Directive

ये एक ऐसा तरीका होता है जिसमें किसी Program का कोई Identifier किसी Predefined String से Replace होता है। Processor ये काम किसी #define Statement के Under में करता है। इस तरह के Statements को **Macro** Definition कहा जाता है। जैसे

```
#define START main(){
```

जब हम हमारे Program में इस प्रकार से किसी **Macro** को Define करते हैं तो Program में जहां भी **Macro** का नाम होता है Program Compile होने से पहले वहां पर Define की गई String Replace हो जाती है। इसे समझने के लिए निम्न Program देखिए–

Program

```
#include <stdio.h>
#include <conio.h>

#define START main()
{
    #define PI        3.14

    START
    float Area, radius;
    printf("Enter Radius");
    scanf("%f", &radius);

    Area = PI * radius * radius;
```

```
        printf("Area of Radius is %f ", Area);
        getch();
    }
```

जब इस Program को Compile किया जाता है तब Program Compile होने से पहले सारे **Macro** Statements Expand होते हैं। यानी ये Program Compile होने से पहले निम्नानुसार Format में Convert होता है

Program

```
    //Expanded File :
    #include <stdio.h>
    #include <conio.h>

    #define START main(){
    #define PI     3.14

    main()
    {
        float Area, radius;
        printf("Enter Radius");
        scanf("%f", &radius);

        Area = 3.14 * radius * radius;

        printf("Area of Radius is %f ", Area);
        getch();
    }
```

Source Code के Expand होने पर START के स्थान पर main(){ o PI के स्थान पर 3.14 मान आ जाता है। मान आने के बाद Program Compile होता है। **Macro** Definition के बाद जो मान लिखा जाता है वह वास्तव में एक String होता है। यानी इस Program में PI का जो मान 3.14 लिखा गया है वह float Value नहीं है बल्कि एक String है। **Macro** Substitution के कई तरीके होते हैं:

Simple Macro Substitution

सामान्यतया इस तरह के **Macro** किसी Constant को Define करने के लिए लिखे जाते हैं। जैसे

```
#define    COUNT    200
#define    TRUE     1
#define    LAMBDA   2.1
#define    NAME     "Raghva"
#define    SIZE     sizeof(float)*6
#define    AREA     (3.1415926 * 12)
```

सामान्यतया सभी **Macros** को Capital Letters में लिखा जाता है ताकि हम जब भी किसी **Macro** को अपने Program में Use करें तो हमें पता रहे कि हम किसी **Macro** को Use कर रहे हैं। सामान्यतया सभी Mathematical Expressions को कोष्ठक में लिखना चाहिए।

कई बार हम **Macro** Define करते समय जो String लिखते हैं, वह String काफी लम्बी हो जाती है। इस स्थिति में यदि हम चाहें कि String को अगली स्पदम में लिख दें, तो हम ये काम Back Slash का प्रयोग करके कर सकते हैं। जैसे

```
#define LONG_STRING    "This string is very long to fit in one \
                       line; so write it in next line."
```

Macros with Arguments

Macros का प्रयोग करके हम और अधिक जटिल Statements लिख सकते हैं। जैसे

```
#define CUBE(x)(x * x * x)
```

यहां CUBE(x) में लिखा गया Argument एक Formal Argument है। जब इस **Macro** को Use किया जाता है तब हम जो Argument CUBE **Macro** के कोष्ठक में देते हैं वह Expand होकर x * x * x हो जाता है। जैसे

```
int lside = 20, total;
total = CUBE(lside);
```

जब हम इस Statement को लिखते हैं तो Program Compile होने से पहले निम्नानुसार Expand होता है और total में 8000 Return करता है–

```
total = ( lside * lside * lside );
```

हम यदि चाहें तो CUBE **Macro** को निम्नानुसार भी Use कर सकते हैं–

```
int total, lside = 10, rside = 5;
```

```
total = CUBE(lside + rside)
```

ये Statement निम्नानुसार Expand होगा–

```
total = (lside + rside * lside + rside * lside + rside);
```

हम देख सकते हैं कि ये Expanding सही Result Generate नहीं करेगा क्योंकि rside का lside से पहले गुणा होगा जबकि हम चाहते हैं कि पहले lside व rside का जोड हो। इस स्थिति में हमें निम्नानुसार CUBE **Macro** को Modify करना होगाः

```
#define      CUBE(x)( (x) * (x) * (x) )
```

अब यदि हम इस **Macro** को अपने Program में Use करें तो ये निम्नानुसार Expand होगाः

```
total = (( lside+rside ) * ( lside+rside) * ( lside+rside ));
```

अब इस **Macro** से जो Result प्राप्त होगा वह सही होगा। हम निम्नानुसार कुछ अन्य **Macro** Define कर सकते हैं जो कि हमारे लिए उपयोगी होते हैंः

```
#define MAX(A,B)    (((a) > (b)) ? (a) : (b))
#define MIN(A,B)    (((a) < (b)) ? (a) : (b))
#define ABS(x)   (((x) > 0 ) ? (x) : (-(x)))
```

Nesting of Macros

अन्य Statements की तरह हम **Macro**s की भी Nesting कर सकते हैं। जैसेः

```
Code:
    #define     X               10
    #define     PI              3.1415
    #define     Y               X+10
    #define     SQUARE(z)       ((z) * (z))
    #define     CUBE(x)         (SQUARE(x) * (z))
```

एक **Macro** को किसी दूसरे **Macro** में Parameter के रूप में भी Use किया जा सकता है। जैसे ऊपर के Code में हमने X व Y को एक **Macro** बनाया है। इन्हे हम निम्नानुसार किसी अन्य **Macro** में Argument के रूप में भी Pass कर सकते हैं–

```
#define MAX(X,Y)    (((X) > (Y)) ? (X) : (Y))
```

इसी **Macro** को हम निम्नानुसार Nested भी कर सकते हैं:

```
int Maximum ;
Maximum = MAX(x,(MAX(y, z))
```

ये Nested **Macro** तीन संख्याओं में से बडी संख्या Return करेगा। एक बात हमेंशा ध्यान रखें कि **Macro** Template व उसके Argument List के बीच किसी तरह का Space नहीं होना चाहिए। जैसे:

```
#define MAX  (X,Y)   (((X) > (Y)) ? (X) : (Y))
```

ये **Macro** काम नहीं करेगा। क्योंकि MAX व उसके Arguments के बीच Space दिया गया है।

Un-defining a Macro

किसी Define किए गए **Macro** को Undefined करने के लिए हमें #undef Directive का प्रयोग करना होता है। इस Directive को हम निम्नानुसार Use कर सकते हैं–

```
#undef  CUBE(x)
#undef  MAX(x, y)
```

हम किसी भी **Macro** को Source File में कहीं भी Undefined कर सकते हैं। यदि हम किसी **Macro** को Program में कहीं पर Undefined कर देते है, तो वह **Macro** वहीं पर Damage हो जाता है। जिस Statement पर किसी **Macro** को Undefined किया जाता है, यदि उस Statement से आगे कहीं पर भी उस **Macro** को Use किया गया है, तो Compiler उस **Macro** को प्राप्त नहीं कर पाता है, क्योंकि हमने उस **Macro** को Undefined कर दिया होता है।

__LINE__ and __FILE__ Predefined Identifiers of Compiler

ये दोनों Identifiers Compiler में पहले से ही Defined किए गए है। __LINE__ Identifier में Currently Compile हो रही Line का Number होता है और __FILE__ Identifier में Currently Compiler हो रही File का नाम होता है। इसे हम निम्न Program द्वारा समझ सकते हैं:

Program

```
// File : Macro.c
#include <stdio.h>
#include <conio.h>

#define MAX 100

void main(void)
{
    #if MAX > 99
        printf("Name of the Program File is %s", __FILE__ );
        printf("\n");
        printf("The Line being Compiled is %d", __LINE__);
    #else
        #error You have defined the MAX Macro Less than 100"
    #endif

    getch();
}
```

Output

```
Name of the Program File is Macro.c
The Line being Compiled is 13
```

यदि हम चाहें, तो इन दोनों Compiler Identifiers के मान Change कर सकते हैं। यानी हम चाहें तो __LINE__ के Number की शुरूआत 100 Number से भी करवा सकते हैं और __FILE__ Identifier में किसी दूसरी File का नाम भी प्रदान कर सकते हैं। इसे समझने के लिए निम्न Program देखते हैं:

Program

```
#include <stdio.h>
#include <conio.h>

#line 100 "inline.cpp"
#define MAX 100

void main(void)
{
    #if MAX > 99
```

```
        printf("Name of the Program File is %s", __FILE__ );
        printf("\n");
        printf("The Line being Compiled is %d", __LINE__);
    #else
        #error You have defined the MAX Macro Less than 100"
    #endif

    getch();
}
```

#line **Macro** #include **Macro** की तरह काम करता है। इसके आगे जिस File का नाम दिया जाता है, Program उस File को भी Expand करता है और हमें Output में ये बताता है कि जो Line हमने इस Program मे Print की है, वह Line Originally inline.cpp File से आई है। सामान्यतया इस Directive का प्रयोग उन Utility Programs में किया जाता है, जो C के Code को Output के रूप में Produce करना चाहते हैं।

Built – In Predefined Macros

1. **__LINE__**
2. **__FILE__**
3. **__DATE__**
4. **__TIME__**
5. **__STDC__**
6. **__cplusplus__**

इन में से दो Predefined **Macros** के बारे में हमने अभी बताया है। शेष **Macros** के कामों को भी हम निम्न Program File द्वारा समझ सकते हैं:

Program

```
#include <stdio.h>
#include <conio.h>

#define __STDC__
#define MAX 100

void main(void)
{
#if MAX > 99
    printf("\nName of the Program File is  : %s", __FILE__);
```

```
        printf("\nThe Line being Compiled is   : %d", __LINE__);
        printf("\nThe Compilation Date of File : %s", __DATE__);
        printf("\nThe Compilation time of File : %s", __TIME__);

    #else
        #error You have defined the MAX Macro Less than 100"
    #endif

    getch();
    }
```

यदि इस File को .C Extension से Save किया जाए, तो File Compile होने से पहले निम्नानुसार एक Error Message आता है और File Compile नहीं होती है।

```
Error:  Macro.c(6,22):Error directive: File should be of .cpp Extension
```

ये Error इसलिए आता है क्योंकि हमने File को .C Extension से Save किया है। यदि हम File को .CPP Extension से Save कर दें, तो File ठीक प्रकार से Run होती है। चूंकि जब हम Header File बनाते हैं, तब कुछ Statements ऐसे होते हैं, जो C++ की Source File के लिए होते हैं और कुछ Statements ऐसे होते हैं, जो C की File के लिए होते हैं। इस स्थिति में हमें ये तय करना होता है कि कौनसा Code C++ की File के लिए है और कौनसा Code C की File के लिए है। __cplusplus **Macro** Identifier द्वारा हम यही तय करते हैं।

यदि हम File को .CPP Extension से Save करते हैं, तो File में __cplusplus **Macro** स्वयं ही Define हो जाता है, जबकि ऐसा ना करने पर ये **Macro** Define नहीं होता है और Compiler #error के आगे लिखे Statements को Error के रूप में Display कर देता है।

साथ ही चूंकि हमने #error Directive का प्रयोग किया है, इसलिए Source File Compile होने से पहले ही Terminate हो जाती है। यानी File Compile नहीं होती है। जब इस File को .CPP Extension से Save करके Run किया जाता है, तब हमें निम्नानुसार Output प्राप्त होता है:

```
Name of the Program File is  : Macro.cpp
The Line being Compiled is   : 16
The Compilation Date of File : Oct 13 2007
The Compilation time of File : 15:27:19
```

__STDC__ **Macro** Identifier Define करने के कारण Compiler केवल Standard C व C++ Codes को ही Source File में Implement करने देता है। यदि हम कोई Nonstandard Code

Source File में Use करें, तो Compiler हमें इस Directive को Defined करने के कारण ऐसा नहीं करने देता है।

जब हम File को Compile करते हैं, तब Compiler Source File को Object File में Convert करने की Date को __DATE__ नाम के **Macro** में Store कर देता है। इसी तरह से __TIME__ **Macro** में वह समय Store हो जाता है, जिस समय File Compile होती है। ये दोनों ही मान String Format में Stored रहते हैं, जिन्हें Output में Print करवाया जा सकता है।

 # and ## Preprocessors

सामान्यतया इनका प्रयोग define के साथ ही होता है। # Operator के बाद जो भी कुछ लिखा जाता है, वह String की तरह Behave करता है तथा ## Concatenating करने का काम करता है। इसे समझने के लिए निम्न उदाहरण देखते हैं:

Program

```
#include <stdio.h>
#include <conio.h>
#
#define str(s) # s
#define concat(x, y) x ## y

void main(void)
{
  int xy = 100;
     printf(str(This is all about Macro));
  printf("\n Value of xy is : %d", concat(x, y));
}
```

Output

```
This is all about Macro
Value of xy is : 100
```

यदि हम किसी Program में केवल # का प्रयोग करें और उसके आगे कुछ ना लिखें तो Compiler अकेले एक # को Ignore कर देता है, जैसाकि इस Program में किया गया है। इसे **Null Directive** कहते हैं।

File Inclusion Directive

#include एक File Inclusion Directive है। इस Directive का प्रयोग करके हम एक Source File में अन्य Source Files या Program File को जोडते हैं। जब Program काफी बडा होता है, तब हम Program को कई भागों में बांट कर, कई Source Files बना लेते हैं, और आवश्यकतानुसार किसी भी File को #include Directive द्वारा Main() Source File में जोड लेते हैं। जैसे कि हम हमारे Program में Input Output से सम्बंधित Functions को प्रयोग कर सकें, इसलिए #include<stdio.h> नाम की Header File को main() Program File में Include करते हैं।

Conditional Compilations

कई बार हम चाहते हैं कि हमारी आवश्यकता के अनुसार कुछ Statement को Execute किया जाए और कुछ को छोड दिया जाए। ऐसी जरूरतों को पूरा करने के लिए "C" में कुछ अन्य **Macros** बनाए गए हैं। ये निम्नानुसार हैं–

```
#ifdef
#endif
```

ये Directive "C" Compiler को बताता है कि यदि **Macro** Define किया गया है तो **Macro** के बाद के Statements को भी Execute करे अन्यथा #ifdef व #endif के बीच के Statements को Compile ना करे व शेष Program को Compile कर दे। इसका syntax निम्नानुसार होता है–

```
#include<header file>
#define Macro template name

main()
{
    Variables Declaration;
    clrscr();
    Statement 1;
    Statement 2;
    Statement 3;

    #ifdef Macro template name
        Statement 4;
        Statement 5;
        Statement 6;
    #endif
```

```
        Statement 7;
        Statement 8;
}
```

यदि **Macro** define किया गया है, तो Statement 4, 5 व 6 भी Compile होंगे अन्यथा "C" Compiler इन Statement को Compile नहीं करेगा। ये **Macro** बिल्कुल if Condition की तरह काम करता है। इसका प्रयोग हम निम्नानुसार किसी File में कर सकते हैं:

Program

```
#include <stdio.h>
#define MAX 100

void main(void)
{
    #if MAX > 99
        printf("You have defined the MAX Macro Greater than 100");
    #endif
}
```

कई बार ऐसी जरूरत होती है कि हमने यदि **Macro** define किया है, तो Statements 4, 5 व 6 का Execution ना हो और यदि **Macro** define नहीं किया हो तो ये Statements 4, 5 व 6 Compile हो जाएं। ऐसी स्थिति में हम #ifndef (if not defined) **Macro** का प्रयोग करते हैं और इसका Syntax निम्नानुसार होता है–

```
#include<header file>
#define Macro template name

main()
{
    Variables Declaration;
    clrscr();
    Statement 1;
    Statement 2;
    Statement 3;

    #ifndef Macro template name
        Statement 4;
```

```
            Statement 5;
            Statement 6;
        #endif
        Statement 7;
        Statement 8;
    }
```

यदि हम इस Concept को भी Implement करना चाहें, तो निम्नानुसार कर सकते हैं:

Program

```
#include <stdio.h>
#define MAX 100

void main(void)
{
    #ifndef MAX
        printf("You have defined the MAX Macro Greater than 100");
    #else
        printf("You have defined the MAX Macro Less than 100");
    #endif
}
```

यदि हम इस Program को Execute करें तो हमें Output में if() Statement का Message दिखाई देगा। लेकिन यदि हम MAX को **Define** ना करें, तो हमें Output में else Statement का Message दिखाई देगा।

इसी प्रकार से हम विभिन्न प्रकार से Compiler को अपनी Source File को Compile करने से पहले ही विभिन्न प्रकार के **Macros** द्वारा ये बता सकते हैं, कि उसे किस Code Block को Execute करना है और किसे नहीं करना है। जिस प्रकार से हम if व else का प्रयोग करते है, उसी प्रकार से हम #else **Macro** को भी प्रयोग कर सकते हैं। जैसे निम्न syntax देखें:

```
#include<header file>
#define Macro template name

main()
{
    Variables Declaration;
    clrscr();
    Statement 1;
```

```
        Statement 2;
        Statement 3;

        #ifdef Macro template name
            Statement 4;
            Statement 5;
        #else
            Statement 6;
        #endif

        Statement 7;
        Statement 8;
    }
```

यहां यदि **Macro** define किया गया हो तो Statement 4 व 5 का Compilation होगा और यदि **Macro** define ना किया गया हो तो Statement 6 का Compilation होगा। जिस प्रकार से हम if व else…if Condition को Use करते हैं उसी प्रकार से हम #if व #elseif Condition को भी use कर सकते हैं और इनके काम करने का तरीका भी बिल्कुल उसी प्रकार का है, जिस प्रकार का if व else if का है। निम्न syntax देखें:

```
    #include<header file>
    #define Macro template name

    main()
    {
        Variables Declaration;
        clrscr();
        Statement 1;
        Statement 2;

        #if Condition
            Statement 3;
        #else
        #if  condition
            Statement 4;
        #else
        #if Condition
            Statement 5;
        #else
```

```
            Statement 6;
        #endif
        #endif
        #endif

        Statement 7;
        Statement 8;
    }
```

इसी **Macro** definition को हम दूसरे तरीके से निम्नानुसार भी लिख सकते हैं:

```
    #include<header file>
    #define Macro template name

    main()
    {
        Variables Declaration;
        clrscr();
        Statement 1;
        Statement 2;

        #if Condition
            Statement 3;
        #elif Condition
            Statement 4;
        #elif Condition
            Statement 5;
        #else
            Statement 6;
        #endif

        Statement 7;
        Statement 8;
    }
```

इस प्रकार से हम एक ही Program को ऐसा बना सकते हैं, कि वही Program किसी खास **Macro** template को define होने पर दूसरा काम करे और **Macro** के define ना होने पर कोई दूसरा।

हमें कई बार ऐसी जरूरतें पडती हैं, कि Program के कुछ Statements को बदलना पडता है, लेकिन किसी कारणवश हमें वापस वही पुराने Statements की जरूरत पड जाती है, जिन्हें हम Delete कर चुके होते हैं। ऐसे में वापस हमें वे सारे Statements लिखने पडते हैं, जो कि एक Boring काम होता है।

हम **Macro** का प्रयोग उस समय करके Statement को delete करने के बजाय ऐसा कर सकते हैं कि Statements तो Program में रहेंगे लेकिन वे Compiler ही नहीं होंगे। इस प्रकार से हमारे समय की काफी बचत हो सकती है और हम उच्च स्तरीय Program लिख सकते हैं।

#error

जब हम Program के Compile होने से पहले ही किसी प्रकार की Error दिखाना चाहते हैं, तब हमें #error का प्रयोग करना होता है। उदाहरण के लिए निम्न Program को देखते हैं:

Program

```
#include <stdio.h>

void main(void)
{
    #if MAX > 99
        printf("You have defined the MAX Macro Greater than 100");
    #else
        #error You have not defined the MAX Macro
    #endif
}
```

इस Program में हमने MAX नाम के **Macro** को Define नहीं किया है। इसलिए जब Program Compile होने से पहले सभी **Macros** को Expand करता है, तब उसे MAX नाम का **Macro** नहीं मिलता है। इसलिए Program निम्नानुसार Program को Compile किए बिना ही एक Error Message प्रदान कर देता है:

```
Error: Error directive: You have not defined the MAX Macro
```

Function And Macros

हमने पिछले कुछ उदाहरणों में देखा है कि हम **Macro** में Arguments Pass कर सकते हैं और किसी संख्या का Square या CUBE आदि ज्ञात कर सकते हैं। **Macro** को Call करना किसी Function को Call करने जैसा ही है।

इन दोनों में अन्तर ये है कि **Macro** के Call में Preprocessor सभी **Macro**s को Expand कर देता है, जबकि किसी Function के Call में जब किसी Function को Call किया जाता है, तभी वह Function Execute होता है और जैसे ही Program Control उस Function से Return होता है, Function व उसके विभिन्न Variables Destroy हो जाते हैं। इससे Memory की बचत होती है। जबकि एक **Macro** पूरे Program में Expand हो जाता है इसलिए काफी Memory Waste होती है।

दूसरा अन्तर ये है कि यदि हम **Macro** का प्रयोग करते हैं तो हमारा Program Control Jump नहीं करता। जबकि यदि हम Functions का प्रयोग करते हैं तो हमारा Program Control एक Program में कई बार Jump करता है। इससे Time अधिक लगता है।

सारांश में कहें तो किसी **Macro** के प्रयोग से समय की बचत होती है लेकिन Memory अधिक Use होती है जबकि Function के Call से Memory कम Use होती है लेकिन Program Control के बार–बार Function पर Jump करने के कारण समय अधिक लगता है। जब हमें छोटे से काम को करना हो तब हमें **Macro** Define करना चाहिए। लेकिन जब हमें काफी जटिल काम करना हो तब हमें **Macro** के स्थान पर Functions का प्रयोग करना चाहिए।

Build Process

"C" की एक Source File की Executable File बनने में कई Steps Follow होते हैं। जब हम "C" की किसी Source File को Compile करके Executable File में Convert करना चाहते हैं, तब सबसे पहले सभी Preprocessor Directives Expand होते हैं। हमारे Source Codes इस Expanded Source File में Store होते हैं। इस Intermediate Expanded Source File का Extension **.i** होता है।

सभी Preprocessor Directives Expand होने के बाद Expanded Source File Compiler पर Compile होने के लिए जाती है। Compilation के दौरान Compiler Expanded Source File को Assembly Language File में Convert करता है। इस File का Extension .asm होता है।

Assembly File Generate होने के बाद इस File से Object File Create होती है। इस Object File का Extension **.obj** होता है। अन्त में सभी Source Files आपस में Link होती हैं और एक Final File Create होती है। इस Final File का Extension **.exe** होता है।

DYNAMIC MEMORY ALLOCATION

Dynamic Memory Allocation

Computer में कोई भी Program इसलिए बनाया जाता है ताकि विभिन्न प्रकार के Data को Process किया जा सके और किसी समस्या का समाधान प्राप्त किया जा सके। विभिन्न प्रकार के Data को Memory में Store करने के लिए हम विभिन्न प्रकार के Variables Create करते हैं।

वास्तविक जीवन में हमेंशा ये निश्चित नहीं होता कि कितने Data के साथ प्रक्रिया करनी है। यानी मानलो कि किसी Company में 20 Employee काम करते हैं। उस Company में आवश्यकतानुसार किसी Employee को Company से निकाला भी जा सकता है और किसी नए Employee को Company में Appoint भी किया जा सकता है।

यानी ये निश्चित नहीं होता कि हमेंशा उस Company में 20 Employee ही काम करेंगे। इसलिए यदि किसी Company के Employees का Record रखने के लिए यदि Computer में कोई Program Develop किया जा रहा है तो Computer में भी हम एक निश्चित संख्या में Variables Declare नहीं कर सकते।

Computer में भी हमें एक ऐसी व्यवस्था की जरूरत होगी जिससे यदि Data बढते हैं तो नए Variables Create हो सकें और यदि Data घटते हैं तो किसी पुराने Variable को Delete किया जा सके ताकि उस Variable द्वारा Reserve की गई Space का कोई अन्य Program उपयोग कर सके।

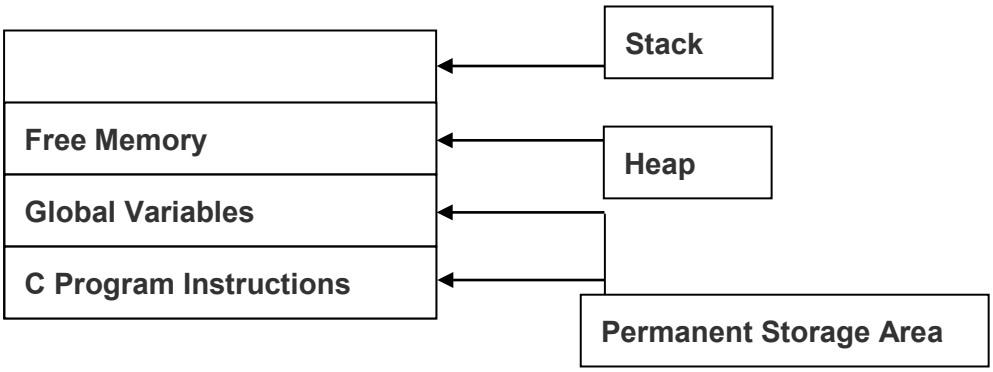
"C" Language में इस काम को करने के लिए Dynamic Memory Allocation Concept को Use किया जाता है। यानी हम Program में अपनी आवश्यकतानुसार Variables Create कर सकते हैं और उन्हें मान प्रदान कर सकते हैं। इस व्यवस्था को **Dynamic Memory Allocation** कहा जाता है।

