LECTURE NOTE -3

INTRODUCTION TO C

Brief History of C

- The C programming language is a structure oriented programming language, developed at Bell Laboratories in 1972 by Dennis Ritchie.
- C programming language features were derived from an 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 is 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.
- Many of C's ideas & principles were derived from the earlier language B, thereby naming this new language "C".

Taxonomy of C Language

WHY IS C POPULAR

- It is reliable, simple and easy to use.
- C is a small, block-structured programming language.
- C is a portable language, which means that C programs written on one system can be run on other systems with little or no modification.
- C has one of the largest assortments of operators, such as those used for calculations and data comparisons.
- Although the programmer has more freedom with data storage, the languages do not check data type accuracy for the programmer.

WHY TO STUDY C

- By the early 1980s, C was already a dominant language in the minicomputer world of Unix systems. Since then, it has spread to personal computers (microcomputers) and to mainframes.
- Many software houses use C as the preferred language for producing word processing programs, spreadsheets, compilers, and other products.
- C is an extremely flexible language—particularly if it is to be used to write operating systems.
- Unlike most other languages that have only four or five levels of precedence, C has 15.

CHARECTERESTICS OF A C PROGRAM

- Middle level language.

High Level	*Middle Level*	*Low Level*
High level languages provide almost everything that the programmer might need to do as already built into the language	Middle level languages don't provide all the built-in functions found in high level languages, but provides all building blocks that we need to produce the result we want	Low level languages provides nothing other than access to the machines basic instruction set
Examples: Java, Python	C, C++	Assembler

- Small size – has only 32 keywords
- Extensive use of function calls- enables the end user to add their own functions to the C library.
- Supports loose typing – a character can be treated as an integer & vice versa.
- Structured language

Structure oriented	*Object oriented*	*Non structure*
In this type of language, large programs are divided into small programs called functions	In this type of language, programs are divided into objects	There is no specific structure for programming this language
Prime focus is on functions and procedures that operate on the data	Prime focus is in the data that is being operated and not on the functions or procedures	N/A
Data moves freely around the systems from one function to another	Data is hidden and cannot be accessed by external functions	N/A
Program structure follows "Top Down Approach"	Program structure follows "Bottom UP Approach"	N/A
Examples: C, Pascal, ALGOL and Modula-2	C++, JAVA and C# (C sharp)	BASIC, COBOL, FORTRAN

- Low level (Bit Wise) programming readily available
- Pointer implementation - extensive use of pointers for memory, array, structures and functions.
- It has high-level constructs.
- It can handle low-level activities.
- It produces efficient programs.
- It can be compiled on a variety of computers.

