Programming with C – Introduction

UNIVERSITY
OF PETROLEUM
& ENERGY STUDIES

PROGRAMMING FUNDAMENTALS

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**

Software Development Method (SDM)

1. Specification of needs
2. Problem analysis
3. Design and algorithmic representation
4. Implementation
5. Testing and verification
6. Documentation

Specification of Needs

- 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.

Problem Analysis

- In the analysis phase, we should identify the following:
  - Inputs to the problem, their form and the input media to be used
  - Outputs expected from the problem, their form and the output media to be used
  - Special constraints or conditions (if any)
  - Formulas or equations to be used

Design and Algorithmic Representation

- 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

Pseudocodes

- 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

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
```

Example 3

- 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
```

Example 4

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
```

Pseudocodes: The Repetition control structure

- Example 5: Summing up 1 to 10

```
set cumulative sum to 0
set current number to 1
while current number is less or equal to 10
     add the cumulative sum to current number
     add 1 to current number
end_while
print the value of cumulative sum
```

Pseudocodes: The **Repetition** control structure

- Subsequently, we can write the previous pseudocodes (example 5) with something like this.

- Example 6: Summing up 10 numbers

```
cumulative sum = 0
current number = 1
while current number is less or equal to 10
      cumulative sum = cumulative sum + current number
      current number = current number + 1
end_while
print the value of cumulative sum
```

- Note that in this algorithm, we are using both the *sequence* and *repetition* control structure

Pseudocodes: The Repetition control structure

- Example 7:

```
Begin
  number of users giving his birth date = 0
  while number of users giving his birth date < 10
        begin
           Read the birth date from the user.
           Calculate the difference between the birth
  date and today's date.
           Print the user age.
           if the age is greater than 55
                print "Pencen"
           else
                print "Kerja lagi"
           end_if
           number of user giving his birth date + 1
        end
  end_while
End
```

Pseudocodes: The Repetition control structure

- Example 8:

```
while user still wants to play
  begin
        Select either to play on network or play against computer
           if play on network
                create connection to remote machine
                play game with connected computer
           else
               select mission
               play game locally
           end_if
          Ask user whether he/she still wants to play
    end
end_while
```

Pseudocodes: The **Repetition** control structure

- Example 9:

