UNIT — I

Computer programming:

Computer programs are written using one of the programming language. A program has a set of instructions written in correct order to get the desired result. The method of writing the instructions to solve the given problem is called programming.

Programming techniques:

There are two types of programming techniques commonly used:

1. Procedural or structural programming
2. Object oriented programming

Object-oriented programming :-

Object-oriented programming (**OOP**) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs. Objective-C, Smalltalk, and Java are examples of object-oriented programming languages.

Structured programming :-

Structured programming is a programming paradigm aimed on improving the clarity, quality, and development time of a computer program by making extensive use of subroutines, block structures and for and while loops—in contrast to using simple tests and jumps such as the go to statement which could lead to "spaghetti code" which is both difficult to follow and to maintain.
At a low level, structured programs are often composed of simple, hierarchical program flow structures. These are sequence, selection, and repetition:

- "Sequence" refers to an ordered execution of statements.
- In "selection" one of a number of statements is executed depending on the state of the program. This is usually expressed with keywords such asif..then..else..endif, switch, or case. In some languages keywords cannot be written verbatim, but must be stropped.
- In "repetition" a statement is executed until the program reaches a certain state, or operations have been applied to every element of a collection. This is usually expressed with keywords such as while, repeat, for or do..until. Often it is recommended that each loop should only have one entry point (and in the original structural programming, also only one exit point, and a few languages enforce this).

Advantages of Structured programming:-

1. **Easy to write**
2. **Easy to debug**
3. **Easy to Understand**
4. **Easy to Change**

Algorithm:

- ❖ An algorithm is a finite sequence of step by step, discrete, unambiguous instructions for solving a particular problem
  - has input data, and is expected to produce output data
  - each instruction can be carried out in a finite amount of time in a deterministic way
- ❖ In simple terms, an algorithm is a series of instructions to solve a problem (complete a task)
- ❖ Problems can be in any form
  - Business
    - ❖ Get a part from Vancouver to Ottawa by morning
    - ❖ Allocate manpower to maximize profit
  - Life
    - ❖ I am hungry. How do I order pizza?

- Explain how to tie shoelaces to a five year old child

Algorithmic Representation of Computer Functions

❖ Input	
– Get information	input
❖ Storage	
– Store information	Given/Result
❖ Process	
– Arithmetic	Let (assignment command)
– Repeat instructions	Loop
– Branch conditionals	If
❖ Output	
– Give information	print

Features of Algorithm:-

According to D.E.Knuth, a pioneer in the computer science discipline, an algorithm has five important features they are as follows

1. Finiteness
2. Definiteness
3. Effectiveness
4. Input
5. Output

Advantages of algorithm

- it is a step-by-step rep. of a solution to a given prblem ,which is very easy to understand
- it has got a definite procedure.
- it easy to first develope an algorithm,&then convert it into a flowchart &then into a computer program.
- it is independent of programming language.
- it is easy to debug as every step is got its own logical sequence.

Disadvantages of algorithm

It is time consuming & cubersome as an algorithm is developed first which is converted into flowchart &then into a computer program.

Example :-

Write an algorithm that reads two values, determines the largest value and prints the largest value with an identifying message.

```
Step 1:         Input VALUE1, VALUE2
Step 2: if (VALUE1 > VALUE2) then
                      MAX ← VALUE1
                else
                      MAX ← VALUE2
                endif
Step 3: Print "The largest value is", MAX
```

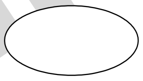
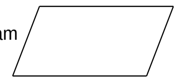
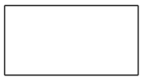
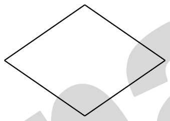
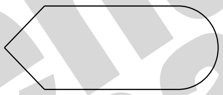
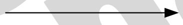
Flowchart:-

A **flowchart** is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.This diagrammatic representation illustrates a solution to a given problem. Process operations are represented in these boxes, and arrows; rather, they are implied by the sequencing of operations. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.[

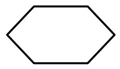
- **(Dictionary) A schematic representation of a sequence of operations, as in a manufacturing process or computer program.**
- **(Technical) A graphical representation of the sequence of operations in an information system or program. Information system flowcharts show how data flows from source documents through the computer to final distribution to users. Program flowcharts show the sequence of instructions in a single program or subroutine.**

Symbols:-

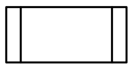
Different symbols are used to draw each type of flowchart.

