[redacted]

LECTURE NOTE-1

[redacted]

INTRODUCTION TO COMPUTERS

Any programming language is implemented on a computer. Right form its inception, to the present day, all computer system (irrespective of their shape & size) perform the following 5 basic operations. It converts the raw input data into information, which is useful to the users.

- *Inputting*: It is the process of entering data & instructions to the computer system.
- *Storing*: The data & instructions are stored for either initial or additional processing, as & when required.
- *Processing*: It requires performing arithmetic or logical operation on the saved data to convert it into useful information.
- *Outputting*: It is the process of producing the output data to the end user.
- *Controlling*: The above operations have to be directed in a particular sequence to be completed.

Based on these 5 operations, we can sketch the block diagram of a computer.

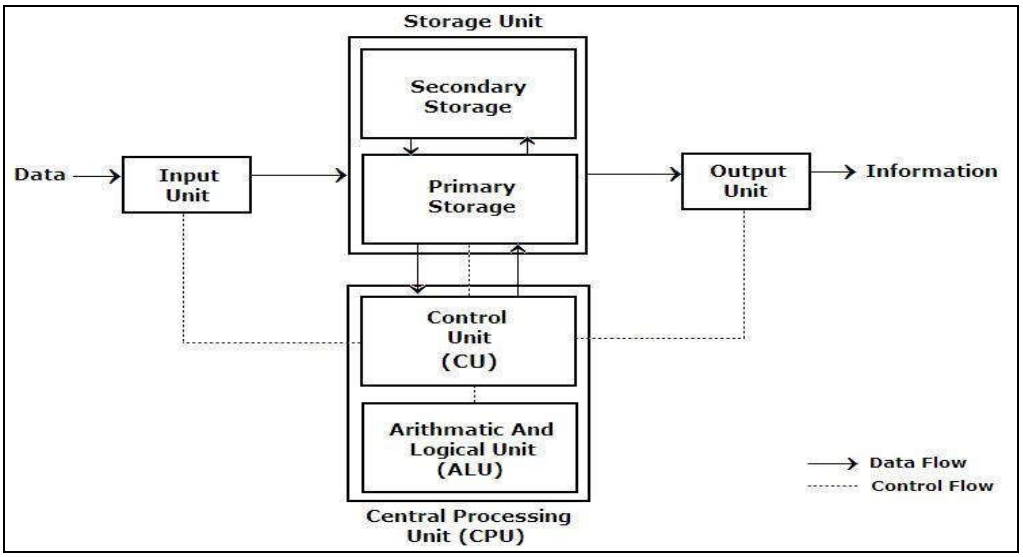


Fig 1: Block Diagram of a Computer

- *Input Unit*: We need to first enter the data & instruction in the computer system, before any computation begins. This task is accomplished by the input devices. (Eg: keyboard, mouse, scanner, digital camera etc). This device is responsible for linking the system with the external environment. The data accepted is in a human readable form. The input device converts it into a computer readable form.

- *Storage Unit*: The data & instruction that are entered have to be stored in the computer. Similarly, the end results & the intermediate results also have to be stored somewhere before being passed to the output unit. The storage unit provides solution to all these issues. This storage unit is designed to save the initial data, the intermediate result & the final result. This storage unit has 2 units: *Primary storage* & *Secondary storage*.

  Primary Storage: The primary storage, also called as the *main memory*, holds the data when the computer is currently on. As soon as the system is switched off or restarted, the information held in primary storage disappears (i.e. it is volatile in nature). Moreover, the primary storage normally has a limited storage capacity, because it is very expensive as it is made up of semiconductor devices.

  Secondary Storage: The secondary storage, also called as the *auxiliary storage,* handles the storage limitation & the volatile nature of the primary memory. It can retain information even when the system is off. It is basically used for holding the program instructions & data on which the computer is not working on currently, but needs to process them later.

- *Central Processing Unit*: Together the Control Unit & the Arithmetic Logic Unit are called as the Central Processing Unit (CPU). The CPU is the brain of the computer. Like in humans, the major decisions are taken by the brain itself & other body parts function as directed by the brain. Similarly in a computer system, all the major calculations & comparisons are made inside the CPU. The CPU is responsible for activating & controlling the operation of other units of the computer system.

