

INTRODUCTION TO PROGRAMMING LANGUAGE

Programming languages are those languages through this we could pass our data and instructions to the computer system.



- ☞ We know that the user can't communicate with computer system directly.
- ☞ If user wants to communicate with computer system, he must have knowledge about any programming language.

So, **Programming languages** can be used to create programs to control the behavior of a machine.

CLASSIFICATION OF PROGRAMMING LANGUAGES:

1. Low Level Languages.
 1. Machine Language.
 2. Assembly Language (ASM).
2. High Level Languages (HLL).
 1. Procedure-Oriented Programming (POP).
 2. Object-Oriented Programming (OOP).

LOW LEVEL PROGRAMMING LANGUAGE

- The term low level means closeness to the way in which the machine has been built.
- Low level languages are machine oriented.
- The low-level languages require extensive knowledge of computer hardware and its configuration.
- Low level languages are divided into **Machine Language** and **Assembly Language**.

1. MACHINE LANGUAGE:

Machine language is the only language that is directly understood by the computer. The machine language code is written as string of **1's** and **0's**. In this not need any translator program.

Example:

1001001000010001111

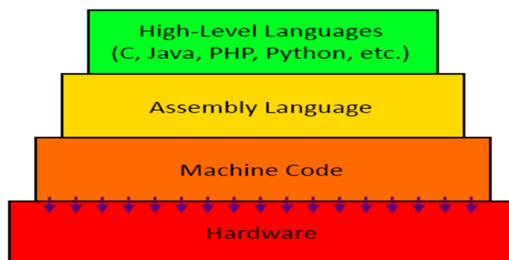
2. ASSEMBLY LANGUAGE:

In this some combination of letters can be used to substitute of machine code, which is called **mnemonics**. Some examples of mnemonics are MOV, CALL, ADD, SUB, MUL, and DIV etc.

2. HIGH LEVEL LANGUAGE (HLL):

In this the instruction are usually written in English like language statement. These languages are employed for easy and speedy development of program. A translator program is required to translate

the HLL to Machine Language. Examples of **HLL** are COBOL, FORTRAN, BASIC, C, C++, Java etc.



HIERARCHY OF PROGRAMMING LANGUAGE

DIFFERENCE BETWEEN LOW-LEVEL AND HIGH-LEVEL LANGUAGE:

S.N.	LOW LEVEL LANGUAGE	HIGH LEVEL LANGUAGE
1.	Low level languages are difficult to learn.	High level languages are easy to learn.
2.	Low level languages are far from human language means these are close to the machine.	High level languages are near to human language.
3.	Program in low level language is fast in execution.	Programs in high-level languages are slow in execution because the translation program is required to translate the HLL code into machine understandable code.
4.	Deep knowledge of hardware is required to write programs.	Knowledge of hardware is not required to write programs.
5.	These languages are normally used to write hardware programs.	These languages are normally used to write application programs.

LANGUAGE TRANSLATOR

A **program** is a set of instructions for performing a particular task. These instructions are just like English words. A program can be written in any programming language. This written program is called the **source program**.

We know that the computer interprets the instructions only as **1's** and **0's**, so we required converting the source program into *machine code*, which is called an **object code** or **binary code**. The language translators are classified in three categories.

1. Assembler.
2. Compiler.
3. Interpreter.

ASSEMBLER:

An Assembler is used to translate the symbolic code of assembly language into machine language instructions.



COMPILER:

The compiler is used to convert the HLL source code into machine code and then saves it to memory. The compiler converts the whole program in one session and reports errors if detected.



INTERPRETER:

Similar to compiler the interpreter also converts the HLL source code into machine code but it reads program in line-by-line manner.



LINKER AND LOADER:

The *linker* and *loader* are the utility program that play a major role in the execution of a program.

The linker is a program that links and merges various object files together in order to make an executable file.

The loader is a part of operating system (OS) and is responsible for loading executable files into memory and execute them.

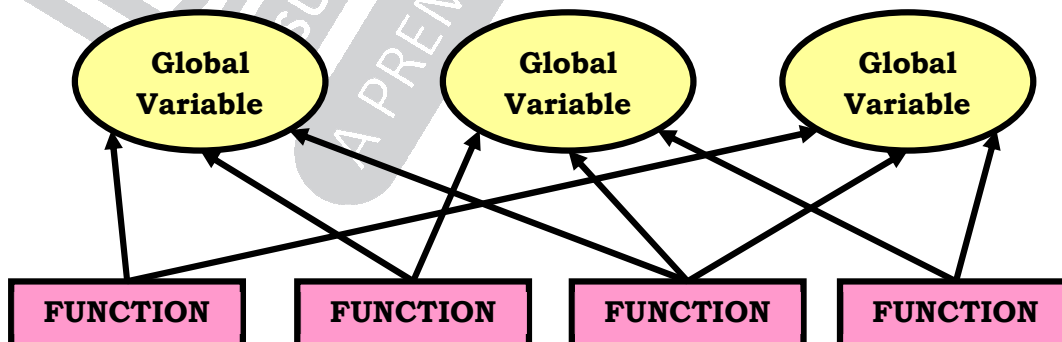


TYPES OF HLL

PROCEDURAL ORIENTED PROGRAMMING:

Procedural language is programming technique based upon the concept of the **procedure** or **function**, means procedural language are those languages which are function oriented.

Functions are a set of instructions which perform some particular tasks. Functions are called repeatedly in a program to execute tasks which performed by these functions.

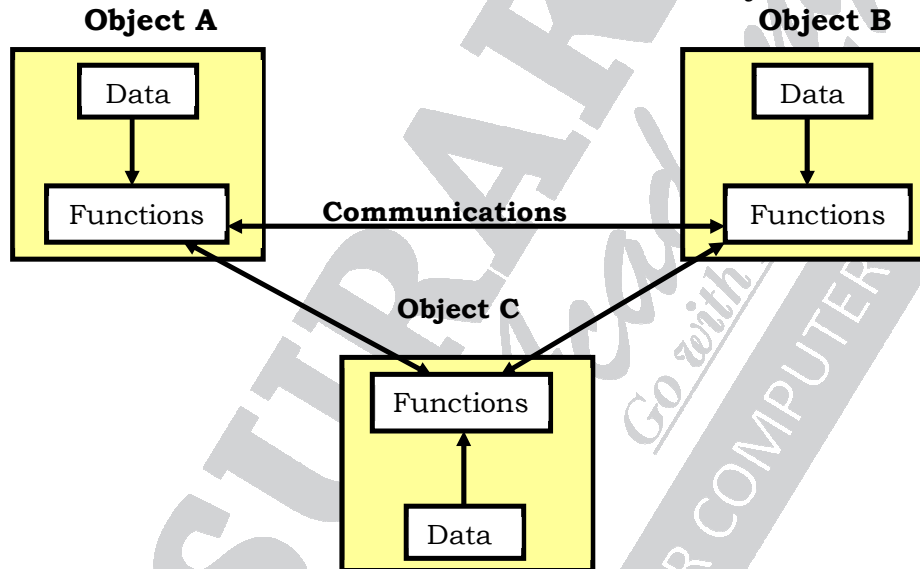


CHARACTERISTICS OF PROCEDURAL LANGUAGE:

- It follows top-down approach.
- Data is given less importance than function.
- Functions shares global data.
- Big program is divided into small modules.

OBJECT ORIENTED PROGRAMMING (OOP):

OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. OOP allows decomposition of a problem into a number of entities called **object** and then builds data and function around these objects.



ORGANIZATION OF DATA AND FUNCTIONS IN OOP

PROBLEM SOLVING TECHNIQUES

There are different ways of representing the logical steps for finding a solution of a given problem.

1. Algorithm.
2. Flowchart.

ALGORITHM

An algorithm is a sequence of **unambiguous** instructions for solving a problem, i.e., for obtaining a required output for any given input in a finite amount of time.

An algorithm is a finite step-by-step process for solving a particular problem.

EXAMPLES

Write an algorithm to add two numbers.

Algorithm:

[ADD TWO NUMBERS]

Step 1: Start.

Step 2: [Input Number.]
Read A and B.

Step 3: [Add Number.]
SUM:=A+B
Step 4: [Display Result.]
Print SUM.
Step5: Exit.

Write an algorithm to find sum of first **N** natural number.

Algorithm:
[ADD FIRST N NATURAL NUMBER]
Step 1: Start.
Step 2: [Initialize.]
Set SUM:=0, I:=1
Step 3: [Input Value of N]
Read N
Step 4: [Loop.]
Repeat Step 5 and 6 while I<=N
Step 5: SUM:=SUM+I.
Step 6: [Increase Counter]
I:=I+1
Step 7: [Display Result]
Print SUM
Step 8: Exit.

Write an algorithm to find factorial of given number.

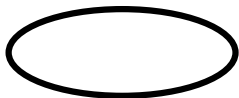



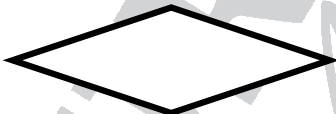
Algorithm:
[CALCULATE THE FACTORIAL OF A NUMBER]
Step 1: Start
Step 2: [Input Number]
Read N
Step 3: [Initialize]
Set I: =1, FACTORIAL: =1
Step 4: [Loop.]
Repeat Steps 5 and 6 until I<=N
Step 5: FACTORIAL: =FACTORIAL*I
Step 6: [Increase Counter]
I:=I+1
Step 7: [Display Result]
Print FACTORIAL
Step 8: Exit.

CHARACTERISTICS OF ALGORITHM:

S.N.	TERM	DESCRIPTION
1.	Finiteness	Terminate after a finite number of steps.
2.	Input	Valid inputs are clearly specified.
3.	Output	Can be proved to produce the correct output based on given valid input.
4.	Effectiveness	Steps are sufficiently simple and basic

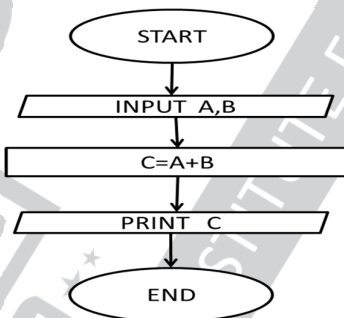
FLOW CHART

Graphical representation of any problem is called flowchart. A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process.

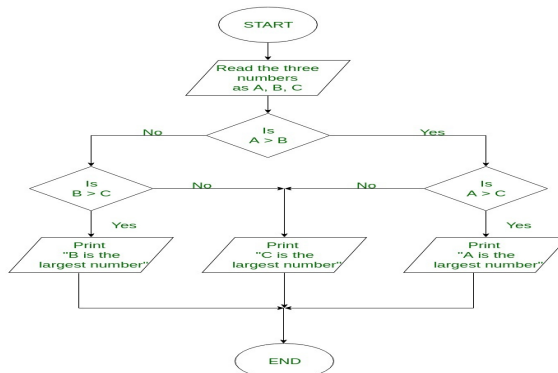
NAME	SYMBOL	USED IN FLOWCHART
Oval		Indicate beginning or end of a program.
Line		Direction of logic flow in program.
Parallelogram		Used as either input or output operation.
Rectangle		Denote process to be carried out.
Diamond		Denotes a decision to be made as yes or no.

EXAMPLES

Draw a flow chart to add two numbers.



Draw a flow chart to find the largest among three input numbers.

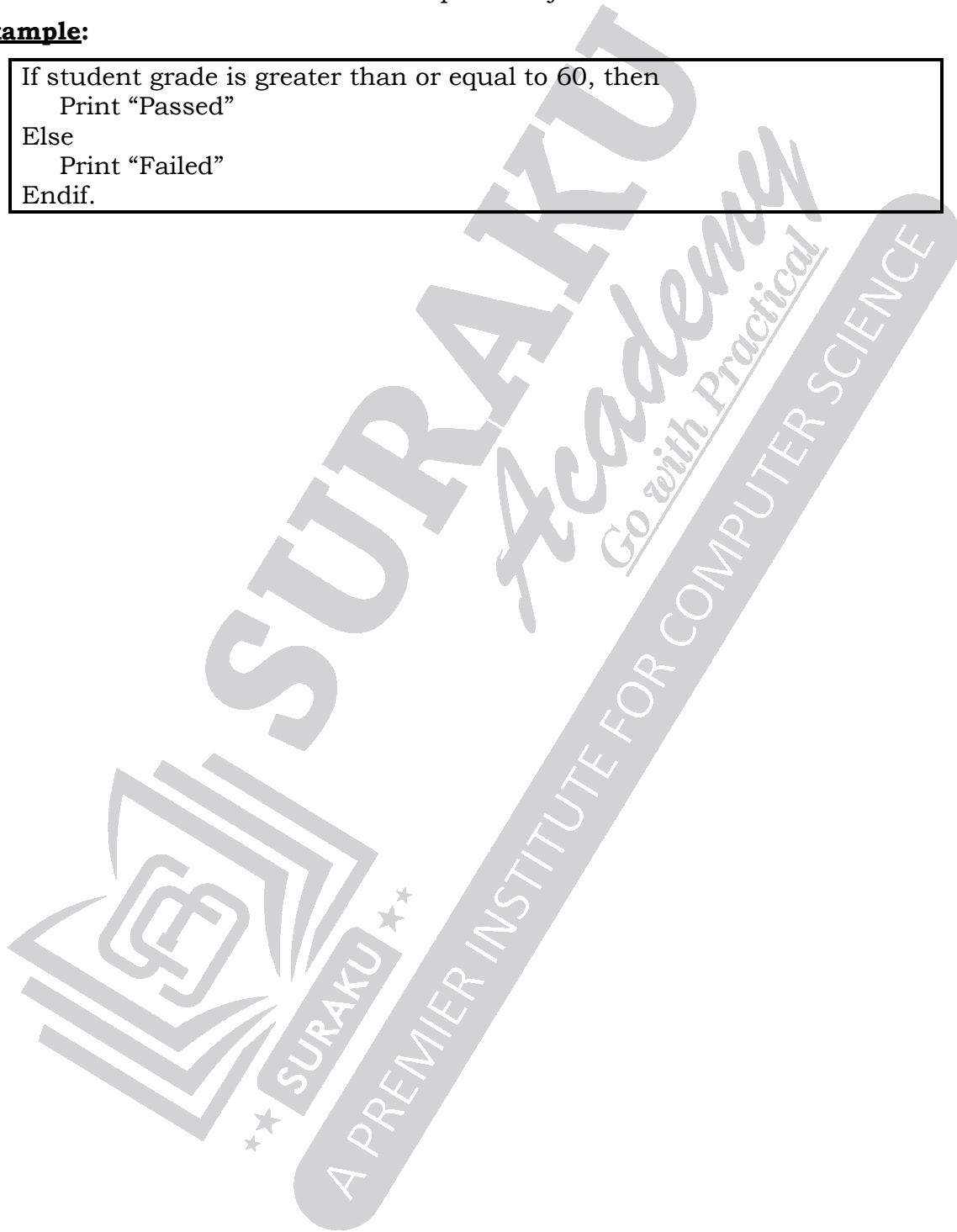


PSEUDO-CODE

Pseudo code is one of the methods that could be used to represent an algorithm. It is not written in a specific syntax that is used by a programming language and therefore cannot be executed in a computer. It just used for human understand.

Example:

```
If student grade is greater than or equal to 60, then
    Print "Passed"
Else
    Print "Failed"
Endif.
```



C is one of the most popular **general-purpose** programming languages. **C** language has been designed and developed by **Dennis Ritchie** at Bell Laboratories, New Jersey, USA, in **1972**.

Several important concepts of **C** are drawn from 'Basic Combined Programming Language' (BCPL) and 'B' language.

S.N.	LANGUAGE	INVENTOR	YEAR
1.	BCPL	Martin Richards	1967
2.	B	Ken Thompson	1969
3.	C	Dennis Ritchie	1972

FEATURES OF C LANGUAGE:

- **C** Language contains the capability of assembly language with the features of high-level language.
- **C** Language is used for developing device driver software, system software etc.
- **C** is the middle level language because it is combination of both **High-Level** and **Low-Level** Language.

STRUCTURE / SKELETON OF C PROGRAM

The execution of **C program** will always begin from **main()** function, which may be access other functions.

1.	Documentation Section
2.	Link Section
3.	Global Declaration Section
4.	The main() function
5.	<p><u>SUB-PROGRAM SECTION</u></p> <p>Function 1</p> <p>Function 2</p> <p>...</p> <p>...</p> <p>Function n</p> <p style="text-align: right;"><u>USER DEFINED FUNCTION</u></p>

1. DOCUMENTATION SECTION:

This section is optional and can be used for writing comments.

Comments are those lines which cannot be compiled by the compiler and only used for give the information about the program.

- **SINGLE LINE COMMENTS:**

- **Example:**

```
//Program to prints "Suraku Academy".
```

- **MULTIPLE LINE COMMENTS:**

- **Example:**

```
/* Program for adding two integers.
The value of variable is taken from user at run time.
*/
```

2. LINK SECTION:

The link section provides instructions to the compiler to link functions from the system library. i.e. #include<stdio.h> or #include<conio.h>.

3. GLOBAL DECLARATION:

This section declares some variables that are used in more than one function. These variables are known as **global** variables.

4. THE main() FUNCTION:

Every **C program** must have a main() function. It has two parts:

- Declaration part.
- Executable part.
- The declaration part declares all the variables used in executable portion of the main function.
- The **main()** is the starting point of the C program execution. The program execution begins from the opening curly brace { and ends at closing curly brace } of main().

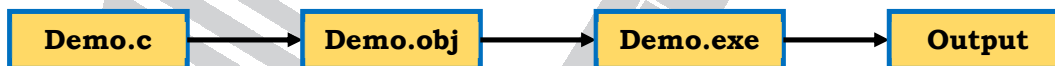
5. SUB-PROGRAM SECTION/USER DEFINED FUNCTION:

The functions defined by the users are called user-defined functions. These functions are defined outside the main () function.

EXECUTING THE 'C' PROGRAM

A **C** program must go through various phases.

- Create the program.
- Compile the program.
- Link the library function.
- Execute the program.



It generates one more file with the name **.bak** (only in 16-bit compiler), which stands for backup file. It is a copy of C source file.

THE C CHARACTER SET

1. Letters.
2. Digits.
3. White spaces.
4. Special Characters.

C CHARACTER SET:

S.NO.	ELEMENT OF 'C' CHARACTER SET
1.	Upper Case Letter: A to Z .
2.	Lower Case Letter: a to z .
3.	Digit: 0 to 9 .
4.	Special Symbol (List Given Below)

SYMBOL	NAME	SYMBOL	NAME
~	Tilde]	Right Square Bracket
<	Less than	/	Forward Slash
>	Greater Than	\	Backward Slash
&	Ampersand	:	Colon
	OR/PIPE	;	Semicolon
#	Hash	+	Plus
<=	Less than equal	-	Minus
>=	Greater than equal	*	Multiply
==	Equal	/	Division
=	Assignment	%	Mod/Moduls
!=	Not Equal	,	Comma
^	Caret	'	Single Quote
{	Left Brace	"	Double Quote
}	Right Brace	>>	Right shift
(Left parenthesis	<<	Left shift
)	Right Parenthesis	.	Period/Dot
[Left Square Bracket	_	Underscore
!	Exclamation Mark	?	Question Mark
@	At the rate	\$	Dollar Sign

TOKENS OF C

The smallest individual unit in a C program is called token.

1. Keywords.
2. Identifiers.
3. Constants.
4. Variables.
5. Strings.
6. Operators.

KEYWORDS:

Keywords are those words whose meaning is already known by the compiler.

- In 'C' language there are 32 keywords.
- We cannot use keywords as names for variable, function, arrays etc.
- All keywords are written in small letters.

Related to Primitive Data Types	Related to Control Statement	Related to Storage Classes	Related to User-Defined Data Type	Other Purpose
int	if	auto	struct	const
char	else	static	union	sizeof
float	switch	register	enum	volatile
double	case	extern	typedef	
short	default			
long	break			

signed	continue
unsigned	goto
void	do
	while
	for
	return

IDENTIFIERS:

Combination of letters becomes an identifier. Identifiers are used for give the names of various program elements like variables, array, functions, structure etc.

RULES FOR WRITING IDENTIFIERS:

- First letter must be an alphabet or underscore.
- From second character onwards, any combination of digit, alphabet or underscore is allowed.
- Other than underscore no any special symbol is allowed.
- Keywords cannot be used as identifiers.

CONSTANTS:

A fixed value that does not change during the execution of a program is refers as constants.

1. Primary Constant/Basic Constant
2. Secondary Constant.

PRIMARY CONSTANT/BASIC CONSTANT:

1. Integer constant.
2. Real constant.
3. Character constant.

1. INTEGER CONSTANTS:

All number either positive or negative without decimal point or fractional part is known as integer constants.

2. REAL CONSTANTS:

They are the numbers with fractional part. They also known as floating point constants.

Example:

34.56, 0.36, 1.2541.

3. CHARACTER CONSTANTS:

They are enclosed in single quotes. They consist of single character or digit.

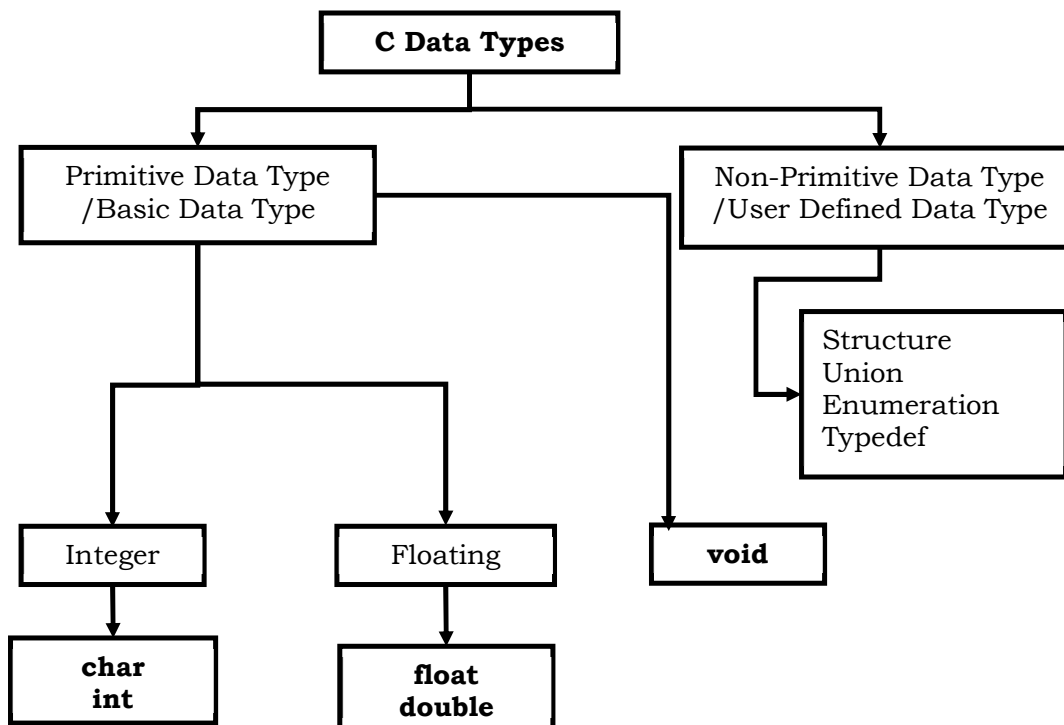
Example:

'3', 'A', '\n'.

SECONDARY CONSTANTS:

Secondary constants are also known as derived constants, as they derived from primary constants. Array, string, pointer, structure, union, enumerations are secondary constants.

DATA TYPES



MODIFIERS/QUALIFIERS:

Modifiers are the keywords used to modify or qualify the properties of data types. Four types of modifiers, they are:

1. The *short* Modifier.
2. The *long* Modifier.
3. The *signed* Modifier.
4. The *unsigned* Modifier.

SIZE AND RANGE OF DATA TYPES:

DATA TYPE	SIZE (BYTES)	FORMAT SPECIFIER
char	1	%c
signed char	1	%c
unsigned char	1	%c
int or short int	2	%d
unsigned int	2	%u
long int	4	%ld
unsigned long int	4	%lu
float	4	%f or %g
double	8	%lf
long double	10	%Lf

CALCULATING RANGE:**FOR signed DATA TYPE:** $(-2^{n-1} \text{ to } 2^{n-1} - 1)$ **FOR unsigned DATA TYPES:** $(0 \text{ to } 2^n - 1)$ Where **n** is numbers of bits and 1 byte is equal to 8 bits.**VARIABLE**

A variable is a named location in memory that is used to hold a value. The value of variable can be modified in the program by the instruction.

VARIABLE DECLARATION:**Data_Type Variable_Name;****Example:**

```
int a;  
float b, c;  
char p, q;  
long int sum;
```

INITIALIZING VARIABLES:

Provides value to the variable is termed as initialization of variable.

Syntax:

Variable_Name = value;

OR

Data_type Variable_Name = value;

Example:

```
int a;  
a=23;  
int y=4;
```

