Unit
2

# Basics of C

**2.1 Introduction**

2.1.1 Basics of C language

2.1.2 History of C

2.1.3 Versions of C

2.1.4 Features Of C Language

**2.2 C Program Development Cycle – Write, Compile and Execute**

2.2.1 Using an Editor

- Using Turbo C

2.2.2 Compiling and Debugging

**2.3 General Structure of C program**

2.3.1 Documentation Section

2.3.2 Link Section

2.3.3 Definition Section

2.3.4 Global Declaration Section

2.3.5 MAIN () Function Section

2.3.6 Subprogram Section

2.3.7 Example program: Printing a message

**2.4 Standard/Preprocessor Directives**

2.4.1 Why is it called Preprocessor?

2.4.2 Why are preprocessor directives useds?

2.4.3 How Pre-Processor directives work?

2.4.4 Categories of Preprocessor directives

2.4.5 Header Files

**2.5 The Main() Function**

2.5.1 Comments

**2.6 Character Set**

2.6.1 'C' tokens/Grammer

2.6.2 Keywords

2.6.3 Identifiers

2.6.4 Constants
- Integer Constants
- Real Constants
- Character Constants
- String Constants
- Escape Sequences

2.6.5 Variables

**2.7 Dynamic Initialization**

2.7.1 Variable scope
- File Scope
- Block Scope

**2.8 Data Types**

2.8.1 Primary data type
- Integer Data Type
- Character Data Type
- Floating Point Types
- Void Type
- Sample program illustrating each data type

2.8.2 User defined data type
- The enum keyword
- typedef (Using Type Definitions)

2.8.3 Derived Data type

2.8.4 Constant
- Volatile Variable

2.8.5 Type Conversion

**2.9 Introduction of Different Types of Operators**

2.9.1 Arithmetic Operators

2.9.2 Relational Operators

2.9.3 Logical Operators

2.9.4 Bitwise Operators

2.9.5 Assignment Operators

2.9.6 Conditional Operators

2.9.7 Special Operators

2.9.8 The size of Operator

**2.10 Priority (Precedence) and Associativity of Operators**

**2.11 Formatted and Unformatted Input and Output in C**

2.11.1 Formatted I/O Functions
- printf() function
- scanf() Function

2.11.2 UnFormatted I/O Functions
- getch() function
- getche() function
- getchar() function
- putchar() function
- gets() and puts() functions

**2.12 Formatted Input and Output in C**
- Formatting Output

◆ **Exercise**

◆ **GTU Exam. Paper Solution**

## 2.1 INTRODUCTION

### ■ Computer Program

A computer program consists of a series of instructions for the computer to run. In order to design a program for a computer, you must determine three basic elements:

1. The instructions that must be performed.
2. The order in which those instructions are to be performed.
3. The data required to perform the instructions.

A programming language (PL) is a set of instructions used to create a program. All programming language languages can be divided into three categories

- High Level Language or Problem Oriented
- Middle Level Language
- Low Level Language or Machine Oriented

### ■ High Level Languages

These languages are more oriented to be user friendly and syntax nearly approaches English. They have poor efficiency and flexibility.

Example High Level – Basic, Pascal, COBOL, FORTRAN

### ■ Low Level Languages

These languages are computer or machine oriented. They allow programmer to code with maximum efficiency and flexibility but coding is quite tough. Its syntax nearly approaches the binary language.

Example Low Level – Assembly Language

### ■ Middle Level Language

The middle level language is one which combines best feature of both high and low level languages. They are easy to code as well as efficient and flexible.

C is a middle level language.

Example Middle Level – C, C++, Java, C#

### 2.1.1 Basics of C Language

C is a computer programming language. C is one of the thousands of programming languages currently in use. C is an easy language to learn.

### 2.1.2 History of C

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972 by Dennis Ritchie.

### 2.1.3 Versions of C

C has two versions.

➢ **Common C ("Bell Labs C" or "K & R C" or "Classic C")**

This is known as Bell Labs C after the most popular compiler of K. & R. It is now often called "Classic C". For many years standard of C was provided with UNIX but as the variety of systems and Operating systems increased each of the manufacturer provided its own version of C language. This became big problem as many variant of same language came into existence.

➢ **ISO/ANSI Standard**

ISO stands for International Standard Organization. ANSI stands for American National Standard Institute. The American National Standards Institute defined a standard for C, eliminating uncertainty about the exact syntax of the language. ANSI C adds few improvements over the old common C.

### 2.1.4 Features Of C Language

The increasing popularity of C is due to its various desirable qualities:

C has high-level constructs.

C is a robust language with rich set of built-in functions and operators.

Programs written in C are efficient and fast (Much faster than BASIC language).

C is highly portable (code written in one machine can be moved to other).

C is highly flexible.

C allows access to the machine at bit level (Low level (Bitwise) programming).

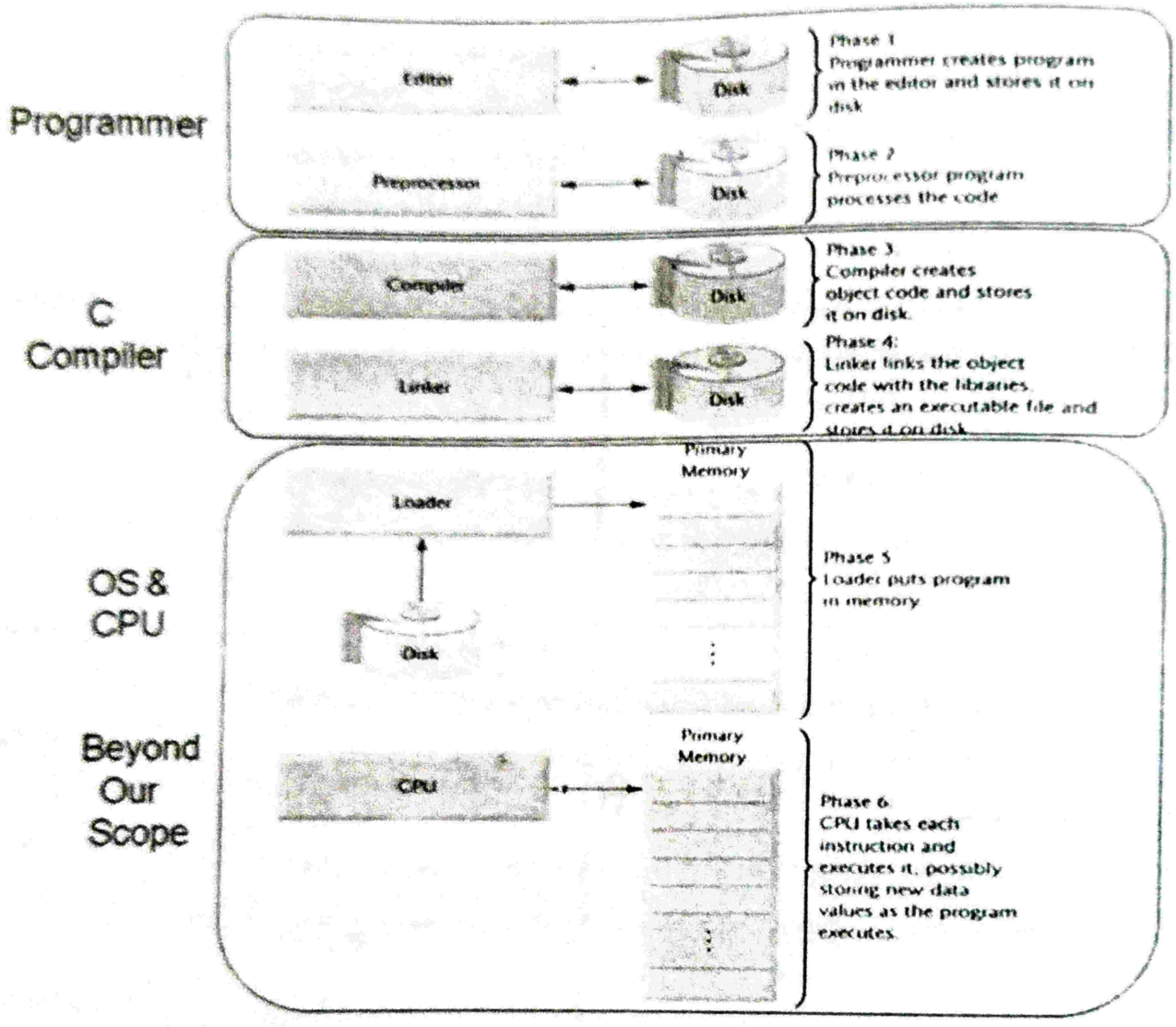
C supports pointer implementation - extensive use of pointers for memory, array, structures and functions.

