Paper – 104 Computer Programming & Programming Methodology (CPPM)

Subject Code -104
Subject title - Computer Programming & Programming Methodology Review/Revision Year: June 2020

UNIT-1: Introduction
1.1 Concepts of Programming Language
1.1.1 Introduction of Source Code, Object Code and executable code
1.1.2 Algorithm and Flowchart
1.1.3 Concepts of Structured Programming Language
1.2 Concepts of Editor, Interpreter and Compiler
1.2.1 Introduction of C program body structure
1.2.2 Character Set, concepts of variables and constants
1.2.3 Identifiers, literals, Key words
1.2.4 Data types (signed and unsigned) (Numeric : int, short int, long, float,double) , (Character type: char, string) and void.
1.2.5 Concepts of source code, object code and executable code.

UNIT-2: Input/Output Statements and Operators
2.1 Input/Output statements:
2.1.1 Concepts of Header files (STDIO,CONIO)
2.1.1.1 Concepts of pre-compiler directives.
2.1.1.2 Use of #include and #define
2.2 Input/Output Statements:
2.2.1 Input statements : scanf(), getc(), getch(), gets(), getchar()
2.2.2 Output Statements: printf(), putc(),puts(), putchar()
2.2.3 Type specifiers (formatting strings) : %d, %ld, %f, %c, %s, %lf
2.3 Operators :
2.3.1 Arithmetic operators (+, -, *, /, %, ++, --)
2.3.2 Logical Operators (&&, ||, !)
2.3.3 Relational Operators (>, <, ==, >=, <=, !=)
2.3.4 Bitwise operators (&, |, ^ , <<, >>)
2.3.5 Assignment operators (=, +=, -=, *=, /=, %=)
2.3.6 Ternary Operator and use of sizeof() function.
2.4 Important Built-in functions:
2.4.1 Use of <string.h> : (strlen, strcmp, strcpy, strcat, strrev)
2.4.2 Use of <math.h> : (abs(), floor(), round(), ceil(), sqrt(), exp(), log(), sin(), cos(), tan(), pow() and trunc())

UNIT-3: Decision Making statements :
3.1 if statements :
3.1.1 simple if statements
3.1.2 if…else statements
3.1.3 if…else if….else statements
3.1.4 Nested if statements.
3.2 Switch..case statements
3.2.1 Use of break and default
3.2.2 Difference between switch and if statements.

UNIT-4: Iterative statements
4.1 Use of goto statement for iteration
4.2 while loop
4.3 do..while loop
4.4 for loop
4.5 Nested while, do..while and for loops
4.6 Jumping statement: (break and continue)

UNIT-5: Concepts of Arrays and pointer
5.1 Concepts of Single-dimensional Array
5.1.1 Numeric single dimensional Array
5.1.2 Numeric single dimensional array operations:
5.1.2.1 Sorting array in ascending or descending. (Bubble and selection)
5.1.2.2 Searching element from array (Linear search)
5.1.3 Character Single dimensional Array
5.1.3.1 Character Single dimensional array operations
5.1.3.2 Use of \0, \n and \t
5.2 Pointers:
5.2.1 Concepts of Pointers
5.2.2 Declaring and initializing int, float, char and void pointer
5.2.3 Pointer to single dimensional numeric array.

Reference Books
1. Programming in C, Balaguruswami – TMH
2. C: How to Program, Deitel & Deitel - PHI
3. C Programming Language, Kernigham & Ritchie -
TMH 4. Programming in C, Stephan Kochan - CBS
5. Mastering Turbo C, Kelly & Bootle - BPB
6. C Language Programming – Byron Gottfried -
TMH 7. Let us C, Yashwant Kanetkar - BPB
Publication 8. Magnifying C, Arpita Gopal - PHI
9. Problem Solving with C, Somashekara - PHI
10. Programming in C, Pradip Dey & Manas Ghosh – Oxford

UNIT-1: Introduction

1.1 Concepts of Programming Language

A program is a set of instructions that help a computer perform tasks.
The languages used to write a program or set of instructions are called "programming languages."
Some examples of programming languages are C, C++, Java, and Python.

1.1.1 Introduction of Source Code, Object Code and executable code

The source code is written by a programmer using a high-level or intermediate language that is human-readable. Source code is easy to read and modify. Computers cannot understand direct source code; they understand machine code and execute it. In simple terms, source code is a set of instructions, commands, and statements written by a programmer using a computer programming language such as C, C++, Java, or Python.

Object code refers to low level code which is understandable by machine. Object code is generated from source code by compiler or other translator. It is the output of a compiler or other translator. We can understand source code but we can not understand object code as it is not in plain text like source code rather it is in binary formats. Example:.obj files.

Object code links with a linker and system library to generate an executable file known as executable code. Example: .exe files.

Source code → compiler → **object code** → linker (runtime library files) → **executable code**

1.1.2 Algorithm and Flowchart

Algorithm- An algorithm is a step-by-step procedure (process) to solve a given problem. Algorithms have a definite beginning and a definite end, and a finite number of steps. An algorithm produces output depending on the basis of given input, and several algorithms can be combined to perform complex tasks .

Features of Algorithm
It should be simple.
It should consist of a finite number of steps.
It should be clear, with no uncertainty or doubt.
It should have the capability to handle unexpected situations that may arise during the solution of a problem.