```
while user still wants to play
begin
Select either to play on network or play against computer
if play on network
create connection to remote machine
play game with connected computer
Else
select mission
play game locally
end_if
Ask user whether he/she still wants to play
end
end_while
```

- For readability, always use proper *indentation!!!*

Flowcharts

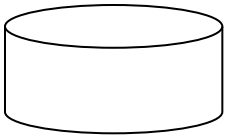
- Flowcharts is a graph used to depict or show a step by step solution using **symbols** which represent a task.
- The symbols used consist of geometrical shapes that are connected by **flow lines**.
- It is an alternative to pseudocoding; whereas a pseudocode description is verbal, a flowchart is graphical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

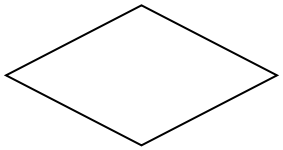
Input-output symbol - shows an input or an output operation.



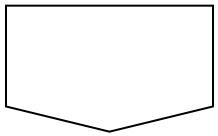
Disk storage I/O symbol - indicates input from or output to disk storage.

Printer output symbol - shows hardcopy printer output.

Flowchart Symbols cont…

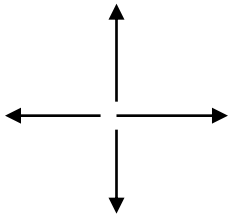


Selection symbol - shows a selection process for two-way selection.



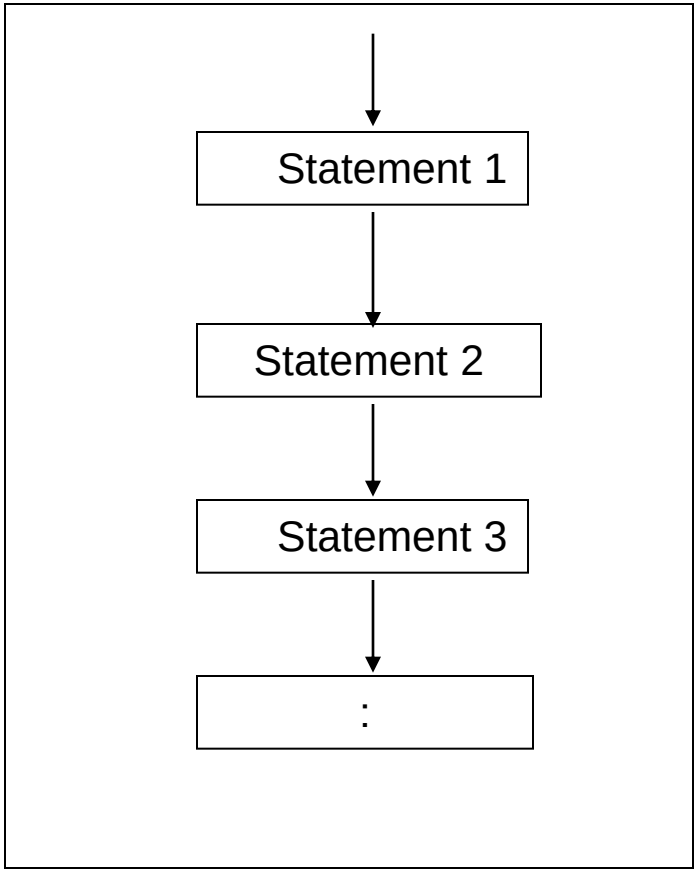
Off-page connector - provides continuation of a logical path on another page.

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

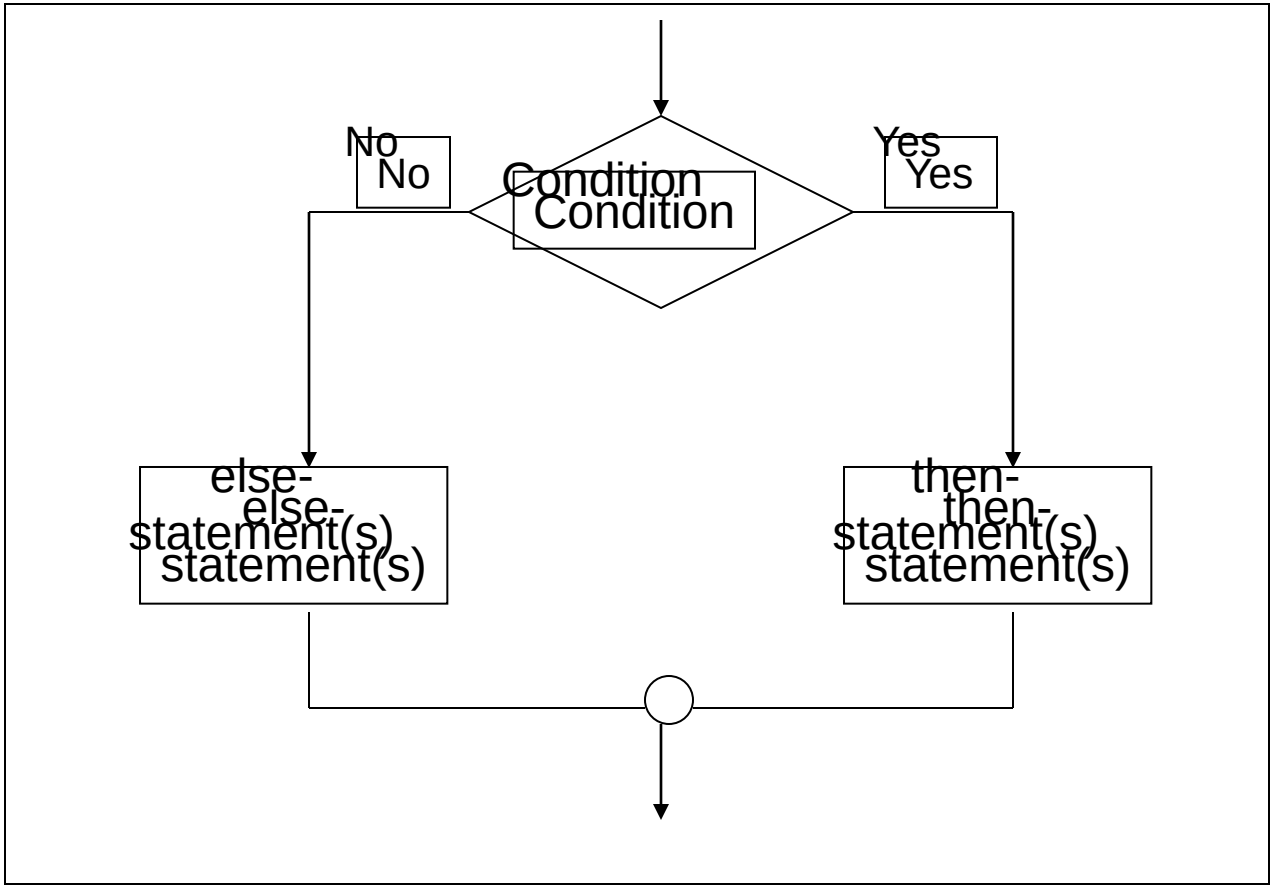