Dynamic Memory Allocation के लिए "C" में कुछ Memory Management Functions हैं जिनका प्रयोग करके हम Program के Run Time में विभिन्न प्रकार के Variables Create करके उन्हें Memory प्रदान कर सकते हैं। ये Function Memory में Data के लिए Storage Space बनाने या Space हटाने का काम करते हैं।

जब कोई "C" Program Execute होता है, तो "C" Compiler सबसे पहले **Static** व **Global Variables**, **Permanent Storage Area** में अपना Space Reserve करते हैं। फिर Local Variables Memory में अपना Space Reserve करते हैं। Local Variables जो Space Reserve करते हैं, उस Space को **Stack** कहा जाता है।

Permanent Storage Area और **Stack** के बीच में हमेंशा कुछ Space बचा रहता है। इस बची हुई Memory Space को **Heap** कहते हैं। Heap का मान Local Variable पर निर्भर करता

है। यानी जब Local Variables Create होता है, तब Heap का मान कम हो जाता है और जब Local Variables Destroy होता है या Local Variable द्वारा Reserve किये गए Memory Space को खाली किया जाता है, तब Heap का मान बढ जाता है। Memory के Management की व्यवस्था को निम्नानुसार चित्र से दर्शाया गया है:



Heap Area का मान Variables के Creation व Deletion के अनुसार घटता या बढता रहता है। कई बार ऐसी स्थिति भी आती है कि Heap Area में नया Variable Create करने के लिए Space नहीं बचता। इस स्थिति को Overflow कहते हैं। Overflow की स्थिति में हमेंशा NULL Return होता है।

malloc() Function

इस Function का प्रयोग करके हम Memory का एक Block Create कर सकते हैं और उसे किसी Variable को Allot कर सकते हैं। जब हम Dynamic Memory Allocation के लिए इस Function का प्रयोग करते हैं, तब ये Function Memory में किसी Specified Data Type का एक Memory Block बनाता है और उस Memory Location या इस **malloc**() Function द्वारा बनाए गए Block Space का Address Return करता है।

इस Address को उसी Data Type प्रकार के Pointer Variable में Store किया जाता है और इस Pointer का उपयोग करके आवश्यकतानुसार Data Input किया जाता है। इस Function का Syntax निम्नानुसार होता है–

```
ptr = ( Data Type * ) malloc ( sizeof (Data Type ));
```

ptr

हमें जिस प्रकार के Data के लिए Memory Allocate करनी है, उसी Data Type का ये एक Pointer Variable होता है, जिसमें malloc() Function द्वारा बनाए गए Memory Block का Address Store होता है।

(Data Type *)

हमें जिस प्रकार के Data के लिए Memory Space Allocate करना होता है, उसी प्रकार के Data Type का Address Pointer **ptr** को Return करना होता है। ये Declaration ptr को Block का Address Return करने का काम करता है।

malloc(sizeof (Data Type))

malloc() Function का प्रयोग Memory में Space Block बनाने के लिए किया जाता है। इस Function में Argument के रूप में ये बताना होता है कि हम जिस प्रकार के Data के लिए Memory में Space Block बनाने जा रहे हैं, वह Data Type Memory में एक Data के लिए कितने Byte लेता है।

जैसे यदि हम int प्रकार के Data के लिए Memory Allocation कर रहे हैं, तो इस Function को बताना होता है कि int प्रकार का Data Type Memory में कितने Byte लेता है।

sizeof Operator का प्रयोग किसी भी Data या Data Type द्वारा Memory में लिए जाने वाले Space की Calculation के लिए किया जाता है। इसीलिए यहां इसका प्रयोग करके malloc() Function को बताया जाता है, कि हम जिस Data Type के लिए Memory Allocation करने जा रहे हैं, वह Data Memory में कितनी Space Reserve करेगा।

Memory Allocation से सम्बंधित सारे Functions "C" की **alloc.h** नाम की Header File में होते हैं। इसलिए Memory Allocation से सम्बंधित कोई भी काम करने के लिए हमें इस Header File को हमारे Program में Include करना जरूरी होता है।

Program

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
main()
{
    int *ptr, j, k;
    clrscr();

    // Memory Allocation For Inputting Integers
    ptr = (int * ) malloc (sizeof ( int ) );
    printf("\n Address of Allocated Memory Block is %u", ptr);
    printf("\n How Many Integers You want to Enter?");
    scanf("%d", &j);
```

```
    for(k=1; k<=j; k++)
    {
        printf("\n Input %d Integer ", k);
        printf("At Address %u :\t", ptr);
        scanf("%d", ptr);
        ptr++;
    }
    printf("\n Address of Last Allocated Memory Block is %u \n", ptr);
    printf("\n Inputted Integers Are :\n");

    for(k=1; k<=j; k++)
    {
        ptr--;
        printf("\t\n\n %d \t ", *ptr);
        printf("At Address %u  \t", ptr);
    }
    free(ptr);
}
```

Output

```
Address of Allocated Memory Block is 2050
How Many Integers You want to Enter?4
Input 1 Integer At Address 2050 :    12
Input 2 Integer At Address 2052 :    22
Input 3 Integer At Address 2054 :    23
Input 4 Integer At Address 2056 :    36

Address of Last Allocated Memory Block is 2058
Inputted Integers Are :
36    At Address 2056
23    At Address 2054
22    At Address 2052
12    At Address 2050
```

आइये अब इस प्रोग्राम का Execution समझते हैं। इस प्रोग्राम में हमें Data, Memory में Space Block को Allocate करके फिर Input करना है, इसलिए alloc.h नाम की Header File को Program में Include किया गया है।

*ptr एक int प्रकार का Pointer Variable है, जिसमें Allocate की गई Space का Address Store करना है, इसलिए इसे Pointer प्रकार का लिया है। चूंकि हमें Program में Integer प्रकार के मान Input करने हैं, इसलिए int प्रकार के मान के लिए निम्नानुसार Memory Allocate की गई है:

```
ptr = (int * ) malloc (sizeof ( int ) );
```

int प्रकार के मान के लिए हमने **sizeof(int)** लिखा है, जो malloc() को बताता है कि हमें int प्रकार के मान Input करने हैं। इसलिए Allocate किया जाने वाला Memory Block 2 Byte का होना चाहिये। साथ ही जो Memory Block बनता है, उस Memory Block का Address हमें ptr में चाहिये।

हम जानते हैं कि Address हमेंशा Pointer Variable में Store होता है, इसलिए इस Address को प्राप्त करने के लिए हमने **(int *)** Statement लिखा है। ये Statement Memory में Allocate की गई Memory Block का Address ptr में Store करता है।

इस प्रकार से ptr = (int *)malloc(sizeof (int)); Statement द्वारा int प्रकार के मान के लिए Memory में एक Space Block बनता है और उस Block का Address ptr Pointer Variable को प्राप्त हो जाता है। "Address of Allocated Memory Block is %u " Statement User को बताता है कि Memory में जो Space बना है, उसका Address क्या है।

अब निम्नानुसार एक Message द्वारा User से ये पूंछा जाता है कि वह कितने मान Input करना चाहता है।

How Many Integers You want to Enter?

यहां User जो भी संख्या Input करता है, For Loop को उतनी ही बार चलाया जाता है। अब Loop में संख्या Input करने के लिए Message आता है। User जब संख्या Input करता है तब वह संख्या, उस Allocate की गई Memory Location पर Store हो जाती है।

अगली संख्या को Input करने के लिए ptr का Increment किया गया है। अब जब वापस Loop चलता है, तो वापस User से मान मांगा जाता है। ये मान ptr के नए Address पर Store होता है, क्योंकि ptr को Increment किया गया है।

इस उदाहरण में हम देख रहे हैं कि जो अन्तिम मान User Input करता है, वह मान Memory Address 2056 पर Store हो रहा है, और "Address of Last Allocated Memory Block is 2058" है।

ऐसा इसलिए होता है क्योंकि Loop में दूसरा मान Input करने से पहले ptr के Address को Increment किया गया है। जब हम अन्तिम मान Input कर देते हैं, उसके बाद भी ptr का Increment होता है, जिससे ptr में अन्तिम Address 1058 Store रहता है। लेकिन Loop Terminate हो जाने से इस Location पर कोई मान Store नहीं रहता।

इसलिए ये जरूरी हो जाता है कि Output में इनका मान Print करने से पहले या तो ptr को वही Address प्रदान किया जाए जो कि Memory Allocation के समय था ताकि ptr प्रथम Memory Block के Address से Increment हो और क्रम से हमें सारे Input किये गये मान प्रदान कर दे, या फिर जिस प्रकार से Loop में ptr को Increment करके मान Input किये गए हैं, उसी प्रकार से Loop का मान चला कर ptr को क्रमशः उल्टे क्रम में Decrement किया जाए और मान Screen पर Print किया जाए।

यहां ptr का अन्तिम मान 1058 है और हमारे मान 2056 तक ही Store हुए हैं, इसलिए सबसे पहले ptr को Decrement किया गया है, फिर क्रम से सारे मान उल्टे क्रम में प्राप्त किये गए हैं। हमने देखा कि सारे मान जिस क्रम में Input किये गए हैं ठीक उसके विपरीत क्रम में प्राप्त किये गए है। इस प्रक्रिया को LIFO (Last In First Out) कहा जाता है।

जब Heap Area में Space नहीं होता है तब malloc() Function NULL Return करता है। जिस प्रकार से हमने int प्रकार के मान के लिए Memory में Space बनाया है, उसी प्रकार से किसी भी प्रकार के Data Type के लिए Memory में Space बनाया जा सकता है। जिस Data Type का Memory Block बनाया जाता है उसी Data Type के Pointer Variable को उस Allocated Memory Block के प्रथम Byte का Address प्रदान करना होता है।

यदि हम चाहें तो एक निश्चित Size की भी Space Allocate कर सकते हैं। इस काम के लिए हमें Data Type का मान व उस Data Type के कितने मानों के लिए Space Allocate करनी है, ये दोनों ही Information, Argument के रूप में malloc() Function को देनी होती है। जैसे हमें 7 Characters की एक String के लिए Memory Allocate करनी है तो हमें निम्नानुसार Statement देना होगा–

```
cptr =  ( char * ) malloc (7 * sizeof (char ) );
```

ये Statement 7 Byte की Contiguous Memory Allocate करेगा और इस Memory Block के प्रथम Byte का Address cptr को दे देगा।

calloc ( ) Function

malloc() Function द्वारा हम एक Memory Block बनाते हैं लेकिन calloc() Function द्वारा हम Data Type प्रकार के समान आकार के कई Memory Block बना सकते हैं। इसका Syntax निम्नानुसार होता हैः

```
ptr = (Data Type * ) calloc ( n, Data Type size );
```

इस Declaration में हम Data Type प्रकार के **n** Memory Block बना सकते हैं, जिसकी Size, Data Type Size के बराबर होती है। इसका काम करने का तरीका ठीक malloc() Function जैसा ही है।

उदाहरण के लिए हमें 10 Characters की एक String Store करने के लिए 10 Memory Block बनाने हैं, तो हम calloc() Function का प्रयोग करके निम्नानुसार Declaration करेंगेः

```
ptr = ( char * ) calloc ( 10, sizeof (char ) );
```

इस Statement से char प्रकार के 10 Memory Blocks **Heap Area** में बन जाएंगे। जब हमें पता होता है कि Input किये जाने वाले Data कितने हैं, तब हम इस प्रकार से Memory Allocation करते हैं।

free() Function

जब हम कोई Memory Allocate करते हैं तो उस Memory को खाली करना भी जरूरी होता है। यदि हमने जो Memory Allocate की है, उसे Release नहीं करते हैं, तो वह Memory किसी अन्य प्रोग्राम द्वारा Use नहीं की जा सकती है। क्योंकि प्रोग्राम के खत्म होने के बाद भी ये Memory उस ptr द्वारा Reserve रहती है। इससे प्रोग्राम की Speed कम हो जाती है। खास करके जब हमारे पास Memory कम होती है, तब Memory Wastage की समस्या परेशानी पैदा करती है।

इसलिए जब हमारे Program का काम समाप्त हो जाए, तब ये जरूरी है कि Allocate की गई Memory को Release किया जाए। Memory को Release करने के लिए free() Function का प्रयोग किया जाता है। इस Function में उस Pointer Variable का नाम दिया जाता है, जिसमे हमने Memory Block का Address Store किया था।

जैसे हमने निम्न प्रोग्राम के अंत में Memory को Release किया है, क्योंकि इस स्थान के बाद Program में कहीं भी ptr की जरूरत नहीं है। malloc() व calloc() Function मे एक मुख्य अन्तर ये है कि malloc() Function से Allocated Memory Block में Garbage Value रहती है जबकि calloc() Function से Allocated Block में Default मान 0 Initialized रहता हैः

Program

```
#include<stdio.h>
#include<alloc.h>

main()
{
    int *ptr, j, k;
    clrscr();

    // Memory Allocation For Inputting Integers
    ptr = (int * ) malloc (sizeof ( int ) );
    printf("\n Address of Allocated Memory Block is %u", ptr);
    printf("\n How Many Integers You want to Enter?");
    scanf("%d", &j);

    for(k=1;k<=j; k++)
    {
        printf("\n Input %d Integer ", k);
        printf("At Address %u :\t", ptr);
        scanf("%d", ptr);
        ptr++;
    }

    printf("\n Address of Last Allocated Memory Block is %u \n", ptr);
    printf("\n Inputted Integers Are :\n");

    for(k=1;k<=j; k++)
    {
        ptr--;
        printf("\t \n \n %d \t ", *ptr);
        printf("At Address %u  \t", ptr);
    }
    free(ptr);
    getch();
}
```

realloc() Function

जब हम malloc() या calloc() Function से Memory Allocate करते हैं, तब कई बार हम आवश्यकता से अधिक या कम Memory Allocate कर लेते हैं। Program में Allocated Memory का मान इस Function से बदला जा सकता है। इस बात को निम्न प्रोग्राम द्वारा समझाया गया है:

Program

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>

main()
{
    char *str;

    /* Allocate Memory for string */
    str = (char *) malloc(10);

    /* copy "Hello" into string */
    strcpy(str, "Hello");

    printf("String is %s\n  Address is %u\n", str, str);
    str = (char *) realloc(str, 20);
    printf("String is %s\n  New address is %u\n", str, str);

    /* free Memory */
    free(str);
}
```

Output

```
String is Hello
  Address is 1502
String is Hello
  New address is 1516
```

malloc(), calloc() व realloc() तीनों ही Functions को Use करने का तरीका एक जैसा ही है। **realloc**() Function में हमें उसी Pointer को नए Memory Block का Address Assign करना होता है, जिसे malloc() या calloc() Function द्वारा हमने Address दिया था।

वास्तव में जब realloc() Function को Use किया जाता है, तब ये Function malloc() या calloc() Function द्वारा Allocate किये गए Address पर Stored मानों को नए Allocate किये गए Memory Block में Transfer कर देता है।

जैसा कि Program के Output में बताया गया है। पहले malloc() Function से 10 Characters के लिए Memory Allocate की गई है, जो कि Memory Location 1502 पर Store होती है। realloc() Function से जब 20 Characters के लिए वापस Memory को Re - Allocate किया गया, तब String नए Location 1516 से Store होना शुरू हो गया है।

Exercise:

1. Preprocessor Directives से आप क्या समझते हैं ?
2. Preprocessor Directives को मुख्यतः कितने भागों में बांटा गया है ? नाम लिखो।
3. Macro Substitution Directives को विस्तार से समझाओ।
4. Conditional Compilation को समझाईए।
5. Function व Macro के बीच के अन्तर को एक सामान्य उदाहरण द्वारा समझाईए।
6. Dynamic Memory Allocation से आप क्या समझते हैं ?
7. **malloc**() Function व **free**() Function को एक उदाहरण Program द्वारा समझाईए।
8. **calloc**() Function व **realloc**() Function के बीच के अन्तर को समझाईए।
9. एक Macro **Swap(x, y)** Define कीजिए जो Integer Type के दो Variables के मानों को आपस में Swap करने का काम करे।
10. एक Macro बनाईए जो तीन संख्याओं में से बडी संख्या Return करे।

STRUCTURE

Structure

हमने देखा है कि एक Array में एक ही प्रकार के Data Type के कई मानों को एक साथ Store किया जा सकता है। लेकिन यदि हम अलग–अलग प्रकार के कई Data Items को एक ही नाम के Variable में Store कर दें, तो बनने वाले Modal को **Structure** कहते हैं।

Structure Logically Related Data Items का एक समूह होता है। इसमें Declare किये गए Variables एक समूह के रूप में होते हैं। Structure एक User Defined Data Type है। किसी Structure में लिखे गए विभिन्न Variables उस Structure के **Members** कहलाते हैं। यह जरूरी नहीं होता है कि किसी Structure के सभी Members का Data Type समान हो। वे भिन्न प्रकार के हो सकते हैं।

Structure एक ऐसा Tool है, जो Logically Related Data Items के एक समूह को एक ही नाम से संचालित कर सकता है। जैसे Student_name, roll_number और marks तीन अलग–अलग प्रकार के Data Type के Data Item हैं, लेकिन ध्यान से देखने पर हम पाते हैं कि ये Logically एक दूसरे से सम्बंधित हैं, क्योंकि ये तीनों मिल कर किसी विधार्थी के सम्बन्ध में कुछ जानकारी देते हैं।

इसलिए इन तीनों को हम Student_record नाम के एक ही Variable में रख कर अलग–अलग प्रकार से संचालित कर सकते हैं। Structure जटिल Data Items के समूह को एक अधिक Meaningful तरीके से संगठित (Organize) करने की सुविधा प्रदान करता है।

Structure Definition

जैसाकि हमने अभी बताया कि Structure एक User Defined Data Type है। यानी जिस तरह से int प्रकार से Declare किये गए Variables Integers मान को ही Store कर सकते हैं, और यदि हमें Memory मे characters Store करने हैं, तो हमें Variable को char प्रकार के Data Type का Declare करना पडता है, उसी प्रकार Structure एक ऐसा User Defined Data Type है, जिसे User अपनी जरूरत के अनुसार बनाता है और इसमें विभिन्न प्रकार के मिश्रित Data Types के Data Items को Store कर सकता है।

सबसे पहले User को एक Structure बनाना होता है। User इस Structure में ना तो सीधे ही मान Store कर सकता है ना सीधे ही इससे मान प्राप्त कर सकता है क्योंकि ये Structure तो मात्र एक प्रकार का Data Type है।

जैसे माना हमें 2 को 3 से जोडना है। 2 व 3 दोनों ही पूर्णांक हैं। हम Computer में इन्हें तभी जोड सकते हैं, जब ये Memory में Store हों। Memory में Store करने के लिए हमें इनके लिए

Variable Declare करने पडते हैं। Variables इसलिए Declare करने पडते हैं क्योंकि दोनों संख्याएं Integers प्रकार की होने के बावजूद हम इन्हे int में Store नहीं कर सकते क्योंकि int कोई Memory Location नहीं है। int तो मात्र एक Data Type है जो Compiler को ये बताता है, कि जो Memory Location int प्रकार के Data Type के लिए Reserve होगी, उस Location पर केवल पूर्णांक मान ही Store होंगे और Variable उस Memory Location का नाम होता है, जहां वह वांछित मान जा कर Store होगा व जहां से मान वापस आवश्यकतानुसार प्राप्त किया जाएगा।

इस प्रकार से हम Data Type व Variable को इस प्रकार से समझ सकते हैं कि Data Type Memory में किसी स्थान पर अपने प्रकार के अनुसार कुछ Space Reserve करता है और ये निश्चित करता है कि उस Space में किस प्रकार के मान Store होंगे और उस Reserved Memory Location को जिस नाम से पहचाना जाएगा, वह नाम Variable का नाम होता है।

इसी Concept के आधार पर यदि हम कह रहे हैं कि Structure एक User Defined Data Type है, तो इसका यही मतलब है, कि Structure Memory में अपनी Size के अनुसार कुछ Space Reserve करेगा और Structure प्रकार के Variables उस Reserve किये गए Space का नाम होंगे जिससे उस Memory Location को पहचाना जाएगा जहां उस Structure प्रकार का मान Store है।

Structure Declaration

किसी भी Structure को निम्न Format में Declare किया जाता हैः

```
struct tag
{
    Member 1;
    Member 2;
    Member 3;
    . . .
    Member n;
};
```

किसी भी Structure को Declare करने के लिए struct Key word का प्रयोग किया जाता है। यह एक ***tag*** होता है, यानी Structure प्रकार के Variables Declare करने के लिए हमें इसी Tag का प्रयोग करना होगा। ठीक उसी प्रकार से जैसे हम Integers प्रकार के Variables Declare करने के लिए int या Character प्रकार के Variable Declare करने के लिए char tag को Use करते है।

कोई भी Structure एक Format बनाता है, जिसे किसी Structure Variable को Declare करने के लिए Use किया जाता है। यानी ये Structure Format ये बताता है कि Structure प्रकार का

कोई भी Variable किस प्रकार के मानों का लेन देन कर सकेगा। ये Structure Format Memory में तब तक किसी प्रकार का कोई Space Reserve नही करता है, जब तक कि इसमें Structure प्रकार के Variables Declare नहीं किए जाते।

किसी Structure के Member कोई भी Variable, Pointer, Array या अन्य प्रकार का Structure हो सकता है। हम एक उदाहरण द्वारा समझते हैं कि Structure किस प्रकार से Define होता है। माना हम एक Book_Database बनाना चाहते हैं, जिसमें Book_name, Author's_name, pages व Price Store करनी है। हम इसके लिए निम्नानुसार एक Structure define कर सकते हैं:

```
struct book_bank
{
    char title[20];
    char author[30];
    int page;
    float price;
};
```

यहां ये Structure इस प्रकार का Format बनाता है, जिसमें चार तरह के Fields title, author, pages व price की Details को Store किया जा सकता है। इस Structure के हर Field को Structure Element या Member कहते हैं और इस Structure का हर Member एक अलग प्रकार के Data Type को Store कर सकता है। book_bank इस Structure का नाम है, जिसे Structure tag कहा जाता है।

ध्यान दें कि अभी तक इस Structure Definition में कोई भी Variable Declare नहीं किया गया है। केवल ये Format बनाया गया है कि कोई Variable किस–किस प्रकार के Data को Accept कर सकेगा। Structure प्रकार के Variable को हम प्रोग्राम में कहीं भी struct Keyword का प्रयोग करके Declare कर सकते हैं। जैसे–

```
book_bank book1,book2, book3;
```

इस प्रकार Structure book_bank प्रकार के तीन Variables book1, book2 व book3 Declare किये गए हैं। किसी भी Structure का Format बनाते समय ही यदि हमें Variables Declare करने हों, तो Closing Bracket के बाद हम सीधे ही Variables को Declare कर देते हैं। उसके बाद Structure को सेमी कॉलन से बन्द करते हैं। जैसे

```
struct book_bank
{
    char title[20];
```

```
        char author[30];
        int page;
        float price;
    } book1, book2, book3;
```

Accessing the Structure Members

Structure के Member स्वयं Variable नहीं होते हैं। इन्हें struct प्रकार के Variable से जोड कर Meaningful बनाया जाता है। यानी इसी Structure को देखें तो title का कोई अर्थ नहीं है। केवल title लिख देने से ये बात पता नहीं चलती है, कि हम किस title की बात कर रहे हैं। लेकिन यदि "book का title" कहा जाए तो ये एक पूरी सूचना होती है कि हम किसी किताब के title की बात कर रहे हैं।

इस प्रकार Variable book व Structure के Member title के बीच एक link बना कर हम Variables को Access करते हैं। ये Link बनाने का काम हम "." (DOT) Operator से करते हैं। जैसे कि book1.title लिखा जाए तो ये Statement book1 के title को Represent करता है। Structure प्रकार के Variables का प्रयोग किसी भी अन्य प्रकार के Variables की तरह ही किया जा सकता है। जैसे

```
    strcpy(book1.title, "Mahabharat");
```

ये Statement book1 Variable में Mahabharat Store कर देगा यानी book1 का title महाभारत हो जाएगा।

Initializing the Structure Members

किसी भी अन्य Data Type के Variable की तरह ही हम Structure Variable को भी प्रारम्भिक मान दे सकते हैं या Initialize कर सकते हैं। Structure को जब Initialize किया जाता है, तब इसे static प्रकार की Storage Class में रखना जरूरी होता है। जब हम ANCI "C" में काम करते हैं तब Structure को auto रखना पडता है। हम एक साधारण प्रारूप द्वारा मान Initialize करना बता रहे हैं।

```
    struct yard
    {
        int width;
        float height;
    }room1= { 30, 40.5};
```

इस Structure में जमीन की कुल चौडाई 30 feet व कुल लम्बाई 40.5 feet Structure Variable room1 को प्राप्त हो जाएगी। जब Variable को Program में कहीं और Initialize करना हो तब इसे निम्न अनुसार initialize करते हैं:

```
static yard room1 = { 30, 40.5};
static yard room2 = { 34, 50.5};                etc...
```

ये बात हमेंशा ध्यान रखें कि हम किसी Structure का Format बनाते समय किसी भी Member को Initialize नहीं कर सकते हैं। "C" हमें इसकी छूट नहीं देता है।