C is extensible

## 2.2 C PROGRAM DEVELOPMENT CYCLE – WRITE, COMPILE AND EXECUTE

The stages of developing C program are as follows. C program requires at least four steps:

| | |
|---|---|
| Step 1 | **Create :** Use an editor to write your source code. C source code files have the extensio .C (e.g, MYPROG.C, DATABASE.C, and so on). |
| Step 2 | **Compile :** Compile the program using a compiler. If the compiler doesn't find any errors in the program, it produces an object file. The compiler produces object files with an .OBJ extension and the same name as the source code file (e.g, MYPROG.C compiles to MYPROG.OBJ). |
| Step 3 | **Link :** Link the program using a linker. If no errors occur, the linker produces an executable program located in a disk file with an .EXE extension and the same name as the object file (for example, MYPROG.OBJ is linked to create MYPROG.EXE). |
| Step 4 | **Execute :** Execute the program. You should test to determine whether it functions properly. If not, start again with step 1 and make modifications and additions to your source code. |




**C Program Development Cycle**

### 2.2.1 Using an Editor

Most compilers come with a built-in editor that can be used to enter source code.

If you're using a UNIX system, you can use editors such as:- ed, ex, edit, emacs, or vi.

If you're using Microsoft Windows, Notepad is available.

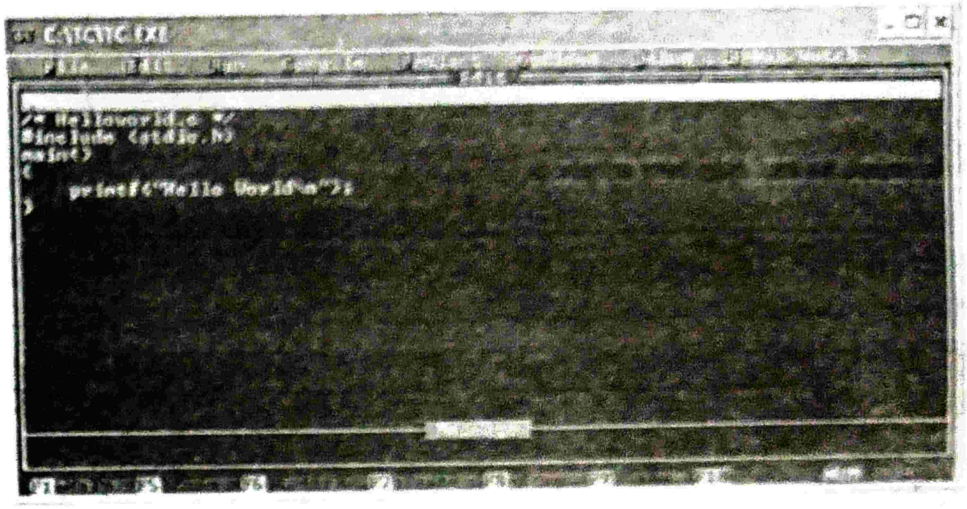
If you're using MS/DOS 5.0 or later, you can use Edit.

When you save a source file, you must give it a name. The name should describe what the program does. In addition, when you save C program source files, give the file a .C extension.

- **USING TURBO C**

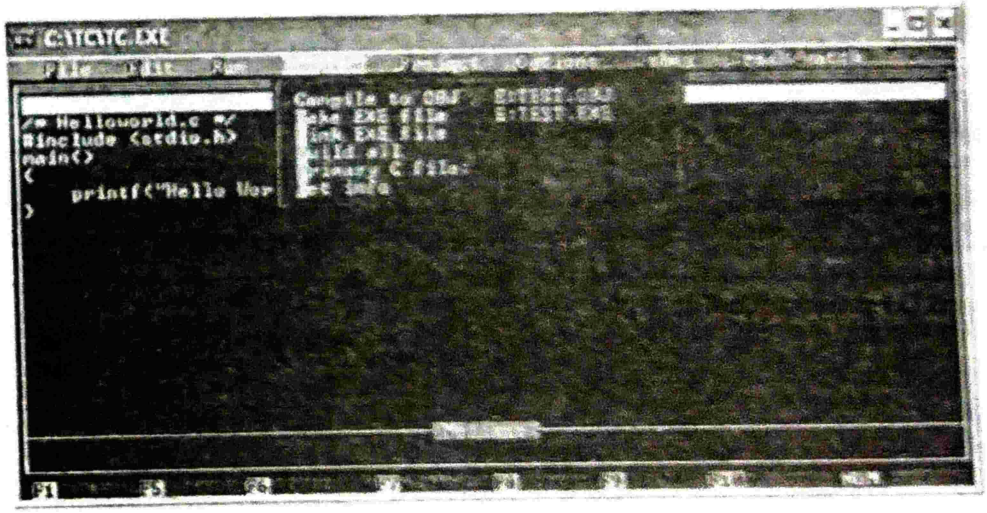
**Edit stage**

Type program in using one of the Borland Turbo C 3.0



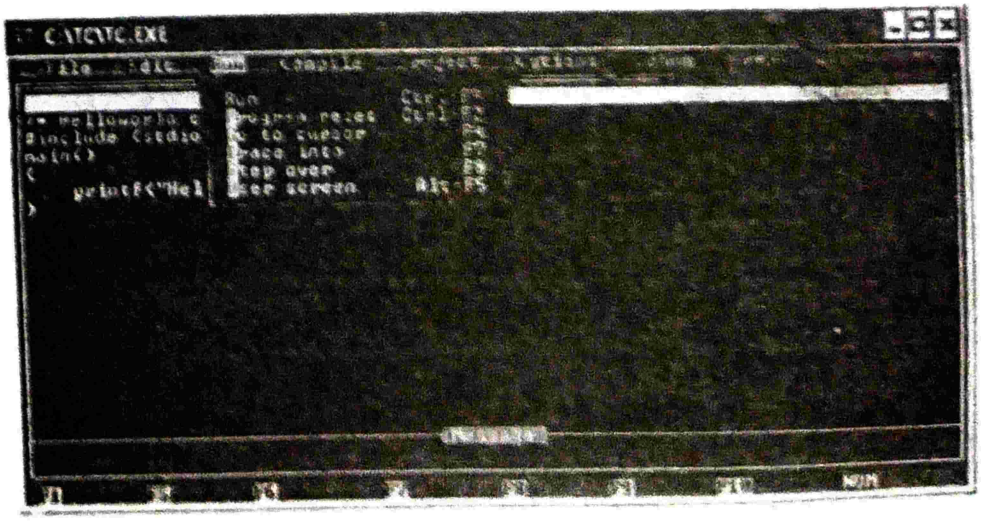

**Compile and link**

Select Compile [or press Alt + F9] from Compile menu. Make option allows you to both compile and link in the same option.




**Execute**

Use the Run menu and select Run option.



### 2.2.2 Compiling and Debugging

Errors in a program are called "bugs" and tracking and removing these errors is called debugging. There can be two main types of errors in a program. They are given below.

