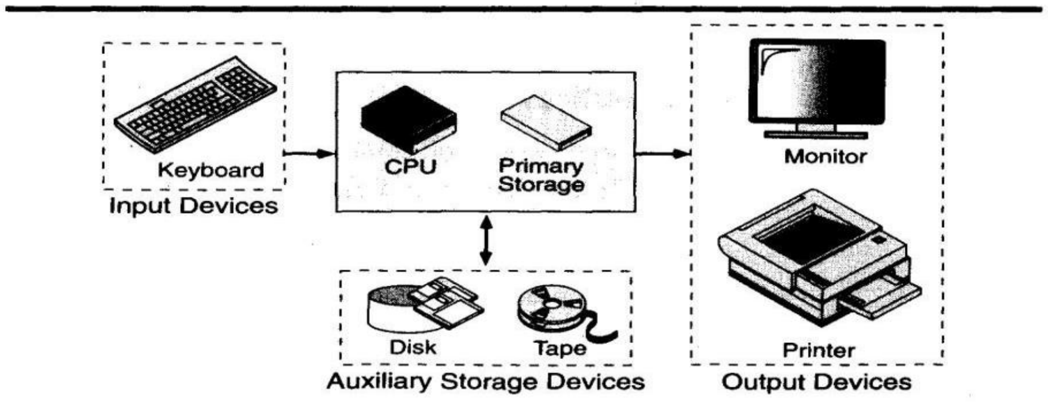
# PROGRAMMING FOR PROBLEM SOLVING USING C
## UNIT I

✓ **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 amouse, 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 amongdata, and movement of data inside the system.
- The **output device** is usually a monitor or a printer to show output. If theoutput 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 storedpermanently. 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: systemsoftware and application software.

- System software manages the computer resources .It provides theinterface between the hardware and the users.
- Application software, on the other hand is directly responsible forhelping 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 havebeen 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 computeris 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 thesame 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 amain 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 integrationof 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 machinelanguages 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 languagesavailable 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 theinternal 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 wouldconvert programs into machine language.

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

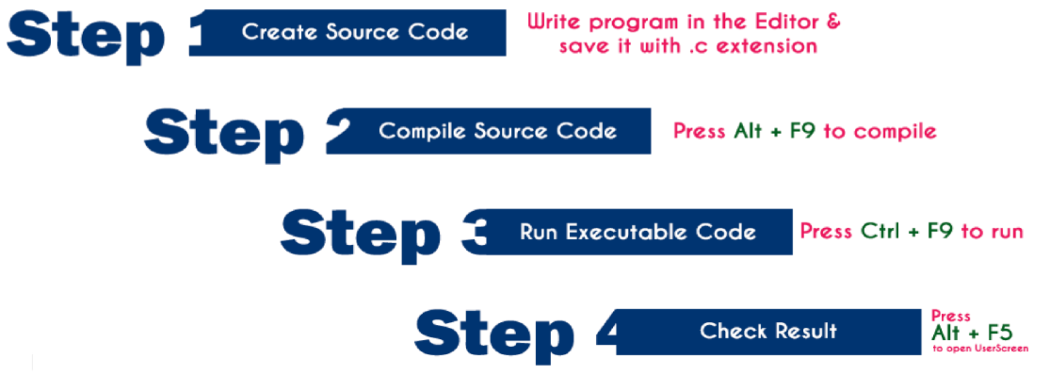
- **High Level Languages:-**

Symbolic languages greatly improved programming effificiency; theystill 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 usingprogramming languages like C, C++, Java, etc., are written using a high-levellanguage likeEnglish. 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., usethe compiler to convert high-level language instructions into low- level languageinstructions.
- 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 WindowsOS…


Step 1 Create Source Code
Write program in the Editor & save it with .c extension
Step 2 Compile Source Code
Press Alt + F9 to compile
Step 3 Run Executable Code
Press Ctrl + F9 to run
Step 4 Check Result
Press Alt + F5 to open UserScreen

### 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 compilerfirst checks for the Errors. If there are any Errors then compiler returns List of Errors, if there are no errors then the source code isconverted 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 generatesan **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.