Example 1. 1 Algorithm: Addition of Two Numbers.

```
Step 1: Start
Step 2: Input two numbers
Step 3: Add the two numbers together
Step 4: Display the result
Step 5: Stop
```

Example 1. 2 Algorithm: Calculate the Area of a Circle.

```
Step 1: Start
Step 2: Input the value of the radius
Step 3: Calculate the area using the formula: area = 3.14 * radius* radius
Step 4: Display the value of the area
Step 5: Stop
```

In the example above, the steps are executed sequentially, but sometimes steps are skipped and executed based on conditions to make decisions.

Example 1. 3 Algorithm: Check if the given number is positive, negative, or zero.

```
Step 1: Start
Step 2: Input a number Num
Step 3: If Num > 0 then
                Display "Num is positive"
        Else if Num < 0 then
                Display "Num is negative"
        Else
                Display "Num is zero"
Step 4: Stop
```

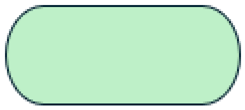
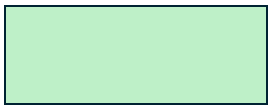
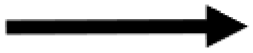
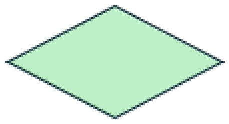
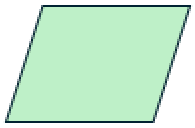
Sometimes we want to repeat one or more statements more than once. This is called looping.

Example 1. 4 Algorithm: Print Numbers from 1 to 10

```
Step 1: Start
Step 2: I = 1
Step 3: Repeat steps (a) and (b) while I <= 10
            (a) Display I
            (b) I = I + 1
Step 4: Stop
```

Flowchart

The flowchart is a graphical way to represent an algorithm, and it is very helpful in solving complicated programming problems. Once the flowchart is drawn, it does become easier to write a program.The table shows the symbols used for drawing flowchart.

	Oval	Represents the start or end of a process
	Rectangle	Denotes a process or operation step
	Arrow	Indicates the flow between steps
	Diamond	Signifies a point requiring a yes/no
	Parallelogram	Used for input or output operations

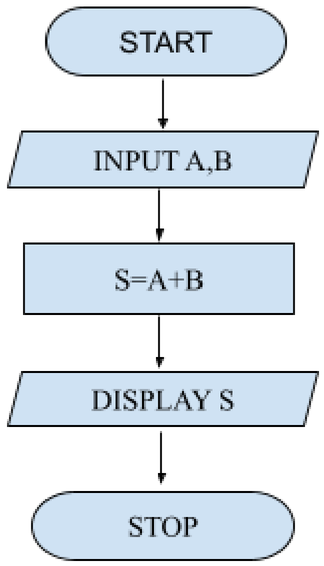
Benefits of Flowcharts:

Flowcharts help in the debugging process.
With the help of flowcharts programs can be easily analyzed.
It provides better documentation.
Easy to trace errors in the software.
Easy to understand.

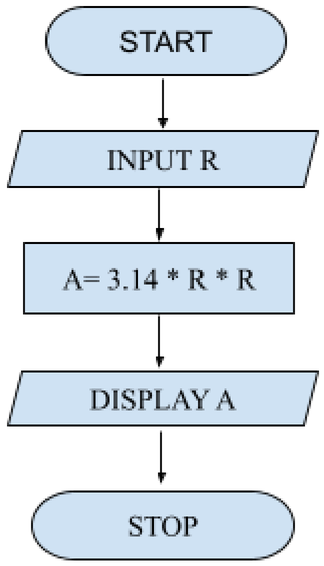
Limitations of Flowcharts:

Complex and detailed flowcharts can sometimes be difficult to draw.
Adding information to a flowchart can be sometimes difficult.

Example 1. 5 Draw the flowchart for the sum of two numbers.



Example 1. 6 Draw the flowchart for calculate the Area of circle



Branching

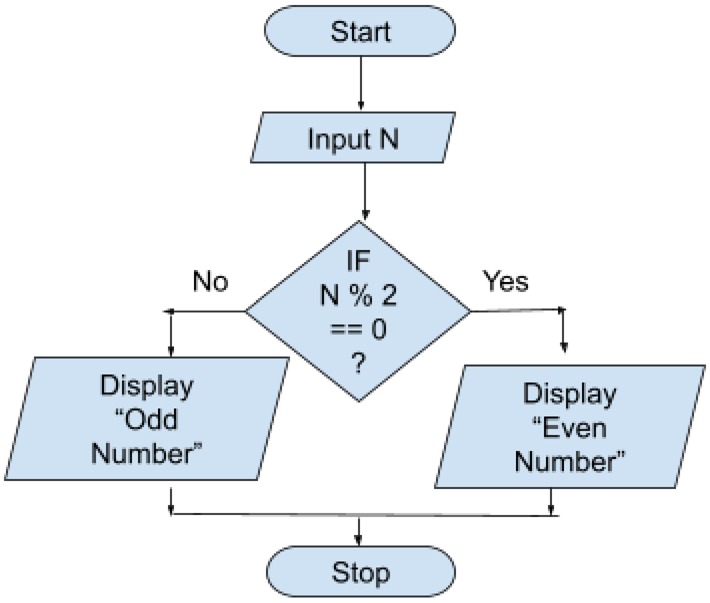
In programming, branching allows for different paths to be taken based on certain conditions. The decision box contains the condition, and if the condition is true, the "yes" branch is executed, otherwise the "no" branch is executed.

