Unit 4
Programming with C – Introduction

Introduction to Problem Solving:

Problem solving is the process of transforming the description of a problem into a solution by using our knowledge of the problem domain and by relying on our ability to select and use appropriate problem-solving strategies, techniques and tools. Computers can be used to help us solving problems.

- To understand exactly:
  - what the problem is
  - what is needed to solve it
  - what the solution should provide
  - if there are constraints and special conditions.

Algortihm: An algorithm is a step-by-step procedure for calculations or particular task. Algorithms are used for calculation, data processing, and automated reasoning. An algorithm is a sequence of a finite number of steps arranged in a specific logical order which, when executed, produces the solution for a problem.

- An algorithm must satisfy these requirements:
  - It may have an **input(s)**
  - It must have an **output**
  - It should not be **ambiguous** (there should not be different interpretations to it)
    - Every step in algorithm must be clear as what it is supposed to do

Pseudocode: A pseudocode is a semiformal, English-like language with limited vocabulary that can be used to design and describe algorithms.

- Criteria of a good pseudocode:
  - Easy to understand, precise and clear
  - Gives the correct solution in all cases
  - Eventually ends

Difference between Algorithm and Pseudocode:

An algorithm is simply a solution to a problem. An algorithm presents the solution to a problem as a well-defined set of steps or instructions. Pseudo-code is a general way of describing an algorithm. Pseudo-code does not use the syntax of a specific programming language, therefore cannot be executed on a computer. But it closely resembles the structure of a programming language and contains roughly the same level of detail.

Pseudocodes: The Sequence control structure:

- A series of steps or statements that are executed in the order they are written in an algorithm.
- The beginning and end of a block of statements can be optionally marked with the keywords *begin* and *end*.
- Example 1:

Begin
    Read the birth date from the user.
    Calculate the difference between the birth date and today's date.

```
        Print the user age.
    End
```

Pseudocodes: The Selection control structure:

- Defines two courses of action depending on the outcome of a condition. A condition is an expression that is, when computed, evaluated to either true or false.
- The keyword used are *if* and *else*.
- Format:

```
if condition
        then-part
else
        else-part
end_if
```

Example 2:

```
if age is greater than 55
        print “Pencen”
else
        print “Kerja lagi”
end_if
```

Pseudocodes: The Selection control structure:

- Sometimes in certain situation, we may omit the else-part.

```
if number is odd number
        print “This is an odd number”
end_if
```

- Nested selection structure: basic selection structure that contains other if/else structure in its then-part or else-part.

```
if number is equal to 1
        print “One”
else if number is equal to 2
        print “Two”
else if number is equal to 3
        print “Three”
else
        print “Other”
end_if
```

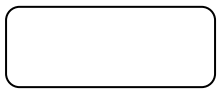
Pseudocodes: The Repetition control structure:

- Specifies a block of one or more statements that are repeatedly executed until a condition is satisfied.
- The keyword used is *while*.
- Format:

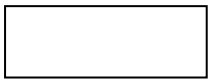
```
while condition
        loop-body
end_while
```

Flowcharts: Flowcharts is a graph used to depict or show a step by step solution using **symbols** which represent a task. The symbols used consist of geometrical shapes that are connected by **flow lines**. It is an alternative to pseudocoding; whereas a pseudocode description is verbal, a flowchart is graphical in nature.

Flowchart Symbols:

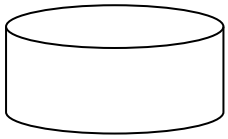


Terminal symbol - indicates the beginning and end points of an algorithm.



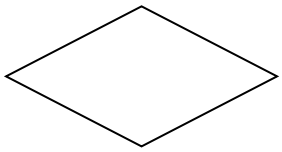
Process symbol - shows an instruction other than input, output or selection.

Input-output symbol - shows an input or an output operation.