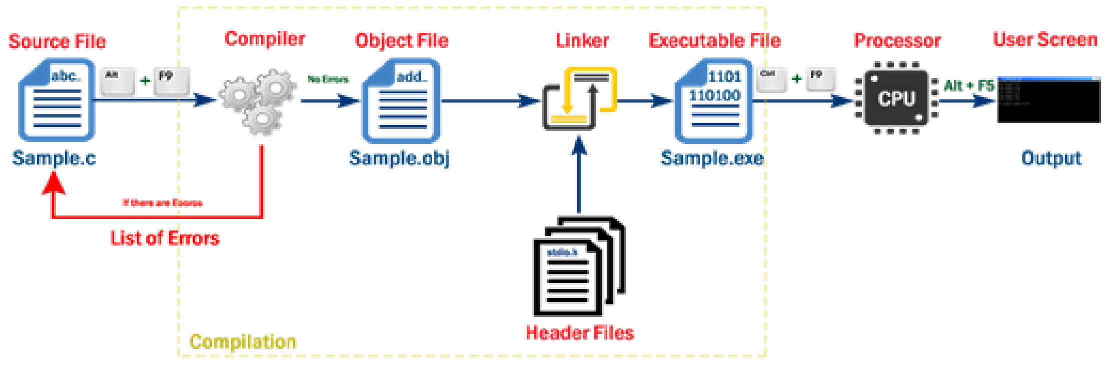
We use a shortcut key **Ctrl + F9** to run a C program. Whenever we press **Ctrl + F9**, the **.exe** file is submitted to the **CPU**. On receiving **.exe** file, **CPU** performs the task according to the instruction written in the file. The result generated from the execution is placed in a window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

## 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 issaved 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, itreturns list of errors, otherwise generates object code in a file withname 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 aprogram 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 writtenin that program and place the result into UserScreen.
- Then we press Alt + F5 to open UserScreen and check the result ofthe program.