- **Syntax Errors**

The rules for writing statements of a computer programming language are called syntax of the language. The program statements are written strictly according to these rules. A single mistake in these rules causes an error. This error is called syntax error. The compiler detects these errors. It does not compile a program that contains syntax errors. These errors are easy to locate and remove.

- **Logical Errors**

The errors in the logic of a program are called logical errors. The compile cannot detect these errors. A program with logical errors runs correctly but it gives wrong result. The logical errors may occur due to the following reasons:

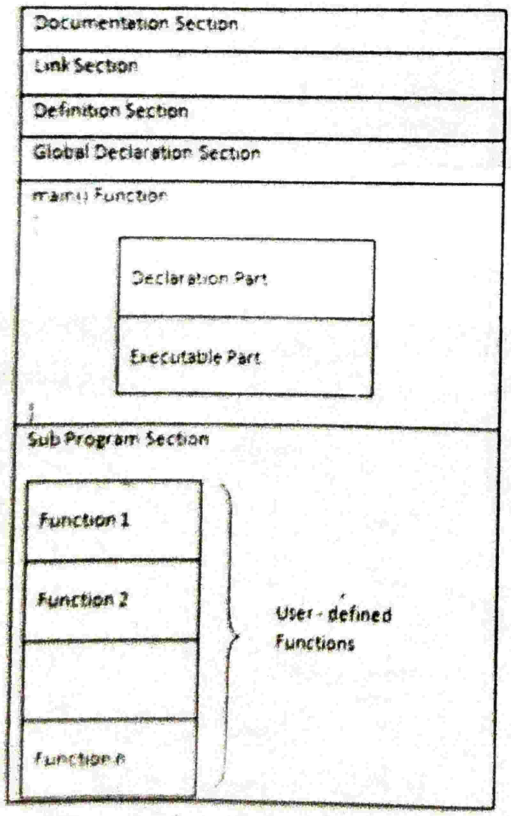
- The input data may be incorrect
- The sequence of instructions used in a program may be correct
- A mathematical formula used in program instructions may be incorrect.

These errors are most difficult to locate and remove.

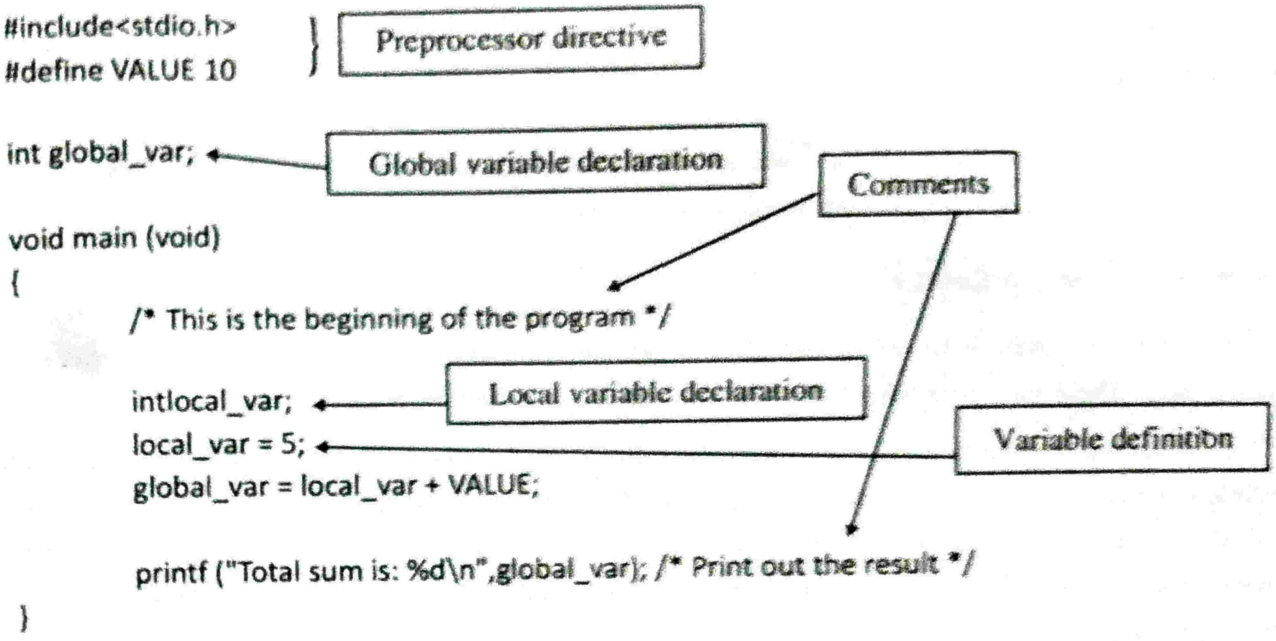
## 2.3 GENERAL STRUCTURE OF C PROGRAM

Every C program is a group of building blocks that are called functions. A function is may include one or more statements to perform a specific task.

C Program is divided into following sections....



Example C program :




### 2.3.1 Documentation Section

Generally this section includes set of comments to give:

The name of the program,

The author name, time of program creation and other details, which the programmer would like to use later

For Example:

```
/*
This is a program to find area of circle created on 12-12-2011
Created By: Jitendra
*/
```

### 2.3.2 Link Section

The link section lists files to be included and preprocessed before compiling the program. These included files provide instructions to the compiler to link functions from the system library.

For Example:

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

### 2.3.3 DEFINITION SECTION

The Definition section defines all symbolic constants; generally capital letters are used to define constants.

For Example :

```
#define PI 3.14
```

Here PI is the symbolic constant.

### 2.3.4 Global Declaration Section

This section is used to declare global variables. Global variables are used in more than one function. Generally they are declared before beginning of any of the function above the main() function.

### 2.3.5 Main () Function Section

Every C program must have one main() function section. This section contains two parts, declaration part and executable part. These two parts must be in between opening and closing braces.

The declaration part declares all the variables and functions, which are to be used in the executable part. The executable part contains all the instructions to be executed. The program execution begins at the opening brace and ends at the closing brace.Remember that you should always declare all the types of variables before using in any executable statement.All statements in the declaration and executable parts end with a semicolon.

### 2.3.6 Subprogram Section

The subprogram section contains all the user defined functions that are called in the main function. User defined functions are generally placed immediately after the end of main function.

### 2.3.7 Example program: Printing a message

```
#include <stdio.h>
int main()
{
/* Printing begins here */
printf ("C is a very good programming language.\n");
/* Printing ends here */
return 0;
}
```

## 2.4 STANDARD/PREPROCESSOR DIRECTIVES

A unique feature of C language is the preprocessor directives or standard directives. A programmer can use the preprocessor directives to make his program easy to read, modify, portable and more efficient. Preprocessor directives are those parts of a C program that processes the code before it passes through the compiler.

### 2.4.1 Why is it called Preprocessor?

Preprocessor directives are placed in the source program before the main() function. Every C program is checked for the preprocessor statements before compiling and if any preprocessor statements are used in the program then they

are processed first and then the preprocessed program is given to the compiler for further compiling. This is the reason why it is called preprocessor directives.

The C Preprocessor is not part of the compiler, but is a separate step in the compilation process. In simplistic terms, a C Preprocessor is just a text substitution tool. All preprocessor lines begin with # and do not require any semicolon at the end.

### 2.4.2 Why are Preprocessor Directives useds?

Preprocessors are used in C because:

It improves the readability of program.

It makes program easy to modify

Increases program portability

The Preprocessor accepts source code as input and is responsible for

Removing comments

Interpreting special preprocessor directives denoted by #.