Name	Symbol	Use in Flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out e.g. addition, subtraction, division etc.
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program

2. Additional Symbols
Related to more advanced programming



Preparation (may be used with "do loops" explained later)

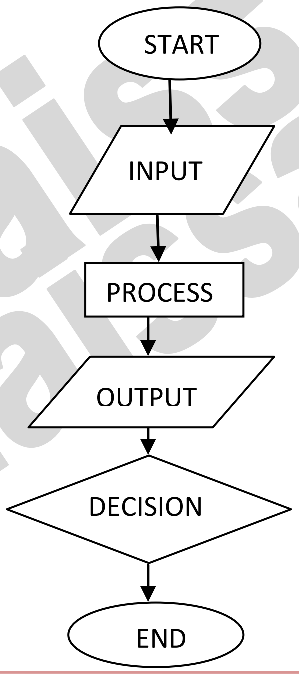


Refers to separate flowchart ("Subprograms"(explained later) are shown in separate flowcharts).

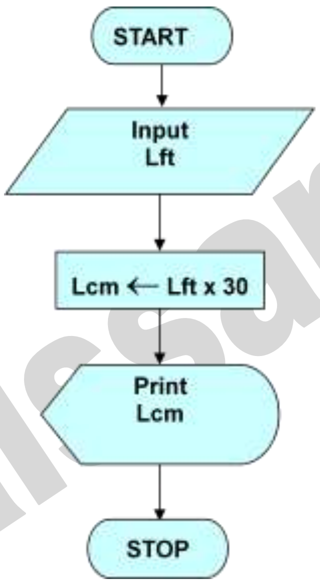
Types of flowchart:-
Sterneckert (2003) suggested that flowcharts can be modeled from the perspective of different user groups (such as managers, system analysts and clerks) and that there are four general types:

- *Document flowcharts*, showing controls over a document-flow through a system
- *Data flowcharts*, showing controls over a data-flow in a system
- *System flowcharts* showing controls at a physical or resource level
- *Program flowchart*, showing the controls in a program within a system

Program Flowchart
- shows the sequence of instructions in a program or subroutine. These instructions are followed to procedure the needed output.



Write an algorithm and draw a flowchart to convert the length in feet to centimeter



Advantages and limitation of flowchart:-

1. Better communication
2. Proper program documentation
3. Efficient coding
4. Systematic debugging
5. Systematic testing

Limitation of flowchart:-

1. Flowchart are very time consuming and laborious to draw.
2. There are no standards determining the amount of detail that should be included in flowchart.
3. Owing to the symbol-string nature of flowcharting, any change or modification in the program logic will usually require a completely new flowchart.

Making of sequence, selection and iteration:

The concept of structured programming says that any programming logic problem can be solved using an appropriate combination of only three programming structures, none of which are complicated. The three structures are known generally as:

- Sequence
- Selection or decision
    - If statement
    - Switch
- Iteration

- Looping

Sequence:-
The general requirement for the sequence structure is that one or more actions may be performed in sequence after entry and before exit.
With the exception discussed below, there may not be any branches or loops between the entry and the exit.
All actions must be taken in sequence.

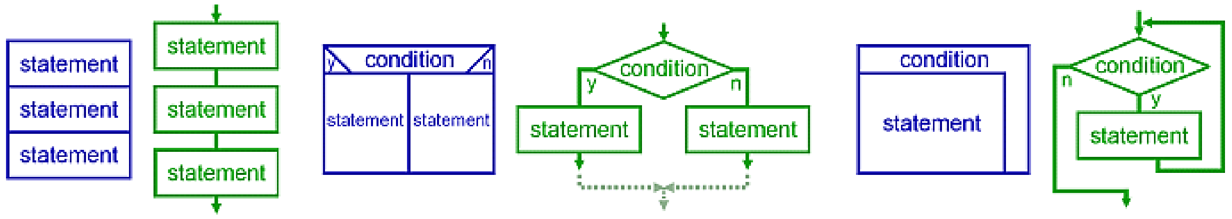
```
Enter
  Perform one or more actions in sequence
Exit
```

Selection or decision:-
There is only one entry point and one exit point.
The first thing that happens following entry is that some condition is tested for true or false.
If the condition is true, one or more actions are taken in sequence and control exits the structure.
If the condition is false, **none**, one or more actions are taken in sequence and control exits the structure.

```
Enter
  Test a condition for true or false
  On true
    Take one or more actions in sequence
  On false
    Take none, one, or more actions in sequence
Exit
```

Iteration:-
Perform the test and exit on false
Perform some actions and repeat the test on true
Each action element may be another structure
Need to avoid infinite loops

```
Enter
  Test a condition for true or false
  Exit on false
  On true
    Perform one or more actions in sequence
    Go back and test the condition again
```



Translator:-

Assembler

An **assembler** translates assembly language into machine code. Assembly language consists of mnemonics for machine opcodes so assemblers perform a 1:1 translation from mnemonic to a direct instruction. For example:
LDA #4 converts to 0001001000100100
Conversely, one instruction in a high level language will translate to one or more instructions at machine level.