  Arithmetic Logic Unit: The actual execution of the instructions (arithmetic or logical operations) takes place over here. The data & instructions stored in the primary storage are transferred as & when required. No processing is done in the primary storage. Intermediate results that are generated in ALU are temporarily transferred back to the primary storage, until needed later. Hence, data may move from the primary storage to ALU & back again to storage, many times, before the processing is done.
  Control Unit: This unit controls the operations of all parts of the computer but does not carry out any actual data processing.It is responsible for the transfer of data and instructions among other units of the computer.It manages and coordinates all the units of the system.It also communicates with Input/Output devices for transfer of data or results from the storage units.

- *Output Unit*: The job of an output unit is just the opposite of an input unit. It accepts the results produced by the computer in coded form. It converts these coded results to human readable form. Finally, it displays the converted results to the outside world with the help of output devices (Eg :monitors, printers, projectors etc..).

So when we talk about a computer, we actually mean 2 things:

- *Hardware*- This hardware is responsible for all the physical work of the computer.
- *Software*- This software commands the hardware what to do & how to do it.

Together, the hardware & software form the computer system.

This software is further classified as system software & application software.

System Software- System software are a set of programs, responsible for running the computer, controlling various operations of computer systems and management of computer resources. They act as an interface between the hardware of the computer & the application software. E.g.: Operating System

Application Software- Application software is a set of programs designed to solve a particular problem for users. It allows the end user to do something besides simply running the hardware. E.g.: Web Browser, Gaming Software, etc.

LECTURE NOTE-2

INTRODUCTION TO PROGRAMMING

A language that is acceptable to a computer system is called a ***computer language*** or ***programming language*** and the process of creating a sequence of instructions in such a language is called ***programming*** or ***coding***. A program is a set of instructions, written to perform a specific task by the computer. A set of large program is called ***software***. To develop software, one must have knowledge of a programming language.

Before moving on to any programming language, it is important to know about the various types of languages used by the computer. Let us first know what the basic requirements of the programmers were & what difficulties they faced while programming in that language.

COMPUTER LANGUAGES

Languages are a means of communication. Normally people interact with each other through a language. On the same pattern, communication with computers is carried out through a language. This language is understood both by the user and the machine. Just as every language like English, Hindi has its own grammatical rules; every computer language is also bounded by rules known as *syntax* of that language. The user is bound by that syntax while communicating with the computer system.

Computer languages are broadly classified as:

- ***Low Level Language***: The term low level highlights the fact that it is closer to a language which the machine understands.

  The low level languages are classified as:

  - ***Machine Language***: This is the language (in the form of 0's and 1's, called binary numbers) understood directly by the computer. It is machine dependent. It is difficult to learn and even more difficult to write programs.

  - ***Assembly Language***: This is the language where the machine codes comprising of 0'sand 1's are substituted by symbolic codes (called mnemonics) to improve their understanding. It is the first step to improve programming structure. Assembly language programming is simpler and less time consuming than machine level programming, it is easier to locate and correct errors in assembly language than in machine language programs. It is also machine dependent. Programmers must have knowledge of the machine on which the program will run.

- ***High Level Language:*** Low level language requires extensive knowledge of the hardware since it is machine dependent. To overcome this limitation, high level language has been evolved which uses normal English, which is easy to understand to solve any problem. High level languages are computer independent and programming becomes quite easy and simple. Various high level languages are given below:

    - BASIC (Beginners All Purpose Symbolic Instruction Code): It is widely used, easy to learn general purpose language. Mainly used in microcomputers in earlier days.
    - COBOL (Common Business Oriented language): A standardized language used for commercial applications.
    - FORTRAN (Formula Translation): Developed for solving mathematical and scientific problems. One of the most popular languages among scientific community.
    - C: Structured Programming Language used for all purpose such as scientific application, commercial application, developing games etc.
    - C++: Popular object oriented programming language, used for general purpose.

PROGRAMMING LANGUAGE TRANSLATORS

As you know that high level language is machine independent and assembly language though it is machine dependent yet mnemonics that are being used to represent instructions are not directly understandable by the machine. Hence to make the machine understand the instructions provided by both the languages, programming language instructors are used. They transform the instruction prepared by programmers into a form which can be interpreted & executed by the computer. Flowing are the various tools to achieve this purpose:

- ***Compiler***: The software that reads a program written in high level language and translates it into an equivalent program in machine language is called as compiler. The program written by the programmer in high level language is called source program and the program generated by the compiler after translation is called as object program.