Structure with Array

Structure का प्रयोग Array के साथ मिलाकर किया जा सकता है। हम जानते हैं कि Structure का प्रयोग Related Data Items के Group को Access करने के लिए होता है। हम Structure के विभिन्न Members को Access करने के लिए Variable Declare करते हैं। जब हमें बहुत सारे Variable किसी प्रोग्राम में Declare करने हों, तो हम इस समस्या से बचने के लिए Array का प्रयोग करते हैं। जैसे यदि किसी कक्षा के सभी Students का नाम व उस Student द्वारा प्राप्त किये गए विभिन्न विषयों में प्राप्त अंकों को Store करना हो, तब Array का प्रयोग करना उपयोगी सिद्ध होता है। जैसे:

```
struct marks
{
    int sub1;
    int sub2;
    int sub3;
};

main()
{
    static struct marks Student[4] = {{41,12,22}, {52,55,66}, {32,55,68}, {55,44,88}};
}
```

यदि हमें चार Students के तीन विषयों में प्राप्त अंको को Store करना हो, तो Variable को हमें Array प्रकार का Declare करना ठीक रहता है। हमने यहां प्रारम्भिक मान प्रदान किया है। यदि हम चाहें तो Loop द्वारा जिस प्रकार पिछले अध्यायों में मान Input किया है, उसी प्रकार यहां भी मान Run Time में Input कर सकते हैं। इस प्रारूप में Store मानों को हम निम्नानुसार Show कर सकते हैं:

```
Student0.sub1      41
Student0.sub2      12
Student0.sub3      22
Student1.sub1      52
Student1.sub2      55
Student1.sub3      66
Student2.sub1      32
Student2.sub2      55
Student2.sub3      68
Student3.sub1      55
Student3.sub2      44
Student3.sub3      88
```

अब हम इसी प्रोग्राम में Data, Run time में Input करेंगे। इस प्रोग्राम में Data, Run Time में Input करने के लिए प्रोग्राम निम्नानुसार बनेगा:

Program

```
struct marks
{
    int sub1;
    int sub2;
    int sub3;
};

struct marks stud[4];
int k;
clrscr();

for( k = 0; k < 4; k++)
{
    printf("\n Enter marks of Student %d ", k+1);
    printf("\n Enter Mark of sub1 ");
    scanf("%d", &stud[k].sub1);
    printf("\n Enter Mark of sub2 ");
    scanf("%d", &stud[k].sub2);
    printf("\n Enter Mark of sub3 ");
    scanf("%d", &stud[k].sub3);
}
```

```
clrscr();

for( k = 0; k < 4; k++)
{
    printf("\n Student %d ",k+1 );
    printf("\t\t Mark of sub1 is %d ",stud[k].sub1);
    printf("\n\t\t\t Mark of sub2 is %d ",stud[k].sub2);
    printf("\n\t\t\t Mark of sub3 is %d ",stud[k].sub3);
}

getch();
```

जिस प्रकार से पहले के प्रोग्रामों में Array का प्रयोग करके मान Input कराया गया है, उसी प्रकार इस प्रोग्राम में भी Array द्वारा मान Input कराया है। सर्वप्रथम एक marks नाम का Structure बनाया है। फिर struct marks प्रकार के Data Type का एक Array Variable stud[10] Declare किया है। ये Array 10 Students के तीन विषयों के Marks Accept करता है। Loop द्वारा एक के बाद एक Student के marks Input करवाए गए हैं। Loop चलाने के लिए एक Variable k int प्रकार का Declare किया है।

जब प्रथम बार Loop Iterate होता है तब "**Enter marks of Student1**" Message आता है। चूंकि प्रथम बार में Loop के Variable k का मान 0 होता है इसलिए k + 1 द्वारा यहां 1 print करवाया गया है।

दूसरे Iteration में k का मान 1 होगा तब फिर से "**Enter marks of Student2**" Message आता है, क्योंकि वापस k + 1 Expression के कारण यहां 2 Print होता है।

हम प्रथम बार जब मान Input करते हैं तब प्रथम विषय में मान जाए, इसे बताने के लिए stud[k].sub1 Expression दिया है, जो Compiler को बताता है कि stud एक Structure प्रकार के Data Type का Variable है और जो मान Input हो रहा है, वह मान sub1 में जाकर Store होगा।

इसी तरह से जब हम दूसरे विषय का मान Input करते हैं, तब stud[k].sub2 Expression के कारण Compiler समझ जाता है कि अब जो मान Input हुआ है, उसे इस Structure के दूसरे विषय में Store करना है। तीसरे Expression stud[k].sub3 से Input होने वाला मान Structure के तीसरे Member मे जा कर Store होता है।

जब प्रथम Student के तीनों विषय के मान Input हो जाते हैं, तब Program Control Loop से बाहर आकर वापस Loop को Check करता है। Loop की Condition अभी सत्य रहती है इसलिए Program Control वापस Loop में चला जाता है। अब k का मान Increment होकर 1 हो चुका है।

इसलिए अब Input होने वाले मान दूसरे Student के Memory Location पर जा कर Store होते हैं, क्योंकि k का Index Number बदल जाने से Variable भी बदल जाता है।

हमने Array Chapter के अंतर्गत बताया था कि किस प्रकार से जब हम एक Array Variable Declare करते हैं, तो वह Array वास्तव में उसी समय अलग–अलग Index Number के साथ एक ही Variable के उतने Variable Declare करता है, जितनी हम उसकी Size Define करते हैं। यही बात यहां भी सत्य है, इसी कारण से stud[0] व stud[1] Structure marks प्रकार के भिन्न–भिन्न Variables हैं। इसीलिए दूसरे Iteration में k का मान 1 हो जाने से जो मान Input होंगे वे एक अलग Memory Location पर जा कर Store होंगे। इस प्रकार से हम Array का प्रयोग Structure के साथ कर सकते हैं।

Input किये गए मानों को वापस प्राप्त करने के लिए वापस for Loop का प्रयोग किया गया है, और जिन अलग–अलग Memory Locations पर Input किये गए मान Store हुए हैं, वहीं से वापस उन मानों को प्राप्त करके Output में Print करवा दिया गया है। इस प्रोग्राम मे जो मान Input हुए हैं, वे मान एक तरह से Two Dimensional Array के रूप में Store होते हैं।

Array within Structure

जिस तरह हमने अभी Structure Variable को Array की तरह प्रयोग किया, उसी प्रकार हम Structure के अंदर भी Array का प्रयोग कर सकते हैं। जैसे ऊपर के प्रोग्राम में हमने तीन विषयों sub1, sub2 व sub3 को Member के रूप में प्रयोग किया। हम इन तीन विषयों को भी एक Array के रूप में Declare कर सकते हैं। प्रोग्राम में इन्हें Access करने के लिए भी एक Loop चलाना होगा और ये प्रोग्राम भी एक Two Dimensional Array की तरह ही होगा।

उदाहरण के लिए हम ऊपर के ही प्रोग्राम को इस प्रकार का बना रहे हैं, जिसमें Structure के अंदर ही One Dimensional Array का प्रयोग किया गया है। ये प्रोग्राम एक Two Dimensional Array की तरह काम करता है। जब इसमें मान Input किया जाता है, तो सबसे पहले stud[0].sub[0] में मान जाता है। फिर stud[0].sun[1] में और इसी क्रम में सभी Array Elements Store होते है।

Program

```
#include<stdio.h>
main()
{
    struct marks
    {
        int sub[3];
    };
    struct marks stud[10];
    int k,    l;
```

```
        clrscr();
            for( k = 0; k < 10; k++)
            {
                printf("\n Enter marks of Student %d ", k+1);

                for(l = 0; l < 3; l++)
                {
                    printf("\n Enter Mark of sub %d ",sl+1);
                    scanf("%d", &stud[k].sub[l]);
                }
            }
            clrscr();
             for( k = 0; k < 10; k++)
            {
                for(l = 0; l < 3; l++)
                {
                    printf("\n Student %d ",k+1 );
                    printf("\t\t Mark of sub %d is %d ",sl+1,stud[k].sub[l]);
                }
            }
        getch();
    }
```

इस Program में विभिन्न Locations पर सभी विषयों के मान सभी Students के अनुसार Store होते हैं।

```
Student[0].sub[1]
Student[0].sub[2]
Student[0].sub[3]
Student[1].sub[1]
Student[1].sub[2]
Student[1].sub[3]
Student[2].sub[1]
Student[2].sub[2]
Student[2].sub[3]
Student[3].sub[1]
Student[3].sub[2]
. . .
. . .
```

```
Student[10].sub[10]
```

इसे ध्यान से देखने पर हम जान सकते हैं कि इस प्रकार से मान प्राप्त करने के लिए या इस प्रकार से विभिन्न Memory Locations पर मान Store करने के लिए हमें दो Loop चलाने होंगे और दूसरा Loop Inner होगा यानी Nested Looping का प्रयोग करना होगा। ताकि जब Outer Loop का Index Number 0 हो तब Inner Loop के Index Number क्रम से तीन Iterations में 0, 1 व 2 हो जाएं और प्रथम Student के तीन Subjects मे Marks Store हो जाएं। फिर जब Outer Loop में Index Number 1 हो जाए तब दूसरे Student के तीनों विषयों के मान Memory में विभिन्न Locations पर Store किये जाएं। यह प्रोग्राम इसी तरीके से बनाया गया है।

Structure Within Structure (Nested Structure)

हम एक Structure में दूसरे Structure को भी प्रयोग कर सकते हैं। जब हम किसी Structure के Block में एक और Structure का प्रयोग करते हैं, तो इस प्रकार की प्रक्रिया को **Nesting of Structure** या **Structure within Structure** कहते हैं। इसे समझने के लिए यहां एक Structure बनाया जा रहा है, जिसमें किसी Employee की salary को Store किया जाएगा।

Structure

```
struct salary
{
    char name[20];
    char department[30];
    int basic;
    int dearness_allowance;
    int house_rent_allowance;
    int city_allowance;
}employee;
```

ये एक सामान्य Structure है। हम देख सकते हैं कि इसमें Allowance से सम्बंधित तीन Members हैं। यदि हम इन तीनों Members को इसी Structure के अंदर एक अन्य Structure बना कर उसके Member बना दें तो Outer Structure के लिए Inner Structure **allowance** एक Member होगा और ये allowance स्वयं एक Structure है जो कि Outer Structure salary का एक Member है। यानी हम ये कह सकते हैं कि salary नाम का एक Structure है, जिसमें allowance नाम का एक Member है और allowance खुद एक Structure है, जिसके अपने तीन Member हैं।

Nested Structure

```
struct salary
    {
```

```
        char name[20];char
        department[30];
        int basic;
        struct
        {
            int dearness_allowance;
            int house_rent_allowance;
            int city_allowance;
        }allowance;
    }employee;
```

यहां salary एक Structure है, जिसके चार Members क्रमशः name, department, basic व allowance हैं और इसी Structure के अंदर allowance नाम का एक और Structure है, जिसके तीन Member क्रमशः dearness_allowance, house_rent_allowance व city_allowance हैं। यानी इस Structure में allowance दो काम कर रहा है। Outer Structure के लिए वह एक Member है, जबकि Structure के अंदर वह स्वयं एक Inner Structure है, जिसके अपने तीन Members हैं। इस प्रकार ये Structure salary एक Nested Structure का उदाहरण है।

जिस प्रकार हम एक Structure के बाहर Structure प्रकार के Variable Declare करते हैं, उसी प्रकार से allowance के भी Variable Declare कर सकते हैं। यहां Nested Structure का एक उदाहरण दिया जा रहा है। इस उदाहरण में दो विधार्थीयों के नाम व उनके तीन विषयों में प्राप्त अंको को Store किया गया हैं व Output में वापस Print किया गया है।

इस उदाहरण में stud नाम का एक Structure बनाया गया है। इसमें एक और Inner Structure बनाया गया है, जिसमें तीन विषयों के मान Store करने के लिए तीन Member हैं। ये Inner Structure, Outer Structure का Member है।

Inner Structure का एक Variable marks है, जिससे इस Inner Structure के Members को access किया जा सकता है। Outer Structure के दो Variables दो Students को Represent कर रहे हैं, जिनका नाम क्रमशः stud1 व stud2 है। ये दोनों Variables Outer Structure stud प्रकार के हैं।

Program

```
#include<stdio.h>
main()
{
    struct stud
    {
```

```
        char name[15];
        int ro_no;
        struct
        {
            int sub1;
            int sub2;
            int sub3;
        }marks;
    };

    struct stud stud1,stud2;
    clrscr();

    printf("\n Enter name of stud1 ");
    scanf("%s", stud1.name);

    printf("\n Enter roll number of stud1 ");
    scanf("%d",&stud1.ro_no);

    printf("\n Enter Marks of sub1 ");
    scanf("%d",&stud1.marks.sub1);

    printf("\n Enter Marks of sub2 ");
    scanf("%d",&stud1.marks.sub2);

    printf("\n Enter Marks of sub3 ");
    scanf("%d",&stud1.marks.sub3);
    clrscr();

    printf("\n Name  of Student1 is %s ", stud1.name);
    printf("\n Ro No of Student1 is %d ", stud1.ro_no);
    printf("\n Marks of Subject1 is %d ", stud1.marks.sub1);
    printf("\n Marks of Subject2 is %d ", stud1.marks.sub2);
    printf("\n Marks of Subject3 is %d ", stud1.marks.sub3);

    getch();
}
```

जब इस प्रोग्राम को Execute किया जाता है, तब सभी प्रकार के Variables व Structure के Declaration के बाद Enter name of stud1 Message आता है। जब हम नाम Input करते हैं, तो वह नाम Structure प्रकार के Variable, जो कि Structure stud प्रकार का एक Variable है, के माध्यम से होते हुए Structure stud के Member name द्वारा Reserve की गई Memory Location पर जा कर Store हो जाता है।

ध्यान दें कि जब हमें Structure में कोई मान भेजना होता है या किसी Structure से कोई मान प्राप्त करना होता है, तो हमें उस मान की Location का पूरा Reference देना होता है। इस प्रकार यहां stud1.name Statement "C" Compiler को Input किये गए नाम को Store करने की पूरी Memory Location बता रहा। यह Statement Program Control को बताता है कि stud1 एक Structure प्रकार का Variable है और stud1 जिस Structure का Variable है, उसमें name नाम का एक Member है। इस Member ने Memory में 15 Byte की Character Type की Space Reserve किया है।

यानी हम इस Member में 15 अक्षरों तक की कोई String Input कर सकते हैं। इस प्रकार Input किया जाने वाला नाम stud नाम के Structure के Structure Member name में जा कर Store हो जाता है।

अब दूसरा Message **"Enter roll number of stud1"** हमसे विधार्थी का Roll Number मांगता है। जब हम Roll Number Input करते हैं, तो वह भी stud नाम के Structure के Member roll_no की Reserve की गई Memory Location पर जा कर Store हो जाता है। जब Program Control हमसे तीसरा मान मांगता है तो वह निम्न Statement द्वारा ये मान Nested Structure में जा कर Store कर देता है।

```
&stud1.marks.sub1
```

यह Statement Program Control को बताता है कि stud1 एक Structure stud के प्रकार का Variable है। इस stud Structure में एक Member है, जिसका नाम marks है और ये Member स्वयं एक Structure है जिसके तीन Member हैं। sub1 उस Inner Structure marks का पहला Member है। Input किया जाने वाला मान उस Memory Location पर Store कर दो जिस Memory Location का नाम sub1 है। इस प्रकार ये मान sub1 के Memory Location पर Store हो जाता है।

इसी प्रकार अन्य विषयों के मान भी Memory में Input हो जाते है। इस प्रोग्राम को इतना समझाने का मकसद ये बताना है, कि हम "." Operator की मदद से ही एक से दूसरे Structure के Members को किस प्रकार Access कर सकते हैं और "." Operator को किस प्रकार प्रयोग किया जाता है।

एक खास बात ध्यान रखें कि जब हमें Inner Structure का कोई Variable Declare करना होता हैं, तो उसे Structure के मंझले कोष्ठक के सेमी colon से पहले ही Declare करना होता है। जैसे नीचे बताए गए प्रारूप में बताया गया हैं। इसमें Inner Structure के तीन Variable allowance, xyz व pqr हैं। ये तीनों ही Variable, Inner Structure प्रकार के हैं।

ध्यान से देखने पर पता चलेगा कि Inner Structure के Data Type के Variable Declare करने के लिए कोई tag नहीं है। यानी Inner Structure का Format ऐसा होता है, जिसमें कोई tag नहीं होता है। इसीलिए Inner Structure प्रकार के Variable हमें Structure बनाते समय ही Declare कर देने होते हैं। ये तीनों ही Variable Outer Structure के लिए Member मात्र हैं जबकि Inner Structure के ये Variable हैं।

Structure Nesting

```
struct salary
    {
        char name[20];char
        department[30];
        int basic;

        struct
        {
            int dearness_allowance;
            int house_rent_allowance;
            int city_allowance;
        } allowance, xyz, pqr ;
    }employee;
```

Structure की Nesting करने का एक अन्य तरीका भी है। इस तरीके में Nested हो रहे Structure का अपना tag भी होता है। वास्तव में हम इस तरीके में दोनों Structures को अलग–अलग ही लिखते हैं।

जिस Structure को दूसरे वाले Structure में Nested करना होता है, हम उस Structure के अंदर पहले वाले Structure के प्रकार के Variable Declare कर देते हैं, जो कि दूसरे वाले Structure के Member होते हैं। इस प्रकार जब हमें पहले वाले Structure प्रकार के Variables को Access करना होता है, तब दूसरे वाले Structure के माध्यम से हम उस Structure के Members को Access करते हैं।

यानी एक Structure प्रकार का Variable यदि दूसरे Structure में Declare किया जाएं, तो ये भी एक प्रकार की Nesting ही हो जाती है। उदाहरण के लिए ऊपर बताए गए प्रारूप पर ही हम इस प्रकार की Nesting प्रयोग कर सकते हैं।

Structure Nesting

```
struct pay
{
    int dearness_allowance;
    int house_rent_allowance;
    int city_allowance;
};

struct salary
{
    char name[20];char
    department[30];
    int basic;
    struct pay allowance;
}employee;
```

यहां salary नाम के Structure में pay नाम के Structure प्रकार का एक Variable allowance Declare किया गया है। इस Variable को दूसरे Structure के अंदर एक Member के रूप में Declare किया गया है। इसलिए हमें जब भी इस pay नाम के Structure के Members को access करना होगा, हमें salary नाम के Structure के allowance नाम के Member को Access करना होगा।

यानी यदि हमें Structure pay के Member को कोई मान भेजना हो या उसके किसी Member से कोई मान प्राप्त करना हो तो हमें ये काम salary नाम के Structure के Structure Member, allowance को Use करके करना होगा जबकि allowance Structure salary का एक Member है। इसलिए हम कह सकते हैं कि Structure pay Structure salary में Nested है।

यहां हम ऊपर दिये गए प्रोग्राम को ही दूसरे प्रकार से दे रहे हैं। इस प्रोग्राम में Inner Structure को हटा कर बाहर ही लिख दिया है और दूसरे Structure, जिसमें कि पहले Structure को Nested किया गया था, वही काम केवल पहले वाले Structure प्रकार के Variable को दूसरे वाले Structure में Declare करके किया गया है। जैसा कि अभी हमने उपर वाले paragraph में बताया है। इसमें बहुत कम परिवर्तन किया गया है।

Program

```
#include<stdio.h>

main()
```

```
{
    struct subjects
    {
        int sub1;
        int sub2;
        int sub3;
    };

    struct stud
    {
        char name[15];
        int ro_no;
        struct subjects marks;
    };

    struct stud stud1,stud2;
    clrscr();

    printf("\n Enter name of stud1 ");
    scanf("%s", stud1.name);
    printf("\n Enter roll number of stud1 ");
    scanf("%d",&stud1.ro_no);
    printf("\n Enter Marks of sub1 ");
    scanf("%d",&stud1.marks.sub1);
    printf("\n Enter Marks of sub2 ");
    scanf("%d",&stud1.marks.sub2);
    printf("\n Enter Marks of sub3 ");
    scanf("%d",&stud1.marks.sub3);
    printf("\n Name  of Student1 is %s ", stud1.name);
    printf("\n Ro No of Student1 is %d ", stud1.ro_no);
    printf("\n Marks of Subject1 is %d ", stud1.marks.sub1);
    printf("\n Marks of Subject2 is %d ", stud1.marks.sub2);
    printf("\n Marks of Subject3 is %d ", stud1.marks.sub3);
    getch();
}
```

इस प्रोग्राम का Output बिल्कुल वैसा ही प्राप्त होता है जैसा कि पहले वाले प्रोग्राम का प्राप्त होता है।

Structure with Function

अन्य Variables की तरह एक Structure Variable को भी Arguments के रूप में किसी Function को भेजा जा सकता है। किसी Function में जब Structure प्रकार के Variable को भेजना होता है तब हमें ये ध्यान रखना होता है कि Function कोई मान Return करेगा या नहीं और यदि Function कोई मान Return करेगा तो वह मान किस Data Type का होगा।

यदि हम Structure प्रकार के Data Type का मान किसी Function द्वारा Return कराना चाहते हैं, तो उस Function को भी हमें Structure प्रकार का Declare करना जरूरी होता है। लेकिन जब हमें किसी प्रकार का कोई मान Calling Function को Return नहीं करवाना होता है या फिर मात्र Standard प्रकार के Data Type जैसे int, float, Double या char प्रकार का ही Data Calling Function को Return करवाना होता है, तब हमें Function को Structure प्रकार का Declare करना जरूरी नहीं होता है।

एक सबसे खास बात ये है कि जब हमें किसी प्रोग्राम में, Structure प्रकार के Variable को, किसी Function में pass करना होता है, तो Structure को main() Program से बाहर ही Define किया जाना जरूरी होता है। क्योंकि यदि Structure main() Function के अंदर Define किया जाता है तो उस Structure को बाहर का Function उपयोग में नहीं ले सकता है, क्योंकि कोई भी User Defined Function हमेंशा main() Program से बाहर ही लिखा जाता है।

जैसा कि Storage Class के अंतर्गत बताया गया था कि केवल Global Variable को ही कोई भी Function उपयोग में ले सकता है। main() Function या किसी अन्य User Defined Function में Declare किये गए Variables को सामान्य तरीके से कोई भी अन्य User Defined Function उपयोग में नहीं ले सकता। उसी प्रकार किसी भी Global Structure को ही कोई भी अन्य User Defined Function अपने उपयोग में ले सकता है।

हम अपनी आवश्यकता के अनुसार प्रोग्राम की जो भी coding main Program से बाहर लिखते हैं, वे सब codes चाहे वह कोई Variable के Declaration के लिए हो, किसी Function को लिखा गया हो या फिर चाहे वह Structure का Definition हो, वे सारी Coding Global हो जाती हैं।

इस प्रकार से Function में Structure प्रकार के Variable को Argument के रूप में तभी भेज कर उपयोग में लाया जा सकता है, जब Structure Global प्रकार से define किया गया हो यानी Structure को main() Program से बाहर लिखा गया हो।

इसे निम्न प्रोग्राम द्वारा समझने की कोशिश करते हैं। इस प्रोग्राम में एक Global प्रकार का Structure define किया गया है और इस inv नाम के इस Structure का एक item नाम का Variable main() Program में Declare किया गया है। इस Structure प्रकार के Variable item में हम किसी सामान का Item Code, सामान की संख्या व सामान की कीमत Store कर सकते हैं।

Program

```
#include<stdio.h>

struct inv
{
    int item, qty, price;
};

main()
{
    int x;
    struct inv item;
    clrscr();

    printf("\n Enter Item code ");
    scanf("%d", &item.item );

    printf("\n Enter Quantity ");
    scanf("%d", &item.qty );

    printf("\n Enter Price ");
    scanf("%d", &item.price );

    x= value(item);

    printf("\n Price is %d ",x);

    getch();
}

value(cal)
struct inv cal;
{
    int val;
    val = cal.qty * cal.price;
    return(val);
}
```

इस प्रोग्राम को Run करने पर Program Control User से सामान का कोड, सामान की मात्रा व सामान की कीमत Input के रूप में लेता है और item नाम के Variable के Structure की Memory Location पर Store कर देता है।

फिर Program Control को value नाम का एक User Defined Function मिलता है। Program Control Argument के रूप में इस Function में Structure प्रकार के Data Type के Variable item को Pass कर देता है। अब Program Control value नाम के इस User Defined Function में आता है। यहां कोष्ठक में formal Argument के रूप में प्राप्त Structure inv प्रकार के Variable item का मान cal नाम के Variable को प्राप्त हो जाता है।

अगले Statement में cal को Structure inv प्रकार का Variable Declare कर दिया गया है। ये सभी काम उसी प्रकार किये जा रहे हैं जैसा कि अन्य Function में किया जाता है। यानी यदि हम int प्रकार का मान User Defined Function को Argument के रूप में Pass करते हैं, तो formal Argument के रूप में वह मान जब User Defined Function के कोष्ठक में लिखे Variable को प्राप्त हो जाता है, तब उस Variable को int प्रकार का Declare किया जाता है।

उसी प्रकार यहां Structure inv प्रकार का मान formal Argument के रूप में User Defined Function value के कोष्ठक में लिखे Variable cal को प्राप्त हो रहा है और फिर इस cal नाम के Variable को Structure प्रकार का Declare किया गया है।