**Overall Process:-**

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

✓ **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 1972at 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 stillretaining 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 thefirst 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-11computers. The language evolved from Martin Richard's BCPL, and oneof 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 andRitchie. 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 dated1999 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 sectionsis 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 keywordsor 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 frommain()

```
{
    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 thefunctions 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 anunderscore. 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 itis .
The format of declaring a variable isData-
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 nocommas 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 thetype 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 onbasic 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 andStructures 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 namesdefined by the user. For Example ,the declaration ,

**typedef int Integer;**

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

**Integer num1,num2;**

Which will be treated by the C compiler as the declaration ofnum1,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 arecalculating 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 Numericconstants 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 1234567899Lis a Long constant.

**CHARACTER CONSTANTS:-**

A character constant is written as one character with in single quotessuch 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 fromthe input device.

**varname=getchar();**

you need to declare varname.

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

**putchar(varname)**

The getchar() is used only for one input and is not formatted. Formattedinput refers to an input data that has been arranged in a particular format, forthat 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 bedeclared 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 within the same storageclass. '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 nothave a memory location).

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

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

**The static Storage Class:-**

The **static** storage class instructs the compiler to keep a local variablein 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 variablethat 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 programminglanguage, your first objective might be to write a program that works.

After you accomplished that, the following are few things you shouldconsider 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 speedof your program.

✓ **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 expressionresults 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 isattempted.

**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 precedencethan 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 * hasa 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 issaid 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, butthis 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**:-

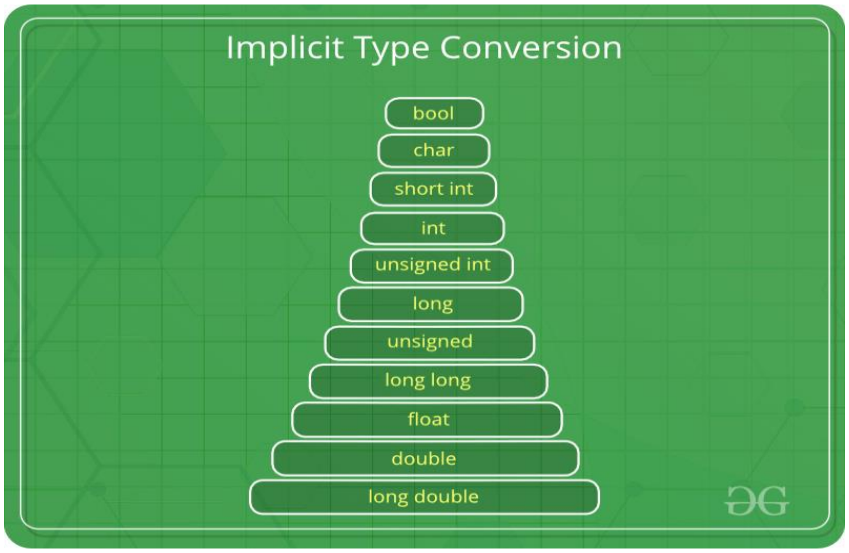
Typecasting is converting one data type into another one. It is also called as data conversion or type conversion. It is one of the important concepts introduced in 'C' programming.

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

1. Implicit type casting
2. Explicit type casting

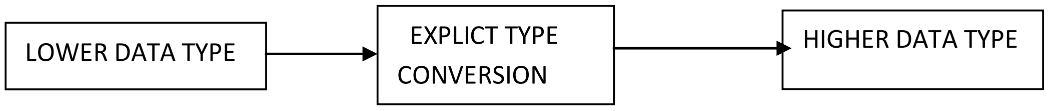
**Implicit type casting**

Implicit type casting means conversion of data types without losing its original meaning. This type of typecasting is essential when you wantto change data types **without** changing the significance of the valuesstored 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 oftwo operands which are declared as an int data type.



**SIMPLE PROGRAM:-**

1.Prime Number program in C.

```
#include<stdio.h>
int main(){
int n,i,m=0,flag=0;
printf("Enter the number to check prime:");
scanf("%d",&n
);
m=n/2;
for(i=2;i<=m;i++)
{
if(n%i==0)
{
printf("Number is not prime");
flag=1;
brea
k;
}
}
if(flag==0)
printf("Number s prime");
return 0;
}
```

**Output:**

```
Enter the number to check prime:56Number 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 yourC programs when they are executed.

  These values are called command line arguments and many timesthey 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 tothe program.

# UNIT II
# Control Flow, Relational Expressions & Arrays

**Concept of Loop:-**

- In looping, a program executes the sequence of statements many times until the stated condition becomes false. A loop consists of two parts, a body of a loop and a control statement.
- The control statement is a combination of some conditions thatdirect the body of the loop to execute until the specified condition

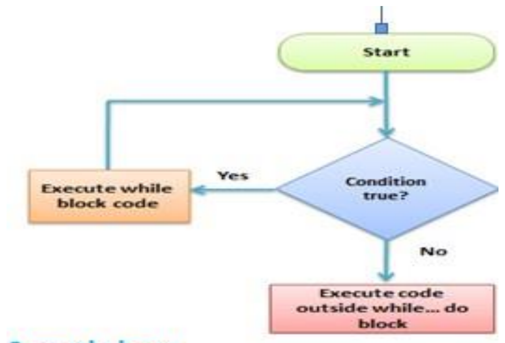
**Types of Loops**

Depending upon the position of a control statement in a program, a loopis classified into two types:

1. Entry controlled loop
2. Exit controlled loop

In an **entry controlled loop,** a condition is checked before executing thebody of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing thebody of a loop. It is also called as a post-checking loop



Sample loop

The control conditions must be well defined and specified otherwise theloop will execute an infinite number of times.