- ***Interpreter***: it also executes instructions written in a high level language. Both complier & interpreter have the same goal i.e. to convert high level language into binary instructions, but their method of execution is different. The complier converts the entire source code into machine level program, while the interpreter takes 1 statement, translates it, executes it & then again takes the next statement.

- ***Assembler***: The software that reads a program written in assembly language and translates it into an equivalent program in machine language is called as assembler.

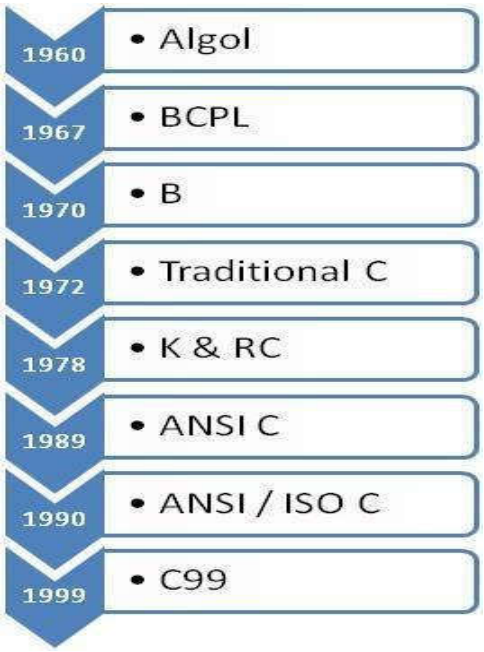
- ***Linker***: A linker or link editor is a computer program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another object file.

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.

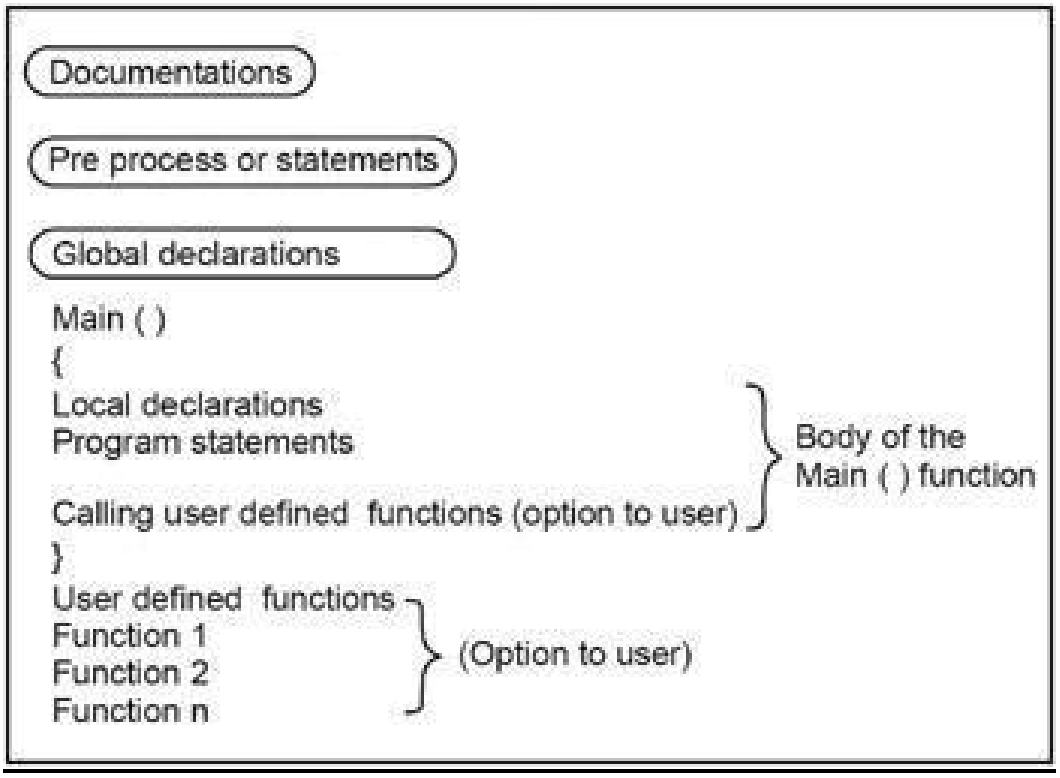
USES

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

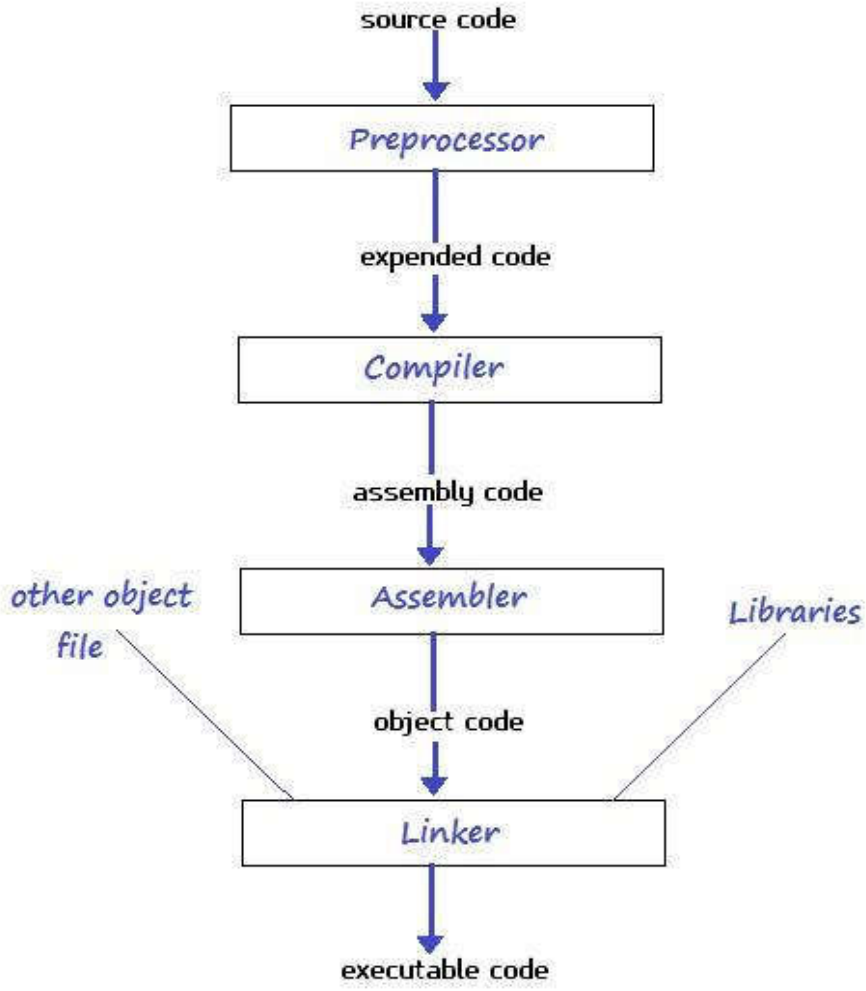
- The printf function is the opposite of scanf.
- It takes text and values from within the program and sends it out onto the screen.
- Just like scanf, it is common to all versions of C and just like scanf, it is described in the system file stdio.h.
- The first parameter to a printf is the format string,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



LECTURE NOTE -4

ELEMENTS OF C

Every language has some basic elements & grammatical rules. Before starting with programming, we should be acquainted with the basic elements that build the language.

Character Set

Communicating with a computer involves speaking the language the computer understands. In C, various characters have been given to communicate.

Character set in C consists of;

Types	Character Set
Lower case	a-z
Upper case	A-Z
Digits	0-9
Special Character	!@#$%^&*
White space	Tab or new lines or space

Keywords

Keywords are the words whose meaning has already been explained to the C compiler. The keywords cannot be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer.

There are only 32 keywords available in C. Below figure gives a list of these keywords for your ready reference.

KEYWORDS

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typeodef	
default	for	short	union	

Identifier

In the programming language C, an identifier is a combination of alphanumeric characters, the first being a letter of the alphabet or an underline, and the remaining being any letter of the alphabet, any numeric digit, or the underline.

Two rules must be kept in mind when naming identifiers.

1. The case of alphabetic characters is significant. Using "INDEX" for a variable is not the same as using "index" and neither of them is the same as using "InDeX" for a variable. All three refer to different variables.
2. As C is defined, up to 32 significant characters can be used and will be considered significant by most compilers. If more than 32 are used, they will be ignored by the compiler.

Data Type

In the C programming language, data types refer to a domain of allowed values & the operations that can be performed on those values. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. There are 4 fundamental data types in C, which are- *char, int, float &, double. Char* is used to store any single character; *int* is used to store any integer value, *float* is used to store any single precision floating point number & *double* is used to store any double precision floating point number. We can use 2 qualifiers with these basic types to get more types.

There are 2 types of qualifiers-

Sign qualifier- signed & unsigned
Size qualifier- short & long

The data types in C can be classified as follows:

Type	**Storage size**	**Value range**
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535

long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

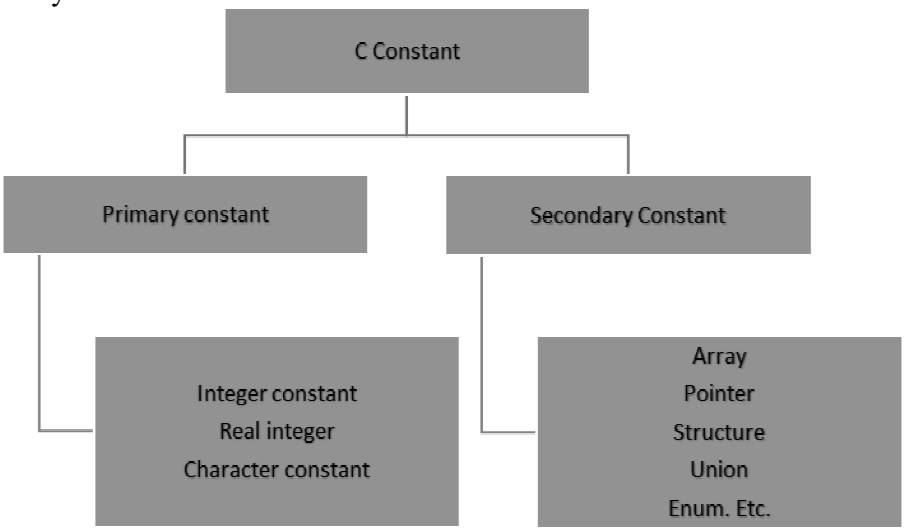
Type	**Storage size**	**Value range**	**Precision**
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal places
double	8 bytes	2.3E-308 to 1.7E+308	15 decimal places
long double	10 bytes	3.4E-4932 to 1.1E+4932	19 decimal places

Constants

A constant is an entity that doesn't change whereas a variable is an entity that may change.

C constants can be divided into two major categories:

- Primary Constants
- Secondary Constants



Here our only focus is on primary constant. For constructing these different types of constants certain rules have been laid down.

Rules for Constructing Integer Constants:

An integer constant must have at least one digit.

a) It must not have a decimal point.
b) It can be either positive or negative.

c) If no sign precedes an integer constant it is assumed to be positive.

d) No commas or blanks are allowed within an integer constant.

e) The allowable range for integer constants is -32768to 32767.

Ex.: 426, +782,-8000, -7605

Rules for Constructing Real Constants:

Real constants are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form.

Rules for constructing real constants expressed in fractional form:

a) A real constant must have at least one digit.

b) It must have a decimal point.

c) It could be either positive or negative.

d) Default sign is positive.

e) No commas or blanks are allowed within a real constant.

Ex. +325.34, 426.0, -32.76, -48.5792

Rules for constructing real constants expressed in exponential form:

a) The mantissa part and the exponential part should be separated by a letter e.

b) The mantissa part may have a positive or negative sign.

c) Default sign of mantissa part is positive.

d) The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.

e) Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Ex. +3.2e-5, 4.1e8, -0.2e+3, -3.2e-5

Rules for Constructing Character Constants:

a) A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.

b) The maximum length of a character constant can be 1 character.

Ex.: ‘M’, ‘6’, ‘+’

LECTURE NOTE-5

VARIABLES

Variables are names that are used to store values. It can take different values but one at a time. A data type is associated with each variable & it decides what values the variable can take. When you decide your program needs another variable, you simply declare (or define) a new variable and C makes sure you get it. You declare all C variables at the top of whatever blocks of code need them. Variable declaration requires that you inform C of the variable's name and data type. Syntax – datatype variablename;

Eg:

```
int page_no;

char grade;

float salary;

long y;
```

➢ **Declaring Variables:**

There are two places where you can declare a variable:

- After the opening brace of a block of code (usually at the top of a function)
- Before a function name (such as before main() in the program) Consider various examples:

  Suppose you had to keep track of a person's first, middle, and last initials. Because an initial is obviously a character, it would be prudent to declare three character variables to hold the three initials. In C, you could do that with the following statement:

```
1. main()
   {
   char first, middle, last;
    // Rest of program follows
   }
```

2. *main()*

   { char first;

   char middle;

   char last;

   // Rest of program follows

   }

- **Initialization of Variables**

  When a variable is declared, it contains undefined value commonly known as garbage value. If we want we can assign some initial value to the variables during the declaration itself. This is called *initialization of the variable.*

  Eg- int pageno=10;

  char grade='A';

  float salary= 20000.50;

Expressions

An expression consists of a combination of operators, operands, variables & function calls. An expression can be arithmetic, logical or relational. Here are some expressions:

a+b – arithmetic operation

a>b- relational operation a

== b - logical operation

func (a,b) – function call

4+21

a*(b + c/d)/20

q = 5*2 x =

++q % 3

q > 3

As you can see, the operands can be constants, variables, or combinations of the two. Some expressions are combinations of smaller expressions, called subexpressions. For example, c/d is a subexpression of the sixth example.

An important property of C is that every C expression has a value. To find the value, you perform the operations in the order dictated by operator precedence.

Statements

Statements are the primary building blocks of a program. A program is a series of statements with some necessary punctuation. A statement is a complete instruction to the computer. In C, statements are indicated by a semicolon at the end. Therefore

legs = 4

is just an expression (which could be part of a larger expression), but

legs = 4;

is a statement.

What makes a complete instruction? First, C considers any expression to be a statement if you append a semicolon. (These are called expression statements.) Therefore, C won't object to lines such as the following:

8;

3 + 4;

However, these statements do nothing for your program and can't really be considered sensible statements. More typically, statements change values and call functions:

x = 25;

++x;

y = sqrt(x);

Although a statement (or, at least, a sensible statement) is a complete instruction, not all complete instructions are statements. Consider the following statement:

x = 6 + (y = 5);

In it, the subexpression y = 5 is a complete instruction, but it is only part of the statement. Because a complete instruction is not necessarily a statement, a semicolon is needed to identify instructions that truly are statements.

Compound Statements (Blocks)

A compound statement is two or more statements grouped together by enclosing them in braces; it is also called a block. The following while statement contains an example:

```
while (years < 100)

{

   wisdom = wisdom * 1.05;

printf("%d %d\n", years, wisdom);

years = years + 1;


}
```

If any variable is declared inside the block then it can be declared only at the beginning of the block. The variables that are declared inside a block can be used only within the block.

LECTURE NOTE -6

INPUT-OUTPUT IN C

When we are saying **Input** that means we feed some data into program. This can be given in the form of file or from command line. C programming language provides a set of built-in functions to read given input and feed it to the program as per requirement.

When we are saying **Output** that means to display some data on screen, printer or in any file. C programming language provides a set of built-in functions to output the data on the computer screen.

Functions *printf()* and *scanf()* are the most commonly used to display out and take input respectively. Let us consider an example:

```
#include <stdio.h>      //This is needed to run printf() function.
int main()
{
   printf("C Programming");  //displays the content inside quotation
return 0;
}
```

Output:

```
C Programming
```

Explanation:

- Every program starts from *main()* function.
- *printf()* is a library function to display output which only works if *#include<stdio.h>*is included at the beginning.
- Here, *stdio.h* is a header file (standard input output header file) and *#include* is command to paste the code from the header file when necessary. When compiler encounters *printf()*function and doesn't find *stdio.h* header file, compiler shows error.
- *return 0*; indicates the successful execution of the program.

Input- Output of integers in C

```
#include<stdio.h>
int main()
{
int c=5;
```

```
printf("Number=%d",c);
   return 0;
}
```

Output

Number=5

Inside quotation of *printf()* there, is a conversion format string "%d" (for integer). If this conversion format string matches with remaining argument, i.e, *c* in this case, value of *c* is displayed.

```
#include<stdio.h>
int main()
{  int c;
   printf("Enter a number\n");
   scanf("%d",&c);
   printf("Number=%d",c);
   return 0;
}
```

Output

Enter a number
4
Number=4

The *scanf()* function is used to take input from user. In this program, the user is asked an input and value is stored in variable *c*. Note the '&' sign before *c*. &c denotes the address of *c* and value is stored in that address.

Input- Output of floats in C

```
#include <stdio.h>
int main()
{
        float a;
        printf("Enter value: ");
        scanf("%f",&a);
        printf("Value=%f",a);    //%f is used for floats instead of %d
        return 0;
}
```

Output

Enter value: 23.45
Value=23.450000

Conversion format string "%f" is used for floats to take input and to display floating value of a variable.