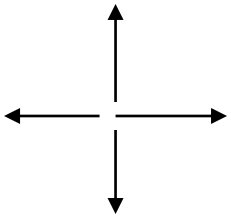
Disk storage I/O symbol - indicates input from or output to disk storage.

Printer output symbol - shows hardcopy printer output.



Selection symbol - shows a selection process for two-way selection.

On-page connector - provides continuation of logical path at another point in the same page.



Flow lines - indicate the logical sequence of execution steps in the algorithm.

C Introduction: C is a programming language for a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.

In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

The C has now become a widely used professional language for various reasons.
- Easy to learn
- Structured language
- It produces efficient programs.
- It can handle low-level activities.
- It can be compiled on a variety of computer platforms.

Facts about C
- C was invented to write an operating system called UNIX.

 C is a successor of B language, which was introduced around 1970.
 The language was formalized in 1988 by the American National Standard Institute. (ANSI).
 The UNIX OS was totally written in C by 1973.

Why to use C?
C was initially used for system development work, in particular the programs that make up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:
 Operating Systems
 Language Compilers
 Assemblers
 Text Editors
 Print Spoolers
 Network Drivers
 Modern Programs
 Databases
 Language Interpreters
 Utilities

C Program Structure:
A C program basically consists of the following parts:
 Preprocessor Commands
 Functions
 Variables
 Statements & Expressions
 Comments

Let us look at a simple code that would print the words "Hello World":

```
 #include <stdio.h>
 int main()
{
 /* my first program in C */
 printf("Hello, World! \n");
  return 0;
}
```

The first line of the program #include <stdio.h> is a preprocessor command, which tells
a C compiler to include stdio.h file before going to actual compilation.
2. The next line int main() is the main function where program execution begins.
3. The next line /*...*/ will be ignored by the compiler and it has been put to add additional
comments in the program. So such lines are called comments in the program.
4. The next line printf(...) is another function available in C which causes the message
"Hello, World!" to be displayed on the screen.
5. The next line return 0; terminates main()function and returns the value 0.

Compile & Execute C Program
Let’s look at how to save the source code in a file, and how to compile and run it. Following are the simple steps:
1. Open a text editor (vi editor in UNIX/SUN Solaris) and add the above-mentioned code.
2. Save the file as hello.c
3. Open a command prompt and go to the directory where you saved the file.
4. Type gcc hello.c and press enter to compile your code.

5. If there are no errors in your code, the command prompt will take you to the next line and would generate a.out executable file.
6. Type a.out to execute your program.
7. You will be able to see "Hello World" printed on the screen.

C Data Types:
Data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows:

S.N.	Types and Description
1	**Basic Types:** They are arithmetic types and consists of the two types: (a) integer types and (b) floating-point types.
2	**Enumerated types:** They are again arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
3	**The type void:** The type specifier *void* indicates that no value is available.
4	**Derived types:** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

Integer Types

Following table gives you details about standard integer types with its storage sizes and value ranges:

Type	Storage size	Value range
Char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
Int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
Short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
Long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating-Point Types

Following table gives you details about standard floating-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision

Float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
Double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The void Type

The void type specifies that no value is available. It is used in three kinds of situations:

S.N.	Types and Description
1	**Function returns as void** There are various functions in C which do not return value or you can say they return void. A function with no return value has the return type as void. For example **void exit (int status);**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept as a void. For example, **int rand(void);**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

C Variables:

Variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable. The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive. Based on the basic types explained in previous chapter, there will be the following basic variable types:

Defining Constants:

There are two simple ways in C to define constants:

1. Using #define preprocessor. #define identifier value
2. Using const keyword. const type variable = value;

C Operators:

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators

Operator	Description	Example

+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A-- will give 9

Relational Operators

Following table shows all the relational operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20, then:

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then:

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0

0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; now in binary format they will be as follows:

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The Bitwise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then:

Show Examples

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators

There are following assignment operators supported by C language:

Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Misc Operators ↦ sizeof & ternary

There are few other important operators including **sizeof** and **? :** supported by C Language.