'C' programming language provides us with three types of loopconstructs:

1. The while loop
2. The do-while loop
3. The for loop

**While Loop**

A while loop is the most straightforward looping structure. The basicformat of while loop is as follows:

```
While (condition )
{
Statements;
}
```

It is an entry-controlled loop. In while loop, a condition is evaluatedbefore processing a body of the loop.

Following program illustrates a while loop:

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int num=1;	//initializing the variable
	 while(num<=10)	//while loop with condition
	{
		printf("%d\n",num);
		num++;		//incrementing operation
	}
	return 0;
}
```

**Output:**

```
1
2
3
4
5
6
7
8
9
10
```

**Do-While loop**

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format/ Syntax of while loop is as follows:

```
do {

 statements

} while (expression);
```

As we saw in a while loop, the body is executed if and only if the condition is true. In some cases, we have to execute a body of the loopat least once even if the condition is false. This type of operation can beachieved by using a do-while loop

In the do-while loop, the body of a loop is always executed at least once.After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

The following program illustrates the working of a do-while loop:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int num=1;      //initializing the
        variabledo      //do-while loop
        {
                        printf("%d\n",2*num);
                        num++;          //incrementing operation
                }while(num<=10);
                return 0;
}
```

Output:

```
2
4
6
8
10
12
14
16
18
20
```

**For loop**

A for loop is a more efficient loop structure in 'C' programming. Thegeneral structure of for loop is as follows:

```
for (initial value; condition; incrementation or decrementation )
{
 statements;
}
```

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares thecounter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) thecounter by a set value.

Following program illustrates the use of a simple for loop:

```
#include<stdio.h>
int main()
{
        int number;
        for(number=1;number<=10;number++) //for loop to print 1-10 numbers
        {
                printf("%d\n",number);          //to print the number
        }
        return 0;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

**Initialization:-**

- Initialization is the process of locating and using the defined valuesfor variable data that is used by a computer program. For example,an operating system or application program is installed with defaultor user-specified values that determine certain aspects of how the system or program is to function.
- The process of the user specifying initialization values issometimes called *configuration*.

**Event and Counter Controlled Loops:-**

**The *Event-Controlled* while loop**

- In this **while** loop an action is repeated until a certain event occurs. This isby far the most frequently used form of the **while** loop.

- There are three different types of **Event-Controlled** while loops

1. **Sentinel-controlled loops -** A *sentinel* variable is initialized to a specific value. The **while** loop continues until, through some action inside the loop,the *sentinel* variable is set to a predefined *termination* value.

2. **End-of-file controlled loops -** This type of **while** loop is usually used whenreading from a file. The loop terminates when the end of the file is detected. This can easily be determined by checking the **EOF()** function after each read from the file. This function will return true when the end-of-file is reached.

3. **Flag controlled loops -** A **bool** variable is defined and initialized to serve asa *flag*. The **while** loop continues until the *flag* variable value flips (true becomes false or false becomes true).

**<u>Count-Controlled Repetition:-</u>**

Count-controlled repetition requires

- control variable (or loop counter)
- initial value of the control variable
- increment (or decrement) by which the control variable is modified eachiteration through the loop
- condition that tests for the final value of the control variable

- A count-controlled repetition will exit after running a certain number oftimes. The count is kept in a variable called an index or counter.
- When the index reaches a certain value (the loop bound) the loop will end.
- Count-controlled repetition is often called definite repetition because thenumber of repetitions is known before the loop begins executing.

**Other Statements Related to Looping:-**

- **break and continue:-**

  - ***break*** and ***continue*** are two C/C++ statements that allow us to furthercontrol flow within and out of loops.
  - ***break*** causes execution to immediately jump out of the current loop, andproceed with the code following the loop.
  - ***continue*** causes the remainder of the current iteration of the loop to beskipped, and for execution to recommence with the next iteration.
  - In the case of ***for*** loops, the incrementation step will be executed next,followed by the condition test to start the next loop iteration.
  - In the case of ***while*** and ***do-while*** loops, execution jumps to the nextloop condition test.

- **Infinite Loops:-**

  - Infinite loops are loops that repeat forever without stopping.
  - Usually they are caused by some sort of error, such as the followingexample in which the wrong variable is incremented:

```
int i, j;
for( i = 0; i < 5; j++ ) printf( "i = %d\n", i );
```

```
printf( "This line will never execute\n" );
```

❖ **Nested Loops**

The code inside a loop can be any valid C code, including other loops.Any kind of loop can be nested inside of any other kind of loop.

**Looping Applications:-**

***Why do we use looping in programming?***

Because we want to repeat something:

- Count from 1 to 10.
- Go through all the words in a dictionary to see whether they'repalindromes.
- For each customer that has an outstanding balance, send out anemail reminder that payment is due.
- For each directory under this one, find music files and add thento the list of known music.

**The Calculator Program**:-

***/*C program to design calculator with basic operations using switch.*/***

```
#include <stdio.h>int main()
{
   int num1,num2;float
   result;
   char ch;    //to store operator choice
    printf("Enter first number: ");
   scanf("%d",&num1); printf("Enter second
   number: ");scanf("%d",&num2);
     printf("Choose operation to perform (+,-,*,/,%): ");scanf("
   %c",&ch);
     result=0;
 switch(ch)
   {
      case '+': result=num1+num2;
         break;

      case '-':
         result=num1-num2;break;

      case '*': result=num1*num2;
         break;

      case '/': result=(float)num1/(float)num2;
         break;


      case '%': result=num1%num2;
         break;
      default:
         printf("Invalid operation.\n");
```

```
        }

        printf("Result: %d %c %d = %f\n",num1,ch,num2,result);return 0;
    }
```

**Output**

```
Enter first number: 10 Enter
second number: 20
Choose operation to perform (+,-,*,/,%): +Result: 10
+ 20 = 30.000000

Enter first number: 10 Enter
second number: 3
Choose operation to perform (+,-,*,/,%): /Result: 10
/ 3 = 3.333333

Enter first number: 10
Enter second number: 3
Choose operation to perform (+,-,*,/,%): >Invalid
operation.
Result: 10 > 3 = 0.000000
```

## Arrays:-

An array is a group of related data items that share a common name.Ex:-Students

- The complete set of students are represented using an array namestudents.
- A particular value is indicated by writing a number called index numberor subscript in brackets after array name.
- The complete set of value is referred to as an array, the individualvalues are called elements.

➢ **Using Array in C:-**

- to store list of Employee or Student names,
- to store marks of students,
- or to store list of numbers or characters etc.

Since arrays provide an easy way to represent data, it is classified amongst the data structures in C. Other data structures in c are structure, lists, queues, trees etc. Array can be used to represent not only simple list of data but also table of data in two orthree dimensions.

- **Array Application:-**
  In c programming language, arrays are used in wide range of applications. Few of them are as follows.
- **Arrays are used to Store List of values**
  In c programming language, single dimensional arrays are used to store list of values of same datatype. In other words, single dimensional arrays are used to store a row of values. In single dimensional array data is stored in linear form
- **Arrays are used to Perform Matrix Operations**
  We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.

- **Arrays are used to Perform Matrix Operations**

  We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.

- **Arrays are used to implement Search Algorithms**

  We use single dimensional arrays to implement search algorihtms like ...

1. **Linear search**
2. **Binary search**

- **Arrays are used to implement Sorting Algorithms**

  We use single dimensional arrays to implement sorting algorithms like ...
  - ✓ Insertion sort
  - ✓ Bubble sort
  - ✓ Selection sort
  - ✓ Quick sort
  - ✓ Merge sort

- **ONE – DIMENSIONAL ARRAYS :-**

  A list of items can be given one variable index is called single subscriptedvariable or a one-dimensional array. The subscript value starts from 0. If we want 5 elements the declarationwill be

  int number[5];

The elements will be number[0], number[1], number[2], number[3], number[4]There will not be number[5]

- **Declaration of One - Dimensional Arrays :**

Type variable – name [sizes];

Type – data type of all elements Ex: int, float etc.,Variable –
name – is an identifier
Size – is the maximum no of elements that can be stored

Ex:- float avg[50]

This array is of type float. Its name is avg. and it can contains 50 elements only.The range starting from 0 – 49 elements.

- **Initialization of Arrays :-**

  Initialization of elements of arrays can be done in same way as ordinaryvariables are done when they are declared.

  Type array name[size] = {List of Value};

  Ex:- int number[3]={0,0,0};

  If the number of values in the list is less than number of elements then only that elements will be initialized. The remaining elements will be set tozero automatically.

- **TWO – DIMENSIONAL ARRAYS:-**

  To store tables we need two dimensional arrays. Each table consists of rowsand columns. Two dimensional arrays are declare as

  type array name [row-size][col-size];

  /* Write a program Showing 2-DIMENSIONAL ARRAY */
  /* SHOWING MULTIPLICATION TABLE */

```
#include<stdio.h>
#include<math.h>
main()
{
int row,cols,prod[3][3];
int i,j;
printf("Multiplication table");
for(j=1;j< =3;j++)
printf("%d",j);
for(i=0;i<3;i++)
{
row = i+1;
printf("%2d|",row);
for(j=1;j < = 3;j++)
{
cols=j;
prod[i][j]= row * cols;
printf("%4d",prod*i+*j+);
}
}
}
```

- **INITIALIZING TWO DIMENSIONAL ARRAYS:-**

  They can be initialized by following their declaration with a list of initialvalues enclosed in braces.

  Ex:- int table[2][3] = {0,0,0,1,1,1};

  Initializes the elements of first row to zero and second row to one. The initialization is done by row by row. The above statement can be written as

  int table[2][3] = {{0,0,0},{1,1,1}};

When all elements are to be initialized to zero, following short-cut methodmay be used.

int m[3][5] = {{0},{0},{0}};

- **Multidimensional Arrays:-**

  C allows for arrays of two or more dimensions. A two-dimensional(2D) array is an array of arrays. A three-dimensional (3D) array is an array of arrays of arrays.

  In C programming an array can have two, three, or even ten or more dimensions. The maximum dimensions a C program can havedepends on which compiler is being used.

- **Declare a Multidimensional Array in C:-**
  A multidimensional array is declared using the following syntax:

  type array_name[d1][d2][d3][d4]………[dn];

  Where each **d** is a dimension, and **dn** is the size of final dimension.

Examples:

1. int table[5][5][20];
2. float arr[5][6][5][6][5];

In Example 1:

✓**int** designates the array type integer.

✓**table** is the name of our 3D array.

✓Our array can hold 500 integer-type elements. This number isreached by multiplying the value of each dimension. In this case: **5x5x20=500**.

In Example 2:

✓Array **arr** is a five-dimensional array.

✓It can hold 4500 floating-point elements (**5x6x5x6x5=4500**).

- Initializing a 3D Array in C:-

  Like any other variable or array, a 3D array can be initialized at thetime of compilation. By default, in C, an uninitialized 3D array contains "garbage" values, not valid for the intended use.

Let's see a complete example on how to initialize a 3D array:-

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

void main()
{
int i, j, k;
int arr[3][3][3]=
      {
        {
        {11, 12, 13},
        {14, 15, 16},
        {17, 18, 19}
        },
        {
        {21, 22, 23},
        {24, 25, 26},
        {27, 28, 29}
        },
        {
```

```
        {31, 32, 33},
        {34, 35, 36},
        {37, 38, 39}
        },};
clrscr();
printf(":::3D Array Elements:::\n\n");
for(i=0;i<3;i++)
{
  for(j=0;j<3;j++)
  {
    for(k=0;k<3;k++)
    {
    printf("%d\t",arr[i][j][k]);
    }
    printf("\n");
  }
  printf("\n");
}
getch();
}
```

- **Programming Example – Calculate Averages:-**

This program takes n number of element from user (where, nis specified by user), stores data in an array and calculates the average of those numbers.