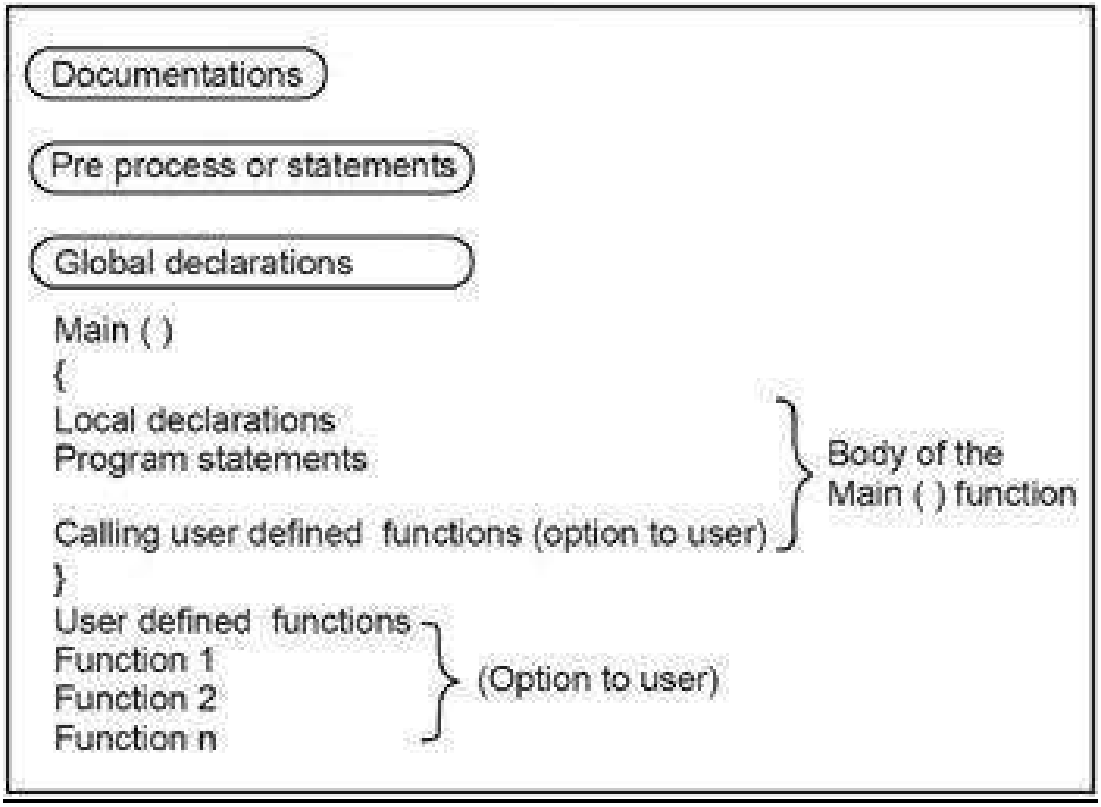
USES

The C programming language is used for developing system applications that forms a 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
- Spreadsheets
- Operating system development
- Compilers and Assemblers
- Network drivers
- Interpreters

STRUCTURE OF A C PROGRAM

The structure of a C program is a protocol (rules) to the programmer, which he has to follow while writing a C program. The general basic structure of C program is shown in the figure below.



Based on this structure, we can sketch a C program.

Example:

```
/* This program accepts a number & displays it to the user*/

#include <stdio.h>
void main(void)
{ int number;
printf( "Please enter a number: " );
scanf( "%d", &number );
printf( "You entered %d", number );
return 0;}
```

Stepwise explanation:

#include

- The part of the compiler which actually gets your program from the source file is called the preprocessor.
    - *#include <stdio.h>*
- #include is a pre-processor directive. It is not really part of our program, but instead it is an instruction to the compiler to make it do something. It tells the C compiler to include the contents of a file (in this case the system file called stdio.h).
- The compiler knows it is a system file, and therefore must be looked for in a special place, by the fact that the filename is enclosed in <> characters

<stdio.h>

- *stdio.h* is the name of the standard library definition file for all STanDard Input and Output functions.
- Your program will almost certainly want to send information to the screen and read things from the keyboard, and stdio.h is the name of the file in which the functions that we want to use are defined.
- The function we want to use is called printf. The actual code of printf will be tied in later by the linker.
- The ".h" portion of the filename is the language extension, which denotes an include file.

void

- This literally means that this means nothing. In this case, it is referring to the function whose name follows.
- Void tells to C compiler that a given entity has no meaning, and produces no error.

main

- In this particular example, the only function in the program is called *main*.
- A C program is typically made up of large number of functions. Each of these is given a name by the programmer and they refer to each other as the program runs.
- C regards the name main as a special case and will run this function first i.e. the program execution starts from *main*.

(void)

- This is a pair of brackets enclosing the keyword *void*.
- It tells the compiler that the function main has no parameters.
- A parameter to a function gives the function something to work on.

{ (Brace)

- This is a brace (or curly bracket). As the name implies, braces come in packs of two - for every open brace there must be a matching close one.
- Braces allow us to group pieces of program together, often called a block.
- A block can contain the declaration of variable used within it, followed by a sequence of program statements.
- In this case the braces enclose the working parts of the function main.

; (semicolon)

- The semicolon marks the end of the list of variable names, and also the end of that declaration statement.
- All statements in C programs are separated by ";" (semicolon) characters.
- The ";" character is actually very important. It tells the compiler where a given statement ends.
- If the compiler does not find one of these characters where it expects to see one, then it will produce an error.

scanf

- In other programming languages, the printing and reading functions are a part of the language.
- In C this is not the case; instead they are defined as standard functions which are part of the language specification, but are not a part of the language itself.
- The standard input/output library contains a number of functions for formatted data transfer; the two we are going to use are scanf (scan formatted) and printf (print formatted).

printf

- The printf function is the opposite of scanf.
- It takes text and values from within the program and sends it out onto the screen.
- Just like scanf, it is common to all versions of C and just like scanf, it is described in the system file stdio.h.
- The first parameter to a printf is the format string, which contains text, value descriptions and formatting instructions.

FILES USED IN A C PROGRAM

- ***Source File-*** This file contains the source code of the program. The file extension of any c file is **.c**. The file contains C source code that defines the *main* function & maybe other functions.

- ***Header File-*** A header file is a file with extension **.h** which contains the C function declarations and macro definitions and to be shared between several source files.
- ***Object File-*** An object file is a file containing object code, with an extension **.o**, meaning relocatable format machine code that is usually not directly executable. Object files are produced by an assembler, compiler, or other language translator, and used as input to the linker, which in turn typically generates an executable or library by combining parts of object files.
- ***Executable File-*** The binary executable file is generated by the linker. The linker links the various object files to produce a binary file that can be directly executed.

COMPLIATION & EXECUTION OF A C PROGRAM