जिस प्रकार से हम item नाम के Structure inv प्रकार के Variable को main() Program मे access करते हैं, उसी प्रकार से अब हम item के स्थान पर cal नाम के Variable को प्रयोग में ले सकते हैं क्योंकि cal नाम का ये Variable item नाम के Variable की ही प्रतिलिपी है। यानी इस प्रतिलिपी पर हम कोई भी काम उसी प्रकार से कर सकते हैं, जिस प्रकार से किसी int प्रकार के formal Variable के साथ हम सामान्य Function में प्रक्रिया करते हैं।

इस User Defined Function value में val नाम का एक Local Variable Declare किया गया है। अब कुल खर्च हुए रूपयों की गणना करने के लिए val = cal.qty * cal.price; Statement लिखा गया है। ये Statement val नाम के Variable में सामान की मात्रा का सामान की कीमत से गुणा करके प्राप्त मान को val नाम के Variable में Store कर देता है।

यदि हमें Function का प्रयोग ना करना होता और सभी सामान के क्रय में खर्च होने वाली रकम को ज्ञात करना होता, तो main() Program में ही इस Statement का प्रारूप कुछ इस प्रकार होताः

```
val = item.qty * item.price;
```

चूंकि इस Function में item के साथ की जाने वाली सारी प्रक्रियाएं अब हमें cal नाम के Variable के साथ करनी है क्योंकि Actual Argument के रूप में item Variable का सारा मान Formal Argument के रूप में User Defined Function के कोष्ठक में लिखे Variable cal को प्राप्त हो गया

है। इसलिए ये Statement *val = cal.qty * cal.price;* उपयोग में लिया गया है। यहां val एक int प्रकार का Variable है और val को main() Function में Return किया गया है।

जब Program Control main() Function में Return होता है, तब val का मान main() Function में Declare Variable x को प्राप्त हो जाता है और main() Function में x का मान print करवा दिया जाता है। जब किसी प्रोग्राम में किसी Function से Structure प्रकार का मान Calling Function को Return करना होता है तब हमें प्रोग्राम में उस Function को Structure प्रकार का Declare करना पडता है। इसे समझने के लिए नीचे लिखे प्रोग्राम को Discus करते हैं।

Program

```
#include<stdio.h>
struct stud
{
    char name[15];
    int ro_no, sub1, sub2, sub3, tot;
};

main()
{
    struct stud stud1,sum();
    clrscr();

    printf("\n Enter name of stud1 ");
    scanf("%s", stud1.name);

    printf("\n Enter roll number of stud1 ");
    scanf("%d",&stud1.ro_no);

    printf("\n Enter Marks of sub1 ");
    scanf("%d",&stud1.sub1);

    printf("\n Enter Marks of sub2 ");
    scanf("%d",&stud1.sub2);

    printf("\n Enter Marks of sub3 ");
    scanf("%d",&stud1.sub3);

    stud1=sum(stud1);
```

```
    clrscr();

    printf("\n Name  of Student1 is %s ", stud1.name);
    printf("\n Ro No of Student1 is %d ", stud1.ro_no);
    printf("\n Marks of Subject1 is %d ", stud1.sub1);
    printf("\n Marks of Subject2 is %d ", stud1.sub2);
    printf("\n Marks of Subject3 is %d ", stud1.sub3);
    printf("\n Total Marks is %d",stud1.tot);

    getch();
}

struct stud sum(stvar) // Function Declaration
struct stud stvar ;
{
    stvar.tot=stvar.sub1+stvar.sub2+stvar.sub3;
    return( stvar );
}
```

इस प्रोग्राम में सर्वप्रथम एक stud नाम का एक Structure बनाया गया है। फिर main() Program में इस stud नाम के दो Variables Declare किये गए हैं। पहला Variable stud1 है जो कि Structure प्रकार का एक सामान्य Variable है, जबकि दूसरा Variable sum() एक Function है। इस Function को यहां Declare करने का मतलब ये है कि ये Function Structure stud प्रकार के Data Type का मान main() Program से Receive करेगा और main() Program को Structure stud प्रकार के Data Type का मान Return करेगा।

इस प्रोग्राम को Execute करके Student का नाम रोल नम्बर व तीन विषयों के अंक Store करने के बाद Program Control को **stud1=sum(stud1); Statement** प्राप्त होता है। इस Statement से Program Control Structure stud प्रकार के Variable stu1 का मान **sum**() Function को Actual Argument के रूप में भेज देता है। sum() Function में Formal Argument को Accept करने के लिए एक formal Variable stvar है।

stud1 का मान stvar को प्राप्त होने के बाद इस Variable को Structure stud प्रकार का Declare किया गया है। अब stvar stud1 Variable के प्रतिनिधि के रूप में काम करेगा यानी जो भी Calculations stud1 पर main() Program में की जा सकती है, वे सभी Calculations इस Function में stvar पर की जा सकती है और stvar के अंदर जो मान है वह stud1 Variable का है।

अब तीनों विषयों में प्राप्त अंकों का मान **stvar.tot=stvar.sub1+stvar.sub2+stvar.sub3;** Statement द्वारा जोड कर stvar नाम के इस Structure प्रकार के Variable में Store कर दिया गया है और stvar के मान को, जो कि Structure प्रकार का मान है, main() Function को Return कर दिया गया है।

इस Function से हम Structure प्रकार का मान Calling Function को Return कर रहें हैं इसलिए Return Data Type के स्थान पर हमने struct stud लिखा हैं। ये Statement Program Control को बताता है कि इस Function से return होने वाला मान stud नाम के Structure प्रकार का है।

जब return Statement द्वारा stvar मे Store मान को main() Function में भेजा जाता है, तब यह मान वापस stud1 नाम के Variable को प्राप्त हो जाता है। यानी stud1 Variable के Structure में tot Member में तीनों विषयों में प्राप्त अंकों का योग Store हो जाता है। Structure के Member tot को मान हमने sum Function द्वारा Calculate करके प्रदान किया है।

इस प्रकार हम इस प्रोग्राम से समझ सकते हैं कि जब एक Structure प्रकार के Variable को किसी Function में भेजा जाता है, तो वास्तव में हम पूरा Structure ही Function को Pass कर रहे होते हैं और जो भी प्रक्रिया हम Declare किये गए नए formal Variable द्वारा करते हैं, वे सारे काम सीधे ही Structure के अंदर होते हैं।

जैसा कि इस प्रोग्राम में हमने tot Member को मान Function में Declare किये गए formal Variable stvar द्वारा प्रदान किया है। stvar में stud1 के सभी Members को प्रदान किया गया मान Function को प्राप्त हुआ है।

इस बात को थोडा ध्यान से समझें कि एक Structure प्रकार का Variable सामान्य Variable से अलग होता है, क्योंकि एक सामान्य Variable में केवल एक ही मान होता है, लेकिन एक Structure प्रकार के Variable में उतने मान होते हैं, जितने Define किये गए Structure के Member होते हैं। इस प्रकार से stud1 से छः मान User Defined Function sum को प्राप्त होते हैं।

पहला name का, दूसरा roll number का तीसरा sub1 का चौथा sub2 का व पांचवा sub3 का व छठा tot का। चूंकि tot में कोई मान हमने Store नहीं किया है इसलिए ये खाली ही रहता है। इस Program में हम देख सकते हैं कि stvar को भी ये ही छः मान formal Argument के रूप में प्राप्त हुए हैं।

ये छः मान वे ही हैं जो कि Structure में हमने Member बनाए हैं। यानी एक Function को जब Structure प्रकार का Variable Argument के रूप में Pass किया जाता है तो वास्तव में हम पूरा Structure ही उस Function को Argument के रूप मे pass कर रहे होते हैं। और उस Structure को use करने के लिए एक Variable की जरूरत होती है। ये Variable Function का formal Variable होता है जो यहां stvar है।

Union

Union व Structure को Define करने का तरीका एक जैसा ही है। जिस तरह Structure को define करने के लिए **struct** key word का प्रयोग करते हैं, उसी प्रकार Union को Define करने के लिए **union** key word का प्रयोग किया जाता है। जो भी काम एक Structure के साथ किया जाता है, वे सभी काम union के साथ किये जा सकते हैं। जैसे **union** प्रकार का Variable Declare करना, उस Variable को मान प्रदान करना, Union के Members को access करना आदि। लेकिन इन दोनों के काम करने के तरीके में अन्तर है।

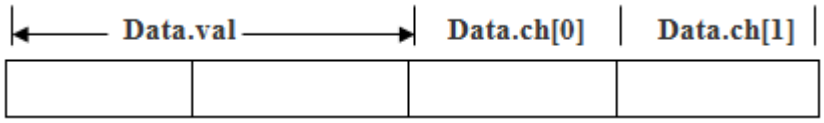
एक Structure के सभी Members की Memory में एक अलग Memory Location होती है। यानी एक Structure के सभी Members Memory में space reserve करते हैं। लेकिन एक Union में Declare किये गए सभी Variable Memory के एक ही Memory Location पर Store होते हैं। एक Union में हम विभिन्न प्रकार के Members को रख सकते हैं, लेकिन एक समय में केवल एक ही प्रकार का मान Union के अंदर किसी Member में Store रह सकता है।

इस कारण से हम एक समय में केवल एक ही Union Member को access कर सकते हैं। हम जब Union define करते हैं, तब Union में सबसे अधिक जगह Store करने वाले Variable के बराबर Memory Reserve हो जाती है। यानी यदि एक Union में एक Member int प्रकार का व दूसरा Member double प्रकार का है, तो ये Union Memory में चार Byte की space Reserve कर लेता है। क्योंकि double प्रकार का Variable Memory में चार Byte लेता है।

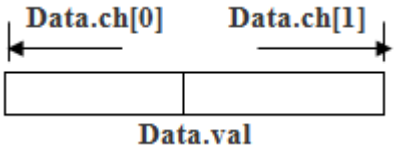
जैसा कि अभी बताया कि एक Union के सभी Members समान Memory Location का प्रयोग करते हैं, इसलिए यदि हम एक ऐसा Union बनाएं, जिसमें एक Member long Double प्रकार का हो, तो हम इस Union में int प्रकार के चार मान, Double प्रकार के दो मान व char प्रकार के आठ मान Store कर सकते हैं। आइये समझते हैं कि किस प्रकार से Structure व Union में मान Store होते हैं। हम समान Members का एक Statement व एक Union बनाते हैं।

```
struct                          union
{                               {
    int val;                        int val;
    char ch[2];                     char ch[2];
}Data;                          }Data;
```

प्रथम प्रोग्राम Structure है। ये Memory में निम्न प्रकार से Variable Data के लिए space reserve करेगा।



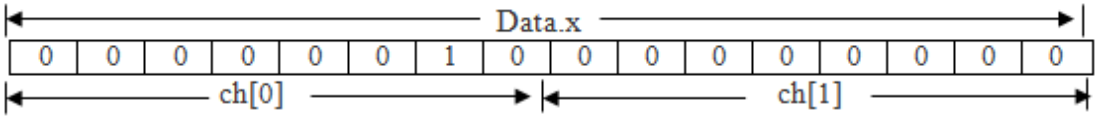
जबकि union Memory में निम्नानुसार Space Reserve करता है।



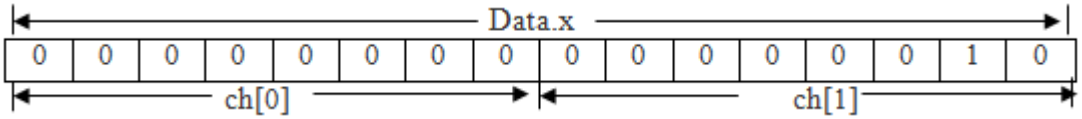
इस प्रकार Union Memory में सिर्फ दो Byte ले रहा है। ध्यान दें कि जो Memory Location Data.val Use कर रहा है, वही Memory Location Data.ch[0] Data.ch[1] भी Use कर रहे हैं। इस प्रकार इन दो Byte को एक साथ या दो अलग–अलग जगह से Use किया जा सकता है।

अब एक उदाहरण देखते हैं। माना Data.val का मान 512 है। हम जानते हैं कि कोई भी मान Memory में Binary Digits के रूप में ही Store होता है। 512 की Binary 1000000000 होती है।

ये बात हमेंशा ध्यान रखनी चाहिये कि Memory में किसी भी मान की Binary Digit के दो भाग होते हैं, जिन्हे क्रमशः Low Byte व High Byte कहते हैं। बडी संख्या को High Byte व छोटी संख्या को Low Byte कहा जाता है। int प्रकार का Variable Memory में दो Byte या 16 Bit में अंको को Store करता है। इस प्रकार आठ Bit हर Byte में Store होते हैं। अंक 512 के भी दो भाग एक–एक Byte में जा कर 00000010 व 00000000 की तरह Store होते हैं। यदि ये संख्या Memory में Store हो तो ये निम्नानुसार Store होनी चाहियेः



लेकिन यदि हम Data.ch[0] व Data.ch[1] को print करते हैं, तो मान क्रमशः 0 व 2 प्राप्त होता है, जो कि इस प्रकार Digit Store होने के कारण क्रमशः 2 व 0 होना चाहिये। ऐसा इसलिए होता है क्योंकि Memory में High Byte, Low Byte से हमेंशा पहले Store होती है। यानी प्रथम byte 00000010 दूसरी Byte 00000000 से बडी है इसलिए ये Memory में निम्नानुसार Store होती हैः



इस प्रकार ch[0] का मान 0 व ch[1] का मान 2 Output में print होता है। इस प्रकार हमने देखा कि अंक 512 किस प्रकार दो भागों में बंट कर 0 व 2 में बदल गया। Union का यही फायदा है कि हम एक प्रकार में मान को Store करके दूसरे प्रकार से भी access कर सकते हैं। ऐसा इसी कारण से होता है क्योंकि union के जितने भी Member होते हैं, वे मान को Store करने के लिए समान Memory Location का प्रयोग करते हैं।

जब हम किसी Union Member को कोई मान प्रदान कर देते हैं तो फिर यदि उसी Union के किसी अन्य Member को कोई मान Input कर दिया जाए, तो पहले वाला मान हट जाता है और दूसरा Inputted मान पहले वाले मान पर Over Write हो जाता है, क्योंकि दोनों मान Memory में Store होने के लिए समान Memory Location का ही प्रयोग करते हैं।

union का प्रयोग सामान्य प्रोग्राम में Use करने के लिए भी किया जाता है, लेकिन इसका प्रयोग अधिकतर "C" द्वारा Hardware के साथ काम करने के लिए किया जाता है। हम Union को Structure के साथ मिला कर भी प्रयोग कर सकते हैं। आवश्यकतानुसार इसकी Nesting कर सकते हैं और Structure के अंदर भी इसे Nested किया जा सकता है साथ ही Union के अंदर Structure को भी Nested किया जा सकता है। Union के साथ ये सारे काम आवश्यकतानुसार किये जा सकते हैं।

Pointers and Structure

हम एक साधारण Variable का Pointer बनाना जानते हैं। ठीक उसी प्रकार से Structure भी एक प्रकार का Data Type है जिसे User स्वयं अपनी जरूरत के अनुसार बनाता है। किसी Structure को Use करने के लिए हमें Structure प्रकार के Variables Declare करने पडते हैं।

जैसे माना int प्रकार का एक Variable Declare करते हैं तो उस int प्रकार के Variable का Address Store करने के लिए हमें int प्रकार का ही एक अन्य Pointer Variable Declare करना पडता है। इसी प्रकार से किसी Structure प्रकार के Variable का Pointer Declare करने के लिए या किसी Structure प्रकार के Variable का Address किसी Pointer में Store करने के लिए हमें Pointer भी Structure प्रकार का ही लेना होगा।

जब हमें किसी Pointer द्वारा किसी Structure के Members को Access करना होता है, तब हमें **->** Arrow Operator का प्रयोग करना पडता है। अब हम एक उदाहरण द्वारा Structure को Pointer द्वारा Access करते हैं।

Program

```
#include<stdio.h>
main()
```

```
{
    struct Address
    {
            char name[15];
            char city[10];
            int room_no;
    };

    struct Address student1={"Kuldeep", "Falna", "122"};
    struct Address *stud1;
    stud1 = &student1;
    clrscr();

    printf("\n Name of student1 is %s ", stud1->name);
    printf("\n City of student1 is %s", stud1->city);
    printf("\n Room Number of student1 is %d", stud1->room_no);
    getch();
}
```

Output

```
Name of student1 is Kuldeep
City of student1 is Falna
Room Number of student1 is 122
```

हमने इस प्रोग्राम में Address नाम का एक Structure बनाया है और इस Structure प्रकार का एक Variable student1 बनाया है, साथ ही इस Variable को मान भी Initialize कर दिया है। फिर एक Pointer Variable Declare किया है जो कि Structure Address प्रकार का है। इस Pointer को Structure के Variable student1 का Address प्रदान किया गया है।

एक Structure और एक Array में केवल इतना ही फर्क होता है कि Array में एक ही प्रकार के ढेर सारे Data Store किये जाते हैं जबकि एक Structure में भिन्न–भिन्न प्रकार के कई Data Store किये जा सकते हैं। एक Structure Memory में निम्नानुसार Store होता है:

student1.name	student1.city	student1.room_no
Kuldeep	Falna	122
4000	1025	4027

इस प्रकार से किसी Structure के Variable का Address उसी प्रकार से किसी Pointer Variable में Store होता है, जिस प्रकार से एक Array में। Array में सभी Element समान Space लेते हैं, लेकिन Structure में Data Type के अनुसार Space Reserve होता है।

हमने इस Structure में **name** को 15 byte की Space दी है, इसलिए यदि Structure **4000** Location पर Store होता है, तो **city** नाम का Member **name** नाम के Member से 15 Byte आगे **4015** पर Store होगा।

इसी प्रकार city को 10 Byte दिया है इसलिए **room_no** नाम का Member **city** Member से 10 byte आगे **4025** पर Store होगा।

इस प्रकार से यदि हम किसी Structure Variable का Address किसी Pointer Variable को देते हैं, तो उस Structure का Base Address ही उस Variable को प्राप्त होता है, जिस तरह किसी Array प्रकार के Variable का Address किसी Pointer Variable को देने पर उस Variable का Base Address ही उस Pointer को प्राप्त होता है।

जिस प्रकार से एक सामान्य Variable को Access किया जाता है, उसी प्रकार से एक Pointer Variable को भी access किया जाता है। लेकिन Pointer Variable को जब Use किया जाता है तब (**.**) Dot Operator के स्थान पर **->** Arrow Operator का प्रयोग करना पडता है।

इस उदाहरण में Pointer का मान Output में उसी प्रकार से Print किया गया है, जिस प्रकार से एक सामान्य Structure Variable को Print किया जाता है। लेकिन चूंकि हम यहां Structure Variable student1 के मानों को Pointer Variable stud1 की सहायता से Print कर रहे हैं, इसलिए यहां **->** Operator का प्रयोग किया गया है।

यदि हम Pointer की सहायता से ये मान Output में Print ना करके Variable student1 की सहायता से Print करते, तो हमें निम्नानुसार Statements में परिवर्तन करके Arrow Operator के स्थान पर Dot Operator का प्रयोग करना पडता।

```
printf("\n Name of student1 is %s ", stud1.name);
printf("\n City of student1 is %s", stud1.city);
printf("\n Room Number of student1 is %d", stud1.room_no);
```

इन Statements का Output भी वही प्राप्त होता जो प्राप्त हुआ है। यानी

```
Name of student1 is Kuldeep
City of student1 is Falna
Room Number of student1 is 122
```

इसी प्रकार से हम किसी Structure Variable के Pointer को किसी User Defined Function में भी Argument के रूप में भेज सकते हैं। जब हम किसी Structure Variable के Pointer को Argument के रूप में किसी Function में भेजते हैं तब भी उस Structure को Access करने का तरीका वही रहता है।

जब हम Pointer द्वारा किसी Function में किसी Structure का Base Address भेजते हैं तब उस Structure के Variables को Use करने के लिए Arrow Operator का प्रयोग करना होता है। हम उपर बताए गये प्रोग्राम में ही Function का प्रयोग करके Output को Function द्वारा Print करने का प्रोग्राम बना रहें हैं जिससे Pointer पद Function को समझा जा सके।

Program

```
#include<stdio.h>
main()
{
    struct Address
    {
            char name[15];
            char city[10];
            int room_no;
    };

    struct Address student1={"Kuldeep", "Falna", "122"};
    struct Address *stud1;
    stud1 = &student1;
    clrscr();

    printf("\n Name of student1 is %s ", stud1->name);
    printf("\n City of student1 is %s", stud1->city);
    printf("\n Room Number of student1 is %d", stud1->room_no);
    getch();
}
```

Output

```
Name of student1 is Kuldeep
City of student1 is Falna
Room Number of student1 is 122
```

इस प्रकार से हम किसी struct प्रकार के Variable को किसी Function में भी भेज सकते हैं। इस Program में Structure Address के Variable student1 का Address Function में भेजा है।

इस Function में Argument को प्राप्त करने के लिए एक Pointer Variable Declare किया है जो कि Structure Address प्रकार का है। stud1ptr Pointer में Variable student1 का Address आ जाता है। अब हम इसे Use करके Structure के सारे Members को Print कर सकते हैं। जिससे कि बताए अनुसार Output प्राप्त हो जाता है। इसी प्रकार से हम उन Structures के साथ भी प्रक्रिया कर सकते हैं जिनमें Array का प्रयोग होता है।

एक बात हमेंशा ध्यान रखें कि Dot Operator के Left Side में हमेंशा Structure का Variable होता है, जबकि Arrow Operator के Left Side में हमेंशा एक Structure Pointer होता है जो कि Structure को Point करता है। Union व Structure को use करने का तरीका बिल्कुल समान होने से Union को Use करना नहीं बताया जा रहा है।

यदि हम **struct** Key Word की जगह **union** Key Word का प्रयोग करें, तो उपर बताया गया उदाहरण ही एक Union with Pointer का उदाहरण हो जाएगा। Union व Structure के Memory में Store होने के तरीके में अन्तर होने से इनके Output में अन्तर होगा लेकिन **Union** व **Structure** को Use करने का तरीके में जरा सा भी अन्तर नहीं हैं।

यदि हमें Structure के लिए Memory Allocate करनी हो तो भी यही बात लागू होती है। जैसेः

```
struct stud *new1;
new1 = ( struct stud * ) malloc ( sizeof (struct stud ) );
```

इस Statement से Structure stud के लिए एक Block Of Memory Allocate हो जाएगा और उसके प्रथम Byte का Address new1 नाम के Structure stud प्रकार के Pointer Variable new1 को प्राप्त हो जाएगा।

Exercise:

1. Structure किसे कहते हैं? Structure का Syntax बनाते हुए Programming में Structure की उपयोगिता को समझाईए।
2. एक Employee का Structure बनाते हुए उस Structure को Initialize करने व उस Structure के Data Members को Access करने के तरीके को एक Program द्वारा समझाईए।
3. एक Student का Structure बनाईए और दस Students की Information को एक Array द्वारा Access करने का Program बनाईए।
4. Array within Structure व Structures का Array दोनों के बीच के अन्तर को उचित उदाहरण द्वारा समझाईए।
5. Structures की Nesting से आप क्या समझते हैं ? एक उदाहरण द्वारा Structures की Nesting को समझाईए।
6. एक Student का Structure बनाईए और इस Structure को एक Display नाम के Function में Argument के रूप में Pass करके Student की Information को Function द्वारा Screen पर Display कीजिए।
7. Union व Structure के बीच के अन्तर को समझाईए।
8. एक Book का Structure बनाईए और इसकी सारी Information एक **Input** Function द्वारा Input कीजिए तथा **Output** Function द्वारा Display कीजिए।

Typedef

इसका प्रयोग किसी Define किये गए Variable को पुनः Define करने के लिए किया जाता है। इस Key Word का प्रयोग करके हम Variables के बडे–बडे नामों को छोटे नामों में Convert कर लेते हैं और उस छोटे नाम का प्रयोग प्रोग्राम में करते हैं जिससे Program को पढना व समझना सरल रहता है। जैसे कि **unsigned long int** प्रकार का कोई भी Variable Declare करने के लिए हमें unsigned long int key word का प्रयोग करना होगा। जैसे

```
unsigned long int num;
```

यदि हम इसे Typedef के प्रयोग द्वारा लिखें तो निम्नानुसार लिख सकते हैं:

```
typedef unsigned long int ULI;
```

अब हम ULI शब्द का प्रयोग करके **unsigned long int** प्रकार के Variables Declare कर सकते हैं। जैसे:

```
ULI num, val;
```

यहां num व val दोनों ही Variables **unsigned long int** प्रकार के Declare हो जाएंगे। Typedef एक ऐसा तरीका है, जिससे Short तरीके से किसी भी Data Type को Declare कर सकते हैं। इसका प्रयोग किसी Structure या Union प्रकार के Variables के Declaration में करके हम लम्बे व जटिल Declaration से बच सकते हैं। जैसे:

```
struct employee
{
    char name[30];
    int age;
    float basic;
};
struct employee e1, e2;
```

Typedef का प्रयोग करके हम इसके Variables को निम्नानुसार भी Declare कर सकते हैं–

```
struct employee
{
    char name[30];
    int age;
    float basic;
};
```

```
Typedef struct employee EMP;
EMP e1,e2;
```