Input - Output of characters and ASCII code

```
#include <stdio.h>
int main()
{
   char var1;
   printf("Enter character: ");
   scanf("%c",&var1);
   printf("You entered %c.",var1);
   return 0;
}
```

Output
Enter character: g
You entered g.

Conversion format string "%c" is used in case of characters.

ASCII code

When character is typed in the above program, the character itself is not recorded a numeric value (ASCII value) is stored. And when we displayed that value by using "%c", that character is displayed.

```
#include <stdio.h>
int main()
{
char var1;
printf("Enter character: ");
scanf("%c",&var1);
printf("You entered %c.\n",var1);
/* \n prints the next line(performs work of enter). */
printf("ASCII value of %d",var1);
```

```
return 0;
}
```

Output:

```
Enter character:
g
103
```

When, *'g'* is entered, ASCII value 103 is stored instead of *g*.

You can display character if you know ASCII code only. This is shown by following example.

```
#include <stdio.h>
int main()
{
int var1=69;
printf("Character of ASCII value 69: %c",var1);
return 0;
}
```

Output

```
Character of ASCII value 69: E
```

The ASCII value of 'A' is 65, 'B' is 66 and so on to 'Z' is 90. Similarly ASCII value of 'a' is 97, 'b' is 98 and so on to 'z' is 122.

OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Increment and decrement operators
- Conditional operators
- Misc Operators

Arithmetic operator:

These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus.

Following table shows all the arithmetic operators supported by C language. Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A – B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2

%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A–will give 9

Relational Operators:

These operators are used to compare the value of two variables.

Following table shows all the relational operators supported by C language. Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators:

These operators are used to perform logical operations on the given two variables.

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are nonzero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and performs bit-by-bit operation. Bitwise operators are used in bit level programming. These operators can operate upon *int* and *char* but not on *float* and *double*.

Showbits( ) function can be used to display the binary representation of any integer or character value.

Bit wise operators in C language are; *&* (bitwise AND), | (bitwise OR), ~ (bitwise OR), ^ (XOR), << (left shift) and >> (right shift).

The truth tables for &, |, and ^ are as follows:

p	***q***	**p & q**	**p | q**	**p ^ q**
0	*0*	0	0	0
0	*1*	0	1	1
1	*1*	1	1	0
1	*0*	0	1	1

The Bitwise operators supported by C language are explained in the following table. Assume variable A holds 60 (00111100) and variable B holds 13 (00001101), then:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators:

In C programs, values for the variables are assigned using assignment operators.

There are following assignment operators supported by C language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A

-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

LECTURE NOTE-9

INCREMENT AND DECREMENT OPERATOR

In C, ++ and – are called increment and decrement operators respectively. Both of these operators are unary operators, i.e, used on single operand. ++ adds 1 to operand and – subtracts 1 to operand respectively. For example:

Let a=5 and b=10

a++; //a becomes 6

a--; //a becomes 5

++a; //a becomes 6

--a; //a becomes 5

When i++ is used as prefix(like: ++var), ++var will increment the value of var and then return it but, if ++ is used as postfix(like: var++), operator will return the value of operand first and then only increment it. This can be demonstrated by an example:

```
#include <stdio.h>
int main()
{
 int c=2,d=2;
printf("%d\n",c++); //this statement displays 2 then, only c incremented by 1 to 3.
Printf("%d",++c);  //this statement increments 1 to c then, only c is displayed.
Return 0;
}
```

Output

```
2
4
```

Conditional Operators (? :)

Conditional operators are used in decision making in C programming, i.e, executes different statements according to test condition whether it is either true or false.

Syntax of conditional operators;

conditional_expression?expression1:expression2

If the test condition is true (that is, if its value is non-zero), expression1 is returned and if false expression2 is returned.

Let us understand this with the help of a few examples:

```
int x, y ;

scanf ( "%d", &x ) ;

y = ( x> 5 ? 3 : 4 ) ;
```

This statement will store 3 in y if x is greater than 5, otherwise it will store 4 in y.

The equivalent if statement will be,

```
if ( x > 5 )

    y = 3 ;

else

    y = 4 ;
```

Misc Operators:

There are few other operators supported by c language.

Operator	Description	Example
sizeof()	It is a unary operator which is used in finding the size of data type, constant, arrays, structure etc.	sizeof(a), where a is integer, will return 4.

&	Returns the address of a variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* &sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<<>>	Left to right
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right

Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right