Show Examples

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.

? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Show Examples

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Input/Output Statements:

Input : In any programming language input means to feed some data into program. This can be given in the form of file or from command line. C programming language provides a set of built-in functions to read given input and feed it to the program as per requirement.

scanf() function

Here %d is being used to read an integer value and we are passing &x to store the vale read input. Here &indicates the address of variavle x.

Output : In any programming language output means to display some data on screen, printer or in any file. C programming language provides a set of built-in functions to output required data.

printf() function

```
printf("%d %s %f %c\n", dec, str, pi,  ch);
```

Here %d is being used to print an integer, %s is being usedto print a string, %f is being used to print a float and %c is being used to print a character.

```
#include <stdio.h>
int main()
{
/* Our first simple C basic program */
printf(“Hello World! “);

return 0;
}
```

Decision Making in C

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

if statement

An if statement consists of a boolean expression followed by one or more statements.

```
 if(boolean_expression)
{
 /* statement(s) will execute if the boolean expression is true */
}
```

If the boolean expression evaluates to true, then the block of code inside the if statement will be executed. If boolean expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Ex:

```
if( a < 20 )
 {
 /* if condition is true then print the following */
 printf("a is less than 20\n" );
 }
 printf("value of a is : %d\n", a);
```

if...else statement

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

```
if(boolean_expression)
{
 /* statement(s) will execute if the boolean expression is true */
}
else
```

```
{
 /* statement(s) will execute if the boolean expression is false */
}
```

```
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m == n) {
      printf("m and n are equal");
  }
  else {
        printf("m and n are not equal");
  }

}
```

The if...else if...else Statement

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind:

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

```
if(boolean_expression 1)
{
 /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2)
{
 /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3)
{
 /* Executes when the boolean expression 3 is true */
}
else
{
 /* executes when the none of the above condition is true */
}
```

```
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m>n) {
      printf("m is greater than n");
  }
  else if(m<n) {
        printf("m is less than n");
  }
  else {
        printf("m is equal to n");
```

```
   }
}
```

Nested if statements

It is always legal in C programming to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

```
if( boolean_expression 1)
{
 /* Executes when the boolean expression 1 is true */
 if(boolean_expression 2)
 {
 /* Executes when the boolean expression 2 is true */
 }
}
```

switch statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

```
switch(expression){
 case constant-expression :
 statement(s);
 break; /* optional */
 case constant-expression :
 statement(s);
 break; /* optional */
  /* you can have any number of case statements */
 default : /* Optional */
 statement(s);
}
```

```
include <stdio.h>

int main ()
{
   int value = 3;

   switch(value)
   {
   case 1:
      printf("Value is 1 \n" );
      break;
   case 2:
      printf("Value is 2 \n" );
      break;
   case 3:
      printf("Value is 3 \n" );
      break;
   case 4:
      printf("Value is 4 \n" );
      break;
   default :
      printf("Value is other than 1,2,3,4 \n" );
   }
```

```
   return 0;
}
```

Output:

Value is 3

The following rules apply to a switch statement:
 The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
 You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
 The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
 When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
 When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
 Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
 A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

The ? : Operator
We have covered conditional operator ? : in previous chapter which can be used to replace if...else statements. It has the following general form:

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.
The value of a ? expression is determined like this: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

```
#include <stdio.h>
int main()
{
        int x=1, y ;
        y = ( x ==1 ? 2 : 0 ) ;
        printf("x value is %d\n", x);
        printf("y value is %d", y);
}
```

LOOPS:
There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths.
A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages

while loop in C
A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages.