इस प्रकार से अब हमें इस Structure के Variables Declare करने के लिए Typedef struct employee Statement नहीं लिखना होगा बल्कि EMP द्वारा भी हम इस Structure प्रकार के Variables Declare कर सकते हैं।

Enumerated Data Type

Enumerated Data Type का प्रयोग करके हम हमारी इच्छा व आवश्यकतानुसार नए Data Types बना सकते हैं और ये define कर सकते हैं कि बनाया गया Data Type किस प्रकार के मान Accept करेगा। इस प्रकार के Data Type बनाने के लिए हमें enum Key word का प्रयोग करना होता है और enum Data Type का definition बिल्कुल उसी प्रकार से किया जाता है जिस प्रकार से एक Structure को define किया जाता है। जैसे कि

```
enum stu_status
{
    pass, fail, supplementary;
};
enum stu_status Student1, Student2;
```

किसी Structure की ही तरह से enum प्रकार के Variables Student1, Student2 Declare किये गए हैं और Structure की ही तरह से ये Declare किया गया है कि ये Data Type किस प्रकार के मान accept करेगा। ये मान **Enumerators** कहलाते हैं। उसी प्रकार से जिस प्रकार से किसी Structure के मान Members कहलाते हैं। अब हम इन Variables को मान प्रदान कर सकते हैं। जैसे

```
Student1 = pass;
Student2 = fail;
```

ये बात हमेंशा ध्यान रखें कि किसी भी Variable को प्रदान किया जाने वाला मान enumerator ही हो सकता है। जैसे

```
Student2 = first_division;
```

ये एक गलत Assignment है क्योंकि first_division enumerator नहीं है। “C” Compiler Internally enumerators को Integers प्रकार से Treat करता हैं। Lost के अन्दर का हर मान

एक Integers द्वारा पहचाना जाता है जो कि 0 के क्रम से शुरू होता है। यानी Enumerator *pass* अंक 0 से सम्बंधित है। enumerator *fail* अंक 1 से सम्बंधित है। enumerator supplementary अंक 2 से सम्बंधित है और ये ही क्रम आगे भी चलता रहता है यदि हमारे पास इससे अधिक Enumerators हैं। हम इन अंको को बदल भी सकते हैं। हम जो अंक Assign करते हैं वे अंक इन अंको पर Over Write हो जाते हैं। जैसे

```
enum stu_status
{
    pass = 10, fail = 20, supplementary = 30;
};
enum stu_status Student1, Student2;
```

अब हम एक प्रोग्राम द्वारा इसका प्रयोग करना समझेंगे।

Program

```
#include<stdio.h>
main()
{
    enum stu_status
    {
        pass, fail, supplementary
    };

    struct Students
    {
    char name[20];
    int age;
    int class1;
    enum stu_status stud;
    };

    struct Students Student1;
    strcpy( Student1.name, "Kuldeep" );
    Student1.age = 20;
    Student1.class1 = 12;
    Student1.stud = pass;

    printf("\n Name %s", Student1.name);
    printf("\n Age %d", Student1.age);
```

```
        printf("\n Class %d", Student1.class1);
        printf("\n Status %d", Student1.stud);

        if(Student1.stud = = fail )
            printf("\n %-s is Fail", stident1.name );
        else
            printf("\n %-s is Not Fail ", Student1.name );
    }
```

Output

```
Name Kuldeep
Age 20
Class 12
Status 0
Kuldeep is Fail
```

इस Program में हमने enum में तीन मान pass, fail व supplementary रखे है। एक Structure बनाया है जिसमें चार Members हैं। चौथे Member के रूप में enum को nested किया है। फिर सभी Members को value assign किया हैं। Student1.stud = fail; किया है। ये Statement Student1.stud = 0; Statement से काफी सरल है जिससे ये समझ में आ जाता है कि Student1 को enumerated Data Type में fail दिया गया हैं।

enumerated Data Type के साथ ये कमी है कि हम किसी Function द्वारा fail या pass सीधे ही enum Data Type से print नहीं करवा सकते हैं। क्योंकि ये int प्रकार से अंकों के रूप में enum में रहते हैं। यदि इन्हें print करवाया जाता है तो ये enum के अंक ही print करते हैं जैसे कि printf("\n Status %d", Student1.stud); Statement द्वारा **pass** को print करवाने पर 0 Print होता है। यदि हमें pass print करवाना हो तो अलग से Statements लिखने होते हैं।

Bit Fields

यदि किसी Program में किसी Variable को केवल दो में से एक मान 0 या 1 Store करने होते हैं, तो हमें इन दोनों मानों को Store करने के लिए केवल एक Bit की जरूरत होती है। इसी तरह से यदि किसी Variable को चार मानों में से केवल एक मान ही Store करना हो, तो हमें केवल दो Bits की जरूरत होती है और यदि किसी Variable को आठ में से केवल एक मान Store करना हो, तो फिर इस जरूरत को केवल 3 Bits का प्रयोग करके पूरा किया जा सकता है।

जब हमारा काम Bits से हो सकता है, तो फिर इनके लिए पूरे Integers प्रकार के 2 Byte (16 Bits) को क्यों Use किया जाए ? लेकिन "C" मे कोई भी ऐसा Data Type नहीं है जो Bits पर

काम करता हो। फिर भी जब ऐसे कई Variables हों, जो पूरा Byte Use ना करते हों, तो उन्हे मिला कर एक ही Byte में विभिन्न Bits Allocate करके, हम हमारा काम कम से कम Memory में कर सकते हैं। इसे एक उदाहरण द्वारा समझने की कोशिश करते हैं:

Male या Female में से एक मान Store करना हो तो 1 Bit काफी है। यदि व्यक्ति Male है तो Variable को 1 कर दें अन्यथा 0. Single, Married, Divorced या Window में से एक को चुनना, हो तो हमें केवल 0, 1, 2, या 3 की जरूरत होती हैं। इसके लिए 2 Bit काफी हैं।

1 आठ में से एक Hobby को चुनना।
2 किसी Company द्वारा दिये जाने वाले 16 में से एक Offer को चुनना।

इसका मतलब हमें Gender Store करने के लिए एक Bit की जरूरत होगी। Marital status Store करने के लिए दो Bits, Hobby के लिए तीन Bits व किसी एक Offer को चुनने के लिए हमें चार Bit की जरूरत होगी। इस प्रकार से हमें इन सभी के लिए केवल 10 Bits की जरूरत होगी। जबकि एक int प्रकार के Data Type की Size 16 Bits की होती है। इसलिए हम एक ही int प्रकार के Variable में इन विभिन्न मानों को Store कर सकते हैं।

जैसा कि हमने पहले भी कहा कि "C" में Bits पर काम करने के लिए कोई Data Type नहीं बनाया गया है। लेकिन "C" में एक व्यवस्था है, जिससे हम किसी Variable के केवल कुछ Bits का ही प्रयोग कर सकते हैं। हमें किसी Variable के जितने Bits का प्रयोग करना हो यदि हम किसी Structure में Member के बाद Colon लगा कर वह संख्या लिख दें, तो "C" Compiler उस Member को लिखे गए अंकों के बराबर Bits Allot कर देता है। जैसे:

```
struct stu_status
{
    unsigned gender : 1;
    unsigned mar_stat : 2;
    unsigned hobby : 3;
    unsigned options : 1;
};
```

इस प्रकार से ये Structure **gender** प्रकार के Member को एक Bit, **mar_stat** प्रकार के Member को 2 Bits **hobby** को **3** व **options** को चार Bits प्रदान कर देगा। अब हम इसी Structure का प्रयोग करके किसी भी Student की status को कम से कम Memory द्वारा Access कर सकते हैं। आइये इसी Concept पर आधारित एक प्रोग्राम देखते हैं।

Program

```
#include<stdio.h>
#define Male 0
```

```
#define Female 1
#define Single 0
#define Married 1
#define Divorced 2
#define Widowed 4

main()
{
    struct stu_status
    {
        unsigned gender : 1;
        unsigned mar_stat : 2;
        unsigned hobby : 3;
        unsigned option : 1;
    };

    struct stu_status Student1;
    Student1.gender = Male;
    Student1.mar_stat = Married;
    Student1.hobby = 6;
    Student1.option = 10;
    printf("\, Gender = %d", Student1.gender );
    printf("\, Marital Status = %d", Student1.mar_stat );
    getch();
}
```

Output

```
Gender = 0
Marital Status = 1
```

Exercise:

1 **typedef** की उपयोगिता को समझाईए।

2 **typedef** व **enumerated** Data Type में क्या अन्तर है? एक उदाहरण Program में इन दोनों को Use करते हुए इनके काम करने के तरीके को समझाईए।

3 **Structure** व **enumerated** Data Type में क्या अन्तर है?

4 Bit-Fields की उपयोगिता को एक उदाहरण द्वारा समझाईए।

Chapter Level Exercise:

1 एक प्रोग्राम लिखिए जिसमें किसी Array में Store सभी मानों का योग एक User Define Function द्वारा main() Function को Return होता है।

2 एक Program बनाओ जिसमें Input की गई String में स्थित कुल Vowels की संख्या Output में Print हो।

3 Two Dimensional Array का Program बनाओ जिसमें User द्वारा Input किया जाने वाला मान व उसका Index Number दोनों Screen पर Display हो।

4 Three-Dimensional Array एक Program बनाओ जिसमें User द्वारा Input किया जाने वाला मान व उसका Index Number दोनों Screen पर Display हो।

5 निम्न Format में गिनती Print करने का Program बनाओ।

```
1     2     3     4     5     6     7     8     9    10
2     4     6     8    10    12    14    16    18    20
3     6     9    12    15    18    21    24    27    30
4     8    12    16    20    24    28    32    36    40
5    10    15    20    25    30    35    40    45    50
6    12    18    24    30    36    42    48    54    60
7    14    21    28    35    42    49    56    63    70
8    16    24    32    40    48    56    64    72    80
9    18    27    36    45    54    63    72    81    90
10   20    30    40    50    60    70    80    90   100
```

6 एक प्रोग्राम बनाइये जो ASCII Value व उसका Character निम्न Format में Print करे।

```
|  65 - A |  66 - B |  67 - C |  68 - D |  69 - E |  70 - F |
|  71 - G |  72 - H |  73 - I |  74 - J |  75 - K |  76 - L |
|  77 - M |  78 - N |  79 - O |  80 - P |  81 - Q |  82 - R |
|  83 - S |  84 - T |  85 - U |  86 - V |  87 - W |  88 - X |
|  89 - Y |  90 - Z |  97 - a |  98 - b |  99 - c | 100 - d |
| 101 - e | 102 - f | 103 - g | 104 - h | 105 - i | 106 - j |
| 107 - k | 108 - l | 109 - m | 110 - n | 111 - o | 112 - p |
| 113 - q | 114 - r | 115 - s | 116 - t | 117 - u | 118 - v |
| 119 - w | 120 - x | 121 - y | 122 - z |
```

7 एक प्रोग्राम बनाइये जो निम्नानुसार Input किये गए अंकों के Matrix को Transpose करे।

```
3 4 5          3 6 9
6 1 2          4 1 8
9 8 7          5 2 7
```

8 एक ऐसा Recursive Function Create कीजिए, जो Input की गई संख्या को Reverse Order में Convert करे।

9 Keyboard से Characters की एक Line Read करो और उस Line के हर Character की Counting को Display करने का Function Create करो।

10 किसी Array में Stored विभिन्न मानों को Reverse Order में Print करने का Function Create करो।

11 Pointer का प्रयोग करते हुए **strlen**(), **strcat**(), व **strcpy**() Functions Create कीजिए।

FILE MANAGEMENT

File Management in C

किसी भी Program का आधारभूत अवयव Data होता है। यानी जो भी Program लिखे जाते हैं वे इसीलिए लिखे जाते हैं ताकि User उसे उपयोग में लेकर अपनी सूचनाओं को Store करके रख सके और उस सूचना का अपनी आवश्यकतानुसार भविष्य में भी उपयोग कर सकें।

अभी तक हमने जो भी Program बनाए हैं, उनमें से किसी भी Program द्वारा कोई Data Hard Disk पर Store नही हुआ है। इस Chapter में हम ऐसे Program बनाना सीखेंगे, जिससे हम हमारे Data को किसी file के रूप में, किसी Storage Device मे store करके रख सकें। हम हमारे Data को दो तरीकों से Storage Device में store करके रख सकते हैं। या तो हम किसी बने बनाए Application Software का उपयोग करके उसमें अपने Data को Store कर लें या फिर आवश्यकतानुसार खुद की जरूरतों को पूरा करने के लिए अपनी खुद की Data File बनाए ।

"C" में ऐसी सुविधा है कि हम हमारे बनाए गए Programs के Data को hard Disk पर Store करके रख सकते हैं और आवश्यकता होने पर पुनः प्राप्त कर सकते हैं। इस Chapter में हम "C" में File Management के बारे में ही पढेंगे।

किसी भी Data File की कुछ मूलभूत जरूरतें होती हैं। जैसे किसी File में नया data जोडना, या किसी पुराने Data को Modify करना या किसी Data को Delete करना। जब हमें किसी "C" Program द्वारा प्राप्त Output को या Input किये गए Data को Hard disk पर Store करके रखना होता है, तब हमें "C" में एक Data File Create करनी पडती है।

Program द्वारा हम जो भी Data Input करते हैं या जो भी Data Output के रूप में हमें प्राप्त होता है, यदि उसे Hard Disk में भविष्य के उपयोग के लिए सुरक्षित करके रखना हो तो Data File create करना जरूरी होता है।

सर्वप्रथम हमें ये तय करना होता है कि हम किस काम के लिए Data File Create करना चाहते हैं। जब हम ये तय कर लेते हैं तब एक Data File Create की जाती है। उस Data File की पहचान के लिए उस Data File को कोई नाम देना जरूरी होता है। अब ये तय करना होता है कि उस Data File में हम Data लिखना चाहते हैं या पहले से लिखे Data को पढना चाहते हैं। यानी File को किस Mode में Open करना है और अंत में File को बंद करना होता है।

Opening a File

इससे पहले कि हम किसी सूचना को लिख या पढ सकें, हमें file को Open करना होगा। कोई भी File, जिसमें हम कुछ लिखना या पढना चाहते हैं, Open करने पर Operating System व Program के बीच एक Link बनता है। Operating System व हमारे Program के बीच जो Link बनता है, वह Link एक Structure होता है, जिसे **FILE** नाम दिया गया है। ये Structure **stdio.h** नाम की एक Header file में Define किया गया है। इसलिए जब भी हमें किसी File के

साथ काम करना हो तो हमें ये Header file अपने Program में Include करना जरूरी होता है। कोई भी File Open करने के लिए हमें निम्न Statement लिखना होता है:

```
FILE *fp;
```

ये Statement fp नाम का एक FILE प्रकार का Pointer Declare करता है। हम जितनी भी File Open करते हैं, उस हर File का एक अलग FILE Structure होता है। इस FILE Structure में File से सम्बंधित सारी जानकारी होती है, कि File कब Create की गई है। FILE की Current Size क्या है, File किस Memory Location पर स्थित है, आदि।

वास्तव में **FILE** का एक Character प्रकार का Pointer होता है, जो कि उस File के Read या Write होने वाले प्रथम Character को Point करता है। यहां **fp** एक Pointer Variable है, जिसमें Structure FILE का Address Stored है। File Pointer के Declaration के बाद अब हमें File Open करने के लिए **fopen**() Function का प्रयोग करना पडता है। इस Function में हमें Argument के रूप में दो parameters देने होते हैं:

- Open की जाने वाली File का नाम
- Open की जाने वाली File का Opening Mode यानी File को Writing Mode में Open किया जा रहा है या पहले से बनी File को Read करने के लिए या किसी अन्य काम के लिए।

ये दोनों parameters string के रूप में होती हैं, इसलिए इन्हें Double Quote में बंद करके इस Function में लिखा जाता है। माना हम Employee.dat नाम की File खोलना चाहते हैं और ये File पहले से Hard disk पर Stored है, इसलिए इस File को Reading Mode में खोलना चाहते हैं। इस स्थिति में हमें निम्नानुसार Argument देने होते हैं:

```
fp = fopen( " Employee.dat ", "r" );
```

ये Statement तीन काम करता है:

- ये Statement Employee.dat नाम की File को Hard disk पर ढूंढता है।
- यदि Hard Disk पर File हो तो उसे Memory में Load करता है। यदि File काफी बडी हो तो उसे टुकडों में बांट कर क्रम से हर भाग को Memory में Load करता है। यदि Disk पर File ना हो तो ये Function NULL Return करता है। NULL stdio.h header file में define किया गया एक macro है, जो ये बताता है कि इस नाम की कोई File Hard Disk पर नहीं है।
- ये एक Character Pointer को Structure FILE के प्रथम character का address दे देता है। यानी ये Pointer उस File के प्रथम character को Point करता है।

जब हम एक File से सम्बंधित सारे काम कर चुके होते हैं, तो Program के अंत में File को Close करना भी जरूरी होता है। File को बंद करने के लिए हमें File Pointer को ही निम्नानुसार Access करना होता है:

```
fclose ( fp );
```

fclose() Function कई काम करता है। जब हम कोई File बंद करते हैं, तो सर्व प्रथम Buffer में स्थित सभी Characters Disk पर Write होते हैं। **Buffer** High level I/O के लिए पहले से ही Define किया गया एक Memory Area है। हम जो कुछ भी किसी File में लिखते हैं, वो सीधे Disk पर जा कर Store नहीं होता, बल्कि Buffer नाम से Reserve की गई Memory में जा कर Store होता है। जब ये Buffer Characters से भर जाता है, तब Buffer से सारा Data एक साथ Disk पर Write हो जाता है।

जब हम File close करते हैं, तब Buffer में चाहे जितने Characters हों यानी Buffer भरा हो या चाहे फिर Buffer में केवल एक ही Character हो, Buffer का सारा Data Disk पर Write हो जाता है। जब File को Close किया जाता है, तब Program Control द्वारा Operating System व हमारे Program के बीच का Link व इस File के लिए Use हो रहा Buffer दोनों Free हो जाते हैं, ताकि इनका प्रयोग किसी अन्य File के लिए हो सके।

File Opening Modes

fopen() Function के साथ निम्न Modes का प्रयोग करके हम आवश्यकतानुसार किसी File को Open कर सकते हैं। सिर्फ "**r**" Mode को छोड कर हर Mode में, यदि File Disk पर नही होती है, तो दिये गए नाम की एक नई File Create हो जाती है। यदि File Disk पर नही होती है, तो fopen() Function **NULL** Return करता है। जबकि यदि File Disk पर होती है, तो उस File को Memory में Load कर देता है।

यदि File काफी बडी हो, तो Program Control File को कई भागों में बांट देता है और क्रम से हर भाग को Memory में Load करता है, साथ ही एक Structure **FILE** के Pointer Variable **pf** को File के प्रथम Character का Address प्रदान करता है या फिर ये कह सकते हैं कि एक Program को Setup करता है, जो File के प्रथम Character को Point करता है। File से सम्बंधित विभिन्न Modes निम्नानुसार हैं:

"**r**" Program Control File को disk पर खोजता है। यदि File प्राप्त हो जाती है, तो उसे Memory में Load करता है और एक File Pointer को File के प्रथम character को point करने के लिए set up करता है। जब हमें पूर्व में किसी File में लिखे गए Data को Read करना होता है, तब हम File को इस Mode में Open करते हैं।

"**w**" Program Control File को disk पर खोजता है। यदि disk पर File हो तो उस File के Data को नए Data से Overwrite कर देता है। यदि Disk पर File नहीं होती है, तो ये दिये गए नाम की एक नई File Create करता है। यदि किसी कारणवश File Open नही हो पाती है, तो ये NULL Return करता है। जब File में कुछ लिखना होता है, तब हम इस Mode का चयन करने हैं।

"**a**" Program Control को जब ये Opening Mode प्राप्त होता है, तो Program Control सबसे पहले दिये गए नाम की File को Disk पर खोजता है। यदि File मिल जाती है तो उसे memory में load करता है व एक File pointer को इस File के प्रथम Character का Address देता है। यदि File disk पर नहीं होती है, तो ये string एक नई File Create करता है। इस Mode का प्रयोग तब किया जाता है, जब किसी File के अंत में नए Data जोडने हों।

"**r+**" इस Mode का प्रयोग Program में तब किया जाता है, जब किसी File से Data Read करने हों, Data को Modify करना हो या फिर नया Data File में write करना हो।

"**w+**" जब इस Mode को use किया जाता है, तब Program Control दिये गए नाम की File को Disk पर खोजता है। यदि Disk पर File स्थित हो, तो ये Mode उस पुरानी File के सारे Data को नष्ट कर देता है। इसका प्रयोग तब किया जाता है, जब हमें File में नया Data Add करना हो, Data को वापस Read करना हो या फिर पुराने Data को Modify करना हो।

"**a+**" जब हमें किसी पुरानी File के अंत में नए Data Add करने हों या File को Read करना हो, तब इस Mode को Use किया जाता है। File के data का Modification इस mode में नहीं किया जा सकता है।

किसी भी File में हम जब Data लिख चुके होते हैं तो File का अंत करने के लिए हमें Function Key F6 या ^z Key Combination का Use करना पडता है। इन Special Keys की ASCII Value 26 होती है और ये ASCII अंक 26 किसी भी File के अंत (EOF) को बताती है।

जब भी Program Control को ASCII Value 26 प्राप्त होती है, तो Program Control समझ जाता है कि User ने File में अपना काम पूरा कर लिया है और User इसमें आगे काम नहीं करना चाहता है। हम File में Data लिखने के लिए एक साधारण सा Program बनाते हैं। इस Program में हम एक data.nam नाम की File को Writing Mode में Open करेंगे और इसमें एक नाम Store करेंगे। Program निम्नानुसार है:

Program

```
#include<stdio.h>
main()
```

```
{
    FILE *fp;
    char ch;
    clrscr();

    printf("\n Input Name :");

    fp = fopen("Data.nam", "w" );

    while((ch = getc (stdin)) != EOF )
        putc (ch, fp);

    fclose (fp);
}
```

आइये पहले इस Program में Use हुए Functions को समझें। इस प्रोग्राम में दो नए Functions को Use किया गया है, जो कि निम्नानुसार हैं:

getc( )

यह Function किसी Stream से Input प्राप्त करने का काम करता है। ये Stream FILE Pointer या फिर stdin (Keyboard) ही हो सकता है। इसका Syntax निम्नानुसार होता है:

```
ch = getc( FILE pointer or Stream );
```

putc( )

यह Function getc() Function से प्राप्त characters को किसी FILE pointer द्वारा, किसी File में Store करता है या किसी stream (stdout or stdprn) पर show करता है। इसका Syntax निम्नानुसार होता है:

```
putc( ch, FILE pointer or Stream );
```

इस प्रोग्राम में FILE प्रकार का एक Pointer Variable Declare किया गया है। पूरे Program में इसी Pointer द्वारा FILE Structure को Access किया जाता है। इस प्रोग्राम में हम एक–एक Character को stdin (Key Board) से Accept करते हैं।

हर Character को Accept करने के लिए हमने char प्रकार का ch नाम का एक Variable Define किया है। फिर हर character को क्रम से Accept करके उसे fp FILE Pointer द्वारा File में Write कर दिया गया हैं। जब इस Program को Execute किया जाता है, तो सभी Definition व Declaration के बाद Program Control **printf**() Function का निम्न Message Screen पर Print करता है।

```
"Input Name :"
```

ये Message Screen पर Print करने के बाद Program Control निम्न Statement को Execute करता है।

```
"fp = fopen("Data.nam", "w" );
```

fopen() Function, **Data.nam** नाम की एक File Writing Mode में Open करता है और **fp** Pointer को इस File के पहले Character का Address दे देता है। ये सवाल दिमाग में आ सकता है कि जब कोई नई File Create होती है, तब उसमें कुछ भी लिखा हुआ नहीं होता। उस समय fp कैसे File के प्रथम Character को Point कर सकता है।

इस सवाल का जवाब ये है कि किसी भी File में कुछ लिखा हो या ना लिखा हो, ये हमेंशा निश्चित होता है, कि यदि कोई Character लिखा जाएगा तो, File में वह सबसे पहला Character किस Location पर जा कर Store होगा।

File Pointer **fp** में उसी Location का Address Stored होता है ना कि File का पहला Character क्योंकि fp एक Pointer है और किसी भी Pointer में किसी अन्य Variable का Address ही Store हो सकता है। इस प्रकार से File Pointer में File के प्रथम Character के Store होने की Location का Address Stored होता है।

हम जानते हैं कि **getc**() Function द्वारा हम एक बार में केवल एक ही Character को Input कर सकते हैं लेकिन हमें तो तब तक Characters Input करने है जब तक कि EOF प्राप्त ना हो जाए।

इसलिए हमने एक **while** Loop का प्रयोग किया है। हमने Loop Chapter के अंतर्गत बताया था कि जब हमें ये पता नहीं होता है कि Loop को कितनी बार चलाना है, तब while Loop को Use करना अधिक अच्छा रहता है। इसीलिए यहां हमने while Loop द्वारा Characters Input करवाए हैं। आइये निम्न Statement को समझते हैं:

```
while((ch = getc (stdin)) != EOF )
```

इस Statement में ch = getc (stdin) Statement का Use किया गया है। ये Statement, Standard Input Device (Key Board) से एक character Accept करता है और उस Character को **ch** नाम के char प्रकार के Variable में Store कर देता है।