Example 1. 7 Write an algorithm and draw a flowchart to check if a given number is even or odd.

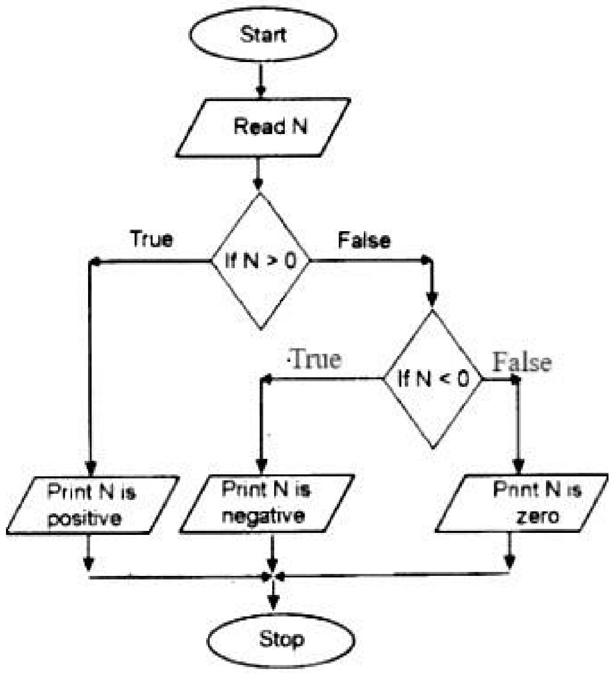
Algorithm

```
Step 1:Start
Step 2:Input a Number (N)
Step 3:IF N % 2 == 0 then
                Print "Even Number"
        Else
                Print "Odd Number"
Step 4: Stop
```

Flowchart



Example 1. 8 Draw a flowchart to check if no is positive, negative and zero.

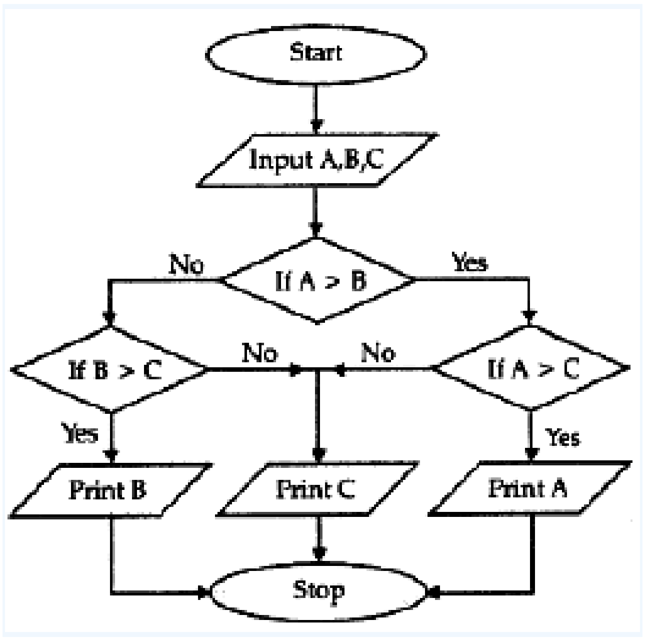


Example 1. 9 Write an algorithm and draw a flowchart to find the largest number among three numbers.

Algorithm

```
Step 1:Start
Step 2:Input A,B,C
Step 3:If (A>B) and (A>C) then print “A is largest”.
       Else if (B>A) and (B>C) then print “B is largest”.
       Else print “C is largest”.
Step 4:  Stop
```

Flowchart

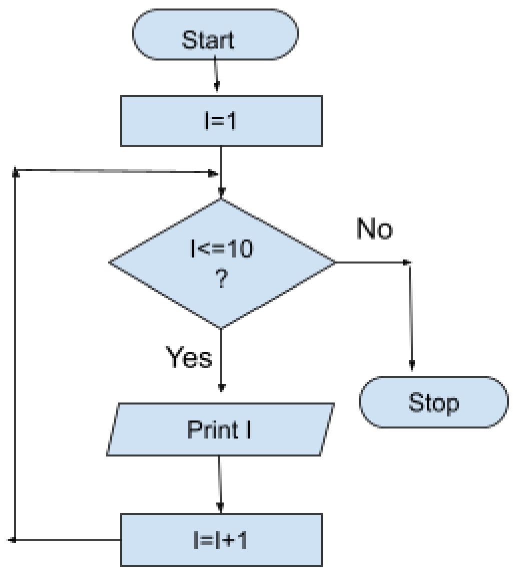


Looping

Looping refers to the repeated use of one or more steps. There are two types of loops: (1) fixed loop and (2) variable loop.

In a fixed loop, a set of statements is executed a fixed number of times until the condition is true, while in a variable loop,statements are executed until the specified condition is met. Loops can jump forward or backward. Loops are defined as follows.

Example 1. 10 Draw a flowchart to print numbers from 1 to 10.