Flow lines - indicate the logical sequence of execution steps in the algorithm.

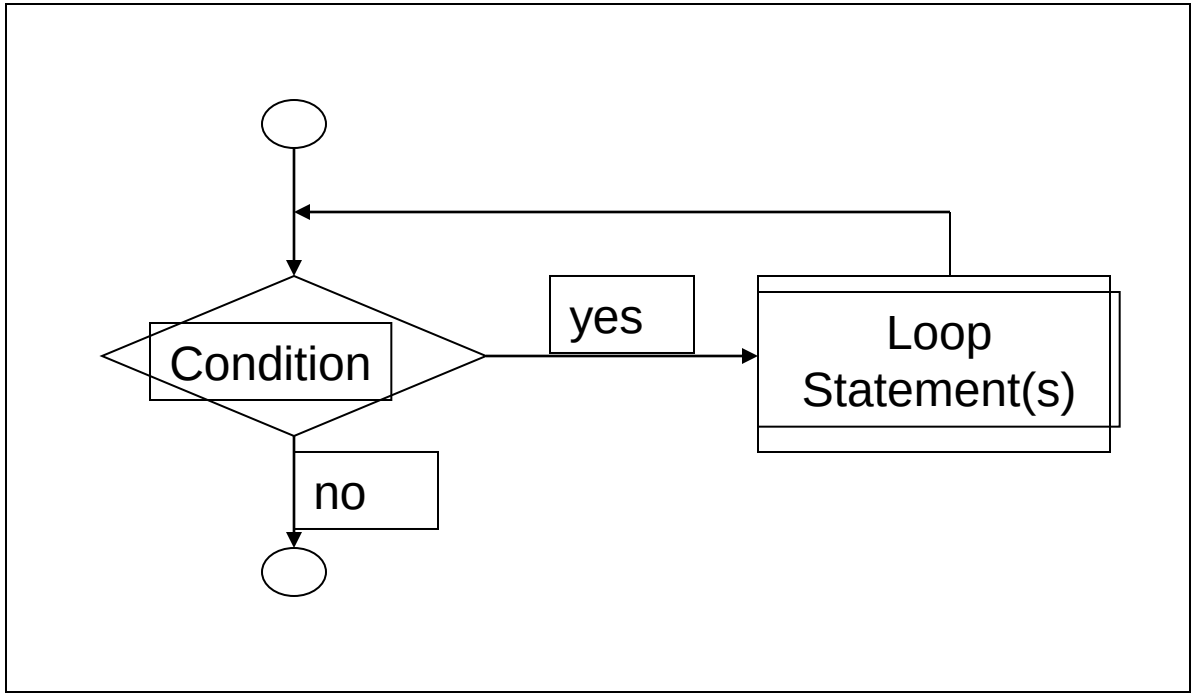
Flowchart – sequence control structure



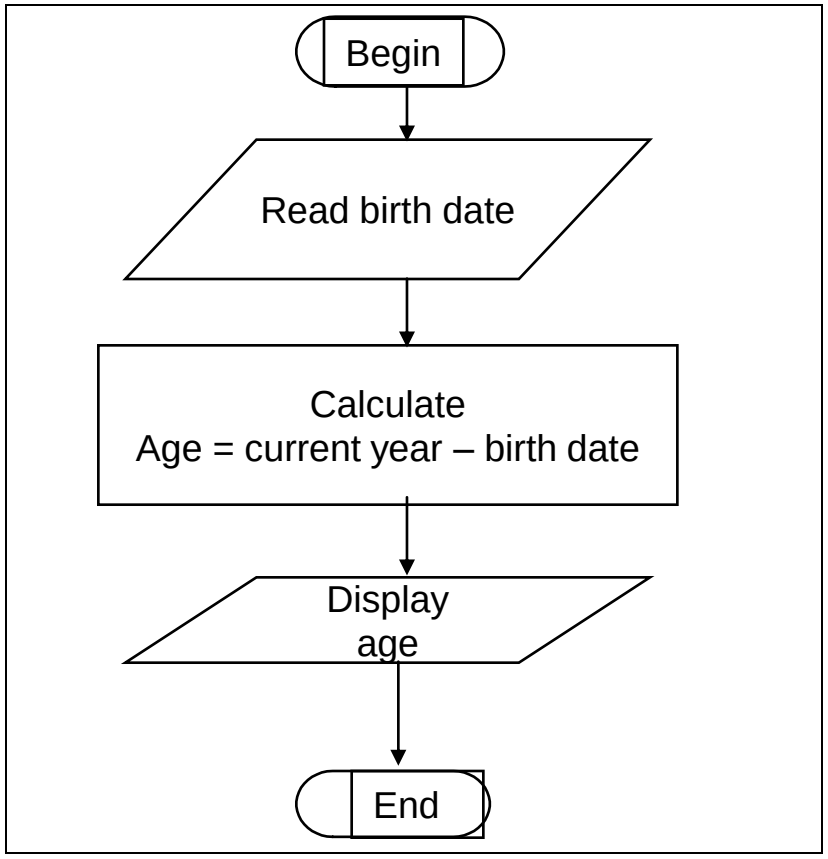
Flowchart – selection control structure



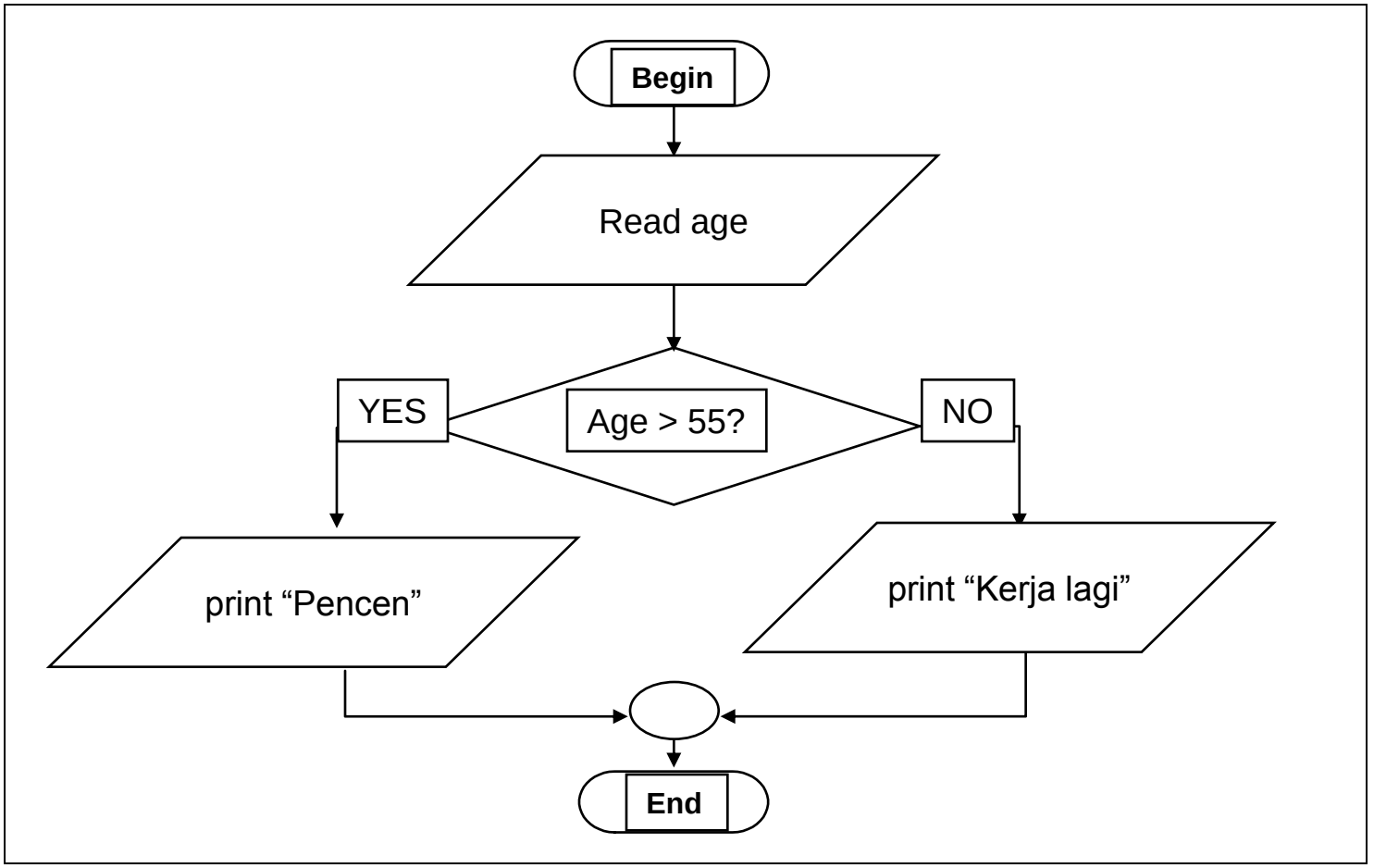
Flowchart – repetition control structure



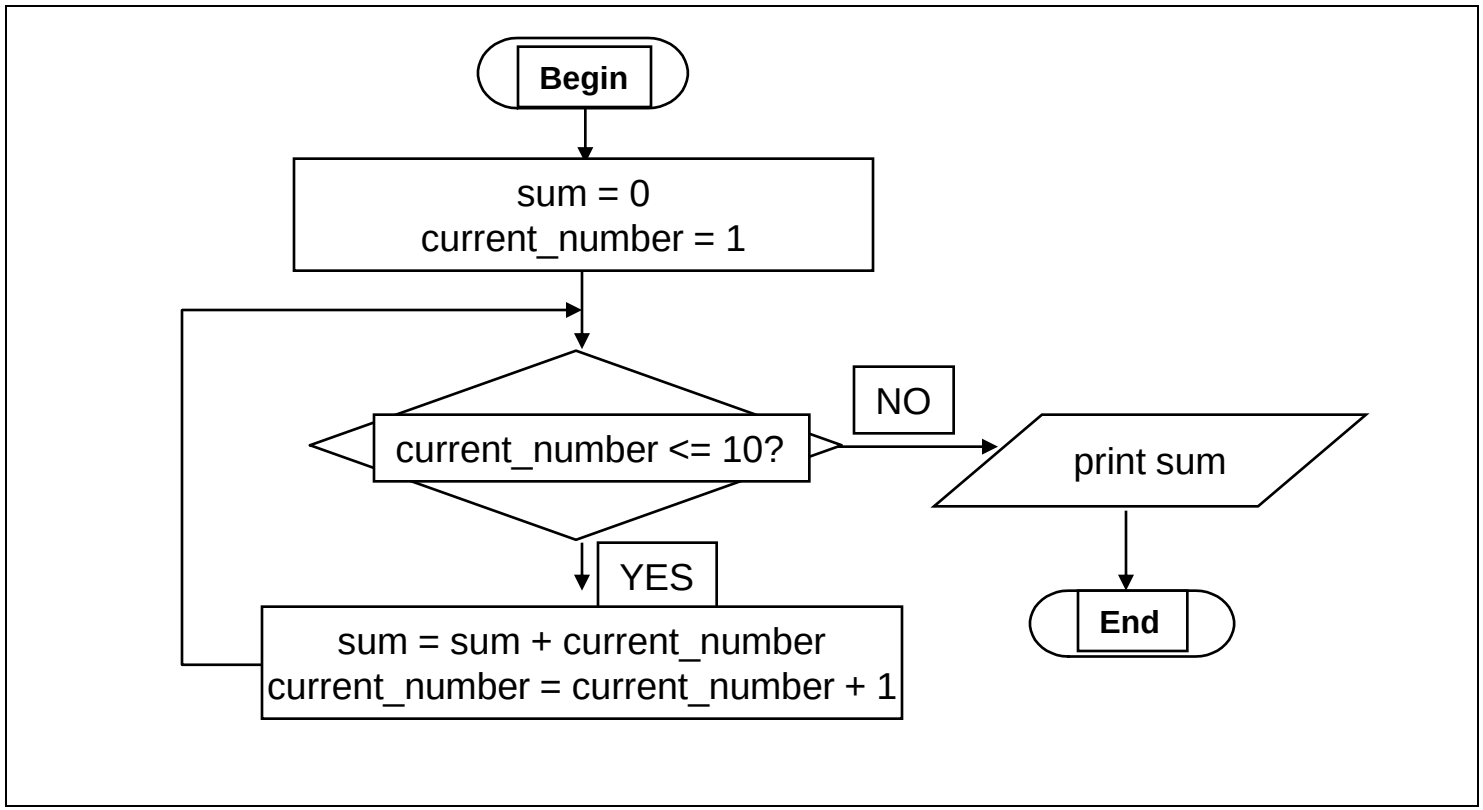
Flowchart – example 1



Flowchart – example 2



Flowchart – example 5



Implementation

- The process of ***implementing*** an algorithm by writing a computer program using a programming language (for example, using C language)

- The output of the program **must** be the solution of the intended problem

- The program must **not** do anything that it is **not** supposed to do
  - *(Think of those many **viruses, buffer overflows, trojan horses,** etc. that we experience almost daily. All these result from programs **doing more** than they were intended to do)*

Testing and Verification

- Program **testing** is the process of executing a program to demonstrate its correctness
- Program **verification** is the process of ensuring that a program meets user-requirement
- After the program is ***compiled***, we must ***run*** the program and test/verify it with **different inputs** before the program can be released to the public or other users (or to the instructor of this class)

Documentation

- Contains details produced at all stages of the ***program development cycle***.