चूंकि while Loop तभी Terminate हो सकता है जब Program Control को **EOF** प्राप्त हो और Program Control को EOF तभी प्राप्त होगा जब User Key Board से **^z** या Function Key **F6** को Press करे। यदि User Key Board से ^z या Function Key F6 को Press नही करता है, तो while Loop की Condition सत्य होती है और Program Control while के Statement Block में प्रवेश करता है।

चूंकि यहां while Condition के सत्य होने पर केवल एक ही Statement को Execute करना है, इसलिए यहां Statement Block का प्रयोग नहीं किया गया है। जब while Condition सत्य होती है, तब Program Control निम्न Statement को Execute करता हैः

```
putc (ch, fp);
```

putc() Function getc() Function से प्राप्त character, जो कि ch नाम के char प्रकार के Variable में Stored है, File pointer द्वारा File में Store कर देता है, साथ ही File Pointer **fp** को File की Next Location का Address प्रदान कर देता है, जिससे File Pointer **fp** File के दूसरे Character की Location को Point करने लगता है।

ये Statement Execute होने के बाद पुनः while Loop Execute होता है और keyboard से Character Accept करता है। putc() Function पुनः getc() Function से प्राप्त Character को File में Store कर देता है और File pointer **fp** को Next Location का Address प्रदान कर देता है। ये क्रम तब तक चलता रहता है जब तक कि User Key Board से Function Key F6 या ^z Press नही कर देता।

इस प्रकार से हम getc() व putc() Function द्वारा एक–एक characters को किसी File में Store कर सकते हैं। यदि हमें ये देखना हो कि हमारे द्वारा Create की गई Data.nam नाम की File Disc पर बनी या नहीं और यदि बनी तो उसमें Inputted Data Store हुए या नहीं, तो इसके लिए हम निम्नानुसार हमारी File के Data को देख सकते हैंः

- File को Dos Prompt पर Type या Edit Command के बाद Data File Data.nam नाम देकर और
- एक और "C" Program बना कर जिसमें Data.nam नाम की File को Reading Mode में Open करके

यहां हम दोनों तरीकों से File को Read करना बता रहे है। पहले तरीके में DOS Prompt पर जाने के लिए Turbo C के File menu में जाएं और वहां Dos Shell नाम का Option Choose करें। इस Option को Choose करते ही हम Temporarily Dos Prompt पर चले जाते हैं। यहां निम्न Command देकर File के Data को देखा जा सकता है:

```
C:\TC\BIN> Type Data.nam
```

इस Command से Data File में Stored सभी Data Screen पर Print हो जाते हैं।

अब हम दूसरे तरीके से Data File के Data को Read करते हैं। इस तरीके में एक Program लिखना होगा। उस प्रोग्राम में Data File को Reading Mode में Open करके, उसके Data को **Character By Character** Screen पर Print करना होगा। यानी जो काम हमने Data Input के लिए किया है ठीक उसका उल्टा अब हमें वापस Data प्राप्त करने के लिए करना होगा।

इसके लिए सर्वप्रथम हमें Data.nam नाम की Data File को Reading Mode में Open करना होगा। फिर उस File के प्रथम Character को Read करके उस Character को Screen पर Print करना होगा। फिर यही क्रम दूसरे character के साथ अपना कर उसे Screen पर Print करना होगा और ये क्रम तब तक अपनाना होगा, जब तक कि Program Control को EOF प्राप्त ना हो जाए। चलिए, अब हम इसी क्रम में Program लिखने की कोशिश करते हैं।

सर्वप्रथम I/O Operations के लिए stdio.h नाम की header file को प्रोग्राम में Include करते हैं। यानी #include<stdio.h> फिर main() लिख कर Statement Block में एक FILE Structure का Pointer Variable *fp Declare करते हैं और Character को Accept करने के लिए एक char प्रकार का Variable ch Declare करते हैं। यानी

```
FILE *fp;
char ch;
```

Screen पर पहले से यदि कोई Matter लिखा हो, तो उसे साफ करने के लिए clrscr(); लिखते हैं। अब Data.nam नाम की Data File जिसे Read करना है, Reading Mode में Open करते हैं और इस File का Base Address FILE Structure के Pointer Variable **fp** को देते है। यानी

```
fp=fopen("Data.nam", "r" );
```

अब while Loop लेते हैं और इसी Loop में एक getc() Function द्वारा Data.nam File के Characters को Read करके Read किये गए Character को ch नाम के Variable में Store करते हैं और Loop को तब तक चलाते हैं, जब तक कि ch में EOF ना आ जाए। यानी

```
while((ch = getc(fp)) != EOF );
```

अब getc() Function द्वारा File Structure से प्राप्त Character जो कि Variable ch में Stored हैं, को **Standard Output VDU** या **Screen** पर Print कर देते हैं। यानी

```
putc( ch, stdout );
```

अंत में Open की गई File को Close करते हैं। यानी

```
close(fp);
```

और Output दिखाते समय Screen तब तक रूकी रहे, जब तक कि हम कोई Key press ना करें, इसके लिए **getch();** Statement लिखते हैं। अब main() Function के Statement Block को बंद करते हैं। इन सभी Statements को जिस क्रम में बताया गया है उसी क्रम में लिख लेने पर Program निम्नानुसार होता है:

Program

```
#include<stdio.h>

main()
{
    FILE *fp;
    char ch;
    clrscr();

    fp = fopen ("Data.nam", "r" );

    while((ch = getc(fp)) != EOF )
        putc( ch, stdout );

    fclose(fp);
    getch();
}
```

यदि हम दो Program ना लिख कर ये चाहें कि पहले प्रोग्राम में ही Data Inputting के बाद Data किस प्रकार Input हुए हैं, ये पता चल जाए तो हम दोनों Programs को मिलाकर निम्नानुसार लिख सकते हैं:

Program

```
#include<stdio.h>
```

```
main()
{
    FILE *fp;
    char ch;
    clrscr();

    fp = fopen ("Data.nam", "w" );
    while((ch = getc(stdin))!= EOF )
        putc( ch, fp );
    fclose(fp);

    fp = fopen ("Data.nam", "r" );
    while((ch = getc(fp)) != EOF )
        putc( ch, stdout );
    fclose(fp);

    getch();
}
```

हम जितनी बार भी इस File को Execute करते हैं तो पुराना Data नए Data पर Overwrite हो जाता है। यदि हम ये चाहें कि हमारा पुराना Data ज्यों का त्यों रहे और नया Data पुराने Data के बाद में File में Store हो, तो हमें हमारे Program में File को Writing Mode में Open ना करके Appending Mode में Open करना होगा। इसी Program में ये Change किया गया है और नया प्रोग्राम निम्नानुसार है। इस Program में पुराना Data ज्यों का त्यों रहता है और नया Data पुराने Data के बाद जुड जाता है:

Program

```
#include<stdio.h>

main()
{
    FILE *fp;
    char ch;
    clrscr();

    fp = fopen ("Data.nam", "a+" );

    while((ch = getc(stdin))!= EOF )
```

```
        putc( ch, fp );

    fclose(fp);

    fp = fopen ("Data.nam", "r" );

    while((ch = getc(fp)) != EOF )
        putc( ch, stdout );

    fclose(fp);
    getch();
}
```

ध्यान दें कि यहां हमने Data.nam File को "a+" Mode में Open किया है। यदि हम चाहें तो "a+" को "+a" भी लिख सकते हैं। इससे Program में कोई Error नहीं आती है।

हमने हमारे Program में getc() Function का प्रयोग किया है। यदि हम चाहें तो इसके स्थान पर getchar() Function को भी प्रयोग किया जा सकता है। getchar() Function का कोष्ठक हमेंशा खाली ही रखा जाता है, क्योंकि ये Function **getc(stdin)** का ही एक Macro है यानी हम **getc(stdin)** का प्रयोग करें या **getchar**() का दोनों का काम एक ही है।

इसी प्रकार से **putchar**() Function को **putc(stdout)** के स्थान पर प्रयोग किया जा सकता है, क्योंकि ये **putc(stdout)** का Macro है। हम **putc**() Function या **putchar**() Function के स्थान पर **printf**() Function का प्रयोग करके भी Characters को Output में Print कर सकते हैं और **scan**() Function का प्रयोग Character को Accept करने के लिए भी कर सकते हैं।

हम **getc**() के स्थान पर हम **fgetc**() Function का भी प्रयोग कर सकते हैं और **putc**() Function के स्थान पर **fputc**() Function का, **getc**() व **putc**() Function क्रमशः **fgetc**() Function व **fputc**() Function के Macro हैं। इस प्रकार से हमने हमारे प्रोग्राम में जहां भी **getc**() व **putc**() Function का प्रयोग किया है, वहां पर क्रमशः **fgetc**() व **fputc**() Function को Replace कर दें, तो Program पर कोई फर्क नहीं पडेगा और Output वही प्राप्त होगा जो पहले प्राप्त हुआ था।

जब हमें किसी Program द्वारा अंकों में मान प्राप्त होता है या फिर हमें अंको से सम्बंधित सूचना को Disk पर Store करना होता है, तब ये काम करने के लिए हम getc() व putc() Function का प्रयोग नहीं कर सकते हैं।

क्योंकि ये Functions केवल Character प्रकार के Data पर काम करते हैं। इस काम के लिए "C" में दो अन्य Function हैं जो कि क्रमशः getc() व putc() Function की तरह ही काम करते हैं, लेकिन ये अंकों के साथ काम करते समय Use होते हैं। ये Function **getw**() व **putw**() हैं।

getw ( )

Syntax Integer_Variable = getw (FILE Pointer or Stream)

ये Function किसी File से या फिर किसी Input stream (stdin) Keyboard से मान प्राप्त करता है और Assignment Operator के Left में स्थित Variable में Store कर देता है।

putw ( )

Syntax putw (Integer_Variable, FILE Pointer or Stream)

ये Function getw() Function द्वारा Integer_Variable में प्राप्त किये गए अंक को किसी File में या फिर standard Output device (Screen या Printer की Buffer Memory) में Store कर देता है।

इन **getw**() व **putw**() Functions का उपयोग समझने के लिए हम एक Program बनाते हैं, जिसमें 20 अंक **Data.dig** नाम की File में Input किया जाएगा और इस Data File से पुनः अंकों को Output में Program द्वारा ही Print किया जाएगाः

Program

```
#include<stdio.h>

main()
{
    FILE *fp1, *fp2;
    int ch, j;
    clrscr();

    fp1 = fopen ("Data.dig", "w" );
    printf("\n Enter Numbers ");

    for ( j=1; j<=20; j++)
    {
        ch = getw( stdin );
        putw ( ch, fp1 );
    }
```

```
        fclose(fp1);
        fp1 = fopen ( "Data.dig", "r" );

        while((ch = getw(fp1) != EOF )
        {
            putw( ch, stdout );
        }
        fclose(fp1);
        getch();
    }
```

इस Program में getw() Function द्वारा data Input किया गया है और putw() Function द्वारा एक बार Data को File में लिखा गया है और एक बार Screen पर। अभी तक हमने while Loop को तब तक चलाया है, जब तक कि EOF प्राप्त नहीं हो जाता। "C" में एक और Function है, जो EOF के लिए प्रयोग किया जा सकता है। ये है feof() Function जिसे निम्नानुसार समझाया गया है:

feof( )

syntax: feof(FILE Pointer)

यह एक Macro है जो तब तक 0 Return करता है जब तक कि Program Control को EOF प्राप्त नहीं हो जाता है। जैसे ही Program Control को EOF प्राप्त होता है, ये Function एक Non-Zero मान Return करता है। while Loop में इसे निम्नानुसार Use किया जा सकता है:

```
    while( !feof )
    {
        Statement 1;
        Statement 2;
    }
```

अब ये Loop तब तक चलेगा जब तक कि Program Control को EOF प्राप्त नहीं हो जाता है।

कई बार हमें ऐसी जरूरतें होती हैं कि पूरा का पूरा String किसी File में लिखना होता है। हम जानते हैं कि String Accept करने के लिए व उस String को Screen पर Print करने के लिए हम **gets**() व **puts**() Functions का प्रयोग करते हैं।

ठीक इसी प्रकार से किसी String को किसी File में लिखने के लिए या किसी File से पूरा का पूरा String प्राप्त करने के लिए हमें **fgets**() व **fputs**() Function Use करना पडता है। चलिए, इन Function को समझते हैं:

fgets ( )

Syntax: **fgets (ch, size, FILE pointer or Stream)**

ch: ये वह Variable है जिसमें किसी File या Stream से प्राप्त किये गए strings को store किया जाता है। ये एक One-Dimensional Array होता है।

size: यहां ch Variable में Store होने वाले characters की संख्या बतानी होती है ताकि ch नाम का Array Over Flow ना हो।

FILE pointer या Stream:

ये या तो कोई FILE Pointer हो सकता है या फिर standard Input Device Keyboard होता है।

fputs ( )

Syntax: **fputs (ch, FILE pointer or Stream)**

ch: ये वह Variable है, जिसमें gets() या fgets() Function द्वारा प्राप्त String होता है।

FILE pointer या Stream: ये या तो कोई FILE Pointer हो सकता है या फिर standard Output Device हो सकता है।

हम अपनी जरूरत के अनुसार इन Functions का प्रयोग कर सकते हैं। लेकिन अभी तक हमने जिन भी Functions को Use किया है, उनमें से कोई भी Function Formatting की सुविधा नहीं देता है।

यदि हमें File में Store होने वाले Data को किसी Format में Store करना हो तो हमें अन्य Functions को Use करना होता है। "C" में Formatted Data Storing के लिए दो Functions हैं, जिनसे एक File में Store होने वाले Data विशेष format में Store होते हैं।

fprintf( )

Syntax: fprintf (fp, "Message And Control Strings" , Arguments);

यहां fp Open की गई File का File Pointer है जो Input होने वाले Data को File में Store कर देता है। हम यहां वे सभी Formatting Use कर सकते हैं, जिन्हे printf() Function में Use किया जाता है। printf() Function व fprintf() Function में यही अन्तर है कि printf() Function सभी मानों को screen पर Print करता है यानी VDU की Memory में store करता है, जबकि fprintf() Function सभी मानों को Open की गई File में Store करता है।

fscanf( )

Syntax: fscanf (fp, "Control Strings" , Arguments);

जिस प्रकार से scanf() Function Keyboard से values Accept करता है, उसी प्रकार से fscanf() Function किसी File में Store किये गए मानों को Accept करता है। अब हम एक Program बनाते हैं, जिसमें Data, Formatted रूप में Data File में store होंगेः

Program

```
#include<stdio.h>

main()
{
    FILE *fp;
    char again = 'y';
    char name[20];
    int age;
    float salary;
    clrscr();

    fp = fopen("Emp.rec", "w");

    while( again =='y' || again == 'Y' )
    {
        printf("\n Enter Name:  Age and Salary");
        scanf("%s%d%f", name, &age,&salary);
        fprintf(fp,"%s%d%f\n", name, age, salary);
        printf("\n Do You Want To Enter Another Record : Y/N ");
        fflush(stdin);
        again = getchar();
    }
    fclose(fp);
}
```

इस प्रोग्राम में किसी भी Employee के Record को Store करने के लिए एक File को Writing Mode में Open किया गया है। इस प्रोग्राम को Execute करने पर यह प्रोग्राम सभी Variables के Definition व Declaration के बाद एक printf() व scanf() Function द्वारा User Data Input करता है।

यहां **fprintf**() Function को Use किया गया है। ये Function उस File को लिखता है, जिस File का File Pointer इन Arguments को Accept करने के लिए **fprintf**() Function में Use किया गया है।

इस Program में हमने fprintf() Function में fp नाम का File Pointer Use किया है, जो कि Employee.rec नाम की Open की गई File का File Pointer है। User जो भी Data Input करता है, वो क्रमशः name, age व salary नाम के Variable में जाकर store हो जाते हैं। फिर इन Variables के Data को fprintf() Function द्वारा File में Write कर दिया जाता है।

इस वजह से ये सभी Data यानी Name, Age व Salary दिए गए Format में ही Emp.rec नाम की File में Store होते है। फिर Program Control User को एक **printf**() Function द्वारा Message देता है, कि User और Data Input करना चाहता है या नहीं।

यदि User **Y** press करता है, तो वापस ये क्रम दोहराया जाता है और वापस Data File में store हो जाते हैं। यदि User N press करता है तो Program Control Program से बाहर आ जाता है।

जिस प्रकार से printf() Function में & Address Operator का प्रयोग नहीं किया जाता है, उसी प्रकार से File में Data Write करने के लिए भी fprintf() Function के साथ केवल उस Variable को लिखा जाता है, जिसमें User द्वारा Input किया गया Data होता है।

इस प्रकार से name, age व salary नाम के Variable में Keyboard से प्राप्त किये गए Data को File में Write कर दिया जाता है। यदि Emp.rec File में हमने जो Data Store किये हैं, उसे उसी Format में प्राप्त करना हो, जिस Format में ये File में Stored हैं, तो हमें fscanf() Function का प्रयोग करना पडता है। ऊपर के प्रोग्राम द्वारा Input किये गए Data को अगले प्रोग्राम द्वारा Screen पर print किया गया है–

Program

```
#include<stdio.h>

main()
{
    FILE *fp;
    char name[20];
```

```
        int age;
        float salary;
        clrscr();

        fp=fopen("emp.rec","r");
        while(fscanf(fp,"%s %d %f",name,&age, &salary) != EOF )
        printf("\n %s %d %f",name, age, salary);
        fclose(fp);

        getch();
    }
```

इस Program से emp.rec नाम की File में Store किये गए Data वापस screen पर ज्यों का त्यों print हो जाते हैं।

Standard DOS Services

किसी File से Data Reading / Writing के लिए हमें fopen() Function का प्रयोग करना होता है, जो कि एक File pointer बनाता है। ये File pointer उस File को Point करता है। MD-DOS में भी पांच predefined Functions हैं। इन्हें Access करने के लिए हमें fopen() Function की जरूरत नहीं होती हैं। ये Standard File pointers निम्नानुसार हैं–

stdin	Standard Input Device (Keyboard)
stdout	Standard Output Device (VDU)
stderr	Standard Error Device (VDU)
stdaux	Standard Auxiliary Device (Serial Ports)
stdprn	Standard Printing Device (Parallel Printer)

ये सभी Standard File Pointers **stdio.h** नाम की Header File मे Define हैं, इसलिए इनको Use करने के लिए इस Header File को Program में Include करना जरूरी होता है। किसी File में अंको को Store करने के लिए हमारे पास **fprintf**() ही एक Function है। यानी हम इसी Function द्वारा किसी File में किसी भी प्रकार के अंक को Store कर सकते हैं। जब हम File में अंकों को Store करते हैं, तो ये अंक File में String के रूप में Store होते हैं।

जैसे float प्रकार का एक मान 123.223 यदि File में Store किया जाए तो ये मान File में 4 Byte ना लेकर 7 Byte की Space लेगा, क्योंकि ये File में Characters के एक समूह के रूप में Store होता है। इसलिए यदि हमें काफी बडी संख्या को File में Store करना हो तो File में बहुत

अधिक Storage Space Reserve हो जाएगा। इस समस्या से बचने के लिए हम File को Text Mode मे Open ना करके Binary Mode में Open कर सकते हैं।

जब हम File को Binary Mode में Open करते हैं, तब हमें **fread**() व **fwrite**() Function का प्रयोग करना चाहिये। ये Functions अंकों को Binary Mode में Store करता है। यानी कोई अंक Memory में जितने Byte लेता है, इन Functions के प्रयोग से वह अंक File में भी उतने ही Byte की Space लेता है।

जब हमें किसी File में Records Store करके रखने होते है, जैसे कि किसी Company के Employees के Records या किसी School के Students के Records, तो हम इन Records को अच्छी तरह से Maintain करने के लिए Structure का प्रयोग कर सकते हैं। उदाहरण के लिए हम पिछले प्रोग्राम को ही Structure के प्रयोग से बना कर Data को File में Store कर रहे हैं:

Program

```
#include<stdio.h>
main()
{
    FILE *fp;
    char again = 'y';
    struct emp
    {
        char name[20];
        int age;
        float salary;
    };
    struct emp e;

    clrscr();

    fp = fopen("Emp.rec", "w");

    while( again ='y' || again == 'Y' )
    {
        printf("\n Enter Name  Age and Salary");
        scanf("%s %d %f", e.name, &e.age, &e.salary);

        fprintf(fp, "%s %d %f\n", e.name, e.age, e.salary);

        printf("\n Do You Want To Enter Another Record : Y/N ");
```

```
        fflush(stdin);
        again = getchar();
    }
    fclose(fp);
}
```

इस प्रोग्राम द्वारा भी हम Employees के Records को File मे Store कर सकते हैं। लेकिन इस प्रोग्राम की दो कमियां हैं:

- इस Program में File को Text Mode में Open किया गया है, इसलिए इस File में **fprintf**() Function द्वारा **salary** को Store किया गया है। इस वजह से File में **salary** String के रूप में Store होती है, जिससे File में अधिक Bytes का उपयोग होता है।
- दूसरी कमी ये है कि यदि Structure के Members बढाए जाएंगे, तो इस Structure को संचालित करना काफी मुश्किल हो जाएगा।

इन समस्याओं से बचने के लिए हम File को Binary Mode में Open करके **fread**() व **fwrite**() Functions का प्रयोग कर सकते हैं।

किसी भी File को Binary Mode में Open करने के लिए हमें Mode String के साथ "**b**" का प्रयोग करना पडता है। हमने इसी Program को थोडा Modify करके फिर से Develop किया है, जिसमें File को Binary Mode में Open किया है और **fprintf**() Function के स्थान पर **fread**() व **fscanf**() Function के स्थान पर **fwrite**() Function का प्रयोग करके प्रोग्राम से ऊपर बताई गई समस्याओं दूर किया गया है। ये Program निम्नानुसार है:

Program

```
#include<stdio.h>

main()
{
    FILE *fp;
    char again = 'y';

    struct emp
    {
        char name[20];
        int age;
        float salary;
    };
    struct emp e;
```

```
    clrscr();

    fp = fopen("Emp.rec", "wb");

    while( again =='y' || again == 'Y' )
    {
        printf("\n Enter Name  Age and Salary");
        scanf("%s %d %f", e.name, &e.age, &e.salary);
        fwrite (&e, sizeof(e), 1, fp);
        printf("\n Do You Want To Enter Another Record : Y/N ");
        fflush(stdin);
        again = getchar();
    }
    fclose(fp);
}
```

इस File को Binary Mode में Open करने के लिए निम्न Statement में "wb" Mode String का प्रयोग किया गया है:

```
fp = fopen("Emp.rec", "wb");
```

"**wb**" Mode String File को Writing Mode में Open करता है और इसमें Store होने वाले Data Binary रूप में File में Store होते हैं। Binary Mode का मतलब ये होता है, कि यदि हम इस File के Data को Read करना चाहते हैं, तो हम इसके Data को साधारण तरीके से Read नहीं कर सकते हैं। इसके Data को Read करने के लिए हमें वापस इसे Binary Mode में ही Open करना होगा। यदि हम इस Data File को साधारण Mode में Open करेंगे, तो हमें File में ना समझ में आने वाले अजीब से चिन्ह दिखाई देंगे ना कि वास्तविक Data जो कि उस File में Store किए गए हैं।

जब हम Keyboard से Data Input करते हैं, तो वह Data **e** नाम के Structure Variable में जा कर Store हो जाते हैं। इस Program में हमने File को Binary Mode में Open किया है, इसलिए Data को Binary रूप में File में Store करने के लिए **fwrite**() Function को Use किया है। इस Function में चार Arguments Pass किए जाते हैं:

1. इस Function में पहला Argument **&e** है, जो कि File में Write किये जाने वाले Structure का Address है। यह Argument "C" Compiler को बताता है कि जो Data File में Store होगा वह Data इस Structure से आएगा।
2. दूसरा Argument Program Control को ये बताता है, कि File में store होने वाले data जिस Variable से आ रहे हैं, उस Variable की Size क्या है ? यानी वह Variable

memory में कितनी Byte Data के लिए Reserve करता है। जैसे यदि Data Int प्रकार का हो तो ये size 2 Byte होगी और यदि Data Float प्रकार का हो तो इसकी Size 4 byte की होगी। चूंकि हम यहां पर एक Structure प्रकार के Variable द्वारा Data को File में Store कर रहे हैं, इसलिए यहां **sizeof**() Operator के अंदर ही Structure Variable **e** को लिख दिया गया है, जिससे Program Control स्वयं ही ये पता लगा ले कि Structure की Size क्या है ? ये Argument हमें ये सुविधा देता है कि यदि Structure के Members बढाए या घटाए जाते हैं, तो इस Statement द्वारा "C" Compiler स्वयं ही बदली हुई Size को पहचान लेता है।

3 तीसरा Argument ये बताता है कि हमें एक बार में कितने Structures को File में Write करना है। यहां हम एक बार में केवल एक Structure को File में Write कर रहे हैं, इसलिए इसकी संख्या 1 रखी है। यदि हम चाहें तो एक Array का प्रयोग करके एक से अधिक Structures को एक समय में एक साथ File में लिख सकते हैं।

4 अन्तिम Argument उस File का File Pointer है, जिसमें हमें Data को Write करना है।

इस प्रकार से इस Program द्वारा हम किसी File में उतने ही Byte का उपयोग करके Data को लिख सकते हैं, जितने Byte का उपयोग वह Program Memory में करता है। अब हम इस Binary Mode File में लिखे गए Data को वापस प्राप्त करने के लिए एक Program लिख रहे हैं। इस Program में **fread**() Function का प्रयोग किया गया है।