A while loop statement in C programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a while loop in C programming language is:

```
 while(condition)
{
 statement(s);
}
```

Statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

```
#include <stdio.h>

int main()
{
  int i=3;

  while(i<10)
  {
      printf("%d\n",i);
      i++;
  }

}
```

Output:

```
3 4 5 6 7 8 9
```

for loop in C

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

The syntax of a for loop in C programming language is:

```
 for ( init; condition; increment )
{
statement(s);
}
```

Here is the flow of control in a for loop:

1. The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
2. Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
3. After the body of the for loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
4. The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
      printf("%d ",i);
  }

}
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

do...while loop in C

The do...while loop in C programming language checks its condition at the bottom of the loop.A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

The syntax of a do...while loop in C programming language is:

```
do
{
 statement(s);
}while( condition );
```

```
#include <stdio.h>

int main()
{
  int i=1;

  do
  {
      printf("Value of i is %d\n",i);
      i++;
  }while(i<=4 && i>=2);

}
```

Output:

```
Value of i is 1
Value of i is 2
Value of i is 3
Value of i is 4
```

break statement in C

The break statement in C programming language has the following two usages:

1. When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
2. It can be used to terminate a case in the switch statement.

If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

The syntax for a break statement in C is as follows:
break;

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
      if(i==5)
      {
          printf("\nComing out of for loop when i = 5");
    break;
      }
      printf("%d ",i);
  }

}
```

Output:

```
0 1 2 3 4
Coming out of for loop when i = 5
```

continue statement in C

The continue statement in C programming language works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control passes to the conditional tests.

The syntax for a continue statement in C is as follows:
continue;

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
      if(i==5 || i==6)
      {
          printf("\nSkipping %d from display using " \
                 "continue statement \n",i);
    continue;
      }
      printf("%d ",i);
  }

}
```

Output:

```
0 1 2 3 4
Skipping 5 from display using continue statement
Skipping 6 from display using continue statement
7 8 9
```

goto statement in C

A goto statement in C programming language provides an unconditional jump from the goto to a labeled statement in the same function.

NOTE: Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten so that it doesn't need the goto.

The syntax for a goto statement in C is as follows:

goto label;

..

.

label: statement;

```
#include <stdio.h>

int main()
{
  int i;

  for(i=0;i<10;i++)
  {
      if(i==5)
      {
          printf("\nWe are using goto statement when i = 5");
          goto HAI;
      }
      printf("%d ",i);
  }

HAI : printf("\nNow, we are inside label name \"hai\" \n");

}
```

Output:

```
0 1 2 3 4
We are using goto statement when i = 5
Now, we are inside label name “hai”
```

The Infinite Loop

A loop becomes infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

C programmers more commonly use the for(;;) construct to signify an infinite loop.

C Functions

Function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

Defining a Function
The general form of a function definition in C programming language is as follows:
return_type function_name(parameter list)
{
 body of the function
}

A function definition in C programming language consists of a function header and a function body. Here are all the parts of a function:
 Return Type: A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
 Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.
 Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
 Function Body: The function body contains a collection of statements that define what the function does.

Function Declarations
A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
A function declaration has the following parts:
return_type function_name(parameter list);
For the above defined function max(), following is the function declaration:
int max(int num1, int num2);
Parameter names are not important in function declaration only their type is required, so
following is also valid declaration:
int max(int, int);
Function declaration is required when you define a function in one source file and you call that function in another file. In such case you should declare the function at the top of the file calling the function.

```
#include<stdio.h>
// function prototype, also called function declaration
float square ( float x );

// main function, program starts from here
int main( )
{

        float m, n ;
        printf ( "\nEnter some number for finding square \n");
```

```
        scanf ( "%f", &m ) ;
        // function call
        n = square ( m ) ;
        printf ( "\nSquare of the given number %f is %f",m,n );

}

float square ( float x )   // function definition
{
        float p ;
        p = x * x ;
        return ( p ) ;
}
```

Output:

```
Enter some number for finding square
2
Square of the given number 2.000000 is 4.000000
```

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.When a program calls a function, program control is transferred to the called function. A called function performs defined task, and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value. For example:

Function call by value

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

#include<stdio.h>

```
// function prototype, also called function declaration
void swap(int a, int b);

int main()
{
    int m = 22, n = 44;
    // calling swap function by value
    printf(" values before swap  m = %d \nand n = %d", m, n);
    swap(m, n);
}

void swap(int a, int b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
    printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

Output:

```
values before swap m = 22
and n = 44
values after swap m = 44
```

```
and n = 22
```

Function call by reference

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.To pass the value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function swap(), which exchanges the values of the two integer variables pointed to by its arguments.

