PROGRAMMING FOR PROBLEM SOLVING USING C
UNIT I

Introduction to Computers: Computer Systems, Computing Environments, Computer languages, Creating and running Programs, Computer Numbering System, Storing Integers, Storing Real Numbers

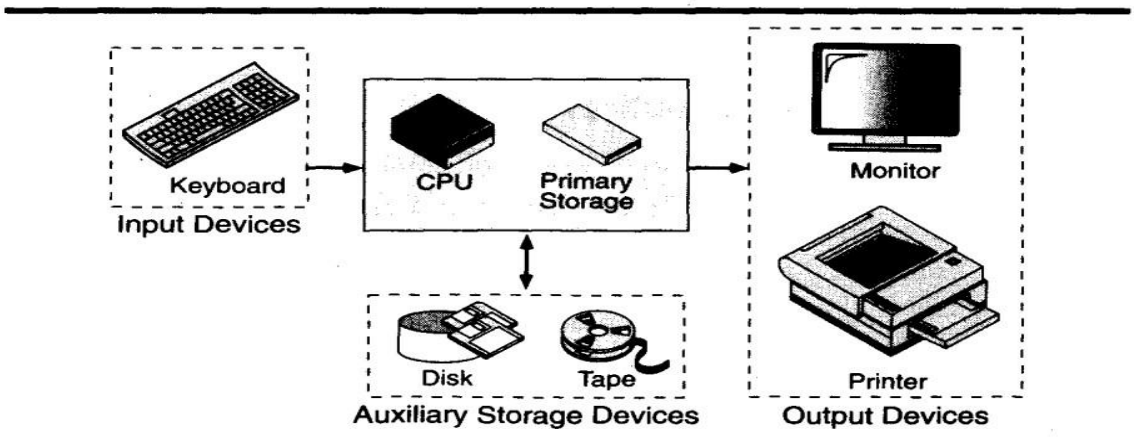
Introduction to the C Language: Background, C Programs, Identifiers, Types, Variable, Constants, Input/output, Programming Examples, Scope, Storage Classes and Type Qualifiers, Tips and Common Programming Errors Key Terms, Summary, Practice Seat.

Structure of a C Program: Expressions Precedence and Associativity, Side Effects, Evaluating Expressions, Type Conversion Statements, Simple Programs, Command Line Arguments Tips and Common Errors, Key Terms, Summary, Practice Sets.

➢ Computer Systems:-

A computer is a system made of two major components: hardware and software. The computer hardware is the physical equipment. The software is the collection of programs (instructions) that allow the hardware to do its job.

❖ **Computer Hardware: -** The hardware component of the computer system consists of five parts: input devices, central processing unit (CPU) ,primary storage, output devices, and auxiliary storage devices.



Basic Hardware Components

- The **input device** is usually a keyboard where programs and data are entered into the computers. Examples of other input devices include a mouse, a pen or stylus, a touch screen, or an audio input unit.
- The **central processing unit (CPU**) is responsible for executing instructions such as arithmetic calculations, comparisons among data, and movement of data inside the system.
- The **output device** is usually a monitor or a printer to show output. If the output is shown on the monitor, we say we have a **soft copy**. If it is printed on the printer, we say we have a hard copy.
- **Auxiliary storage**, also known as **secondary storage**, is used for both input and output. It is the place where the programs and data are stored permanently. When we turn off the computer, or programs and data remain in the secondary storage, ready for the next time we need them.

❖ **Computer Software :-**

Computer software is divided in to two broad categories: system software and application software.

- System software manages the computer resources .It provides the interface between the hardware and the users.
- Application software, on the other hand is directly responsible for helping users solve their problems.

➢ Computing Environments:-

Computing Environment is a collection of computers / machines, software, and networks that support the processing and exchange of electronic information meant to support various types of computing solutions. With the advent if technology the computing environments have been improved.

Types of Computing Environments:-

1. Personal Computing Environment
2. Time sharing Environment
3. Client Server Computing Environment
4. Distributed Computing

1. Personal Computing Environment:-

Personal means, all the computer stuff will be tied together i.e computer is completely ours, no other Connections.