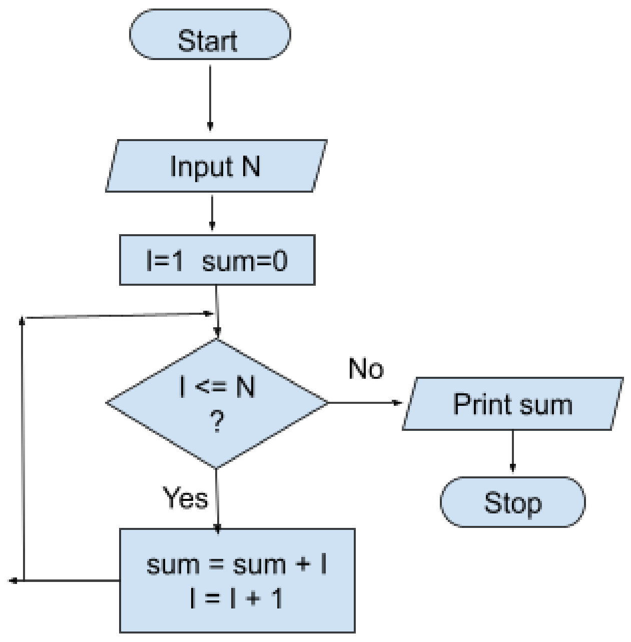


Example 1.11 Write an algorithm and draw a flowchart to find the sum of N numbers, where the sum is calculated as sum = 1 + 2 + 3 + ... + N.

Algorithm:

```
Step 1: Start
Step 2: Input a number N.
Step 3: Declare sum to 0 and I to 1
Step 4: Repeat steps (a) to (b) until I <= N
            (a)  sum = sum + I
            (b)  I=I+1
Step 5: Print sum
Step 6: Stop
```

Flowchart



1.1.3 Concepts of Structured Programming Language

C is called a structured programming language because, to solve a large problem, the C programming language divides the problem into smaller modules called functions or procedures, each of which handles a particular responsibility. The program that solves the entire problem is a collection of such functions. Here is an example of a Matrix addition program, which is divided into these sub-procedures: input matrix, display matrix, add matrix, save result matrix to file.

Advantages:
1. Readability: Structured programming in C enhances code readability by breaking down complex problems into smaller, more manageable modules.
2. Reusability: Functions or procedures in structured programming can be reused in different parts of the program, promoting code reusability.
3. Maintainability: Dividing a program into smaller modules makes it easier to maintain and update the codebase.
4. Debugging: Structured programming simplifies the debugging process as issues can be isolated to specific functions or procedures.

Disadvantages:
1. Limited Flexibility: Structured programming may limit the flexibility of the program structure, making it challenging to implement certain complex algorithms or data structures.
2. Overhead: Breaking down a program into multiple functions can introduce overhead in terms of function calls and memory management.
3. Complexity: While structured programming promotes code organization, it can also introduce complexity when dealing with interconnected functions and dependencies.

1.2 Concepts of Editor, Interpreter and Compiler

1.2.1 Introduction of C program body structure

A C program may contain one or more sections. as illustrated in Example 1.12 below.

Example 1.12

```
// Example of C program body structure // Documentation Section
#include <stdio.h>  //Link Section
#include <conio.h>

#define pi 3.14 // Defination Section

int a=10; // Global Declaration Section
void display();

int main() // Main Function Section
{
            int b=20; //Declaration Part
            clrscr(); //Executable Part
            display();
            printf("\n Sum is %d",a+b);
            getch();
            return(0);
}

void display()
{
            printf("\n C program body structure");
}

 Output : C program body structure
          Sum is 30
```

Documentation Section: The documentation section consists of a set of comment lines providing the program's name, the author, and other details that the programmer may wish to reference later. It includes two types of comments: single-line (//) and multiline (/*...*/).

Link Section: The link section provides instructions to the compiler for linking functions from the system library, such as #include <stdio.h>.

Definition Section: The definition section defines all symbolic constants, for example, #define pi 3.14.

Global Declaration Section: Variables used in multiple functions, known as global variables, are declared in the global declaration section, outside of all functions. This section also declares all user-defined functions.

Main() Function Section: Every C program must contain one main function section, which contains two parts: the declaration part and the executable part.

Declaration Part: This part declares all variables used in the executable part.

Executable Part: Contains at least one statement and must be enclosed within opening and closing braces. Program execution starts at the opening brace and ends at the closing brace of the main function, which is the logical end of the program. All statements in the declaration and executable part end with a semicolon.

Subprogram Section: This section contains all user-defined functions called in the main() function. User-defined functions are typically placed immediately after the main() function, although they can appear in any order.
All sections, except the main() function section, may be absent when not required.

1.2.2 Character Set, concepts of variables and constants

Character set
The character set is the set of alphabets, letters, and some special characters that are valid in the C language.

Alphabets
Uppercase: A B C .... X Y Z
Lowercase: a b c .... x y z

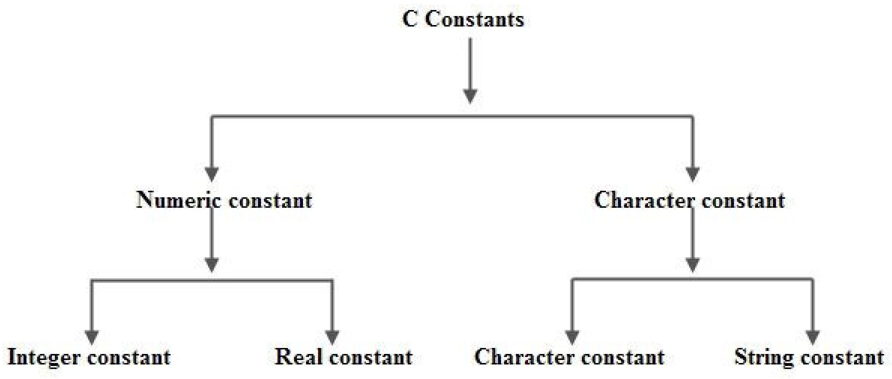
Digits
0 1 2 3 4 5 6 8 9

Special Characters
Special Characters in C language
, < > . _ ( ) ; $: % [ ] # ?
' & { } " ^ ! * / | - \ ~ +

White space Characters
White space characters include blank space, new line, horizontal tab, carriage return and form feed.

Constant and variable –needs and definition

Constants -needs and definition
constant cannot be changed during the execution of a program. In C, constants can be classified as:



Numeric constants are divided into two types: integer and real constants.
Integer constants are numeric constants without any fractional part. They may be positive or negative. Spaces, commas (,), and non-digit characters are not allowed.
For example: 123, +12, -12.

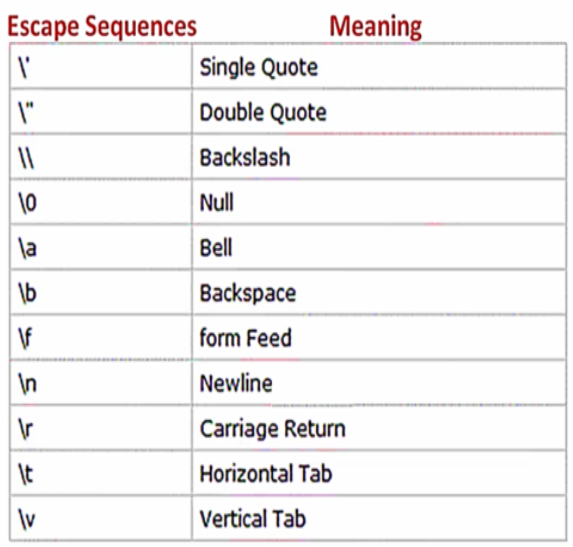
There are three types of integer constants in the C language: decimal constants (base 10), octal constants (base 8), and hexadecimal constants (base 16).
Decimal digits: 0 1 2 3 4 5 6 7 8 9
Octal digits: 0 1 2 3 4 5 6 7
Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F.
For example:
Decimal constants: 0, -9, 22 etc
Octal constants: 021, 077, 033 etc
Hexadecimal constants: 0x7f, 0x2a, 0x521 etc

Real (Floating-point) constants are numeric constants with a fractional part. They may be positive or negative. Spaces, commas (,), and non-digit characters are not allowed. For example: 12.3, +1.2, -1.2.

Character constants are divided into two types: Character and String constants.
Character constants are constants that use single quotation marks around character. For example: 'a', 'l', 'm', 'F' etc.

String constants are constants that are enclosed in a pair of double quotation marks.
For example:
"good" //string constant
"" //null string constant
" " //string constant of six white space
"x" //string constant having a single character.
"Earth is round\n" //prints string with newline

Escape Sequences Character / Backslash Character Constants
The escape sequence in C is the characters or the sequence of characters that can be used inside the output function. A list of escape sequence character constants is given in the following table.



Escape Sequences	Meaning
\'	Single Quote
\"	Double Quote
\\	Backslash
\0	Null
\a	Bell
\b	Backspace
\f	form Feed
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab

Variables –needs and definition

A variable is a data name that may be used to store a data value. Value of the variable may be changed during program execution.

The rules for writing a variable name are as follows:

- The first character must be a letter, and the remaining characters may be letters, numbers, or underscores.
- Uppercase and lowercase letters are considered different.
- Spaces and special characters, except for underscores, are not allowed.
- The variable name should not be a keyword and should be short and meaningful.

Some examples of valid variable names are mark and sum_1
Invalid examples include 1_sum and $name

Variables can be assigned using the assignment operator with the syntax variable = value.
For example, A = 10.
It is also possible to assign a value at the time of variable declaration with the syntax
Datatype variable = value.
For example, int A = 10.
In C, it is possible to initialize more than one variable in one statement using multiple assignment
operators.
Example void main()

```
    {
            int a,b,c;
            a=b=c=10;
    }
```

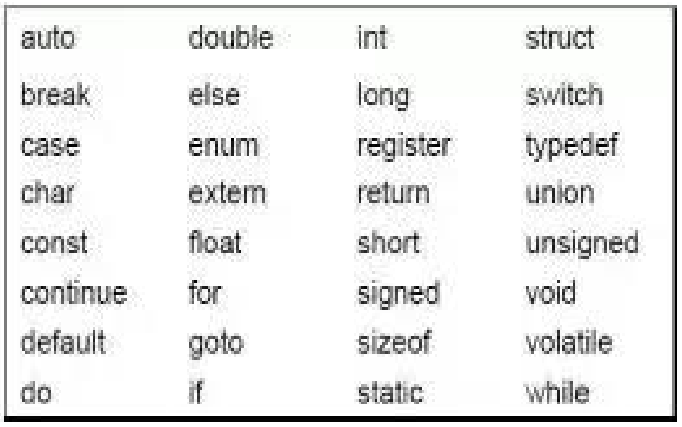
1.2.3 Identifiers, literals, Key words

Identifiers

Identifiers are names given to variables, functions, structures etc. For example: int A;
Here, A is an identifier that denotes a variable of type integer.

Keywords

Keywords are reserved words with fixed meanings that cannot be altered by the user.
For example, in the statement int A; the keyword "int" indicates that 'A' is of type integer.
C programming is case-sensitive; all keywords must be written in lowercase. Here is a list of all keywords predefined by C.



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

1.2.4 Data types (signed and unsigned) (Numeric : int, short int, long, float,double) , (Character type: char, string) and void.

In C, each variable is associated with a data type that specifies the type of data the variable can store, such as integer, character, floating point, double, etc. Each data type requires varying amounts of memory and has specific operations that can be performed on it.

In C, data types can be classified as follows:

Primitive Data Types: These are the most basic data types used to represent simple values like integers, floats, and characters.
User-Defined Data Types: These are defined by the user. Example class, structure, union , enum and typedef.
Derived Data Types: These types are derived from primitive or built-in data types. Example
array, function, pointer and reference.

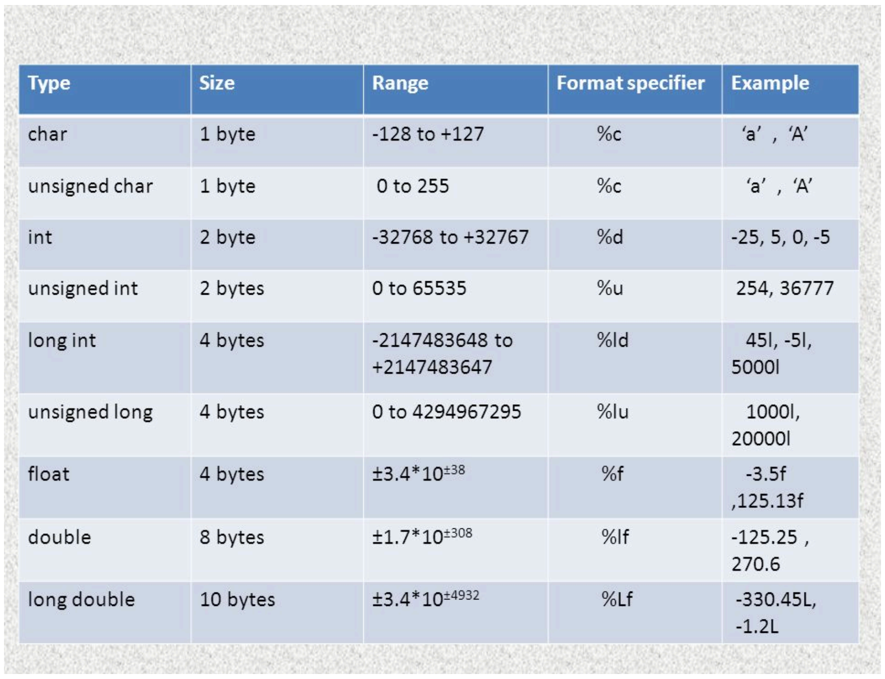
Syntax for declaration of a variable   Datatype variable_name;

Integer data types are declared using the keyword "int" to specify variables with integer values. For example, in the statement int A;, A represents a variable of integer type.

Floating point variables can store real values, such as 2.34 or -9.382. The keywords "float" or "double" are used to declare floating point variables.For example, you can declare a float variable as float B; and a double variable as double C;In this case, both B and C are floating point variables.
If the precision provided by a float is not enough, a double type is used. If even a double is insufficient, a long double type is used.

For character types, the keyword "char" is used to declare variables of character type. For example: char n='h'; char name[10]=”Kavita”

In this case, n and name are character type variables. The table below displays the size and range of various data types.



Type	Size	Range	Format specifier	Example
char	1 byte	-128 to +127	%c	‘a’ , ‘A’
unsigned char	1 byte	0 to 255	%c	‘a’ , ‘A’
int	2 byte	-32768 to +32767	%d	-25, 5, 0, -5
unsigned int	2 bytes	0 to 65535	%u	254, 36777
long int	4 bytes	-2147483648 to +2147483647	%ld	45l, -5l, 5000l
unsigned long	4 bytes	0 to 4294967295	%lu	1000l, 20000l
float	4 bytes	$\pm 3.4*10^{\pm 38}$	%f	-3.5f ,125.13f
double	8 bytes	$\pm 1.7*10^{\pm 308}$	%lf	-125.25 , 270.6
long double	10 bytes	$\pm 3.4*10^{\pm 4932}$	%Lf	-330.45L, -1.2L

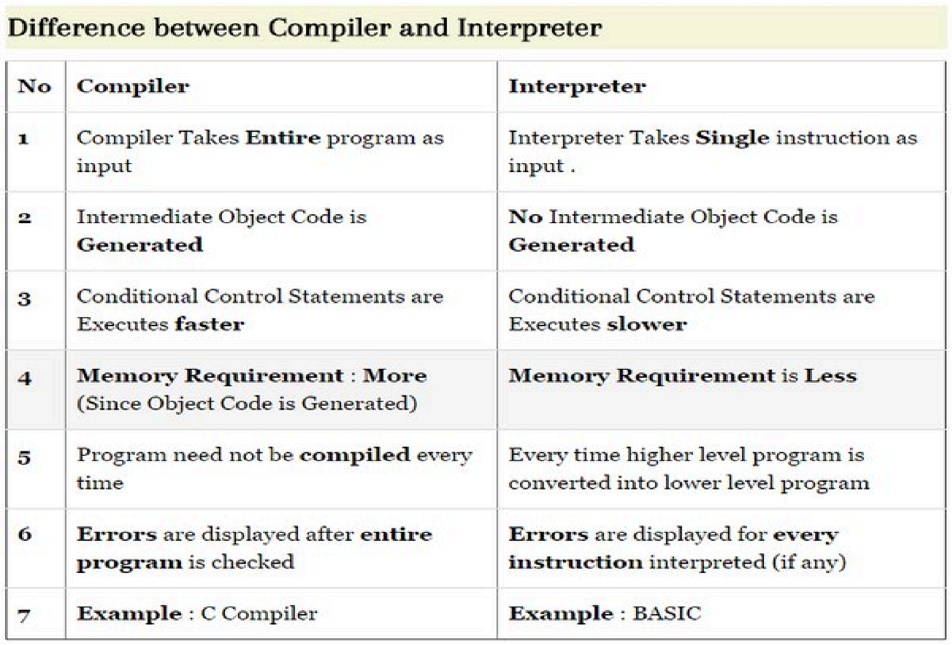
The void type has no values and is typically used to specify the type of functions. A function is said to have a void type when it does not return any value to the calling function.

1.2.5 Concepts of source code, object code and executable code.

Source code: The program written in a high-level language (HLL) is known as source code. The source code is provided as input to the compiler.

Object code: The program written in HLL and converted to machine-level language (MLL) is known as object code. Object code is also referred to as binary code or machine code.

Translator: It converts the source program (HLL) into the object program (MLL). Both compiler and interpreter serve as translators. The main difference between a compiler and an interpreter is explained below.



Difference between Compiler and Interpreter

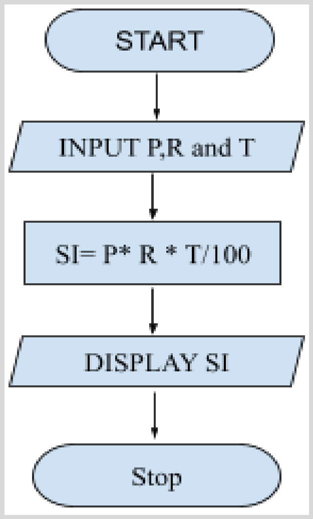
No	Compiler	Interpreter
1	Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input .
2	Intermediate Object Code is **Generated**	**No** Intermediate Object Code is **Generated**
3	Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
4	**Memory Requirement** : **More** (Since Object Code is Generated)	**Memory Requirement** is **Less**
5	Program need not be **compiled** every time	Every time higher level program is converted into lower level program
6	**Errors** are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)
7	**Example** : C Compiler	**Example** : BASIC

Algorithm and flowchart exercises with solutions.

1. Write an algorithm and draw a flowchart to compute the simple interest on principal P, rate R% and duration T years.

   Algorithm
   Step 1: Start
   Step 2: Input the values of P, R, and T.
   Step 3: Calculate the simple interest using the formula: SI = P * R * T / 100.
   Step 4: Display the value of SI.
   Step 5: Stop

Flowchart

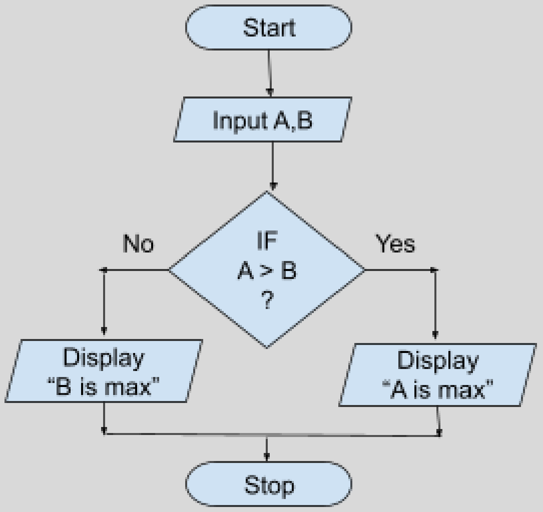


2. Write an algorithm and draw a flowchart to find the max number from two numbers.

   Algorithm
   Step 1: Start

```
Step 2: Input the values of A and B.
Step 3: If A is greater than B, then
            Print "A is max".
        Else
            Print "B is max".
Step 4: Stop
```