Program

```
#include<stdio.h>
main()
{
    FILE *fp;
    struct emp
    {
        char name[20];
        int age;
        float salary;
    };
    struct emp e;
    clrscr();
    fp = fopen ("emp.rec" , "rb");
    while(fread(&e, sizeof(e), 1, fp ) == 1 )
        printf("\n %s\t %d\t %f\t", e.name, e.age, e.salary);
    fclose(fp);
    getch();
}
```

यह Program emp.rec नाम की File में Store Data को Output में Screen पर दिखा देता है। इस Program में File से Data को Read करने के लिए **fread**() Function का प्रयोग किया है। इस Function में भी चार Arguments Pass किए जाते हैं:

- पहला Argument &e Program Control को बताता है, कि जो Data File से Read किया जाना है, वह Data e नाम के एक Variable में है और ये Variable एक Structure प्रकार का Variable है।
- दूसरा Argument Program Control को बताता है कि Read हो रहे Data की Size क्या है।
- तीसरा Argument Program Control को बताता है कि File से कितने Data एक साथ प्राप्त करने हैं और
- चौथा Argument उस File का File pointer है, जिसमे Data Stored है।

ध्यान दें कि **fread**() व **fwrite**() दोनों ही Function में दिये गए Arguments एक समान हैं। ये Argument हमेंशा एक समान ही होने चाहिये, तभी सही तरीके से Data Reading होती है।

इस Program में fread() Function को while Loop के अंदर Condition के रूप में लिखा गया है। जब fread() Function File से कोई Data Read करता है, तब fread() Function **1** Return करता है। लेकिन जब **fread**() File के अंत में पहुंचता है, जहां उसे Read करने के लिए कोई Data नहीं होता तो **fread**() Function **0** Return करता है और जब **fread**() 0 Return करता है, तब while Loop Terminate हो जाता है।

यानी जब Program Control **EOF** पर पहुंचता है, तो Read करने के लिए कुछ भी नहीं होता है। ऐसे में **fread**() Function **0** Return करता है, जिससे while Loop Terminate हो जाता है और इसका मतलब होता है कि File से सभी Data Read किये जा चुके हैं।

rewind( );

अभी तक हमने ये देखा है कि जब भी File के साथ कोई काम करना होता है, तब File Open करनी पडती है। इससे Program की लम्बाई काफी बढ जाती है और Program देखने में काफी मुश्किल लगता है।

इस समस्या से बचने के लिए "C" में एक **rewind**() नाम का Library Function है, जो कि **stdio.h** नाम की Header File में Define किया गया है। Program Control File में किसी भी Location पर स्थित हो, इस Function के प्रयोग से File की शुरूआत में पहुंच जाता है। यानी ये

Function File Pointer को File की शुरूआत पर ले जाता है, जिससे File को बार–बार Open व Close नहीं करना पडता है। इसका Syntax निम्नानुसार होता है:

```
rewind( File_pointer );
```

ferror( );

यदि File Reading के समय सही प्रकार से Read ना हो, तो ये Function **Non-Zero** Return करता है और यदि Data ठीक से Read हो रहा होता है तो **0 Return** करता है। इसे Use करने का Syntax निम्नानुसार होता है:

```
Statement of File Reading
    if(ferror())
    {
        printf("Could Not Read Data Properly";
        break;
    }
```

जब File से Data reading के समय कोई Error आती है तब ferror() Function Execute होता है और "**Could Not Read Data Properly**" Message Print कर देता है। साथ ही Program Control को Data Reading Loop से बाहर निकाल देता है।

यदि हम चाहें तो "**Could Not Read Data Properly**" Message के स्थान पर "C" Compiler द्वारा दिया जाने वाला Error Message भी Print करवा सकते हैं। System द्वारा दिया जाने वाला Error Message **perror**() Function द्वारा दिया जाता है। इस Function का Syntax निम्नानुसार होता है:

```
    if(ferror())
    {
        perror("Programmer Message");
    }
```

Programmer Message में Double Quotes के बीच कुछ Message Programmer द्वारा देना जरूरी होता है। जैसे यदि हम एक ऐसी File को Open करना चाहते हैं, जो Disk पर है ही नहीं, तो हम निम्नानुसार System Error के साथ अपना Message दे सकते हैं:

Program

```
#include<stdio.h>
main()
```

```
{
    FILE *fp;

    Fp = fopen("Error.err", "r");
    if(fp==0);
    perror("File Could Not Open");
    fclose(fp);
}
```

यदि **Error.err** नाम की File Disk पर नहीं होगी तो निम्नानुसार Error Output में प्राप्त होगाः

File Could Not Open : No such file or directory

इस Error में "**No such file or directory**" Message System द्वारा दिया जा रहा है।

fseek( );

Syntax: **fseek(fp, offset, Position);**

ये Function हमें File के एक Record से दूसरे Record पर लाने व ले जाने का काम करता है। ये Function तीन Arguments लेता हैः

1. इस Function में पहले Argument के रूप में Open किए गए File का File Pointer होता है।
2. दूसरे Argument के रूप में हमें "C" Compiler को ये बताना होता है, कि File Pointer को File में कितना Byte आगे या पीछे ले जाना है। यदि Minus (-) चिन्ह का प्रयोग किया जाता है, तो File Pointer अपनी Current Position से पीछे की तरफ Move होता है। ये एक **long int** प्रकार का मान होता है। यहां जो संख्या लिख दी जाती है, File Pointer File में उतने ही Byte Move होता है।
3. तीसरे Argument के रूप में हम **SEEK_END, SEEK_CUR** या **SEEK_SET** नाम के तीन Macros में से किसी एक को आवश्यकतानुसार Use करना होता है। ये Argument File Pointer को क्रमशः File के अंत, File की वर्तमान Location या File की शुरूआत पर ले जाता है। यदि हम SEEK_END Use करते हैं, तो File Pointer File के अंत में पहुंच जाता है। यदि हम SEEK_SET करते हैं, तो File Pointer File की शुरूआत पर चला जाता है और यदि हम SEEK_CUR करते हैं, तो File Pointer अपनी वर्तमान स्थिति पर ही रहता है। ये तीनों Macros हैं और इन्हे **stdio.h** नाम की Header File में Define किया गया है। यदि हम चाहें तो SEEK_SET के स्थान पर 0, SEEK_CUR के स्थान पर 1 व SEEK_END के स्थान पर अंक 2 का भी प्रयोग कर सकते हैं।

ftell( );

Syntax: **tot_bytes = ftell (fp);**

ये Function File Pointer की Current Position बताता है। जब हम File को Binary Mode में Open करते हैं, तब ये Function कुल Use हो रही Bytes को बताता है। **fwrite**() हमेंशा वहां से File में Data लिखना शुरू करता है, जहां पर Currently File Pointer होता है। उसी प्रकार से **fread**() File में वहां से Data Reading शुरू करता है, जहां पर Currently File Pointer होता है।

यदि हम ये जानना चाहते हैं कि Currently File Pointer File में कहां पर स्थित है या File में कितने Bytes का Data लिखा जा चुका है, तो हम इस Function को Use करके इसका पता लगा सकते हैं। ये Function **long int** प्रकार का मान **return** करता है। यदि ये Function successful काम नहीं करता है, तो **-1** Return करता है।

किसी भी File में एक विशेष प्रकार का Data एक क्रम में Store किया जाता है। जैसे हम किसी File में int प्रकार के Data Store कर रहे हैं, तो File में किसी खास Data को खोजने के लिए ये जरूरी होता है, कि हम **fseek**() Function में Offset का मान **2** रखें क्योंकि File में Store होने वाला Data **int** प्रकार का है, जो कि Memory में 2 Bytes की जगह लेता है।

यदि हम ये मान 2 के बजाय 1 रखेंगे, तो File से सही प्रकार से Data Output में प्राप्त नहीं होगा। क्योंकि यदि हमने File में Data **fwrite**() Function द्वारा Input किया है, तो हर Data दो Byte के स्थान में Store होगा। ऐसे में हम यदि **offset** का मान **1** रखेंगे, तो दो Byte का Data एक Byte के रूप में प्राप्त नहीं किया जा सकेगा।

यदि हमने किसी File में किसी Record को Structure द्वारा Input किया है और उसी File से जब Data प्राप्त करना हो या किसी विशेष Record को Search करना हो, तो ये जरूरी हो जाता है कि हम **fseek**() Function में **offset** के स्थान पर उस Structure की पूरी **size** लिखें, ताकि जब **fseek**() Function मे File Pointer Move हो, तब वह हर Movement में एक Record आगे या पीछे Move हो।

यदि हम offset के स्थान पर Structure की **size** नहीं लिखते और Offset के स्थान पर माना कोई संख्या जैसे **6** लिख देते हैं, तो Output में हमें सही Record कभी प्राप्त नहीं होगा, क्योंकि हो सकता है, कि Structure से जो Data File में Write हो रहा है, उसकी Size 18 Bytes हो। ऐसे में File Pointer क्रम से 6–6 Bytes की Location पर Point करेगा और हमें हमारा वांछित परिणाम प्राप्त नहीं होगा क्योंकि हमारा Record तो 6–6 Bytes के तीन भागों में बंट चुका होगा।

किसी भी समय हमें ये पता नहीं होता है, कि **fseek**() Function का प्रयोग करते समय **offset** का मान क्या रखा जाए। इसलिए हमें हमेंशा **sizeof**() Operator के प्रयोग से **fseek**() Function को

खुद ही कुल Offset का मान तय करने देना चाहिये। ऐसा करने के लिए हमें उस Variable का नाम **sizeof**() Operator के Function मे Argument के रूप में दे देना चाहिये। जैसा कि पूर्व के प्रोग्राम में Structure प्रकार के Variable **e** को **fread**() व **fwrite**() Function में **sizeof(e)** लिख कर Use किया गया है।

अभी तक हमने जितने भी प्रोग्राम बनाए हैं, उनमें सभी Functions का अलग–अलग उपयोग किया है। अब हम एक ऐसा Program बनाते हैं, जिसमें इन सभी Function का उपयोग हो रहा है। इससे हम ये बताना चाहते हैं, कि "C" द्वारा हम एक Database को किस प्रकार से Handle कर सकते हैं।

किसी भी **Database Management System** में हमें कुछ काम जैसे नए Records को File में Add करना, किसी Record को Modify करना, कोई अनावश्यक Record Delete करना, File के Records को देखना आदि अवश्य ही करने पडते हैं। यहां कुछ Points दिये जा रहे हैं, जो इन कामों को पूरा करने में मदद करेंगे और एक प्रोग्राम द्वारा इन्हे समझाया भी गया है:

- जब भी कभी हमें किसी File में नया Record Add करना हो तो उसे Data File के अंत में Add करना चाहिये। जैसे कि हम किसी Register में नया Record हमेंशा सबसे अंत में लिखते हैं।
- Records को जब screen पर show करना हो तो प्रथम Record से अन्तिम Record तक के सभी Records Screen पर दिखाए जाने चाहियें।
- जब भी किसी Record को Modify करना हो तो पुराने Record पर नया Record Over write किया जाना चाहिये।
- जब किसी File से किसी Record को Delete करना हो, तो जिस Record को Delete करना है, उसे छोड कर शेष सभी records को एक Temporary File में store कर लेना चाहिये। फिर पुरानी File को Delete करके उस Temporary File का नाम पुरानी File के नाम से Rename कर लेना चाहिये।
- File को Reading व Writing के लिए Binary mode में ही Open करना चाहिये। ये कई प्रकार से अच्छा रहता है।
- एक File को सिर्फ एक बार ही Open करना चाहिये और एक ही बार Program के अंत में File को बंद करना चाहिये।

इन्ही सब बातों पर निर्भर करते हुए यहां एक Program बनाया गया है, जिसमें हम Records को **Add, Modify, List** व **Delete** कर सकते हैं।

Program

```
#include<stdio.h>
main()
{
```

```
FILE *fp, *ft;
char another, choice;
struct emp
{
    char name[20];
    int age;
    float salary;
};

struct emp e;
char e_name[20];
long int recsize;

fp =fopen("Emp.dat", "rb");
{
    fp = fopen("Emp.dat", "wb");
    if(fp == NULL)
    perror("Could Not Open File");
}

recsize = sizeof(e);
while(1)
{
    clrscr();
    gotoxy(30,10);
    printf("1    Add Records");
    gotoxy(30,12);
    printf("2    List Records");
    gotoxy(30,14);
    printf("3    Modify Records");
    gotoxy(30,16);
    printf("4    Delete Records");
    gotoxy(30,18);
    printf("0    Exit");
    gotoxy(30,20);
    printf("Enter Your Choice");
    fflush(stdin);
    choice = getchar();
    switch(choice)
```

```
{
    case '1':
        fseek(fp, 0, SEEK_END);
        another = 'y';

        while(another == 'y')
        {
            printf("Enter Name Age and Salary:");
            scanf("%s %d %f", e.name, &e.age, &e.salary);
            fwrite(&e, recsize, 1, fp );
            printf("Do You Want To Add Another Record: Y/N");
            another = getche();
        }
        break;

    case '2':
        rewind(fp);
        while(fread(&e, recsize, 1, fp ) == 1)
        printf("\n %s\t %d\t %f\t", e.name, e.age, e.salary);
        break;

    case '3':
        another = 'y';

        while(another == 'y')
        {
            printf("Enter Name :");
            scanf("%s ", e_name);
            rewind(fp);
            while(fread(&e, recsize, 1, fp) ==1)
            {
                if(strcmp(e.name, e_name) == 0)
                {
                    printf("\n Enter New Name, Age and Salary :");
                    scanf("%s %d %f", e.name, &e.age, &e.salary);
                    fseek(fp, -recsize, SEEK_CUR);
                    fwrite(&e, recsize, 1, fp);
                    break;
                }
```

```
            }

            printf("\n Do You Want To Modify Another Record");
            fflush(stdin);
            another = getche();
        }
        break;

    case '4':
        another = 'y';

        while(another == 'y')
        {
            printf("\n Enter name of Employee to Delete");
            scanf("%s", e_name);
            ft = fopen("TEMP.DAT", "wb")'
            rewind(fp);

            while(fread(&e, recsize, 1, fp) == 1)
            {
                if(strcmp(e.name, e_name) != 0)
                fwrite(&e, recsize, 1, ft);
            }

            fclose(fp);
            fclose(ft);
            remove("Emp.dat");
            rename("TEMP.DAT", "Emp.dat");
            fp = fopen("Emp.dat","rb+");
            printf("\n Delete Another Record : Y/N );
            fflush(stdin);
            another = getche();
        }
        break;

    case '0' :
        fclose(fp);
        exit();
}
```

```
        }
    }
```

इस Program में एक Structure के द्वारा File में Data Input किया जाता है और इसी Structure का प्रयोग करके File से Data प्राप्त भी किया जाता है। सभी Variable के Declaration व Definition के बाद सर्वप्रथम Program Control एक File **Emp.dat** को **Binary Reading Mode** मे Open करता है और इस File का Address एक Pointer **fp** को Assign करता है। फिर निम्न Statement द्वारा Structure Variable **e** की Size ज्ञात करके, उस Size को **recsize** नाम के Variable में Store करता है:

```
recsize = sizeof(e);
```

फिर Program Control को **while(1)** Statement प्राप्त होता है। जब किसी While के कोष्ठक में **0** के अलावा कोई भी मान लिखा जाता है, तो while Loop की Condition हमेंशा सत्य रहती है, जिससे इस Loop के Statement Block में लिखे Statement हमेंशा Screen पर Print रहते है।

इस कारण से इस Loop के Statement Block में लिखे सारे Statements Screen पर हमेंशा रहते हैं, जब तक कि Program Control को **exit**() Function प्राप्त नहीं हो जाता। इन Statements में से हमें जो भी काम करना होता है, उस अंक को चुन लेते हैं। जैसे हमें कोई Record Add करना हो तो अंक **1** Press करते हैं।

जब हम अंक 1 Press करते हैं, तब Program Control ये मान **getche**() Function द्वारा **choice** नाम के Variable में Store कर देता है। फिर **switch** Statement द्वारा ये check किया जाता है कि **choice** का मान switch Statement Block की किस **Case** से मेल करता है। जिस Case से मेल करता है, Program Control उस Case के Statements को Execute कर देता है।

जब हम अंक 1 Input करते हैं, तब Program Control switch के **Case 1:** के Statements को Execute करता है। यहां सबसे पहला Statement निम्न होता है, जो File pointer को 0 record के step में File के अंत से Move करता है:

```
fseek(fp, 0, SEEK_END);
```

अब यहां एक While Loop का प्रयोग किया गया है। इस Loop के द्वारा Name Age व Salary को Input किया जाता है। जब Record Input किया जाता है तब ये Input किये गए मान Structure में जाकर Store हो जाते हैं। फिर निम्न Statement द्वारा ये Data File में Write किया जाता है:

```
fwrite(&e, recsize, 1, fp );
```

यदि और Data Input करना हो, तो यहां एक Message आता है। यदि यहां **Y** Input किया जाता है तो हम वापस इसी Loop में पहुंच जाते हैं और वापस यहीं प्रक्रिया चलती है। यदि हम Y के अलावा कोई भी Key Press करते हैं, तो इस Loop से बाहर आ जाते हैं। इस Loop के बाहर आते ही Program Control को **break** मिलता है और हम **switch** Statement से बाहर वापस **Main Menu** पर आ जाते हैं।

जब हमें इस File की पूरी List देखनी होती है, तब हम दूसरा Option Choose करते हैं। जब इस प्रोग्राम का दूसरा Output Choose किया जाता है, तब **switch** Statement का दूसरे **Case 2:** के Statements Execute होते हैं, जहां निम्न Statement द्वारा File के सारे Data Screen पर Print हो जाते हैं:

```
while(fread(&e, recsize, 1, fp ) = = 1)
```

जब हम तीसरा Option Choose करते हैं, तब **switch** के तीसरे **Case 3:** के Statements Execute होते हैं। यहां भी एक **while** Loop का प्रयोग किया गया है, ताकि यदि हमें एक से अधिक Records Modify करने हों, तो किया जा सके। यहां निम्न Statement द्वारा उस व्यक्ति का नाम, जिसका Record Modify करना है, e_name नाम के Variable में लिया जाता है:

```
Enter Name :
```

फिर निम्नानुसार अगले Statement द्वारा पुनः File Pointer को File की शुरूआत पर ले जाया जाता है:

```
rewind(fp);
```

फिर निम्न Statement द्वारा क्रम से File के हर Record पर File Pointer को Move किया जाता है:

```
while(fread(&e, recsize, 1, fp) ==1)
```

जैसे ही File Pointer एक Record Move होता है, निम्न Statement द्वारा ये Check किया जाता है कि Variable **e_name** में जो नाम है, वह नाम File के उस Record से मेल करता है या नहीं जिस Record पर Currently File Pointer स्थित है:

```
if(strcmp(e.name, e_name) = = 0)
```

यदि **e_name** का String File के उस Record में नहीं मिलता, जहां पर File Pointer स्थित है, तो File Pointer अगले Record को Point करने लगता है। लेकिन यदि File pointer जिस

Record पर है, उस Record में वही नाम है, जो नाम **e_name** नाम के Variable में है, तो ये Statement **0** मान Return करता है।

यानी File के Record के नाम व **e_name** नाम के Variable में स्थित नाम की तुलना की जाती है। जब दोनों नाम एक समान होते हैं, तो **strcmp**() Function **0** Return करता है। जैसे ही ये Function **0** return करता है, **if** Condition सत्य हो जाती है और Program Control **if** Statement Block में प्रवेश करता है।

if Statement Block में आते ही Program Control को निम्न Statement प्राप्त होती है:

```
fseek(fp, -recsize, SEEK_CUR);
while(fread(&e, recsize, 1, fp) = =1);
```

इस **while** Statement से **fseek**() तब तक अगले Record पर Move करता रहता है, जब तक कि इसे Records क्रम से प्राप्त होते जाते हैं। **if** Condition Block में प्रवेश करने से पहले ही **fp** अगले Record को Point करने लगता है। इसलिए जब हमें वह Record प्राप्त हो जाता है, जिसे Modify करना है, तब तक **fseek**() Function अगले Record को Point करने लगता है। इसलिए ये जरूरी हो जाता है, कि File Pointer को एक Record पीछे सरकाया जाए। निम्न Statement द्वारा हम ये काम करते हैं, यानी एक Record पीछे उस Record पर आ जाते हैं, जिसे Modify करना है।

```
fseek(fp, -recsize, SEEK_CUR);
```

अब निम्न Statement द्वारा हम जो Data Input करते हैं, वो Data पुराने Data पर Over Write हो जाता है:

```
fwrite(&e, recsize, 1, fp);
```

अब वापस ये Message आता है कि क्या और Records Modify करने हैं या नहीं। यदि हम **Y** Press करते हैं, तो वापस यही प्रक्रिया होती है और पुराना Data नए Data से Over Write हो जाता है लेकिन यदि हम **Y** Press नहीं करते हैं, तो इस **while** Loop से बाहर आ जाते हैं। बाहर आते ही Program Control को **break** मिलता है और Program Control वापस **Main Menu** पर आ जाता है।

जब हमें किसी Employee का Record File से Delete करना होता है, तब हम अंक 4 Press करके **switch** के **Case '4':** के Statements को Execute करते हैं। जब चौथा Option Choose किया जाता है, तब हमें उस Employee का नाम Input करना होता है, जिसका Record Delete करना है। ये नाम **e_name** नाम के Variable में Store हो जाता है।

अब एक और नई File Open करते हैं, जिसका File Pointer **ft** है और **rewind**() Function द्वारा वापस से **fp** File Pointer को **Emp.dat** नाम की File के प्रारम्भ में ले जाया जाता है। निम्न Statement द्वारा वापस क्रम से File Pointer को हर Record पर Move किया जाता है:

```
while(fread(&e, recsize, 1, fp) == 1)
```

और वापस से **e_name** में Stored String को File में निम्न Statement द्वारा खोजा जाता है:

```
if(strcmp(e.name, e_name) != 0)
```

जैसे ही **e_name** नाम के Variable में Stored String File में स्थित किसी Record से मेल करती है, उस मेल करने वाले Record को छोड कर हर Record को क्रम से निम्न Statement द्वारा Open की गई दूसरी File जिसका File Pointer ft है, में Copy कर दिया जाता है:

```
fwrite(&e, recsize, 1, ft);
```

अब दोनो Files को Close करते समय निम्न Statement द्वारा **Emp.dat** नाम की हमारी पहली File को Delete कर दिया जाता है:

```
remove("Emp.dat");
```

और निम्न Statement द्वारा उस File को, जिसे हमने बाद में Open किया था और Delete करने वाले Record को छोड कर शेष Data को जिस File में Copy कर लिया था, उसे **Emp.dat** नाम से Rename कर लेते हैं:

```
rename("TEMP.DAT", "Emp.dat");
```

अब इस File में Deleted Data के अलावा सभी Data हैं, यानी जो Data हमें Delete करना था, वह Delete हो चुका है। निम्न Statement द्वारा वापस से **Emp.dat** नाम की File को Open कर लिया जाता है:

```
fp = fopen("Emp.dat","rb+");
```

यदि हमें और Records Delete करने हों तो इस चौथे Case के साथ भी एक while Loop प्रयोग किया गया है, जिसमें ये Message आता है कि क्या आप और Records Delete करना चाहते हैं। यदि यहां **Y** Press करें तो वापस पूरी की पूरी प्रक्रिया दोहराई जाती है और यदि हम और Record Delete करना नहीं चाहते तो यहां y के अलावा कुछ भी Press कर देते हैं।

इस Case से बाहर आते ही हमें वापस **break** मिलता है, जिससे हम वापस **Main Menu** में पहुंच जाते हैं। यदि हमें इस Program से बाहर आना हो तो हम **0** press करते है। 0 Press करते ही **switch** के पांचवे **Case 0:** के Statements Execute होते हैं जहां Program Control को **exit**() Function मिलता है और Program Control Program से बाहर आ जाता है।

Command Line Argument

वे Arguments जो Commands के साथ DOS Prompt पर देकर कोई काम किया जाता है, **Command Line Arguments** कहलाते हैं। जैसे जब हम किसी File को Command Prompt पर किसी File को Rename करते हैं, तो हमें **Rename** Command के साथ Source File व Target File का नाम देना होता है। ये Command व File Names **Command Line Arguments** कहलाते हैं।

हम कोई भी File जब "C" में बना कर उसे Compile कर लेते हैं, तो "C" उस File की उसी नाम से एक Executable File बना देता है। इस Executable File को Command Prompt से सीधे ही Execute किया जा सकता है। हम "C" में भी ऐसे Program बना सकते हैं, जिनमें Command Line Argument Accept करके उससे सम्बंधित काम किया जा सकता है। इसे समझने के लिए हम एक File Copy Program बनाते हैं।

Program

```
#include<stdio.h>

main()
{
    FILE *fs, *ft;
    char ch;

    fs = fopen("Source.c", "r");
    if( fs == NULL )
    {
        perror("\n Could Not Open File");
    }

    ft = fopen("Target", "w");
    if( ft == NULL )
    {
        perror("\n Could Not Open File");
        fclose( fs );
```

```
        }

        while(1)
        {
            ch = fgetc( fs );
            if(ch == EOF)
            break;
            else
            fputc(ch, ft);
        }

        fclose(fs);
        fclose(ft);
        printf("\n File Copied Successfully"
    }
```