2. Time sharing Environment:-

In computing, time-sharing is the sharing of a computing resource among many users by means of multi programming and multi-tasking at the same time. Mam users are connected to one or more computers.

3. **Client Server Computing Environment**:-

A client/server system is "a networked computing model that distributes processes between clients and servers, which supply the requested service." A client/server network connects many computers, called clients, to a main computer, called a server. Whenever client requests for something, server receives the request and process it.

4. **Distributed Computing**:-

A Distributed Computing Environment Provides a seamless integration of computing functions between different servers and clients. the servers are connected by internet all over the world.

➢ Computer Languages:-

To write a program for a computer, we must use a computer language. Over the years computer languages have evolved from machine languages to natural languages.

1940's Machine level Languages
1950's Symbolic Languages
1960's High-Level Languages

❖ Machine Languages:-

In the earliest days of computers, the only programming languages available were machine languages. Each computer has its own machine language, which is made of streams of 0's and 1's.

Instructions in machine language must be in streams of 0's and 1's because the internal circuits of a computer are made of switches transistors and other electronic devices that can be in one of two states: off or on. The off state is represented by 0 , the on state is represented by 1.

The only language understood by computer hardware is machine language.

❖ Symbolic Languages:-

In early 1950's Admiral Grace Hopper, A mathematician and naval officer developed the concept of a special computer program that would convert programs into machine language.

Computer does not understand symbolic language it must be translated to the machine language. A special program called assembler translates symbolic code into machine language.

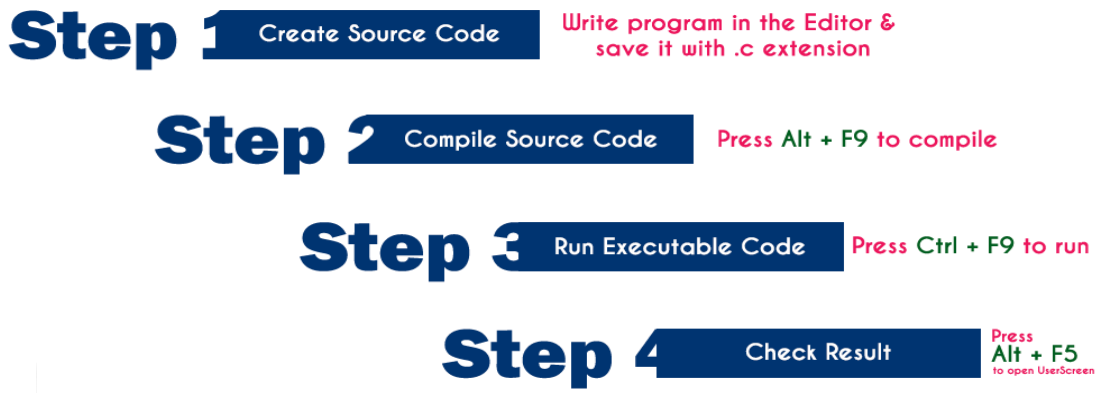
❖ **High Level Languages:-**

Symbolic languages greatly improved programming effificiency; they still required programmers to concentrate on the hardware that they were using.

Working with symbolic languages was also very tedious because each machine instruction has to be individually coded. The desire to improve programmer efficiency and to change the focus from the computer to the problem being solved led to the development of high-level language.

➢ Creating and running Programs:-

- Generally, the programs created using programming languages like C, C++, Java, etc., are written using a high-level language like English. But, the computer cannot understand the high-level language.
- It can understand only low-level language. So, the program written in high-level language needs to be converted into the low-level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.
- Popular programming languages like C, C++, Java, etc., use the compiler to convert high-level language instructions into low-level language instructions.
- To create and execute C programs in Windows Operating System, we need to install Turbo C software. We use the following steps to create and execute C programs in Windows **OS...**



Step 1: Creating Source Code

Source code is a file with C programming instructions in high level language. To create source code, we use any text editor to write the program instructions. The instructions written in the source code must follow the C programming language rules. The following steps **are used to create source code file in Windows OS...**

- Click on **Start** button
- Select **Run**
- Type **cmd** and press Enter
- Type **cd c:\TC\bin** in the command prompt and press **Enter**
- Type **TC** press **Enter**
- Click on **File -> New** in C Editorwindow
- Type the **program**
- Save it as **FileName.c** (Use shortcut key **F2** to save)

Step 2: Compile Source Code (Alt + F9)

Compilation is the process of converting high level language instructions into low level language instructions. We use the shortcut key **Alt + F9** to compile a C program in **Turbo C**.