Flowchart

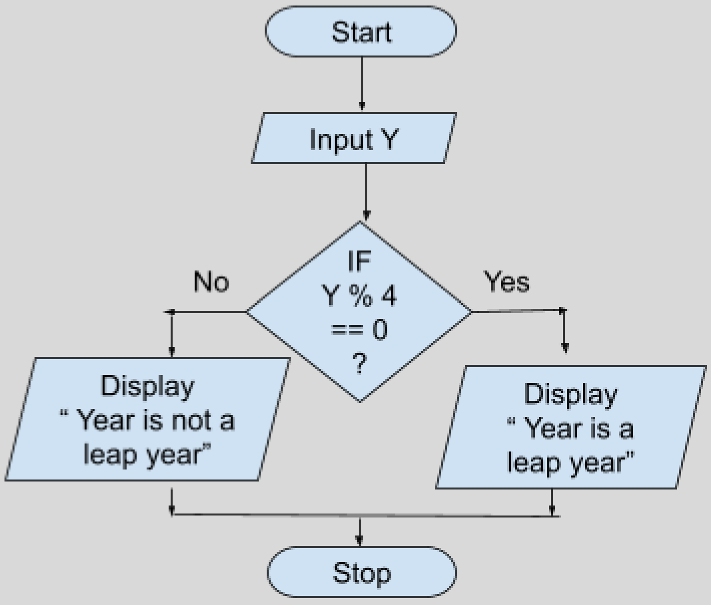


3. Write an algorithm and draw a flowchart to determine whether a year is a leap year.

Algorithm

```
Step 1: Start
Step 2: Input a year(Y).
Step 3: If  Y % 4 == 0, then
            Print "Year is a leap year".
        Else
            Print "Year is not a leap year".
Step 4: Stop
```

Flowchart

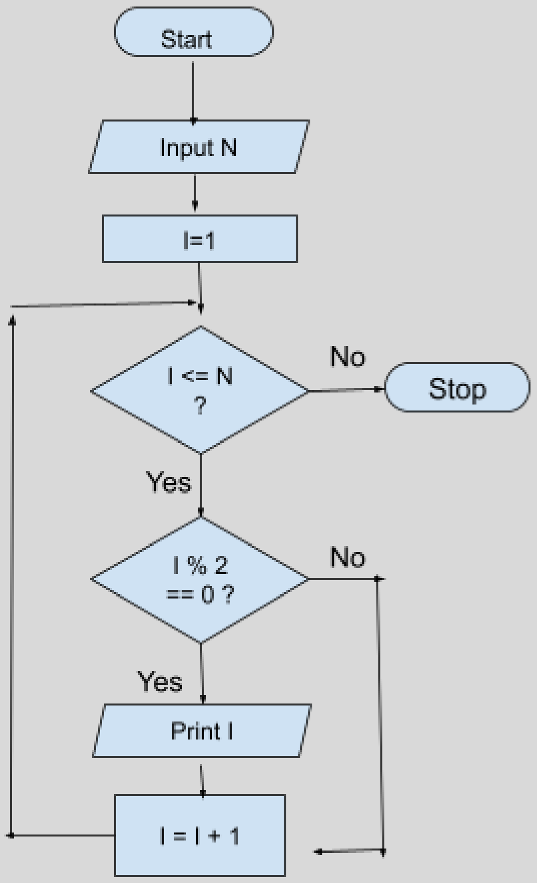

Start
Input Y
IF Y % 4 == 0 ?
No
Yes
Display " Year is not a leap year"
Display " Year is a leap year"
Stop

4. Write an algorithm and draw a flowchart to print an even number between 1 to N.

Algorithm

```
Step 1: Start
Step 2: Input a number N.
Step 3: Declare I to 1.
Step 4: Repeat steps (a) and (b) until I<=N
            (a) if I % 2 == 0
                    print I
            (b) I=I+1
Step 5: Stop
```

Flowchart

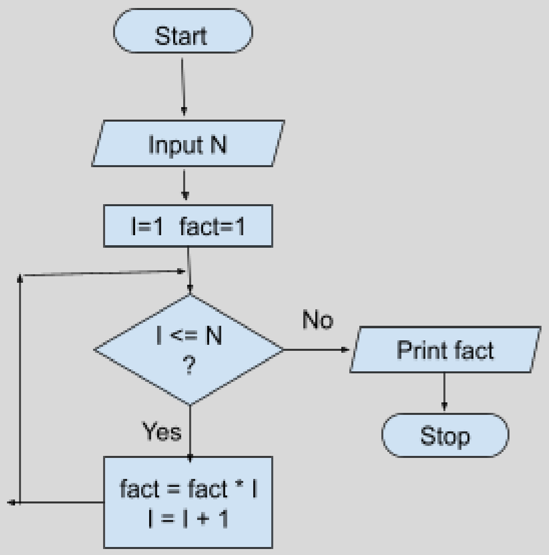


5. Write an algorithm and draw a flowchart to calculate the factorial of N.

Algorithm

```
Step 1: Start
Step 2: Input a number N.
Step 3: Declare fact to 1 and I to 1
Step 4: Repeat steps (a) to (b) until I <= N
            (a) fact= fact * I
            (b) I=I+1
Step 5: Print fact
Step 6: Stop
```

Flowchart


Start
Input N
I=1 fact=1
I <= N ?
No
Print fact
Stop
Yes
fact = fact * I
I = I + 1