### 2.4.3 How Pre-Processor directives work?

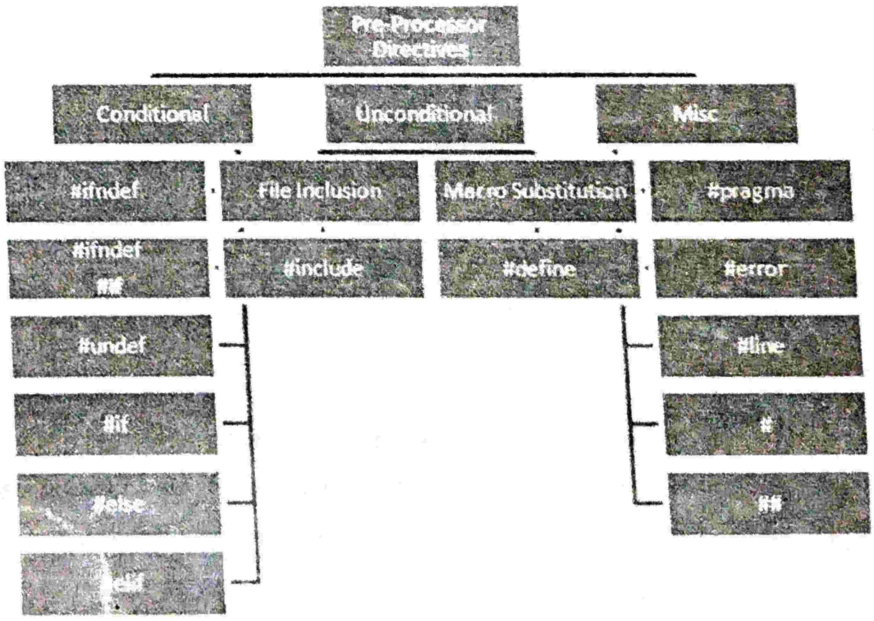
All the preprocessor directives are processed before the staring of actual compilation and converted into an intermediate file. In the intermediate file all preprocessor is converted in term of actual C code. A preprocessor must be in its own separate line. So following code below is not valid-

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

Also the preprocessor does not need a semi-colon since preprocessor is not a executable statement but an instruction to compiler so they don't need a semi-colon.

### 2.4.4 Categories of Preprocessor directives

A set of commonly used preprocessor directives are :



The conditional directives are:

#ifdef - If this macro is defined

#ifndef - If this macro is not defined

#if - Test if a compile time condition is true

#else - The alternative for #if

#elif - #else an #if in one statement

#endif - End preprocessor conditional

The unconditional directives are:

#include - Inserts a particular header from another file

#define - Defines a preprocessor macro

#undef - Undefines a preprocessor macro

Other miscellaneous directives include:

# - Stringization, replaces a macro parameter with a string constant

## - Token merge, creates a single token from two adjacent ones

### 2.4.5 Header Files

Code defined in standard library is generally included using header files. For example,

#include <math.h> — standard library maths file.

#include <stdio.h> — standard library I/O file

Following C source code shows the #include preprocessor in action -

```
#include <stdio.h> /* Include a standard header file.*/
#include "C:\\TC\\myfile.c" /* Include a user defined file.*/
int main ()
{
 prinf ("Notice the use of #Include preprocessor");
 return 0;
}
```

## 2.5 THE MAIN() FUNCTION

All C programs must have a main function. main () is the starting point of the program. You can only have one main() function, but you can place it anywhere within the code. The opening and closing braces ({ }) indicates the beginning and ending of the main() function. Different way of writing main() functions:

| | |
|---|---|
| main()<br>{ ……..<br>……..<br>} | Simple C main function |
| void main(void)<br>{<br>……..<br>……..<br>return;<br>} | Main function returning nothing. and accepting no arguments |
| int main(void)<br>{<br>……..<br>……..<br>return 0;<br>} | Main function with returning 0 indicating successful completion of the program to the OS |
| int main(void)<br>{<br>……..<br>……..<br>return <non-zero value>;<br>} | Main function with returning value other than 0 indicating unsuccessful completion of the program to the OS |
| int main(int argc, char *argv[])<br>{<br>……..<br>……..<br>return 0;<br>} | Main function accepting character array of arguments. |

Functions return a value too, in other words, function passes back a value to the program. If a function returns nothing, its return type is of type void - i.e. nothing is returned.

### 2.5.1 Comments

A comment is a note that you put into your source code. All comments are ignored by the compiler. Comments are useful for the readability and understandability of the program. It is a very good practice to include comments in all the programs to make the users understand what is being done in the program. Comments are used primarily to document the meaning and purpose of your source code.

In C, the start of a comment is signaled by the /* character pair. A comment is ended by */.

For exampl e:

```
/* This is a comment. */
```

Comments can be written on multiplelike:

```
/* Hi, Here is
a multi-line comment */
```

Here are examples of commented code:

```
/* Comments spanning several */
/* lines can be commented*/
/* out like this!*/
/* But this is a
 simpler way
 of doing it! */
// These are C++
// style comments
// and should NOT
// be used with C!!
/* /* NESTED COMMENTS
ARE ILLEGAL!! */ */
```

## 2.6 CHARACTER SET

Every language has its own words (Keywords) and specific statements, which can be used in the program. These words are made up using certain symbols, characters, digits etc. and are known as Character Set of the language.

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

- Letters & Digits
- Special Characters
- White Spaces

**Letters & Digits**

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

**Special Characters**

| , | Comma | & | Ampersand |
|---|---|---|---|
| . | Period | ^ | Caret |
| ; | Semicolon | * | Asterisk |
| : | Colon | - | Minus Sign |
| ? | Question Mark | + | Plus Sign |
| ' | Aphostrophe | < | Opening Angle (Less than sign) |
| " | Quotation Marks | > | Closing Angle (Greater than sign) |
| ! | Exclaimation Mark | ( | Left Parenthesis |
| \| | Vertical Bar | ) | Right Parenthesis |
| / | Slash | [ | Left Bracket |
| \ | Backslash | ] | Right Bracket |
| ~ | Tilde | { | Left Brace |
| - | Underscore | } | Right Bracket |
| $ | Dollar Sign | # | Number Sign |
| &% | Percentage Sign | . | . |

**White Spaces**

White Spaces are ignored by the compiler. White Space may be used to separate words.

Blank Space

Horizontal Tab

Carriage Return

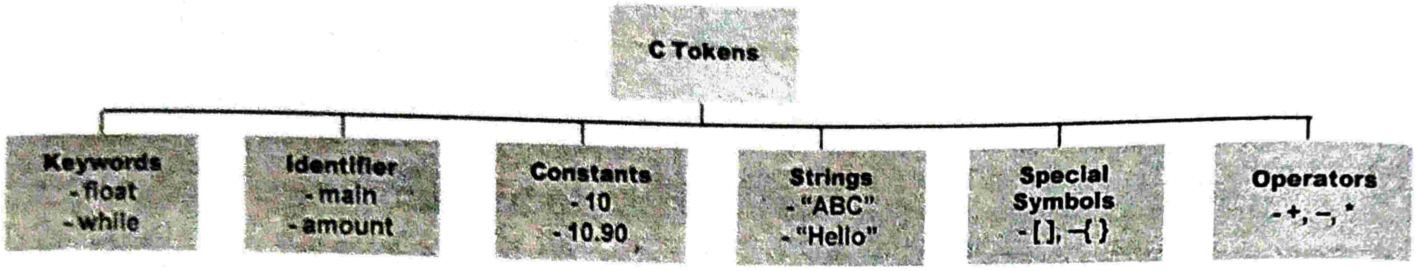
New Line

Form Feed

The above things are used in the character set of "C".

### 2.6.1 'C' Tokens/Grammer

C source code consists of several elements. The smallest individual elements of the C program are called tokens. A token can be variable names, keywords, constants, punctuation marks, operators, etc. C has six types of tokens as shown in fig. given below




- Tokens can be words, such as for, return, main, and i,.
- Tokens can be constants such as 1 and 10.
- Tokens can be operators such as =, +, and.
- Tokens can be other punctuation characters (often called delimiters), such as parentheses and braces { }.