Whenever we press **Alt + F9**, the source file is going to be submitted to the Compiler. On receiving a source file, the compiler first checks for the Errors. If there are any Errors then compiler returns List of Errors, if there are no errors then the source code is converted into **object code** and stores it as file with **.obj** extension. Then the object code is given to the **Linker**. The Linker combines

both the **object code** and specified **header file** code and generates an **Executable file** with **.exe** extension.

Step 3: Executing / Running Executable File (Ctrl + F9)

After completing compilation successfully, an executable file is created with **.exe** extension. The processor can understand this **.exe** file content so that it can perform the task specified in the source file.

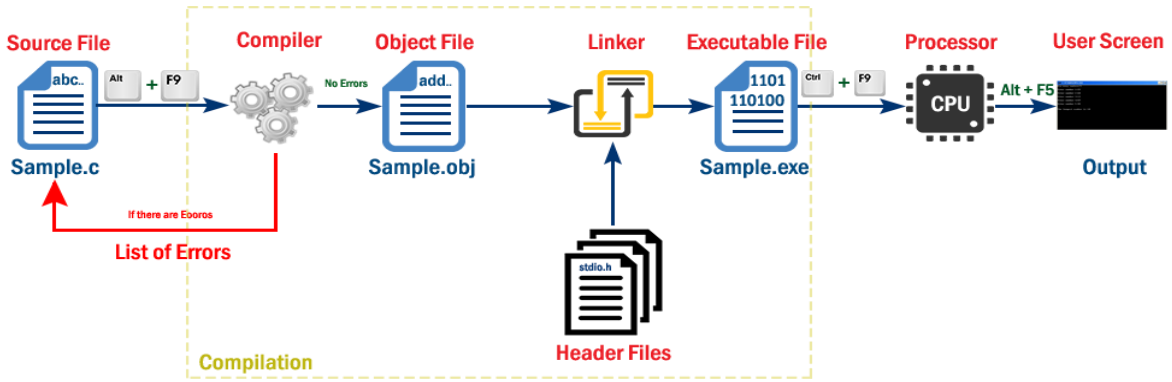
We use a shortcut key **Ctrl + F9** to run a C program. Whenever we press **Ctrl + F9**, the **.exe** file is submitted to the **CPU**. On receiving **.exe** file, **CPU** performs the task according to the instruction written in the file. The result generated from the execution is placed in a window called **User Screen**.

Step 4: Check Result (Alt + F5)

After running the program, the result is placed into **User Screen**. Just we need to open the User Screen to check the result of the program execution. We use the shortcut key **Alt + F5** to open the User Screen and check the result.

Execution Process of a C Program

When we execute a C program it undergoes with following process...



- The file which contains c program instructions in high level language is said to be source code. Every c program source file is saved with .c extension, for example Sample.c.

- Whenever we press Alt + F9 the source file is submitted to the compiler. Compiler checks for the errors, if there are any errors, it returns list of errors, otherwise generates object code in a file with name Sample.obj and submit it to the linker.
- Linker combines the code from specified header file into object file and generates executable file as Sample.exe. With this compilation process completes.
- Now, we need to Run the executable file (Sample.exe). To run a program we press Ctrl + F9. When we press Ctrl + F9 the executable file is submitted to the CPU.
- Then CPU performs the task according to the instructions written in that program and place the result into UserScreen.
- Then we press Alt + F5 to open UserScreen and check the result of the program.

Overall Process:-

- Type the program in C editor and save with **.c extension** (Press **F2** to save).
- Press **Alt + F9** to compile the program.
- If there are errors, correct the errors and recompile the program.
- If there are no errors, then press **Ctrl + F9** to execute / run the program.
- Press **Alt + F5** to open User Screen and check the result.