Advantages of using an Assembler:

- Very fast in translating assembly language to machine code as 1 to 1 relationship
- Assembly code is often very efficient (and therefore fast) because it is a low level language
- Assembly code is fairly easy to understand due to the use of English-like mnemonics

Disadvantages of using Assembler:

- Assembly language is written for a certain instruction set and/or processor
- Assembly tends to be optimised for the hardware it's designed for, meaning it is often incompatible with different hardware
- Lots of assembly code is needed to do relatively simple tasks, and complex programs require lots of programming time

Compiler

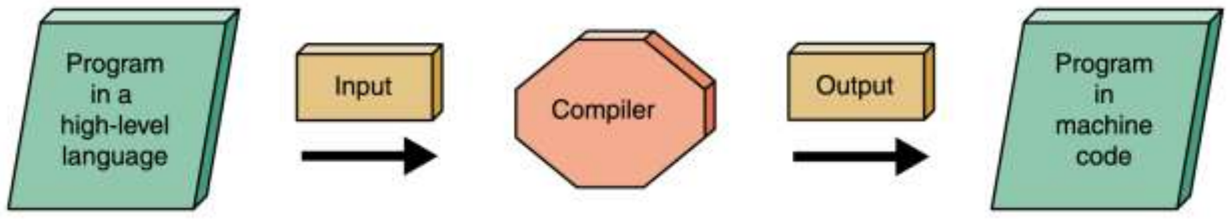
A **Compiler** is a computer program that **translates code** written in a high level language to a lower level language, object/machine code. The most common reason for translating source code is to create an executable program (converting from a high level language into machine language).

Advantages of using a compiler

- ource code is not included, therefore compiled code is more secure than interpreted code
- Tends to produce faster code than interpreting source code
- Produces an executable file, and therefore the program can be run without need of the source code

Disadvantages of using a compiler

- Object code needs to be produced before a final executable file, this can be a slow process
- The source code must be 100% correct for the executable file to be produced



Compilation process

Interpreter

An interpreter program executes other programs directly, running through program code and executing it line-by-line. As it analyses every line, an interpreter is slower than running compiled code but it can take less time to interpret program code than to compile and then run it — this is very useful when prototyping and testing code. Interpreters are written for multiple platforms, this means code written once can be run immediately on different systems without having to recompile for each. Examples of this include flash based web programs that will run on your PC, MAC, games console and Mobile phone.

Advantages of using an Interpreter

- Easier to debug(check errors) than a compiler.
- Easier to create multi-platform code, as each different platform would have an interpreter to run the same code.
- Useful for prototyping software and testing basic program logic.

Disadvantages of using an Interpreter

- Source code is required for the program to be executed, and this source code can be read making it insecure
- Interpreters are generally slower than compiled programs due to the per-line translation method

Linker and Loader:-

A linker or link editor is a computer program that takes one or more object files generated by a compiler and combines them into a single executable program.
A loader brings object program into memory for execution. System program that performs Loading. Some loaders also do relocation and linking.

Absolute loader:- No linking or relocation. All functions are performed in one pass.
E.g. a Bootstrap Loader

Computer programs typically comprise several parts or modules; all these parts/modules need not be contained within a single object file, and in such case refer to each other by means of symbols. Typically, an object file can contain three kinds of symbols:

- defined symbols, which allow it to be called by other modules,
- undefined symbols, which call the other modules where these symbols are defined, and

- local symbols, used internally within the object file to facilitate relocation.

For most compilers, each object file is the result of compiling one input source code file. When a program comprises multiple object files, the linker combines these files into a unified executable program, resolving the symbols as it goes along.

Linkers can take objects from a collection called a *library*. Some linkers do not include the whole library in the output; they only include its symbols that are referenced from other object files or libraries. Libraries exist for diverse purposes, and one or more system libraries are usually linked in by default.

The linker also takes care of arranging the objects in a program's address space. This may involve *relocating* code that assumes a specific base address to another base. Since a compiler seldom knows where an object will reside, it often assumes a fixed base location (for example, zero). Relocating machine code may involve re-targeting of absolute jumps, loads and stores.
The executable output by the linker may need another relocation pass when it is finally loaded into memory (just before execution). This pass is usually omitted on hardware offering virtual memory: every program is put into its own address space, so there is no conflict even if all programs load at the same base address. This pass may also be omitted if the executable is a position independent executable.

Unit II

History of C:-

C language is a structure oriented programming language, was developed at Bell Laboratories in 1972 by Dennis Ritchie

C language features were derived from earlier language called “B” (Basic Combined Programming Language – BCPL)

C language was invented for implementing UNIX operating system

In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C Programming Language” and commonly known as K&R C

In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

C standards

C89/C90 standard – First standardized specification for C language was developed by American National Standards Institute in 1989. C89 and C90 standards refer to the same programming language.

C99 standard – Next revision was published in 1999 that introduced new futures like advanced data types and other changes.

Features of C language:

- Reliability
- Portability
- Flexibility
- Interactivity
- Modularity
- Efficiency and Effectiveness

Uses of C language:

C language is used for developing system applications that forms major portion of operating systems such as Windows, UNIX and Linux. Below are some examples of C being used.

- Database systems
- Graphics packages
- Word processors
- Spread sheets
- Operating system development
- Compilers and Assemblers
- Network drivers
- Interpreters

We are going to learn a simple "Hello World" program in this section. Functions, syntax and the basics of a C program are explained below.

C Basic Program:

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

void main()
{
/* Our first simple C basic program */
printf("Hello World! ");
getch();
}
```

Output:

```
Hello World!
```

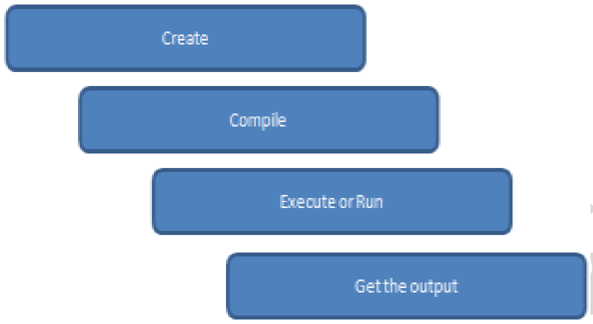
Explanation for above C basic Program:

Let's see all the sections of the above simple C program line by line.

S.no	Command	Explanation
1	#include <stdio.h>	This is a preprocessor command that includes standard input output header file(stdio.h) from the C library before compiling a C program
2	#include <conio.h>	Header file used mostly by MS-DOS compilers to provide console input/output.
3	int main()	This is the main function from where execution of any C program begins.
4	{	This indicates the beginning of the main function.
5	/*_some_comments_*/	whatever is given inside the command "/* */" in any C program, won't be considered for compilation and execution.
6	printf("Hello_World!");	printf command prints the output onto the screen.
7	getch();	This command waits for any character input from keyboard.
8	return 0;	This command terminates C program (main function) and returns 0.
9	}	This indicates the end of the main function.

Steps to write C programs and get the output:

Below are the steps to be followed for any C program to create and get the output. This is common to all C program and there is no exception whether its a very small C program or very large C program.



Creation, Compilation and Execution of a C program:

Prerequisite:

- ✶ If you want to create, compile and execute C programs by your own, you have to install C compiler in your machine. Then, you can start to execute your own C programs in your machine.
- ✶ You can refer below link for how to install C compiler and compile and execute C programs in your machine.
- ✶ Once C compiler is installed in your machine, you can create, compile and execute C programs as shown in below link.

Basic structure of C program:

Structure of C program is defined by set of rules called protocol, to be followed by programmer while writing C program. All C programs are having sections/parts which are mentioned below.

1. Documentation section
2. Link Section
3. Definition Section
4. Global declaration section
5. Function prototype declaration section
6. Main function
7. User defined function definition section

Printf and Scanf

C printf():

The printf() function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen.

To display the value of an integer variable, we use printf statement with the `%d` format specifier.

Similarly `%c` for character, `%f` for float variable,`%s` for string variable, `%lf` for double, `%x` for hexadecimal variable.

To generate a newline,we use \n in C printf statement.

%d got replaced by value of an integer variable(no),
%c got replaced by value of a character variable(ch),
%f got replaced by value of a float variable(flt),
%lf got replaced by value of a double variable(dbl),
%s got replaced by value of a string variable(str),

C scanf :

scanf() function is used to read a character, string, numeric data from keyboard.
Consider the below example where the user enters a character and assign it to the variable ch and enters a string and assign it to the variable str.
The format specifier %d is used in scanf statement so that the value entered is received as an integer and %s for string.
Ampersand is used before the variable name ch in scanf statement as &ch. It is just like in a pointer which is to point to the variable.

Data Types

- ✶ C data types are defined as the data storage format that a variable can store a data to perform a specific operation.
- ✶ Data types are used to define a variable before to use in a program.
- ✶ Size of variable, constant and array are determined by data types.

C – data types:
There are four data types in C language. They are,

S.no	Types	Data Types
1	Basic data types	int, char, float, double
2	Enumeration data type	enum
3	Derived data type	pointer, array, structure, union
4	Void data type	void

1. Basic data types
Integer data type:

This data type allows a variable to store numeric values. **int** keyword is used to refer integer data type. The storage size of int data type is 2 or 4 or 8 byte. It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for int data type. Like wise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for int.

int(2 byte) can store values from -32,768 to +32,767
int(4 byte) can store values from -2,147,483,648 to +2,147,483,647.
If you want to use the integer value that crosses the above limit, you can go for long int and long long int for which the limits are very high.

Character data type:
This data type allows a variable to store only one character. Storage size of char data type is 1. We can store only one character using char data type. char keyword is used to refer character data type.
For example, 'A' can be stored using char datatype.
You can't store more than one character using char data type.

Floating point data type:
Floating point data type consists of 2 types. They are, float and double.
float:
Float data type allows a variable to store decimal values. storage size of float data type is 4. This also varies depend upon the processor in the CPU. We can use upto 6 digits after decimal using float data type.
For example, 10.456789 can be stored in a variable using float data type.

double:
Double data type is also same as float data type which allows upto 10 digits after decimal. and the range is from 1E–37 to 1E+37.

C – sizeof() function:
sizeof () operator is used to find the memory space allocated for each C data types.

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

int main()
{

   int a;
   char b;
   float c;
   double d;
   printf("Storage size for int data type:%d \n",sizeof(a));
   printf("Storage size for char data type:%d \n",sizeof(b));
   printf("Storage size for float data type:%d \n",sizeof(c));
   printf("Storage size for double data type:%d \n",sizeof(d));
   return 0;
}
```

Output:

```
Storage size for int data type:4
Storage size for char data type:1
Storage size for float data type:4
Storage size for double data type:8
```

C – Modifiers

The
amount of memory space to be allocated for a variable is derived by modifiers. Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
For example, storage space for int data type is 4 byte for 32 bit processor.
We can increase the range by using long int which is 8 byte.
We can decrease the range by using short int which is 2 byte.

There are 5 modifiers available in C language. They are,

1. short
2. long
3. signed
4. unsigned
5. long long

Below table gives the detail about the storage size of each C basic data type in 16 bit processor. Please keep in mind that storage size and range for int and float will vary depend on the CPU processor (8,16, 32 and 64 bit)

S.No	C Data types	storage Size	Range
1	char	1	–127 to 127
2	unsigned char	1	0 to 255
3	int	2	–32,767 to 32,767
4	float	4	1E–37 to 1E+37 with six digits of precision
5	double	8	1E–37 to 1E+37 with ten digits of precision
6	long double	10	1E–37 to 1E+37 with ten digits of precision

7	long int	4	–2,147,483,647 to 2,147,483,647
8	short int	2	–32,767 to 32,767
9	unsigned short int	2	0 to 65,535
10	signed short int	2	–32,767 to 32,767
11	long long int	8	–(2power(63) –1) to 2(power)63 –1
12	signed long int	4	–2,147,483,647 to 2,147,483,647
13	unsigned long int	4	0 to 4,294,967,295
14	unsigned long long int	8	2(power)64 –1

2. Enumeration data type:

Enumeration data type consists of named integer constants as a list. It start with 0(zero) by default and value is incremented by 1 for the sequential identifiers in the list.2. To invent own data type and define the values of the variable

```
Eg. Enum mar_status
{
Single, married, divorced, widowed
};
Enum mar_status person1, person2;
```

The first part declares the data type and specifies its possible values
The second part declares variable of this data type.
We can give values

```
Person1=married;
Person2=divorced;
```

3. Derived data type:

Array, pointer, structure and union are called derived data type.

4. Void data type:

Void is an empty data type that has no value. This can be used in functions and pointers.

C – Tokens and keywords
C Tokens:

C Tokens: These are the basic buildings blocks in C language which are constructed together to write a C program.

Each and every smallest individual unit in a C program is known as C Tokens.

C Tokens are of six types.

1) Keywords	(eg: int, while),
2) Identifiers	(eg: main, total),
3) Constants	(eg: 10, 20),
4) Strings	(eg: “total”, “hello”),
5) Special symbols	(eg: (), {}),
6) Operators	(eg: +, /,-,*)

C – Identifiers:
Each program elements in a C program are given a name called identifiers.
Names given to identify Variables, functions and arrays are examples for identifiers. eg. x is a name given to integer variable in above program.

Rules for constructing identifier name:
1. First character should be an alphabet or underscore.
2. Succeeding characters might be digits or letter.
3. Punctuations and special characters aren’t allowed except underscore.
4. Identifiers should not be keywords.

C – Keywords
Keywords are pre-defined words in a C compiler.
Each keyword is meant to perform a specific function in a C program.
Since keywords are referred names for compiler, they can’t be used as variable name.

C language supports 32 keywords which are given below.

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

C – Constant
C Constant are also like normal variables except their values cannot be modified by the program once they are defined. They refer to fixed values. They are also called as literals

They may be belonging to any of the data type. Please see below table for constants with respect to their data type.

Types of C constant:

1. Integer constants
2. Real or Floating point constants
3. Octal & Hexadecimal constants
4. Character constants
5. String constants
6. Backslash character constants

S.no	Constant type	data type	Example
1	Integer constants	int unsigned int long int long long int	53 , 762 , -478 , etc 5000u , 1000U , etc 1000L , -300L , etc 5555555LL
2	Real or Floating point constants	float, doule	111.22F , 2.22e-2f 111.22 , 4.0 , -0.34565
3	Octal constant	int	013 /* starts with 0 */
4	Hexadecimal constant	int	0×90 /* starts with 0x */
5	character constants	char	‘A’ , ‘B’, ‘C’
6	string constants	char	“ABCD” , “Hai”

Rules for constructing C constant:

✶ **Integer Constants**
1. An integer constant must have at least one digit.
2. It must not have a decimal point.
3. It can either be positive or negative.
4. No commas or blanks are allowed within an integer constant.
5. If no sign precedes an integer constant, it is assumed to be positive.
6. The allowable range for integer constants is -32768 to 32767

✶ **real Constants**
1. A real constant must have at least one digit
2. It must have a decimal point
3. It could be either positive or negative
4. If no sign precedes an integer constant, it is assumed to be positive.

5. No commas or blanks are allowed within a real constant.

✶ **character constants**

1. A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas. Both the inverted commas should point to left.
2. For example, 'C' is a valid character constant whereas 'C' is not.
   The maximum length of a character constant is 1 character

Backslash Character Constants:

There are some characters which have special meaning in C language. They should be preceded by back slash symbol to make use of special function of them. Given below is the list of special characters and their purpose.

Character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\a	Alert bell
\t	Horizontal tab

How to use constants in a C program?

We can define constants in a C program in the following ways.

1. By "const" keyword
2. By "#define" preprocessor directive

C – Variable

✶ C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable.

✶ The value of the C variable may get change in the program.

✶ C variable might be belonging to any of the data type like int, float, char etc.

✶

Rules for naming C variable:

1. Variable name must begin with letter or underscore.
2. Variables are case sensitive
3. They can be constructed with digits, letters.
4. No special symbols are allowed other than underscore.
5. sum, height, _value are some examples for variable name

Declaring & initializing C variable:

✶ Variables should be declared in the C program before to use.
✶ Memory space is not allocated for a variable while declaration. It happens only on variable definition.
✶ Variable initialization means assigning a value to the variable.

S.No	Type	Syntax	Example
1	Variable declaration	data_type variable_name;	int x, y, z; char flat, ch;
2	Variable initialization	data_type variable_name = value;	int x = 50, y = 30; char flag = ‘x’, ch=’l’;

There are three types of variables in C program They are,

1. Local variable
2. Global variable

   Environment variable

Operators and Expressions

An operator is a symbol that performs specific mathematical or logical manipulations in program C is flexible and powerful language, because it has rich set of operators.
The symbols which are used to perform logical and mathematical operations are called C operators.
These C operators join individual constants and variables to form expressions. Operators, functions, constants and variables are combined together to form expressions.
Example : A + B * 5
where,

✶ +, * - operators
✶ A, B - variables
✶ 5 – constant
✶ A + B * 5 - expression

Types of C operators:

C language offers many types of operators. They are,

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators

8. Special operators

1) Arithmetic Operators:
These operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus.

S.No.	Operator	Operation
1	+	Addition
2	-	Subtraction
3	*	multiplication
4	/	Division
5	%	Modulus

Operand 1 Operators Operand 2
Eg. 6 + 4
7 x 3
5 / 2
6 / 2

2) Assignment operators: The symbol = is also an operators, and it is evaluated last. Here the equal TO (=) symbol should not be treated as in mathematics. Here it means that the value of the expression in the right-hand side is taken by the variable to the left side.

eg : a = 2 + 7 * 3

3) Increment and Decrement Operator: C provides two special operators not generally found in other languages. These are the increment (++) and decrement the value of a variable by l. These operators are called unary operators and require only one operand. They are of equal precedence.

These operators can be used both as prefix and postfix.

Eg. ++ variable
Variable ++
-- variable
variable --

4) Relational operators : In C there are 6 operators which are used to compare two values or expressions. The values of two expressions with the relational operator forms a relational expressions. These 6 operators.

These operators are used to find the relation between two variables. That is, used to compare the value of two variables.

Operator	Example
>	x > y
<	x < y
>=	x >= y
<=	x <= y
==	x == y
!=	x != y

5) Logical operators: The logical operators are sued to support the basic logical operations of AND, OR C NOT according to the truth table. For combining expressions into logical expression logical operators are used. These are

Symbol	Name
&&	logical AND
! !	logical OR
!	logical Not

(a) Logical AND operators: This is used to combine two expressions in the form **exp 1 & & exp 2.**
If both exp 1 and exp 2 are true then the whole expression is considered as true. If any one of them is false, the whole expression will considered as false.

(b) Logical OR operator: It also combines the two expressions. If any one of the expression is true the whole expression is considered as true.

Exp1 !! Exp2

(c) Logical NOT operator: It inverts the logical value of an expression.

! Exp

6) Bitwise operators: One of C's powerful features is a set of bit manipulation operators. These permit the programmer to access and manipulate individual bits within a piece of data. The various bitwise operators available in C are :

Operator	Meaning
&	bitwise AND
!	bitwise OR
^	bitwise exclusive OR
<<	shift left
~	one's complement

Bitwise operators can operator upon int and char but not on float and double. All are binary operators except ~ which is unary.

(a) Bitwise AND (&): The bitwise AND (&) operator yields 1 if the corresponding bits in both values are 1 otherwise, it yields 0.

eg.
```
   1 0 1 0
 & 0 1 1 0
   -------
   0 0 1 0
```

The use of AND operator is to check whether a particular bit of an operand is ON/OFF.

```
Eg. #include <stdio.h>
    int main()
  {
    int a = 12, b = 25;
    printf("Output = %d", a&b);
  }
```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Output - 00001000 = 8

(b) Bitwise OR (!): Another important bitwise operator is OR operator which is represented by ! The rules that given the value of the resulting bit obtained after OR of two bits is shown below :

!	0	1
0	0	1
1	1	1

The result of an inclusive OR is a 1 if either (or both) of the corresponding bits is 1.

eg.
```
  1 0 1 0
  0 1 1 0
  -------
  1 1 1 0
```

Bitwise OR operator is usually used to put ON a particular bit in a number.

```
Eg. #include <stdio.h>
   int main()
    {
       int a = 12, b = 25;
       printf("Output = %d", a|b);
    }
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Output -  00011101  = 29
```

(c) Bitwise XOR (^): The XOR operator is represented as ^ and is also called on exclusive OR operator. The OR operator returns 1, when any one of the two bits or bith the bits are 1, whereas XOR returns 1 only if one of the two bits is 1. The truth table for the XOR operator is given below :

^	0	1
0	0	1
1	1	0

```
Eg. #include <stdio.h>
   int main()
    {
       int a = 12, b = 25;
       printf("Output = %d", a^b);
    }
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Output -  00010101  = 21
```

(e) Right Shift Operator: Like one's complement operator right shift operator also operates on a single variable. It's represented by >> and it shifts each bit in the operand to right. The number of places the bits are shifted depends on the number following the operand.

Thus, num >>3 would shift all bits three places to the right. Similarly num >> would shift all bits five places to the right.

eg. If num = 110011110 then

num >> 1	given	011001111
num >> 3	gives	000110011
num >> 5	gives	000001100

While bits are shifted to right blanks are created on the left. These left blanks must be filled with zeros.

(e) Left Shift operator: This is similar to the right shift operator the only difference being that the bits are shifted to the left and for each bit shifted, added to the right of the number.

eg. If num =11010111 then

num << 1	gives	10101110
num << 3	gives	10111000
num << 5	gives	11100000

Precedence and order of evaluation of operators :

(f) Bitwise compliment operator: This is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011
____
11011100 = 220 (In decimal)

7) Conditional operators:

Conditional operators return one value if condition is true and returns other value is condition is false.

This operator is also called as ternary operator.

Syntax : (Condition? true_value: false_value);

Example : (A > 100 ? 0 : 1);
Here, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

8. Special Operators:

Below are some of special operators that the C language offers.

S.No.	Operators	Description
1	**&**	This is used to get the address of the variable. Example: &a will give address of a.
2	*****	This is used as pointer to a variable. Example: * a where, * is pointer to the variable a.
3	**Size of ()**	This gives the size of the variable. Example: size of (char) will give us 1.
4	**ternary**	This is ternary operator and act exactly as if … else statement.
5	**Type cast**	Type cast is the concept of modifying variable from one data type to other.

Preprocessor directives

Preprocessor Directives extends the power of C programming language. C Preprocessors Directives is a special set of instruction in program which is processed before handing over the program to the compiler. These instructions are always preceded by a pound sign (#) and NOT terminated by a semicolon. The preprocessor directives are executed before the actual compilation of code begins. Different preprocessor directives (commands) tell the preprocessor to perform different actions. We can categorize the Preprocessor Directives as follows:
File inclusion directives
Macro substitution directives
Conditional compilation directives

Advantages:
It make program development easy
It enhance the readability of the program
It make modification easy
It increase the transportability of the program between different machine architectures.
File inclusion directives
The file inclusion directive is used to include files into the current file. When the preprocessor encounters an #include directive it replaces it by the entire content of the specified file.The file inclusion directive (#include) can be used in following two ways:
#include “file-name”
#include <file-name>

We can see that #include can be used in two ways angular brackets (<>)and inverted commas (“”). The only difference between both expressions is the location (directories) where the compiler is going to look for the file. In the first case where the file name is specified between double-quotes, the compiler will look for the in the current directory that includes the file containing the directive. In case if it is not there the compiler will look the file in the default include directories where it is configured to look for the standard header files. When #include is written with <> it means the file is searched directly where the compiler is configured to look for the standard header files. Therefore, standard header files are usually included in angle-brackets, while other specific header files are included using quotes.

Standard Header Files
It is the set of header files those used for performing various common and standard operations.
stdio.h - Defines core input and output functions
string.h - Defines string handling functions.
time.h - Defines date and time handling functions
math.h - Defines common mathematical functions.
User Defined Header Files
In C Programming Language user can have their own custom header file that provide additional capabilities.

Macro substitution directives
Macro substitution directives are used to define identifier which is being replaced by a pre defined string in program.Macro substitution is process in which preprocessor replaces identifiers by one, or more program statements (like functions) and they are expanded inline.
There are two directives for Macro Definition:
#define – Used to define a macro
#undef – Used to undefine a macro
#define preprocessor
To define preprocessor macros we can use `#define`.
Conditional compilation directives(#ifdef, #ifndef, #if, #endif, #else and #elif)
Conditional Compilation Directives allows you to include or exclude certain part of code only when certain condition met.These macros are evaluated on compile time.

The following directives are included in this category:
- #if
- #elif
- #endif
- #ifdef
- #ifndef

Header Files

A header file is a file with extension **.h** which contains C function declarations and macro definitions and to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that come with your compiler.

You request the use of a header file in your program by including it, with the C preprocessing directive**#include** like you have seen inclusion of **stdio.h** header file, which comes along with your compiler.

Including a header file is equal to copying the content of the header file but we do not do it because it will be very much error-prone and it is not a good idea to copy the content of header file in the source files, specially if we have multiple source file comprising our program.

A simple practice in C or C++ programs is that we keep all the constants, macros, system wide global variables, and function prototypes in header files and include that header file wherever it is required.

Include Syntax

Both user and system header files are included using the preprocessing directive **#include**. It has following two forms:

```
#include <file>
```

This form is used for system header files. It searches for a file named file in a standard list of system directories. You can prepend directories to this list with the -I option while compiling your source code.

```
#include "file"
```

This form is used for header files of your own program. It searches for a file named file in the directory containing the current file. You can pretend directories to this list with the -I option while compiling your source code.