ये प्रोग्राम यदि Disk पर Source.c नाम की File होगी तो उसे Target.c नाम से Copy कर देगा। लेकिन चूंकि हमने दोनों Files का नाम Program के अंदर ही लिख दिया है, इसलिए हम इस Program को Dos Prompt पर जितनी भी बार Execute करेंगे, यही Source.c नाम की File Target.c नाम की File के रूप में Copy होगी। लेकिन हम चाहते हैं कि ये Program उस Files को Copy करे जिसे हम Argument के रूप में Command prompt पर दें।

इस प्रकार की जरूरत के लिए "C" में दो Library Variables हैं, जो main() Function को Argument Pass करने की क्षमता रखते हैं। यानी जिस प्रकार से हम किसी User Defined Function को Arguments Pass करते हैं, वैसे ही हम **argv[]** व **argc** नाम के दो Library Variables का प्रयोग करके Command Prompt से **main**() Function को Argument Pass कर सकते हैं।

argc ये एक Argument Counter है। इसे हमेंशा **int** प्रकार का Declare किया जाता है। ये Variable इस बात का ध्यान रखता है कि Command Prompt से कितने Arguments **main**() Function को प्राप्त हो रहे हैं।

argv[ ] ये एक char प्रकार का Pointer Array है जिसकी Size **argc** के मान के बराबर होती है। ये एक प्रकार का "**Pointers Of Array To String**" होता है।

अब हम अभी बताए गए File Copy Program को ही इस प्रकार का लिख रहे हैं जिससे Command Prompt पर ही मनचाही File का नाम Enter करके उसे किसी और नाम से दूसरी File में Save कर सकते हैं:

Program

```
#include<stdio.h>
main( int argc, char *argv[])
{
    FILE *fs, *ft;
    char ch;
    if( argc != 3 )
    {
        printf("Arguments Mismatch")
        exit();
    }

    fs = fopen("argv[1]", "r");
    if( fs == NULL )
    {
        perror("Could Not Open File");
    }
    ft = fopen("argv[2]", "w");

    if( ft == NULL )
    {
        perror("Could Not Open File");
        fclose( fs );
    }

    while(1)
    {
        ch = fgetc( fs );
        if(ch == EOF)
            break;
        else
            fputc(ch, ft);
    }
    fclose(fs);
    fclose(ft);
    printf("File Copied Successfully");
}
```

इस File को **Filecopy.c** नाम से Save करके Compile करें व इस File को Execute करने के लिए DOS Prompt पर जाएं। वहां पर इस File को ठीक वैसे ही Use करें जिस प्रकार से हम Dos के Copy Command को किसी File को Copy करने के लिए करते हैं। माना हमें a.c नाम की file को b.c नाम की File के नाम से copy करना है तो हम निम्नानुसार Command Line पर Argument देंगेः

```
c:\tc\bin>filecopy a.c b.c        Press Enter
File Copied Successfully
```

अब Dir command से check करें। A.c नाम की File B.c नाम से Copy हो चुकी है। आइये अब समझते है कि ये Program किस प्रकार काम कर रहा हैं। हमने main() Function के कोष्ठक में **int argc, char *argv[]** Statement दिया है। **int argc** कुल Input किये जाने वाले Arguments को Count करता है।

Dos Prompt पर किसी Argument के बीच Space मान्य नहीं है, इसलिए जैसे ही Command Line पर Space दिया जाता है, तो इस **argc** का मान एक अंक बढ जाता है। जब हम कोई File Name लिखते हैं तो वह नाम Memory में एक One-Dimensional Array के रूप में Store हो जाता है और इस Array का Base Address ***argv** में **argc[0]** यानी Array **argc** के Index Number **0** पर जाकर Store हो जाता है। यह एक बिना Define की गई Size का Array है, जिसमें उतने ही One-Dimensional Array के Base Address, Elements के रूप में Store होते हैं, जितने Argument के रूप में Dos Prompt पर दिये जाते हैं।

जब हम कोई File Name Dos Prompt पर लिख कर Space देते हैं, तो वह File Name जिस One Dimensional Array में जाकर Store हो रहा होता है, वह One-Dimensional Array Terminate हो जाता है। क्योंकि Space के मिलते ही यह Array उस Space को Null Character में Terminate कर देता है, जिससे String का अंत हो जाता है।

इस Space के मिलते ही **int argc** का मान बढ कर **1** हो जाता है। अब Space के बाद जब हम दूसरा नाम लिखते हैं, तो ये नाम भी Memory में किसी Location पर One Dimensional Array के रूप में Store हो जाता है और इस One Dimensional Array का Base Address भी Pointer Array **argv** के Index Number **1** पर जा कर Store हो जाता है।

जब हम यहां पर नाम Input करने के बाद वापस Space Press करते हैं, तो वापस **int argc** का मान बढ जाता है। ये क्रम तब तक चलता रहता है जब तक कि हम Command Prompt पर Arguments देते रहते हैं।

यहां हमने ये बताने की कोशिश की है, कि Dos Prompt से किस प्रकार से main() Function को Arguments Pass होते हैं। एक बात हमेंशा ध्यान रखें कि जब हमें main() Function को कोई

Argument Pass करना होता है, तो प्रथम Argument के रूप में हमें हमेंशा उस File का नाम लिखना होता है, जिसे Use करना है।

argv[0] में हमेंशा Executable File का नाम Argument के रूप में **main**() Function को प्राप्त होना चाहिये। इसलिए इस Filecopy Program को उपयोग में लेते समय सबसे पहले Argument के रूप में इस EXECUTABLE File यानी **filecopy** लिखना होगा। यह नाम हमेंशा उस Program को Execute कर देगा जिससे हम किसी File को Copy कर सकेंगे। अब हम ये देखते हैं कि ये File Copy Program किस प्रकार Execute होता है और एक File को दूसरी File में Copy करता है।

सर्वप्रथम **main**() Function के कोष्ठक मे **int argc, char *argv[]** लिखा गया है, ताकि DOS Prompt से Arguments प्राप्त किये जा सकें। फिर दो File Pointer लिए गए हैं और एक char प्रकार का Variable **ch** Declare किया गया है।

हमें Command Prompt से कुल तीन Argument Accept करने हैं, पहला इस Executable File का नाम, दूसरा उस File का नाम जिसका Matter Copy करना है और तीसरा उस File का नाम जिसमें Matter को Copy करना है। इसलिए यहां एक **if** Condition दी गई है, कि यदि Command Prompt से आने वाले Arguments की संख्या तीन से कम या अधिक हो यानी तीन के बराबर ना हो, तो एक Error Message "**Arguments Mismatch**" Screen पर Print हो जाए व Program Control Program को Terminate कर दे।

फिर हमने निम्न Statement द्वारा Command Prompt से प्राप्त दूसरे Argument के नाम की File को Reading Mode में Open किया है:

```
fs = fopen("argv[1]", "r");
```

क्योंकि Pointer Array के Index Number **1** पर उस File के नाम के One-Dimensional Array का Address है, जिसे Copy करना है। इस Statement से Copy की जाने वाली File Open हो जाती है। यदि जो File Name हमने Copy करने के लिए दिया है, वो File Disk पर मौजूद नहीं है, तो एक Error Message आता है और Program Terminate हो जाता है।

इसी प्रकार एक और File Open की है और इस File को Writing Mode में Open किया है। यह वह File है, जिसमें पहले वाली File का Matter Copy करना है। यहां हमने निम्न Statement दिया है:

```
ft = fopen("argv[2]", "w");
```

argv[2] में उस One-Dimensional Array के नाम का Address है, जिस नाम से हमें File को Copy करना है। यदि ये File Open नहीं हो पाती है, तो भी एक Error Message आता है व Program Terminate हो जाता है।

अब **while(1)** Statement दिया है, जिससे ये Loop तब तक सत्य रहेगा जब तक कि किसी अन्य तरीके से Loop को Terminate ना कर दिया जाए। ये Loop तब Terminate होता है, जब निम्न Statement मिलता हैः

```
if(ch = = EOF)
```

EOF के मिलते ही Program Control को **break;** मिलता है और Program Terminate हो जाता है। बाकी का Program सामान्य File Copy Program जैसा ही है। यानी निम्न Statement द्वारा One By One Character ch में Store होता हैः

```
ch = fgetc( fs );
```

और निम्न Statement द्वारा Target File में Characters को One by One तब तक लिखता रहता है जब तक कि EOF प्राप्त नहीं हो जाताः

```
fputc(ch, ft);
```

इस प्रकार से हम "C" में कई ऐसे Program लिख सकते हैं, जिन्हें Command Prompt से Execute करके Use किया जा सकता है।

Low Level Disk I/O

अभी तक हमने जो भी पढा वो High Level Disk I/O के बारे में था। हमने जो Filecopy Program बनाया है, उस Program से केवल Text File ही Copy की जा सकती है। **.Exe** या **.Com** Files इस Program से Copy नहीं की जा सकती हैं।

जिस प्रकार से हम High Level Disk I/O में Character By Character Data को File पर Read या Write करते हैं, उस प्रकार से हम Low Level Disk I/O में नही कर सकते हैं। यदि हम ऐसा करते हैं, तो काम करने की गति बहुत ही कम हो जाती हैं। इसलिए Low Level Disk I/O के लिए हमें एक विशेष प्रकार के तरीके का प्रयोग करना पडता है।

इस तरीके में सबसे पहले एक निश्चित मात्रा की Memory Assign की जाती है। इस Assign की गई Memory को **buffer** कहते हैं। हम Low level Disk I/O के लिए अपना Data इस buffer में रखते हैं, जब तक कि buffer Full ना हो जाए। जब Buffer Full हो जाती है, तब इस buffer से

Data को एक साथ Disk पर Write कर दिया जाता है। High level Disk I/O में ये buffer पहले से ही होता है, जो कि Invisible रहता है, जबकि Low level Disk I/O में हमें ये buffer Declare करना पडता है।

DOS में buffer की Optimal Size **512** characters की होती है। हम इसे अपनी सुविधानुसार कम या ज्यादा कर सकते हैं। लेकिन एक हद से अधिक करने पर **Stack Over Flow** की Problem आती है। इसलिए buffer Size Default रूप में **512** रखना ही ठीक रहता है। अभी हमने जो Filecopy Program बनाया है, उसी Program को **Low Level Disk I/O** के लिए File Copy प्रोग्राम में बदला जा रहा है। साथ ही समझाया जा रहा है कि किस प्रकार से Low Level Disk I/O के साथ काम किया जाता है।

Program

```
#include <fcntl.h>
#include <types.h>
#include <stat.h>
main( int argc, char *argv[])
{
    char buffer[512];
    int input, output, bytes;
    input = open(argv[1], O_RDONLY | O_BINARY);

    if( input == -1 )
    {
        perror("Could Not Open File");
    }
    output=open(argv[2],O_CREAT | O_BINARY | O_WRONLY | S_IWRITE);
    if( output == -1)
    {
        perror("Could Not Open File");
        close( input );
    }

    while(1)
    {
        bytes = read(input, buffer, 512 );
        if( bytes > 0 )
            write ( output, buffer, 512);
        else
        break;
```

```
        }
        close(input);
        close(output);
        printf("File Copied Successfully");
    }
```

इस प्रोग्राम में हमने सर्वप्रथम **char buffer[512];** का Declaration किया है। ये एक Array है, जो Low Level Disk I/O के लिए **buffer** का काम करता है। जो भी Data Disk से प्राप्त होता है, वो Data यहीं आ कर Store होता है। उसके बाद उस Data की Processing होती है।

Low Level Disk I/O में भी सारे काम उसी प्रकार से करने होते हैं, जिस प्रकार से High Level Disk I/O में किया जाता है। यानी सर्वप्रथम File Open की जाती है। Low level Disk I/O में File Open करने के लिए **fopen**() Function के बजाय केवल **open**() Function का प्रयोग किया जाता है। ये Function fopen() Function से बिल्कुल अलग है।

open() Function में Mode के रूप **O_FLAGS** का प्रयोग किया जाता है। ये O_FLAGES **fcntl.h** नाम की Header File में Define किये गए हैं। इसलिए Low Level Disk I/O के लिए इस Header File को Program में Include करना जरूरी होता है। जब दो या दो से अधिक Flags को Use करना होता है तब इनके बीच **Bitwise OR** (|) Operator का प्रयोग करना जरूरी होता है। इस Header File में निम्नानुसार Mode Flags होते हैं:

O_APPEND	एक File में नए Data Add करने के लिए Open करता है।
O_CREAT	एक नई File Create करता है। यदि File पहले से हो तो भी कोई फर्क नहीं पडता है।
O_RDONLY	Reading के लिए एक नई File बनाता है।
O_RDWR	Reading व Writing दोनों के लिए एक File Create करता है।
O_WONLY	Writing के लिए एक File Create करता है।
O_BINARY	Binary Mode मे एक File को Open करता है।
O_TEXT	Text Mode में File Create करता है।

निम्न Statement द्वारा एक File Open किया जाता है, जिसका नाम Command Line Argument के दूसरे Argument से प्राप्त होता है:

```
input = open(argv[1], O_RDONLY | O_BINARY);
```

इस Statement में Flage **O_RDONLY** से File Reading Mode में Open होगी और **O_BINARY** Flage के कारण ये File Binary Mode में Open होती है। जिस प्रकार से High Level Disk I/O में जब किसी File को Open किया जाता है, तो उस File का Address एक File Pointer में Return होता है।

उसी प्रकार से जब Low Level Disk I/O में कोई File Open की जाती है, तब एक Integer Value Return होती है। जब कोई File Low level Disk I/O में Create किया जाता है, तो Program Control हर File को एक Integer संख्या दे देता है। इस संख्या को **File Handle** कहते हैं।

जब कोई File ठीक से Open नही हो पाती है, तब **open**() Function File Handle के रूप में संख्या **–1** Return करता है, जो कि ये बताता है कि File ठीक से Open नहीं हो सकी है। इसके बाद निम्न Statement द्वारा दूसरी File Open करते हैं:

```
output = open(argv[2], O_CREAT | O_BINARY | O_WRONLY | S_IWRITE);
```

ये Target File है। **O_CREAT** Flag द्वारा इस Target File को Create किया गया है। हम इस File में Data Write करना चाहते हैं, इसलिए इस File को **O_WRONLY** Flage द्वारा केवल Writing Mode में Open किया गया है। **O_BINARY** Flage द्वारा File को Binary Mode मे Open किया गया है।

जब भी हम **O_CREAT** Flage का प्रयोग करते हैं, तो हमें इस Flage के साथ एक और Flage का प्रयोग करना पडता है, जो "C" Compiler को ये बताता है, कि जो File Create की गई है, उस File के साथ कैसा काम करना है।

यानी इस File को Read करना है या इस File में Data Write करना है। इसे **Permission Argument** कहते हैं और इन Permission Argument को Use करने के लिए हमें **stat.h** व **ctype.h** नाम की Header File को हमारे Program में Include करना जरूरी होता है। Permission Argument निम्न में से कोई एक होता है:

S_IWRITE जब हम File में कुछ लिखना चाहते हैं।
S_IREAD जब हम File से केवल Data पढना चाहते हैं।

आइये अब देखते हैं कि एक Low Level Disk I/O में File से Data कैसे Read किया जाता है। Low Level Disk I/O में Data Read करने के लिए हमने निम्न Statement प्रयोग किया है:

```
bytes = read(input, buffer, 512 );
```

read() Function Low Level Disk I/O में Data Reading का काम करता है। इसे Argument के रूप में **input** में Source File की संख्या, Buffer का नाम जिसमें Data Read करके Store करने हैं और Buffer की Size देनी होती है।

यहां हमने Source File के रूप में जो File Open की है, उस File का **File Handle** यानी **open**() Function ने उस File को जो संख्या प्रदान की है, उस संख्या को हमने **int** प्रकार के Variable **input** में Store किया है। इसलिए प्रथम Argument के रूप में **read**() Function में **input** लिखा है।

हमने File से Data Read करके Memory में Store करने के लिए **buffer** नाम का एक One-Dimensional Array लिया है। जो भी Data Read किया जाता है, वो Data पहले इसी Buffer में आकर Store होता है। इसलिए दूसरे Argument के रूप में हमने **read**() Function में **buffer** लिखा है।

तीसरा Argument Buffer में Store होने वाले कुल Characters की संख्या पूंछता है, कि buffer में अधिकतम कितने Characters Store किये जा सकते हैं। हमने char प्रकार के One-Dimensional Array की Size **512** रखी है, इसलिए **read**() Function में तीसरे Argument के रूप में हमने buffer की Size 512 लिखी है।

यानी **read** होने वाली File से पहले **512** Characters इस **buffer** नाम के Array में Store होंगे। फिर जब ये buffer पूरी तरह से Characters से भर जाएगा, तब इस buffer में Stored सारे Characters को Disk पर दूसरी File में Write कर दिया जाएगा।

प्रथम File से Data Read करने के बाद buffer में Stored Data को दूसरी File में Write करना है। Low Level Disk I/O में किसी File में Buffer में Stored Data को Write करने के लिए हमें **write**() Function को Use करना होता है।

यहां हमने निम्न Statement द्वारा **buffer** में Stored Characters को, जिसमें अधिकतम 512 Characters हो सकते हैं, उस File में Disk पर Write कर दिया है, जिस File का File **Handle** अंक **output** नाम के **int** प्रकार के Variable में स्थित है:

```
write ( output, buffer, 512);
```

यानी हमने Copy होने वाले Matter को जिस File में Store करने के लिए एक दूसरी File, जिसे हमने Writing Mode में Open किया था और जिसका **File Handle** या **File** संख्या हमने **output** नाम के **int** प्रकार के Variable में Store किया था, उस File में **read**() Function से लिया गया Data, जो कि buffer में Store है, **write** कर दिया गया है। यहां **write**() Function में तीसरे Argument के रूप में buffer में अधिकतम कितने Characters हो सकते हैं, इसकी संख्या बताई गई है।

इस प्रकार से **read** व **write** दोनो ही Function के Argument एक जैसे ही हैं। केवल उन **File Handle** को ध्यान रखना होता है कि किस File Handle से Data को Read करना है और किस File Handle मे Data को Write करना है।

Read Function से जब Data Read किये जाते हैं, तब उस Data के मान को **byte** नाम के Variable को दिया जाता है। फिर इस byte को Check किया जाता है कि Bytes की मात्रा **0** से अधिक है या नहीं। यदि File में कोई Data Read करने को नहीं बचता है, तो bytes नाम के Variable में कोई Byte नहीं रहती जिससे **if(bytes>0**) Condition असत्य हो जाती है, और Program Control while Loop से बाहर आ जाता है।

साधारण रूप से हम Program मे Input Keyboard द्वारा करते हैं और Output Screen पर प्राप्त करते हैं। लेकिन DOS में एक Special Operator है, जिससे हम किसी File में सीधे ही Data Write कर सकते हैं, और किसी File के Data को सीधे ही Screen पर या Printer पर Print कर सकते हैं। DOS में इस Operator को **Indirection** कहते हैं। हम एक Program बनाते हैं, जिससे Key Board से Data Read किया जाएगा और सीधे ही किसी File में Store किया जा सकेगा। ये Program निम्नानुसार है:

```
#include<stdio.h>
    main()
    {
        char ch;
        while((ch = getc(stdin))!= EOF)
            putc(ch, stdout);
    }
```

इस Program को जब Execute किया जाएगा तब हम Keyboard से Data Receive करेंगे और वह Data Screen पर Print होगा। इसे Program को हम **directfil.c** नाम से Save करके Compile करते हैं तो हमें **directfil.Exe** नाम की Executable File प्राप्त होती है।

हमने इस Program में कोई File Open नहीं की है, लेकिन DOS के Indirection Operator द्वारा हम इस Program से सीधे ही File में Data Write कर सकते हैं। आइये देखते हैं कैसे ? DOS Prompt पर निम्न Command दीजिए:

```
C:\tc\bin>directfil > text.txt          Press Enter
```

इस Command से **text.txt** नाम की एक File बनेगी और हम जो भी Data लिखेंगे वो Screen पर भी दिखेगा और text.txt नाम की File में भी Store होता जाएगा जब तक कि हम **F6** Function Key या **^Z** Key Combination Press करके File का अंत नहीं कर देते। > व < के चिन्ह ही DOS में Indirection Operation का काम करते है।

यदि हम देखना चाहें कि हमने जो Matter Type किया था, वो File में Store हुआ या नहीं, तो Type Command द्वारा देख सकते हैं। यदि हम चाहें कि हम जो भी Matter Type करें वो सीधे ही Printer पर Print हो जाए, तो हमें निम्न प्रकार से Direction को Use करना होगाः

```
C:\tc\bin>directfil > PRN          Press Enter
```

इसी प्रकार से हम < चिन्ह का प्रयोग करके किसी File के Data को Screen पर Print कर सकते हैं या सीधे ही Printer पर Print निकाल सकते हैं। जैसे हमने text.txt नाम की जिस File में अभी Matter Input किया है उसी File के Matter को वापस Screen पर देखना हो तो हमें निम्न प्रकार से Direction को Use करना होगाः

```
C:\tc\bin>directfil < text.txt          Press Enter
```

यदि हम चाहें तो दोनों ही Direction को एक साथ प्रयोग किया जा सकता है। जैसे कि हम text.txt नाम की File का Matter Screen पर Print करने की बजाए एक अन्य File **data.txt** में Store करना चाहें तो हम निम्नानुसार Directions को use कर सकते हैं–

```
C:\tc\bin>directfil < text.txt > data.txt       Press Enter
```

ये Command एक तरह से File Copy का काम करता है। जब हम कई Directions का प्रयोग एक साथ कर रहे हों तो कभी भी दोनो File का नाम समान नहीं देना चाहिये, क्योंकि Output File पहले Erase हो जाती है, फिर उस पर Data Write होता है। ऐसे में दोनो Files मे कोई Matter नहीं रह जाएगा। यदि हम चाहें तो किसी File को Screen पर बिना दिखाए निम्न Command द्वारा सीधे ही Printer पर Print कर सकते हैंः

```
C:\tc\bin>directfil < text.txt > PRN       Press Enter
```

हम एक File के Data मे दूसरी File के Data को Directly जोड सकते हैं, इसके लिए हमें | Pipe Operator का प्रयोग करना पडता है।

Exercise:

1 Programming में File Management के Concept को समझाईए।

2 File Open करने का तात्पर्य समझाईए। File Opening Modes की आवश्यकता व कार्यप्रणाली पर प्रकाश डालिए।

3 एक Program बनाईए, जिसमें Message.txt नाम की एक File Create कीजिए। इस Program में **getc**() व **putc**() Functions का प्रयोग करते हुए Message.txt नाम की इस Create होने वाली File में "**Hello World**" Message को Write कीजिए और File में लिखे गए Contents को फिर से Screen पर Display कीजिए।

4 किसी Student की Mark Sheet के विभिन्न Subjects के Marks को Marks.dat नाम की एक File में Store करने का Program बनाईए। Marks को File में Store करने के लिए getw() Function को Use कीजिए तथा File में Stored Marks को Screen पर Display करने के लिए putw() Function का प्रयोग करते हुए File में Stored Student की Mark Sheet के विभिन्न Subjects के Marks को Screen पर Display कीजिए।

5 **fgets**() Function व **fputs**() Function को एक उचित Program बनाते हुए, समझाईए।

6 **rewind()** Function का प्रयोग क्यों किया जाता है ? एक उचित उदाहरण Program द्वारा समझाईए।

7 **fscanf**() Function व **fprintf**() Function को एक उचित Program द्वारा समझाईए।

8 **fseek**() Function व **ftell**() Function को एक उदाहरण Program द्वारा समझाईए।

9 Command Line Arguments किसे कहते हैं? एक Program बनाइए जो Command Line पर Argument के रूप में एक नाम ले और Screen पर उस नाम वाले व्यक्ति को "**Hello"** कहें

10 Low-Level Disk I/O से आप क्या समझते हैं ? High-Level I/O व Low-Level I/O के अन्तर को स्पष्ट कीजिए। साथ ही Low-Level I/O के File Opening Mode Flags का वर्णन कीजिए।

Last but not Least. There is more…

इस पुस्तक में हमने न केवल "C" Language को Best तरीके से Describe करने की कोशिश की है, बल्कि हमने ये मानते हुए ये पुस्तक लिखी है कि आप Computer Programming से पूरी तरह से अनभिज्ञ हैं और इसीलिए हमने किसी भी Program के एक–एक Step को Describe करते हुए Progeram के Flow को समझाने की कोशिश की है, ताकि आप न केवल "C" Progreamming Language को ठीक तरीके से समझ सकें, बल्कि आप Computer Programming के मूल Concepts को भी बेहतर तरीके से समझ सकें।

"C" Language अपने आप में इतना बडा व Versatile Subject है कि "C" Language के विकास के बाद जितनी भी Programming Languages को विकसित किया गया है, वे सभी लगभग "C" Language के Syntax पर ही आधारित हैं।

इसलिए यदि आप "C" Language को ठीक से समझते हैं, तो **C++, Java, C#, PHP, JSP, JavaScript** आदि जैसी विभिन्न Programming व Scripting Languaghe बडी ही आसानी से सीख सकते हैं। उम्मीद है, इस पुस्तक ने आपके Development व Programming के ज्ञान को जरूर बढाया होगा।