Finally, all of the preceding elements can be separated by whitespace: spaces, tabs, and the "carriage returns" between lines.

For example, in the following program segment:

```
main()
{
    int a, b;
}
```

main, {, }, int, a, b and punctuation marks (,) and (;) are tokens of the program.(total 10 tokens)

### 2.6.2 Keywords

There are reserved words in 'C', which are called keywords. The compiler already knows meaning of keywords. The meaning of keyword cannot be changed. All the keywords must be written in lowercase. Some examples of keywords are :

| | | | |
|---|---|---|---|
| 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 |

All keywords are categorized according to group as follows:

Flow control (6): if, else, return, switch, case, default

Loops (5): for, do, while, break, continue

Common types (5): int, float, double, char, void

For dealing with structures (3): struct, typedef, union

Counting and sizing things (2): enum, sizeof

Rare but still useful types (7): extern, signed, unsigned, long, short, static, const

Keywords which are rarely use are (4): goto, auto, register, volatile

Total keywords: 32

### 2.6.3 Identifiers

Identifier are name of user-defined variables, array and functions. A variable should be a sequence of letters and or digits and the variable name should begin with a character. Both uppercase and lowercase letters are permitted. The underscore character is also permitted in identifiers.The identifiers must conform to the following rules :

First character must be an alphabet (or underscore)

Identifier names must consists of only letters, digits and underscore.

An identifier name should have less than 31 characters.

Any standard C language keyword cannot be used as a variable name.

An identifier should not contain a space.

### 2.6.4 Constants

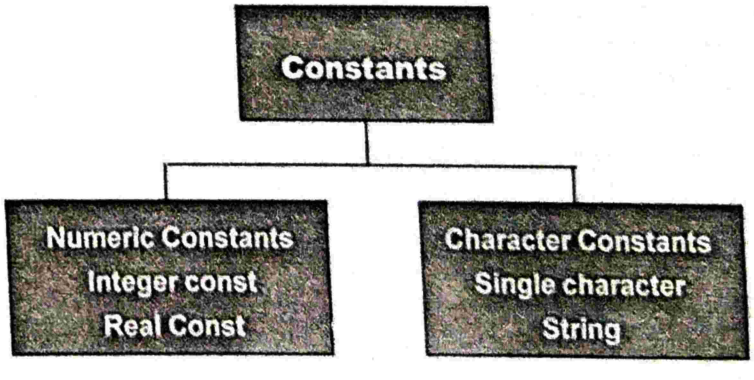
A constant value is the one which does not change during the execution of a program. C supports several types of constants.

Integer Constants

Real Constants

Single Character Constants

String Constants



## ■ INTEGER CONSTANTS

An integer constant is a sequence of digits. There are 3 types of integer namely decimal integer, octal integers and hexadecimal integer.

DECIMAL INTEGER consists of a set of digits 0 to 9. Examples of valid decimal constants:

```
123
-31
0
562321
+ 78
```

Some examples for invalid decimal integer constants are

```
15 750
20,000s
Rs. 1000
```

OCTAL INTEGER constant consists of any combination of digits from 0 through 7 with 0 at the beginning. Some examples of octal integers are

```
026
0
0347
0676
```

HEXADECIMAL INTEGER constant is preceded by OX or Ox, they may contain alphabets from A to F or a to f. The alphabets A to F refer to 10 to 15 in decimal digits.

Examples of valid hexadecimal integers are :

```
0X2
0X8C
0Xbcd
0x
```

Certain suffixes can be given to constants, which identify them for the type

| Type | Suffix |
|---|---|
| long | I or L |
| unsigned long | Ul or UL |

For example

```
5000U       unsigned (decimal)
0x5000U     unsigned (hex)
1234UL      unsigned long (decimal)
0777U       unsigned (octal)
```

## ■ REAL CONSTANTS

Floating point constant contains a decimal point or an exponent. It could be either negative or positive. No commas or blanks are allowed within a real constant. Examples of real constants are :

```
0.0026
-0.97
435.29
+487.0
```

A real number may also be used for exponential (or scientific) notation.

The General form is

```
+/- M e n
```

Here M is mantissa part and n is exponent part.

For example: 250000 can be written as 25e4 or 2.5e5

Suffixes for real constants, which identify them for the type

| Type | Suffix |
|---|---|
| Float | f or F |
| long double | l or L |

## ■ CHARACTER CONSTANTS

A Single Character constant represent a single character which is enclosed in a pair of quotation symbols.

Examples for character constant are:

```
'5'
'x'
';'
' '
```

All character constants have an equivalent integer value which is called ASCII Values

| Character | ASCII |
|---|---|
| 'A' | 65 |
| 'B' | 66 |

## ■ STRING CONSTANTS

A string constant is a set of characters enclosed in double quotation marks. The characters in a string constant sequence may be a alphabet, number, special character and blank space. Example of string constants are

```
"INDIA"
"12345"
"God Bless U"
"!.....?"
```

## ■ ESCAPE SEQUENCES

Backslash character constants are special characters used in output functions. Although they contain two characters they represent only one character. Given below is the table of escape sequence and their meanings.

| Esc. Seq | Purpose | Description |
|---|---|---|
| \n | New line | Causes a new line to begin. |
| \b | Backspace | Moves the cursor one position to the left |
| \f | Form feed | Advances computer stationary to the top of next page |
| \' | Single quote | Causes a single quote to be printed |
| \\ | Backslash | Causes a back slash to be printed |
| \" | Double Quote | Causes a double quote to be printed |
| \t | Tab | The cursor jumps over 8th character with this esc. Seq. |
| \r | Carriage return | Takes the cursor to the beginning of the line in which it is placed. |
| \a | Audible alert | Alerts the user by sounding the speaker inside the computer |

Here is an example program to implement escape sequence characters to print "India" in different forms. Use "\n" in each string to insert a new line.

```
#include<stdio.h>
#include<conio.h>
main()
{
clrscr();
printf ("\aIndia\n");
printf ("\"India\");
printf ("\'India\'\n");
printf ("I\tn\td\n");
printf ("\\India\\");
getch();
}
```

**Output**

```
India
"India"
'India'
I       n       d
\India\
```

### 2.6.5 Variables

A variable is a name that can change its value in the program. It is label given to a memory location and used to store a data value. Variable names are case sensitive e.g. 'NAME' and 'name' are different variables.

- **Rules for defining variables**

The following are the rules for writing a variable name in a program in C:

The first character may be an alphabetor an underscore (_).

The first character of variable name cannot be a digit.

Blank spaces are not allowed.

Special characters, such as arithmetic operators, #, ^, cannot be used.

Reserved C words cannot be used as variable names.

The maximum length of a variable name is up to 31 characters. However, the maximum length depends upon the compiler.

Following are some examples of the valid and invalid variable names.

| Variable Name | Valid / Invalid | Remarks |
|---|---|---|
| India | Valid | |
| Perform | Valid | |
| double | Invalid | C reserved word |
| Foxpro | Valid | |
| switch | Invalid | C reserved word |
| _large | Valid | |
| Float | Valid | The keyword float is in lower case |
| int | Invalid | C reserved word |
| 3taq | Invalid | Starts with a numeral |
| unsigned | Invalid | C reserved word |
| x-y | Invalid | Special character is not allowed |
| Taq Ahd | Invalid | Space is not allowed |

- **Declaration of Variables**

Giving the name of a variable and the data type is called declaration of the variable. The format for the declaration of a variable is as follows:

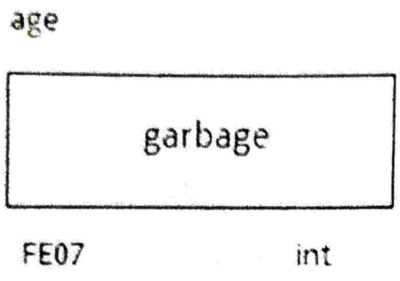
```
data_type variable-name;
```

For example, to declare a variable "age" of integer type, the statement is written as:

```
int age;
```

Visualization of the variable declaration:

int age;




All variables used in a program must be declared before use. The variables are used to store values.

- **Declaration of Multiple Variables**

More than one variable of the same data type can also be declared in one statement. For example,

```
int  second, minute, hour;
```

In this case. the names of the variables are written after the keyword separated by commas.

- **Assigning a value into a Variable**

After declaring a variable we can assign a value to a variable using assignment operator. For example:

```
int age;
age = 24;
```

When you assign a value into a variable, the previous value in that variable is overwritten and therefore destroyed.

- **Initialization of Variables**

Assigning a known value to a variable at the time of its declaration is called initializing of the variable.

For Example,

```
int a=10;
```

a | 10

```
float num=67.3f;
```

num | 67.3

Initializing a variable is a process of declaring as well as defining it. It is when a real memory is allocated to it. Multiple initialization of variable is also allowed as:

```
int a = 0, b = 10, c, d = 20;
int a=100, b=60, c;
```

## 2.7 DYNAMIC INITIALIZATION

Dynamic initialization is the process of declaring and assigning the value to variable in the expression itself. For example,

```
int a,b;
a=10;
b=20;
int c=a+b;
```

Here the variable c is declared and initialized in the addition statement, this is called dynamic initialization of variable.

### 2.7.1 Variable scope

Scope of a variable is the portion of the program where a defined variable can be accessed.Scope depends on the location of the variable declaration. C scope rules can be covered under the following two categories.

There are basically 4 scope rules:

| Scope | Meaning |
|---|---|
| File Scope | Scope of a Identifier starts at the beginning of the file and ends at the end of the file. It refers to only those Identifiers that are declared outside of all functions. The Identifiers of File scope are visible all over the file Identifiers having file scope are global |
| Block Scope | Scope of a Identifier begins at opening of the block / '{' and ends at the end of the block / '}'. Identifiers with block scope are local to their block |
| Function Prototype Scope | Identifiers declared in function prototype are visible within the prototype |
| Function scope | Function scope begins at the opening of the function and ends with the closing of it. Function scope is applicable to labels only. A label declared is used as a target to goto statement and both goto and label statement must be in same function |

- **FILE SCOPE**

These variables are usually declared outside of all of the functions and blocks, at the top of the program and can be accessed from any portion of the program. These are also called the global scope variables as they can be globally accessed.

➢ **Example 1:**

```
#include <stdio.h>
// Global variable
int global = 5;
// global variable accessed from within a function
void display()
{
printf("%d\n", global);
}
```

```
// main function
int main()
{
    printf("Before change within main: ");
    display();

    // changing value of global variable from main function
    printf("After change within main: ");
    global = 10;
    display();
}
```

**Output :**

```
Before change within main: 5
After change within main: 10
```

➢ **Example 2 :**

```
// filename: file1.c
int a;
int main(void)
{
    a = 2;
}
// filename: file2.c
// When this file is linked with file1.c, functions of this file can access a
extern int a;
int myfun()
{
    a = 2;
}
```

## ■ BLOCK SCOPE

A Block is a set of statements enclosed within left and right braces i.e. '{' and '}'. Blocks may be nested in C(a block may contain other blocks inside it). A variable declared inside a block is accessible in the block and all inner blocks of that block, but not accessible outside the block. Basically, these are local to the blocks in which the variables are defined and are not accessible outside.

**Example:**

```
#include <stdio.h>
int main()
   {
       {
      int x = 10, y = 20;
      {
         // The outer block contains declaration of x and y, so following statement
         // is valid and prints10 and 20
         printf("x = %d, y = %d\n", x, y);
         {
            // y is declared again,so outer block y isnot accessible in this block
            int y = 40;
            // Changes the outer blockvariable x to 11
x=x+1;
            // Changes this block's variable y to 41
y=y+1;
            printf("x = %d, y = %d\n", x, y);
         }
         // This statement accessesonly outer block'svariables
         printf("x = %d, y = %d\n", x, y);
      }
   }
   return 0;
}
```

**Output :**

```
x = 10, y = 20
x = 11, y = 41
x = 11, y = 20
```

- **Difference between local and global variable :**

| Local Variable | Global Variable |
|---|---|
| Declared inside function body. | Declared outside function body. |
| Not initialized automatically. | Initialized automatically by value 0. |
| Use of local variables advisable. | Too much use of global variables make program difficult to debug, so use with care. |

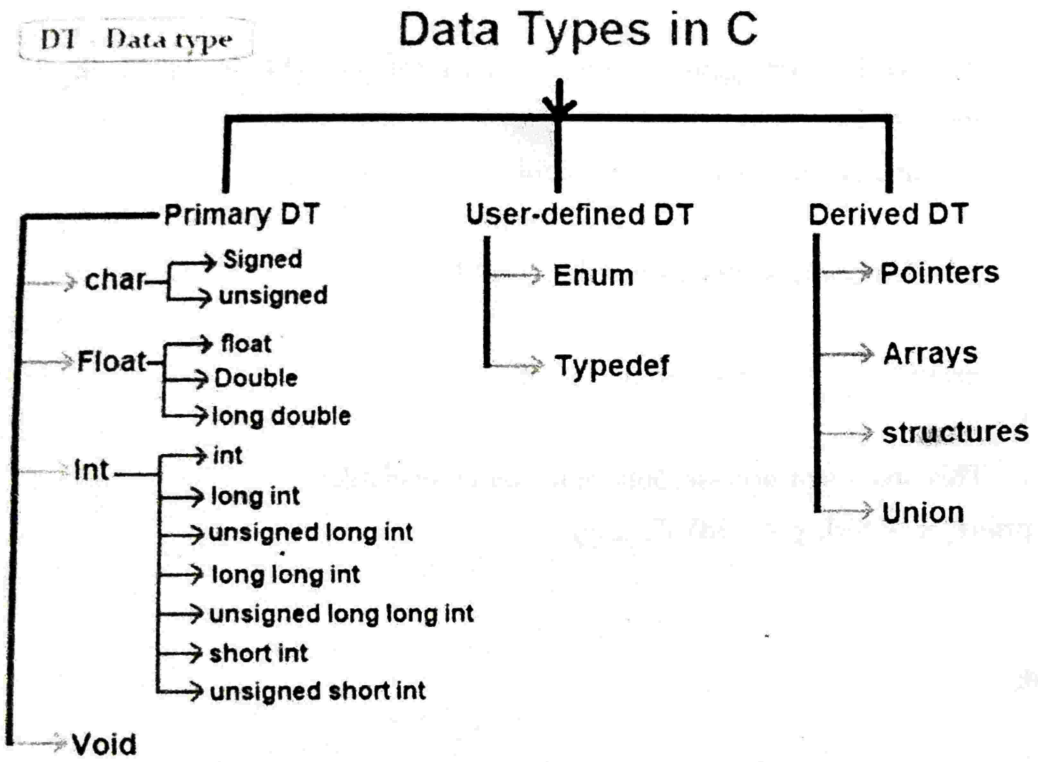
## 2.8 DATA TYPES

Data types indicate the type of value stored in a variable, the number of bytes to be reserved in memory, the range of values that can be represented in memory.

Data types in C Language are classified into three types as follows.

1. **Primitive Data Types :** Integer, character, float, void. All these are called primitive data types.
2. **Derived Data Types :** Array, String, Pointer,Structure, unionetc. come under derived data types.
3. **User-Defined Data Types :** typedef, enum, etc. are comes under user-defined data types.

**C Language data types.**




### 2.8.1 Primary Data Type

As we already discussed the Primitive Data Types are classified into four types are as follows.

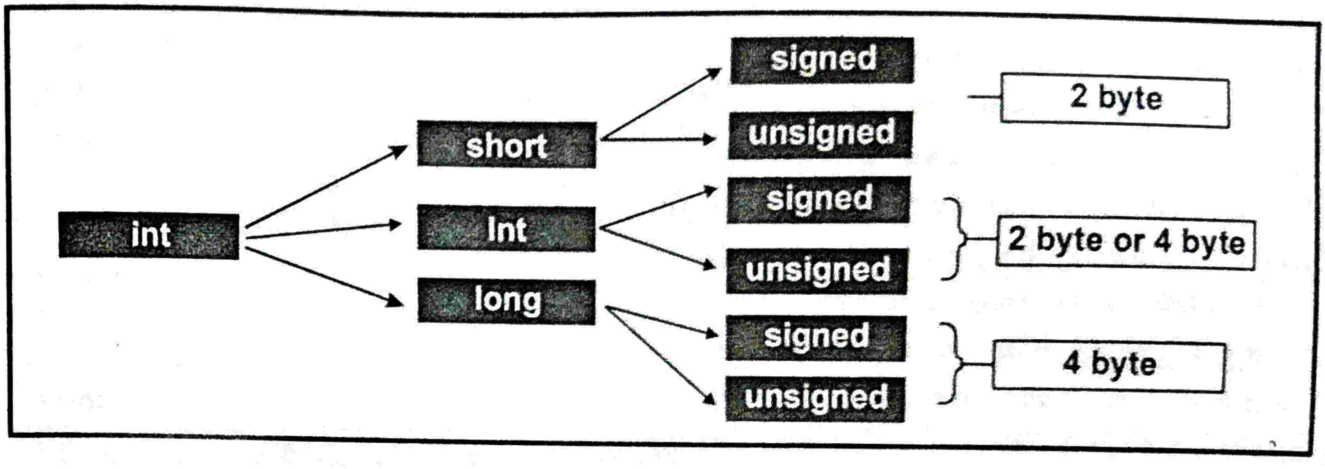
- Integer
- Character
- Float
- Void

■ **INTEGER DATA TYPE**

Again, Integer is divided into three types are as follows.

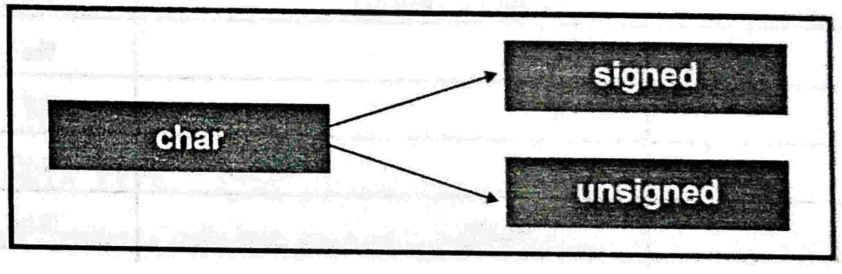
1. **Short**
2. **Int**
3. **Long**

Again, the short data type is divided into two types i.e. signed short and unsigned short. Same for int and long i.e. signed int, unsigned int, signed long, and unsigned long. So, one integer data type is again subdivided into 6 types. For a better understanding of the integer data types, please have a look at the below image.



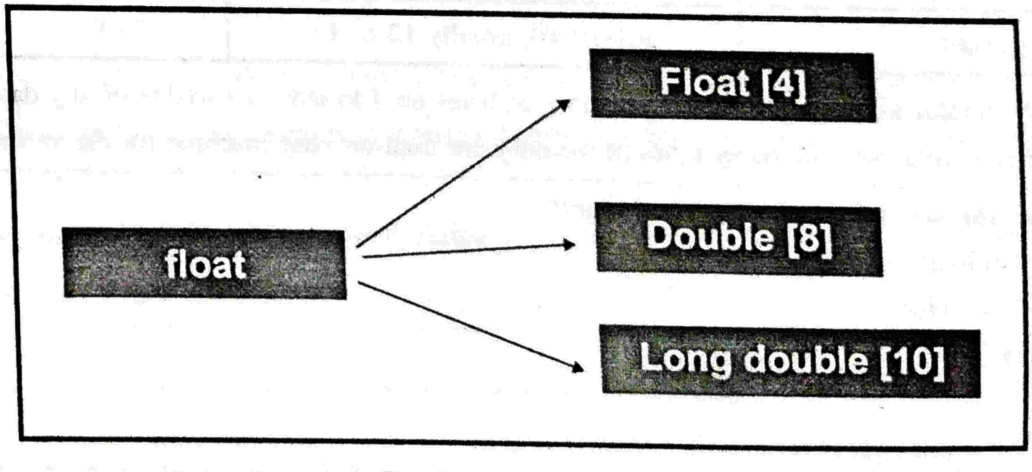

- **Character Data Type in C Language:**

The Character Data Type in C language is divided into two types. One is a signed character and the second one is an unsigned character. Both are of size 1 byte. For a better understanding of the Character data types, please have a look at the following image.




- **Float Data Type in C Language**

The Float Data Type in C language is divided into three types one is float type, the second one is double and the last one is long double. Float is of size 4 bytes; double is of size 8 bytes and long double is of size 10 byte. For better understanding, please have a look at the following diagram.



**Example to demonstrate the Built-in Data Types in C Language**

```
#include <stdio.h>
int main()
{
int a = 4000; // positive integer data type
float b = 5.2324; // float data type
char c = 'Z'; // char data type
long d = 41657; // long positive integer data type
long e = -21556; // long -ve integer data type
int f = -185; // -ve integer data type
short g = 130; // short +ve integer data type
short h = -130; // short -ve integer data type
double i = 4.1234567890; // double float data type
float j = -3.55; // float data type
}
```

Here's a table containing commonly used types in C programming for quick access.

| Type | Size (bytes) | Format Specifier |
|---|---|---|
| int | at least 2, usually 4 | %d, %i |
| char | 1 | %c |
| float | 4 | %f |
| double | 8 | %lf |
| short int | 2 usually | %hd |
| unsigned int | at least 2, usually 4 | %u |
| long int | at least 4, usually 8 | %ld, %li |
| unsigned long int | at least 4 | %lu |
| signed char | 1 | %c |
| unsigned char | 1 | %c |
| long double | at least 10, usually 12 or 16 | %Lf |

The sizeof operator allows us to find the number of bytes used to store a variable of any data type. The following program is to find out how many bytes of memory are used on your machine for the various data types.

```
/*Compute the size of the fundamental types*/
#include <stdio.h>
#include <conio.h>
int main(void)
{
clrscr();
```

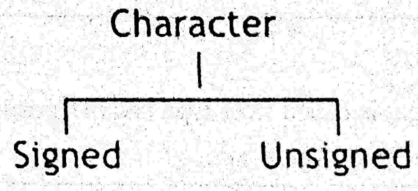
```
printf("The size of some fundamental types is computed.\n\n");
printf("char:       %3d byte \n", sizeof(char));
printf("short:          %3d byte \n", sizeof(short));
printf("int:        %3d byte \n", sizeof(int));
printf("long:       %3d byte \n", sizeof(long));
printf("unsigned:       %3d byte \n", sizeof(unsigned));
printf("float:      %3d byte \n", sizeof(float));
printf("double:         %3d byte \n", sizeof(double));
printf("long double:    %3d byte \n", sizeof(long double));
return 0;
}
```

**Output :**

```
The size of some fundamental types is computed.
char:         1 byte
short:        2 byte
int:          2 byte
long:         4 byte
unsigned:     2 byte
float:        4 byte
double:       8 byte
long double:  10 byte
```

## ■ CHARACTER DATA TYPE

Character type data consists of alphabetic characters, numeric digits and special characters. Characters are stored in 8 bits of memory. The qualifier signed or unsigned can be applied to char.




Unsigned characters have values between 0 and 255, signed characters have values from –128 to 127.

All the characters on your keyboard have a unique numerical code (ASCII Code) associated with it. For example, the ASCII code for the character 'm' is 109

### ➢ Declaring and Initializing Character Variables

If you declared c as a character as

```
char c;
```

then you can assign A in different ways as follows:

```
c = 'A';
c = 65;
c = '\x41';   /* Hexadecimal representation */
c = '\101';   /* Octal representation */
```

Example to demonstrate the character data type:

```
#include <stdio.h>
int main(void)
{
char c1, c2;
char c3, c4;
c1 = 65;
c2 = 97;
c3 = 'B';
c4 = 'b';
printf("Character that has the ASCII value of 65 is: %c.\n",c1);
printf("Character that has the ASCII value of 97 is: %c.\n",c2);
printf("The ASCII value of the character 'B' is %d.\n", c3);
printf("The ASCII value of the character 'b' is %d.\n", c4);
}
```

**Output**

```
Character that has the ASCII value of 65 is: A.
Character that has the ASCII value of 97 is: a.
The ASCII value of the character 'B' is 66.
The ASCII value of the character 'b' is 98.
```

## ■ FLOATING POINT TYPES

The data consisting of real numbers is called float type data. Depending upon the maximum and minimum values, there are three types of real type data in C. These are:

```
float
double
long double
```

float and double are used to hold real numbers.

```
float salary;
double price;
```

In C, floating-point numbers can also be represented in exponential. For example,

```
float Factor = 22.442e2;
```

You can use the format %f for printing floating numbers. For example, printf("%f\n", f); %f prints output with 6 decimal places. If you want to print output with 8 columns and 3 decimal places, you can use the format %8.3f. For printing double you can use %lf.

### ■ VOID TYPE

Using void data type, we can specify the type of a function. It is a good practice to avoid functions that does not return any values to the calling function.

**SAMPLE PROGRAM ILLUSTRATING EACH DATA TYPE**

```
#include < stdio.h >
main()
{
        int    sum;
        float money;
        char   letter;
        double pi;
        sum = 10;  /* assign integer value */
        money = 2.21;  /* assign float value */
        letter = 'A'; /* assign character value */
        pi = 2.01E6;/* assign a double value */
        printf("value of sum = %d\n", sum );
        printf("value of money = %f\n", money );
        printf("value of letter = %c\n", letter );
        printf("value of pi = %e\n", pi );
}
```

**Sample program output**

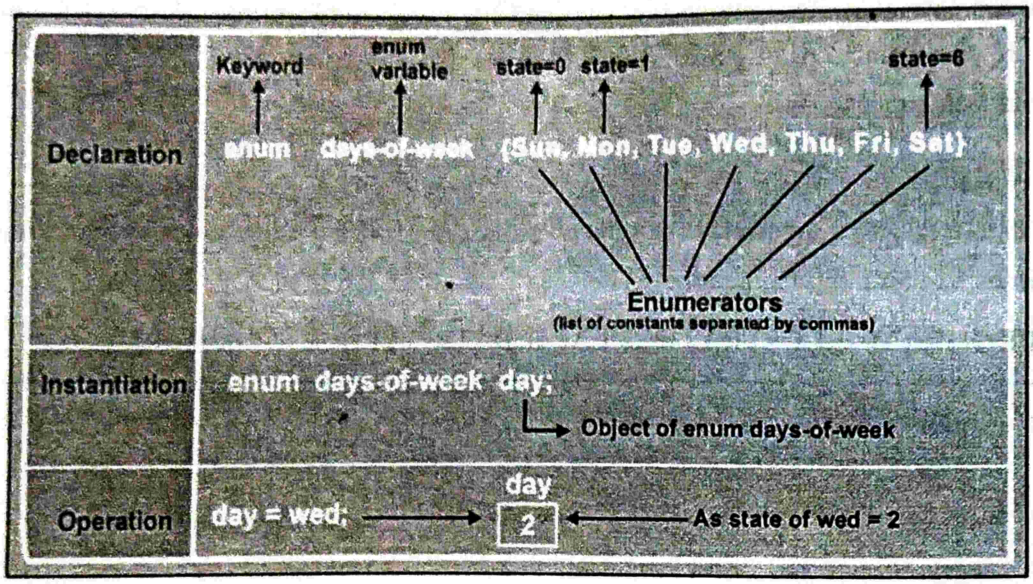
```
value of sum = 10
value of money = 2.210000
value of letter = A
value of pi = 2.010000e+06
```

## 2.8.2 User Defined Data Type

In C language a user can define an identifier that represents an existing data type. The user defined datatype identifier can later be used to declare variables.

### ■ THE ENUM KEYWORD

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.




The general syntax is as follows:

```
enum identifier {value1, value2 .... value n};
```

Here's an example:

```
enum colors {RED, YELLOW, GREEN, BLUE};
```

In above example now colors are a new data type defined by me. Now I can use color variables having any of the above given values. If you don't assign a value to a constant, the default value for the first one in the list - RED in our case, has the value of 0.The rest of the undefined constants have a value 1 more than the one before, so in our case, YELLOW is 1, GREEN is 2 and BLUE is 3 But you can assign values if you wanted to:

```
enum colors {RED=1, YELLOW, GREEN=6, BLUE };
```

Now RED=1, YELLOW=2, GREEN=6 and BLUE=7, Here is an example program:

```
#include <stdio.h>
int main()
{
  enum {RED=5, YELLOW, GREEN=-4, BLUE};
  printf("RED = %d\n", RED);
  printf("YELLOW = %d\n", YELLOW);
  printf("GREEN = %d\n", GREEN);
  printf("BLUE = %d\n", BLUE);
  return 0;
}
```

**Output :**

```
RED = 5
YELLOW = 6
GREEN = -4
BLUE = -5
```

## TYPEDEF (USING TYPE DEFINITIONS)

In C language a user can define an identifier that represents an existing data type. The user defined datatype identifier can later be used to declare variables. As its name implies, we can define our own types. The general syntax is:

```
typedef type identifier;
```

For example:

```
typedef int salary;
typedef float average;
```

Here salary symbolizes int and average symbolizes float. They can be later used to declare variables as follows:

```
salary dept1, dept2;.
average section1, section2;
```

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

Here is an example program

```
#include <stdio.h>
typedef unsigned short int USHORT;
int main()
{
  USHORT i = 2004;
  printf("Year %d\n", i);
  return 0;
}
```

**Output :**

```
Year 2004
```

### 2.8.3 Derived Data Type

Data types that are derived from fundamental data types are called derived data types.Derived data types do not create new data types. Instead, they add some functionality to the existing data types.Given below are the various derived data types used in C:

1. **Arrays** : An array is an ordered sequence of finite data items of the same data type that share a common name.
2. **pointers :** A pointer is a special type of variable used to hold the address of another variable.
3. **Structures :** A structure is a collection of different data type items stored in a contiguous memory allocation.
4. **Unions :** A union is similar to a structure where the memory allocated to the largest data type is reused for other types in the group.

## 2.8.4 CONSTANT

In C programming language, constants can be declared or defined in two ways one is using a keyword "const" and the other is using #define preprocessor.

- Using the "define" Directive
- Using the "const" Directive

***Using define directive***

Syntax:

```
#define symbolic_name value
```

For Example:

```
#define PI  3.1415
```

Here PI is symbolic constant. Valid examples of constant are:

```
#define TOTALMARKS 100
#define TRUE  1
#define FALSE 0
#define NAME_SIZE 20
#define FLAG  1
#define ANGLE_MIN 0
#define ANGLE_MAX 360
```

➢ **Example**

```
#include <stdio.h>
#define LENGTH 20
#define WIDTH  30
int main() {
int area_r, area_s;
area_r = LENGTH * WIDTH;
printf("Area of rectange is l * b =  %d\n", area_r);
area_s = LENGTH * LENGTH;
printf("Area of square is l * b =  %d", area_s);
return 0;
}
```

**Output :**

```
Area of rectangle is l*b = 600
Area of square is l*b = 400
```