- Can be done in 2 ways:
  - Writing comments between your line of codes
  - Creating a separate text file to explain the program

- Important not only for other people to use or modify your program, but also for you to understand your own program after a long time *(believe me, you will forget the details of your own program after some time ...)*

Documentation cont…

- Documentation is so important because:
  - You may return to this program in future to use the whole of or a part of it again
  - Other programmer or end user will need some information about your program for reference or maintenance
  - You may someday have to modify the program, or may discover some errors or weaknesses in your program

- Although documentation is listed as the last stage of software development method, it is actually an ongoing process which should be done from the **very beginning** of the software development process.

C Introduction

- The C programming language was designed by Dennis Ritchie at Bell Laboratories in the early 1972
- Influenced by
  - ALGOL 60 (1960),
  - CPL (Cambridge, 1963), -
  - BCPL (Martin Richard, 1967), - Basic Combined Programming Language
  - B (Ken Thompson, 1970)
- Procedural language also known as structured programming
- Standardized in 1989 by ANSI (American National Standards Institute) known as ANSI C

DATA TYPES

- **Integer data type**
  - Short integer(short) 2 bytes -32,768 to +32,767
  - Integer(Int) 4 bytes -2,147,483,648 to +2,147,483,647
  - Long Integer(long) 4 bytes -2,147,483,648 to +2,147,483,647

  Integer data types again classified into signed and unsigned. Signed can store -ve values. Unsigned can store +ve values

- **Floating point data type**
  - float 4 bytes 3.4e-38 to 3.4e+37
  - double 8 bytes 17e-308 to 1.7e+308
  - Long double 10 bytes

- **Character data type**
  - signed char(char) 1 byte -128 to +127
  - Unsigned char 1 byte 0 to 255

User defined data type declaration

- To define user define data type:

typedef type identifier;

Where **typedef** is a keyword

type is data type

identifier is a new name

Eg., **typedef int integer;**

tyedef float real;

integer num1, num2;

real avg, per;

The advantage of typedef is that we can create meaningful data type names for increasing readability of the program

User defined data type declaration

- Another user defined data type is enumerated data type:

  enum identifier{ value1, value2, …….. Valuen};

  Where enum is keyword

  Identifier is a new name

  Value1, value2, ……. Valuen are different values

  Compiler automatically assign integer values starting from 0

  Eg., enum day{mon, tue, wed, thu, fri, sat, sun};

  i.e., mon =0, tue = 1, wed = 2, ….. Sun = 6

  You can also override the values like

  enum day{mon =1, tue, wed, thu, fri, sat, sun};

  i.e., mon = 1, tue = 2, …….. Sun = 7

Standard Input / Output in C

- C has a notation of input coming from a place known as **standard input** and output goes to a place known as **standard output**

- **stdin** **stdout** standard I/O functions. Predefined streams automatically opened when the program is started.

- For Input/output functionality need to add #include<stdio.h>

Standard Input / Output in C

- Some standard functions are defined in C programming for stdin and stdout are

getchar(), putchar(), getc(), putc()

getchar() retrieves a single character from standard input

Eg., char c= getchar();

This function return value will be one of the next input character

putchar() sends a single character to standard output

Eg., putchar(c);

The character to be output is passed as a parameter.

Functions getc(), putc() are much same.

Eg: c = getc(stdin); putc(c,stdout);

Standard Input / Output in C

- The mentioned functions are for single character I/O

- <stdio.h> gives a access **printf(), scanf()** functions
- **printf()** is a function that converts formats and prints the arguments on a standard output
- Eg., printf("The value of x is %d, and value of y is %d\n",x,y);
- The output will appear as: If x is 10, y is 5

  The value of x is 10 and value of y is 5

  d to print decimal number

  c to print character **s** to print string

  x to print hexadecimal

Standard Input / Output in C

- Eg:

```
#include<stdio.h>
main()
{
    int x = 5000;
    printf(%d\n”, x);
    printf(%10d\n”,x);
}
```

Output is

5000

6spaces and 5000 (will be printed as right alignment with 10 spaces)

Standard Input / Output in C

- **scanf()** is similar to printf(), but it reads input.

scanf(“%d”, &x); x is an integer

scanf(“%f”, &y); y is a real

scanf(“%c”, &z); z is a character

scanf(“%s”, &u); u is a word

Basic Operators

- The precedence rules of operator are:

( ) *,/ +,-

In addition to arithmetic (*,/,+,-) operators % for modulus

Assignement operators

- Assignment operator to assign the value of a variable or expression

  var = expression; **var = var;**

- C support short hand format of assignment operators +=, -=, *=, /=, %=

Bitwise operators

- Bitwise AND (&):
  - For integral types, ANDs each corresponding pair of bits
    - 0 & 0 == 0
      0 & 1 == 0
      1 & 0 == 0
      1 & 1 == 1
    - Eg., d1=4 and d2 = 6
    - The binary representation of d1 is 0000000000000100
    - The binary representation of d2 is 0000000000000110
    - d1 & d2 is 0000000000000100
    - i.e., 4

Bitwise operators

- Bitwise OR (|):
  - For integral types, ORs each corresponding pair of bits

    0 | 0 == 0
    0 | 1 == 1
    1 | 0 == 1
    1 | 1 == 1

    Eg., d1 is 4 and d2 is 6. d1|d2 is 6
- Bitwise XOR (^):
  - For integral types, XORs each corresponding pair of bits
    - 0 ^ 0 == 0 0 ^ 1 == 1 1 ^ 0 == 1 1 ^ 1 == 0
    - Eg., d1 is 4 and d2 is 6. d1^d2 is 2

Bitwise operators

- **Bitwise Complement (~):**

Bitwise complement (~) is an unary operator. It can be defined by replacing 0 by 1 and 1 by 0.

eg., d1 is 4

4 is in binary 0000000000000100

complement of d1 is 1111111111111011

Bitwise operators

- **Left shift (<<):**

  It is a binary operator. In this operator the second operand specifies the number of bits position to be shifted in the left of the first operand. Add 0 in the right position of shifted bits.

  Eg., d1 is 4 means 0000000000000100

  d1<<2 means 0000000000010000 = 16

- **Right shift (>>):**

  Add 0 in the left position of the shifted bits.

  Eg., d1 is 4 means 0000000000000100

  d1 >> 2 means 0000000000000001 = 1

Increment & decrement operators

- ++ and --

Eg., ++ count ; /* prefix */

count --; /* postfix */

Eg., x = 10;

y = ++x; y = 11

y = x++; y = 10

y = --x; y = 9

y = x--; y = 10

(* Not mixed all the statements. Each is taken as separate)

Relational operators

==	equal to	!=	not equal to
<	less than	<=	less than equal
>	greater than	>=	greater than equal

Logical operators

- Negation !
- Logical AND &&
- Logical OR ||

Order of precedence is !, &&, ||

Ternary operator or Conditional operator

- (condition) ? Expression 1 : expression 2

- Eg., value = (x < 0) ? -1 : x +5

  If x is less than 0 then value = -1

  If x is greater than 0 then value = x +5

Thanks