```
#include<stdio.h>
// function prototype, also called function declaration
void swap(int *a, int *b);

int main()
{

    int m = 22, n = 44;
    //  calling swap function by reference
    printf("values before swap m = %d \n and n = %d",m,n);
    swap(&m, &n);

}

void swap(int *a, int *b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    printf("\n values after swap a = %d \nand b = %d", *a, *b);
}
```

Output:

```
values before swap m = 22
and n = 44
values after swap a = 44
and b = 22
```

C Arrays

Cprogramming language provides a data structure called the array, which can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows:

type arrayName [arraySize];

This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and

type can be any valid C data type. For example, to declare a 10-element array called balance of type double, use this statement:
double balance[10];
Now balance is a variable array which is sufficient to hold up-to 10 double numbers

Initializing Arrays

You can initialize array in C either one by one or using a single statement as follows:
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array:
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write:
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
You will create exactly the same array as you did in the previous example.
balance[4] = 50.0;
The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th i.e. last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representation of the same array we discussed above:

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example:
double salary = balance[9];
The above statement will take 10th element from the array and assign the value to salary variable.
Following is an example which will use all the above mentioned three concepts viz. declaration, assignment and accessing arrays:

```
#include<stdio.h>

int main()
{
    int i;
    int arr[5] = {10,20,30,40,50};
    // declaring and Initializing array in C
    //To initialize all array elements to 0, use int arr[5]={0};
    /* Above array can be initialized as below also
       arr[0] = 10;
       arr[1] = 20;
       arr[2] = 30;
       arr[3] = 40;
       arr[4] = 50;
    */
    for (i=0;i<5;i++)
    {
        // Accessing each variable
        printf("value of arr[%d] is %d \n", i, arr[i]);
    }
}
```

Output:

```
value of arr[0] is 10
value of arr[1] is 20
value of arr[2] is 30
value of arr[3] is 40
value of arr[4] is 50
```

Multi-dimensional Arrays

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

type name[size1][size2]...[sizeN];

For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array:

int threedim[5][10][4];

Two-Dimensional Arrays

The simplest form of the multidimensional array is the two-dimensional array. To declare two dimensional array x and y values to be specified.

type arrayName [x][y];

```
#include<stdio.h>
int main()
{
    int i,j;
    // declaring and Initializing array
    int arr[2][2] = {10,20,30,40};
    /* Above array can be initialized as below also
       arr[0][0] = 10;   // Initializing array
       arr[0][1] = 20;
       arr[1][0] = 30;
       arr[1][1] = 40;
    */
    for (i=0;i<2;i++)
    {
       for (j=0;j<2;j++)
       {
          // Accessing variables
          printf("value of arr[%d] [%d] : %d\n",i,j,arr[i][j]);
       }
    }
}
```

Output:

```
value of arr[0] [0] is 10
value of arr[0] [1] is 20
value of arr[1] [0] is 30
value of arr[1] [1] is 40
```

C Pointers

Pointers in C are easy and fun to learn. Some C programming tasks are performed more easily with pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect C programmer. Let's start learning them in simple and easy steps.As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.

What Are Pointers?

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address. The general form of a pointer variable declaration is:

type *var-name;

- Programming basics, algorithms,
- use of Pseudo-codes & Flowcharting,
- Data types, operators, expression,

I/O statements,

```
#include <stdio.h>

int main()
{
        int *ptr, q;
        q = 50;
        /* address of q is assigned to ptr       */
        ptr = &q;
        /* display q's value using ptr variable */
        printf("%d", *ptr);
        return 0;
}
```

Output: