Binary System:

The information, which it is stored or manipulated by the computer memory it will be done in binary mode.

RAM:

This is also called as real memory, physical memory or simply memory. In order to run any program, it has to be loaded into RAM. Whatever we store in this device, it will be available as long as power supply is available for it. So this memory is also called as volatile memory.

RAM contains elementary memory devices known as Flip-Flop's. A Flip-Flop can be considered as a bi-state device such as an electrical switch. That is, we can set ON or unset OFF the Flip-Flop state. If it is set, value ONE is assumed to be stored, otherwise ZERO is assumed to be stored. That is, we can store one bit in a Flip-Flop. A series of Flip-Flops in the processor is called as register.

Present computers allocate memory in multiples of eight Flip-Flops only. If the number what we like to store in the computer memory is not exact multiple of eight Flip-Flops, then ZERO padding will be done to the left hand side of the number and then the number is stored in the computer memory.

Number System:

4 Bits = 1 Nibble

8 Bits = 1 Byte

2^{10} Bytes = 1024 Bytes = 1 Kilo Byte

2^{20} Bytes = 2^{10} KB = 1024 KB = 1 Mega Byte

2_{30} Bytes = 2^{20} KB = 2^{10} MB = 1024 MB = 1 Giga Byte

2^{40} Bytes = 2^{30} KB = 2^{20} MB = 2^{10} GB = 1024 GB = 1 Tara Byte

2^{50} Bytes = 2^{40} KB = 2^{30} MB = 2^{20} GB = 2^{10} TB = = 1024 TB = 1 Peta Byte

ROM:

Read Only Memory is another important component in the computer system. Whatever we store in this device cannot be modified or erased through software. However by applying hardware methods such as sweeping through laser gun we can erase the content in ROM. This can be used to store Power On Self Test (POST) software.

Power On Self Test:

Whenever we switch on a machine, we may require some special software to test the workability of all the parts of the machine. This is called as POST software. This is usually kept on ROM, because users cannot change this software. If all the parts are working fine then the Operating System (OS) such as Disk Operating System, Microsoft Windows, Unix etc., are loaded into RAM from the secondary memory device such as hard disks. Then the users can utilize the computer services.

Basics of C Language

The programming language C was originally developed by **Dennis Ritchie** of **Bell Laboratories** and was designed to run on a PDP-11 with a UNIX operating system. Although it was originally intended to run under UNIX, there has been a great interest in running it under the MS-DOS operating system on the IBM PC and compatibles. Due to the simplicity and ease of writing a C compiler, it is usually the first high level language available on any new computer, including microcomputers, minicomputers, and mainframes.

C is not the best beginning language because it is somewhat puzzling in nature. It allows the programmer a wide range of operations from high level down to a very low level, approaching the level of assembly language. There seems to be no limit to the flexibility available.

Powerful features, simple syntax, and portability make C a preferred language among programmers for business and industrial applications. Portability means that C programs written for a computer with a particular kind of processor say Intel can be executed on computers with different processors such as Motorola, Sun Sparc, or IBM with little or no modification.

C language is widely used in the development of operating systems. An **Operating System** is software that controls the various functions of a computer. Also it makes other programs on your computer work.

Behind the Name – 'C' Language:

BCPL (Basic Combined Programming Language) is a computer programming language designed by Martin Richards of the University of Cambridge in 1966. B is a programming language that was developed at Bell Labs. It is almost extinct, as it was replaced with the C language. It was mostly the work of Ken Thompson, with contributions from Dennis Ritchie, and first appeared circa 1969. The first version of UNIX was written in the low-level PDP-7 assembler language. Then they developed a High-level language called 'B' and it got its name since it was developed based on BPCL language.

Then Dennis Ritchie developed the Language 'C' due to the arrival of PDP-11 computer. They named it as 'C' since the previous language they used was 'B' and they want to be named in sequential order that is after 'B', 'C,' which follows sequential order.

C Language Usages:

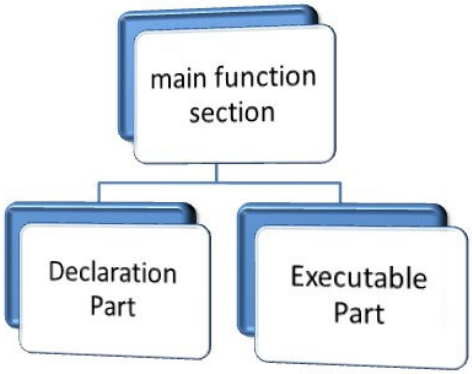
'C's ability to communicate directly with hardware makes it a powerful choice for system programmers. And the popular operating systems such as Unix and Linux are written entirely in C. Additionally, even compilers and interpreters for other languages such as FORTRAN, Pascal, and BASIC are written in C. However, C's scope is not just limited to developing system programs. It is also used to develop any kind of application, including complex business ones.

The following is a partial list of areas where C language is used:

Structure of C Program

The C language program should start with main() function. And the programmers have to follow the structure and syntax of the C language. The below one is the structure of the C language program.

```
Preprocessor directives
Global Variables Declaration
void main()
{
Local variables declaration
Executable Statements & Expressions
}
Return type subfunction (arguments)
{
Local variables declaration
Statements & Expressions
}
```



Preprocessor directives are lines included in the code of our programs that are not program statements but directives for the preprocessor. These lines are always preceded by a hash sign (#). The preprocessor is executed before the actual compilation of code begins, therefore the preprocessor digests all these directives before any code is generated by the statements. Example of this preprocessor directive is **#include<stdio.h>**.

These preprocessor directives extend only across a single line of code. As soon as a newline character is found, the preprocessor directive is considered to end. No semicolon (;) is expected at the end of a preprocessor directive. The only way a preprocessor directive can extend through more than one line is by preceding the newline character at the end of the line by a backslash (\).

The variables which we are declaring before the main function are called as **global variables**. These variables can be used in all the functions in the entire program.

The function **main** is very important, and must appear once, and **only once** in every C program. This is the point where execution is begun when the program is run. Following the main program name is a pair of parentheses which are an indication to the compiler that this is a function. The two curly brackets are called as braces, are used to define the limits of the program itself. The actual program statements go between the two braces and in this case, there are no statements because the program does absolutely nothing.

Constant and Variables

Instructions in C language are formed using syntax and keywords. It is necessary to strictly follow C language Syntax rules. Any instruction that mismatches with C language Syntax generates an error while compiling the program. All programs must confirm to rules pre-defined in C Language. Keywords as special words which are exclusively used by C language, each keyword has its own meaning and relevance hence, Keywords should not be used either as Variable or Constant names.

Character Set:

The character set in C Language can be grouped into the following categories.

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

White Spaces are ignored by the compiler until they are a part of string constant. White Space may be used to separate words, but are strictly prohibited while using between characters of keywords or identifiers.

C Character-Set Table:

Letters	Digits
Upper Case A to Z	0 to 9
Lower Case a to z	

Keywords:

Every word in C language is a keyword or an identifier. Keywords in C language cannot be used as a variable name. They are specifically used by the compiler for its own purpose and they serve as building blocks of a c program. The following are the Keyword set of C language.

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**	**size of**	**volatile**
do	**if**	**static**	**while**

Therefore dept1 and dept2 are indirectly declared as integer datatype and section1 and section2 are indirectly float data type.

The second type of user defined datatype is enumerated data type which is defined as follows.

Enum identifier {value1, value2 …. Value n};

The identifier is a user defined enumerated datatype which can be used to declare variables that have one of the values enclosed within the braces. After the definition we can declare variables to be of this 'new' type as below.

enum identifier V1, V2, V3, …… Vn

The enumerated variables V1, V2, ….. Vn can have only one of the values value1, value2 ….. value n

Examples:

```
enum day {Monday, Tuesday, …. Sunday};
enum day week_st, week end;
week_st = Monday;
week_end = Friday;
if(wk_st == Tuesday)
week_en = Saturday;
```

Defining Symbolic Constants:

A symbolic constant value can be defined as a preprocessor statement and used in the program as any other constant value. The general form of a symbolic constant is

 # define symbolic_name value of constant

Valid examples of constant definitions are:

```
# define marks 100
# define total 50
# define pi 3.14159
```

These values may appear anywhere in the program, but must come before it is referenced in the program. It is a standard practice to place them at the beginning of the program.

Use of printf() and scanf() functions:

In order to read and write, we can use standard library functions **printf()** and **scanf()**. Both these functions definition is available in **stdio.h**. The printf function is used to print the data on the console. The **printf()** function is a standard in built function for printing a given line which appears inside the double quotes. Whatever message we wanted to be displayed on the screen is to be included in between two double quotes and send to printf. And printf call also has to be terminated with a semicolon.

Example Program:

```
#include<stdio.h> //include header file stdio.h
void main() //tell the compiler the start of the program
{
        int num1, num2, sum, sub, mul, div, mod; //declaration of variables
        printf(" Please Enter Two Numbers :: "); //asking user to enter no's
        scanf ("%d %d", &num1, &num2); //inputs the operands
        sum = num1+num2; //addition of numbers and storing in sum.
        printf("\n The sum is = %d", sum); //display the output
        sub = num1-num2; //subtraction of numbers and storing in sub.
        printf("\n The difference is = %d", sub); //display the output
        mul = num1*num2; //multiplication of numbers and storing in mul.
        printf("\n The product is = %d", mul); //display the output
        div = num1/num2; //division of numbers and storing in div.
        printf("\n The division is = %d", div); //display the output
        mod = num1%num2; //modulus of numbers and storing in mod.
        printf("\n The modulus is = %d", mod); //display the output
 }
```


Output

```
Please Enter Two Numbers :: 10 5
The sum is = 15
The difference is = 5
The product is = 50
The division is = 2.0
The modulus is = 0
```

Integer Arithmetic:

When an arithmetic operation is performed on two whole numbers or integers than such an operation is called as integer arithmetic. It always gives an integer as the result. Let x = 27 and y = 5 be 2 integer numbers. Then the integer operation leads to the following results.

x + y = 32

x – y = 22

x * y = 115

x % y = 2

x / y = 5

output

x = 10.00

y = 7.00

z = 4.00

Type Conversions

Some times it is required to convert the one data type to another data type. That is called as **type conversion**. Type conversion occurs when the expression has data of mixed data types, for example, converting an integer value into a float value, or assigning the value of the expression to a variable with different data types.

Implicit type conversion:

C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance. This automatic type conversion is known as implicit type conversion

During evaluation it holds to very strict rules and type conversion. If the operands are of different types the lower type is automatically converted to the higher type before the operation proceeds. The result is of higher type.

In type conversion, the data type is promoted from lower to higher because converting higher to lower involves loss of precision and value. For type conversion, C maintains a hierarchy of data types using the following rules:

1. Integer types are lower than floating-point types.
2. Signed types are lower than unsigned types.
3. Short whole-number types are lower than longer types.
4. The hierarchy of data types is as follows: double, float, long, int, short, char.

The following rules apply during evaluating expressions:

All short and char are automatically converted to int then

1. If one operand is long double, the other will be converted to long double and result will be long double.
2. If one operand is double, the other will be converted to double and result will be double.
3. If one operand is float, the other will be converted to float and result will be float.
4. If one of the operand is unsigned long int, the other will be converted into unsigned long int and result will be unsigned long int.
5. If one operand is long int and other is unsigned int then

Bitwise AND Operator:

A bitwise AND takes two binary representations of equal length and performs the logical AND operation. The bitwise-AND operator compares each bit of its first operand to the corresponding bit of its second operand. If both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

For example:

0101 (decimal 5)
(AND) & 0011 (decimal 3)
= 0001 (decimal 1)

In the C programming language family, the bitwise AND operator is "&" (ampersand). Again, this operator must not be confused with its Boolean "logical and", which treats its operands as Boolean values, and is written "&&" (two ampersands).

Bitwise OR Operator:

A bitwise OR takes two bit patterns of equal length, and produces another one of the same length by matching up corresponding bits and performing the logical inclusive OR operation. And it compares each bit of its first operand to the corresponding bit of its second operand. If either bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

For example:

0101 (decimal 5)
(OR) | 0011 (decimal 3)
= 0111 (decimal 7)

In the C language family, the bitwise OR operator is "|" (pipe). Again, this operator must not be confused with its Boolean "logical or", which treats its operands as Boolean values, and is written "||" (two pipes).