➢ Computer Numbering System:-

The technique to represent and work with numbers is called **number system**. **Decimal number system** is the most common number system. Other popular number systems include **binary number system, octal number system, hexadecimal number system,** etc.

Decimal Number System:-

Decimal number system is a **base 10** number system having 10 digits from 0 to 9. This means that any numerical quantity can be represented using these 10 digits. Decimal number system is also a **positional value system**.

Binary Number System:-

The easiest way to vary instructions through electric signals is two-state system – on and off. On is represented as 1 and off as 0, though 0 is not actually no signal but signal at a lower voltage. The number system having just these two digits – 0 and 1 – is called **binary number system**.

Octal Number System:-

Octal number system has eight digits – 0, 1, 2, 3, 4, 5, 6 and 7. Octal number system is also a positional value system with where each digit has its value expressed in powers of 8, as shown here −

8^5	8^4	8^3	8^2	8^1	8^0

Decimal equivalent of any octal number is sum of product of each digit with its positional value.

$726_8 = 7\times8^2 + 2\times8^1 + 6\times8^0$

$= 448 + 16 + 6$

$= 470_{10}$

Hexadecimal Number System:-

Octal number system has 16 symbols – 0 to 9 and A to F where A is equal to 10, B is equal to 11 and so on till F. Hexadecimal number system is also a positional value system with where each digit has its value expressed in powers of 16, as shown here −

16^5	16^4	16^3	16^2	16^1	16^0

➢ Storing Integers:-

Integers are commonly stored using a word of memory, which is 4 bytes or 32 bits, so integers from 0 up to 4,294,967,295 ($2^{32} - 1$) can be stored.

Below are the integers 1 to 5 stored as four-byte values (each row represents one integer).

```
0  :  00000001 00000000 00000000 00000000  |  1
 4  :  00000010 00000000 00000000 00000000  |  2
 8  :  00000011 00000000 00000000 00000000  |  3
12  :  00000100 00000000 00000000 00000000  |  4
16  :  00000101 00000000 00000000 00000000  |  5
```

This may look a little strange; within each byte (each block of eight bits), the bits are written from right to left like we are used to in normal decimal notation, but the bytes themselves are written left to right! It turns out that the computer does not mind which order the bytes are used (as long as we tell the computer what the order is) and most software uses this left to right order for bytes.[7.3]

Two problems should immediately be apparent: this does not allow for negative values, and very large integers, 2^{32} or greater, cannot be stored in a word of memory.

➢ Real numbers:-

Real numbers (and rationals) are much harder to store digitally than integers.

Recall that k bits can represent 2^k different states. For integers, the first state can represent 0, the second state can represent 1, the third state can represent 2, and so on. We can only go as high as the integer 2^k - 1, but at least we know that we can account for all of the integers up to that point.

Unfortunately, we cannot do the same thing for reals. We could say that the first state represents 0, but what does the second state represent? 0.1? 0.01? 0.00000001? Suppose we chose 0.01, so the first state represents 0, the second state represents 0.01, the third state represents 0.02, and so on. We can now only go as high as 0.01 x (2^k - 1), *and* we have missed all of the numbers between 0.01 and 0.02 (and all of the numbers between 0.02 and 0.03, and infinitely many others).

➢ INTRODUCTION TO 'C' LANGUAGE:-

- C language facilitates a very efficient approach to the development and implementation of computer programs. The History of C started in 1972 at the Bell Laboratories, USA where Dennis M. Ritchie proposed this language. In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce "an unambiguous

and machine independent definition of the language C " while still retaining it's spirit .

- C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C. Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on.
- C was first designed by Dennis Ritchie for use with UNIX on DEC PDP-11 computers. The language evolved from Martin Richard's BCPL, and one of its earlier forms was the B language, which was written by Ken Thompson for the DEC PDP-7. The first book on C was *The C Programming Language* by Brian Kernighan and Dennis Ritchie, published in 1978.
- In 1983, the American National Standards Institute (ANSI) established a committee to standardize the definition of C. The resulting standard is known as *ANSI C,* and it is the recognized standard for the language, grammar, and a core set of libraries. The syntax is slightly different from the original C language, which is frequently called K&R for Kernighan and Ritchie. There is also an ISO (International Standards Organization) standard that is very similar to the ANSI standard.
- It appears that there will be yet another ANSI C standard officially dated 1999 or in the early 2000 years; it is currently known as "C9X."

➢ BASIC STRUCTURE OF C LANGUAGE:-

- The program written in C language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

1. Documentation section
2. Linking section
3. Definition section
4. Global declaration section
5. Main function section

```
{
Declaration section
Executable section
}
```

6.Sub program or function section

1. DOCUMENTATION SECTION : comes first and is used to document the use of logic or reasons in your program. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with /* and */ . Whatever is written between these two are called comments.

2. LINKING SECTION : This section tells the compiler to link the certain occurrences of keywords or functions in your program to the header files specified in this section.
e.g. #include <stdio.h>

3. DEFINITION SECTION : It is used to declare some constants and assign them some value.
e.g. #define MAX 25
Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

4. GLOBAL DECLARATION SECTION : Here the variables which are used through out the program (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program)
e.g. int i; (before main())

5. MAIN FUNCTION SECTION : It tells the compiler where to start the execution from

```
main()
{
   point from execution starts
}
```

main function has two sections

1. declaration section : In this the variables and their data types are declared.
2. Executable section : This has the part of program which actually performs the task we need.

6. SUB PROGRAM OR FUNCTION SECTION : This has all the sub programs or the functions which our program needs.

SIMPLE 'C' PROGRAM:

```
/* simple program in c */
#include<stdio.h>
main()
{
printf("welcome to c programming");
} /* End of main */
```

IDENTIFIERS :-

- Names of the variables and other program elements such as functions, array, etc, are known as identifiers.

There are few rules that govern the way variable are named (identifiers).

1. Identifiers can be named from the combination of A-Z, a-z, 0-9, _(Underscore).
2. The first alphabet of the identifier should be either an alphabet or an underscore. digit are not allowed.
3. It should not be a keyword.
Eg: name, ptr, sum
After naming a variable we need to declare it to compiler of what data type it is .
The format of declaring a variable is

Data-type id1, id2, ...idn;

where data type could be float, int, char or any of the data types.
id1, id2, id3 are the names of variable we use. In case of single variable no commas are required.

```
Eg      float a, b, c;
        int e, f, grand total;
        char present_or_absent;
```

DATA TYPES :-

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. C supports four different classes of data types namely

1. Basic Data types
2. Derives data types
3. User defined data types
4. Pointer data types

BASIC DATA TYPES:

All arithmetic operations such as Addition , subtraction etc are possible on basic data types.

```
E.g.: int a,b;
      Char c;
```

DERIVED DATA TYPES:-

Derived datatypes are used in 'C' to store a set of data values. Arrays and Structures are examples for derived data types.

```
Ex:   int a[10];
      Char name[20];
```

USER DEFINED DATATYPES:

C Provides a facility called typedef for creating new data type names defined by the user. For Example ,the declaration ,

typedef int Integer;

makes the name Integer a synonym of int.Now the type Integer can be used in declarations ,casts,etc,like,

Integer num1,num2;

Which will be treated by the C compiler as the declaration of num1,num2as int variables.

"typedef" ia more useful with structures and pointers.

POINTER DATA TYPES:-

Pointer data type is necessary to store the address of a variable.

➢ VARIABLES :-

A quantity that can vary during the execution of a program is known as a variable. To identify a quantity we name the variable for example if we are calculating a sum of two numbers we will name the variable that will hold the value of sum of two numbers as 'sum'.

➢ CONSTANTS : -

A quantity that does not vary during the execution of a program is known as a constant supports two types of constants namely Numeric constants and character constants.

NUMERIC CONSTANTS:

1. Example for an integer constant is 786,-127
2. Long constant is written with a terminal 'l'or 'L',for example 1234567899L is a Long constant.

CHARACTER CONSTANTS:-

A character constant is written as one character with in single quotes such as 'a'. The value of a character constant is the numerical value of the character in the machines character set.

➢ INPUT AND OUTPUT STATEMENTS :-

The simplest of input operator is getchar to read a single character from the input device.

varname=getchar();

you need to declare varname.

The simplest of output operator is putchar to output a single character on the output device.

putchar(varname)

The getchar() is used only for one input and is not formatted. Formatted input refers to an input data that has been arranged in a particular format, for that we have scanf.

scanf("control string", arg1, arg2,...argn);

Control string specifies field format in which data is to be entered.

arg1, arg2... argn specifies address of location or variable where data

Eg scanf("%d%d",&a,&b);

%d used for integers
%f floats
%l long
%c character

for formatted output you use printf

printf("control string", arg1, arg2,...argn);

/* program to exhibit i/o */

```
#include<stdio.h>
main()
{
int a,b;
float c;
printf("Enter any number");
a=getchar();
printf("the char is ");
putchar(a);
printf("Exhibiting the use of scanf");
printf("Enter three numbers");
scanf("%d%d%f",&a,&b,&c);
printf("%d%d%f",a,b,c);
}
```

- **Scope:-**

  A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language –

  - Inside a function or a block which is called **local** variables.

- Outside of all functions which is called **global** variables.
- In the definition of function parameters which are called **formal** parameters.

Local Variables:-

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

Global Variables:-

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function.

➢ Storage Classes:-

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program −

- auto
- register
- static
- extern

The auto Storage Class

The **auto** storage class is the default storage class for all local variables.

```
{
   int  mount;
   auto int month;
}
```

The example above defines two variables with in the same storage class. 'auto' can only be used within functions, i.e., local variables.

The register Storage Class:-

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```
{
   register int  miles;
}
```

The register should only be used for variables that require quick access such as counters.

The static Storage Class:-

The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope..

The extern Storage Class

The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

Type Qualifiers:-

The keywords which are used to modify the properties of a variable are called type qualifiers.

TYPES OF C TYPE QUALIFIERS:
There are two types of qualifiers available in C language. They are,

1. const
2. volatile

1. CONST KEYWORD:

- Constants are also like normal variables. But, only difference is, their values can’t be modified by the program once they are defined.
- They refer to fixed values. They are also called as literals.
- They may be belonging to any of the data type.

Syntax:
const data_type variable_name; (or) const data_type *variable_name;

VOLATILE KEYWORD:

- When a variable is defined as volatile, the program may not change the value of the variable explicitly.
- But, these variable values might keep on changing without any explicit assignment by the program. These types of qualifiers are called volatile.

Syntax:
volatile data_type variable_name; (or) volatile data_type *variable_name;

Tips and Common Programming Errors:-

When you start writing your code in C, C++ or any other programming language, your first objective might be to write a program that works.

After you accomplished that, the following are few things you should consider to enhance your program.

1. Security of the program
2. Memory consumption
3. Speed of the program (Performance Improvement)

This article will give some high-level ideas on how to improve the speed of your program.

Few general points to keep in mind:-

- You could optimize your code for performance using all possible techniques, but this might generate a bigger file with bigger memory footprint.
- You might have two different optimization goals, that might sometimes conflict with each other. For example, to optimize the code for performance might conflict with optimize the code for less memory footprint and size. You might have to find a balance.
- Performance optimization is a never-ending process. Your code might never be fully optimized. There is always more room for improvement to make your code run faster.
- Sometime we can use certain programming tricks to make a code run faster at the expense of not following best practices such as coding standards, etc. Try to avoid implementing cheap tricks to make your code run faster.

➢ Expressions:-

An expression is a combination of variables constants and operators written according to the syntax of C language.

In C every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable.

Evaluation of Expressions:-

Expressions are evaluated using an assignment statement of the form.

```
Variable = expression;
```

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side.

. All variables used in the expression must be assigned values before evaluation is attempted.

Example of evaluation statements are

X=a*b-c
Y=b/c*a
Z=a-b/c+d;

➢ Precedence and Associativity:-

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

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a 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

Side Effects of c:-

In C and more generally in computer science, a function or expression is said to have a **side effect** if it modifies a state outside its scope or has an *observable* interaction with its calling functions or the outside world. By convention, returning a value has an effect on the calling function, but this is usually not considered as a side effect.

Some side effects are:

- Modification of a global variable or static variable
- Modification of function arguments
- Writing data to a display or file
- Reading data
- Calling other side-effecting functions

Type Conversion Statements:-

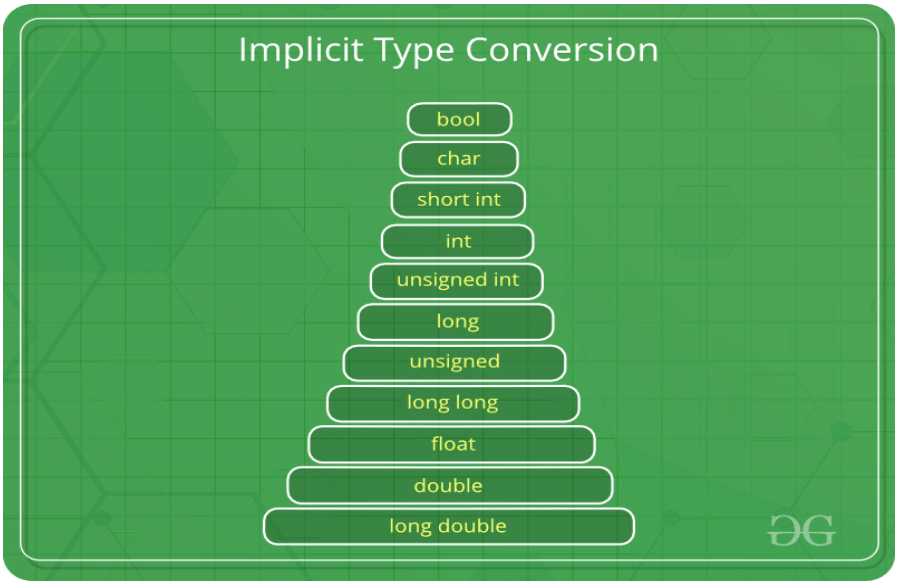
Typecasting is converting one data type into another one. It is also called as data conversion or type conversion. It is one of the important concepts introduced in 'C' programming.

'C' programming provides two types of type casting operations:

1. Implicit type casting
2. Explicit type casting

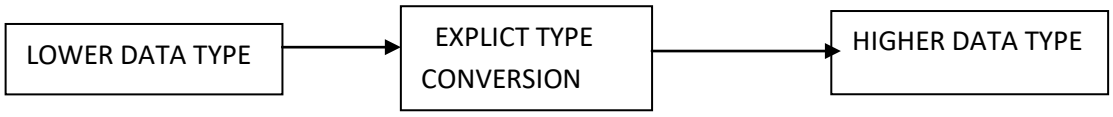
Implicit type casting

Implicit type casting means conversion of data types without losing its original meaning. This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.



Explicit type casting

In implicit type conversion, the data type is converted automatically. There are some scenarios in which we may have to force type conversion. Suppose we have a variable div that stores the division of two operands which are declared as an int data type.



SIMPLE PROGRAM:-

1.Prime Number program in C

```
1. #include<stdio.h>
2. int main(){
3. int n,i,m=0,flag=0;
4. printf("Enter the number to check prime:");
5. scanf("%d",&n);
6. m=n/2;
7. for(i=2;i<=m;i++)
8. {
9. if(n%i==0)
10.{
11.printf("Number is not prime");
12.flag=1;
13.break;
14.}
15.}
16.if(flag==0)
17.printf("Number is prime");
18.return 0;
19. }
```

Output:

```
Enter the number to check prime:56
Number is not prime
Enter the number to check prime:23
Number is prime
```

- **Command Line Arguments:-**

It is possible to pass some values from the command line to your C programs when they are executed.

These values are called command line arguments and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

The command line arguments are handled using main() function arguments where argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument passed to the program.