**Source Code to Calculate Average Using Arrays:-**

```
#include <stdio.h>

int main()
{
  int n, i;
  float num[100], sum = 0.0, average;
  printf("Enter the numbers of elements: ");
  scanf("%d", &n);

  while (n > 100 || n <= 0)
  {
    printf("Error! number should in range of (1 to
    100).\n");printf("Enter the number again: ");
    scanf("%d", &n);
  }

  for(i = 0; i < n; ++i)
  {
    printf("%d. Enter number: ",
    i+1);scanf("%f", &num[i]);
    sum += num[i];
  }
  average = sum / n; printf("Average = %.2f", average);
  return 0;
}
```

**Output**

```
Enter the numbers of elements: 6
1. Enter number: 45.3
2. Enter number: 67.5
3. Enter number: -45.6
4. Enter number: 20.34
5. Enter number: 33
6. Enter number: 45.6
Average = 27.69
```

# UNIT III
# Strings

**Strings:-**

A String is an array of characters. Any group of characters (except doublequote sign ) defined between double quotes is a constant string.

Ex: “C is a great programming language”.

If we want to include double quotes.

Ex: “\”C is great \” is norm of programmers “.

- **Declaring and initializing strings :-**

  A string variable is any valid C variable name and is always declared asan array.

  char string name [size];

  size determines number of characters in the string name. When thecompiler assigns a character string to a character array, it automaticallysupplies a null character (‘\0’) at end of String.

  Therefore, size should be equal to maximum number of character inString plus one.

  There are two ways to declare a string in c language.

  1. By char array
  2. By string literal

  1. char city*10+= ,‘N’,’E’,’W’,’ ‘,’Y’,’O’,’R’,’K’,’\0’-;

  2. char city*10+= “NEW YORK’;

C also permits us to initializing a String without specifying size.

Ex:- char Strings*+= ,‘G’,’O’,’O’,’D’,’\0’-;

- **String Input / Output Functions:-**
  - ✓ C provides two basic ways to read and write strings
  - ✓ First we can read and write strings with the formatted input/output functions,scanf/fscanf and prinf/fprinf.
  - ✓ Second we can use a special set of strin only functions ,get string(gets/fgets)and put string(puts/fputs).

**Formatted string Input/Output:**

- ✓ Formatted String Input:scanf/fscanf:
- ✓ **int fscanf(FILE ****stream***, const char ****format***, ...);**
- ✓ int scanf(const char **format*, ...);
- ✓ The ..scanf functions provide a means to input formatted informationfrom a stream.
- ✓ **fscanf** reads formatted input from a stream

- ✓ **scanf** reads formatted input stdin

  These functions take input in a manner that is specified by the format argument and store each input field into the following argumentsin a left to right fashion.

**String Input/Output**

In addition to the Formatted string functions,C has two sets of string functions that read and write strings without reformatting any data.These functions convert text file lines to strings and strings to textfile lines

**gets():**

Declaration:

char *gets(char **str*);

Reads a line from **stdin** and stores it into the string pointed toby *str*. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first. The newline character is not copied to the string. A null character is appended to the end of the string.

**puts():**

Declaration:

int puts(const char **str*);

Writes a string to **stdout** up to but not including the null character.
A newline character is appended to the output.
On success a nonnegative value is returned. On error **EOF** is returned.

- **STRING HANDLING/MANIPULATION FUNCTIONS:-**

| | |
|---|---|
| strcat( ) | Concatenates two Strings |
| strcmp( ) | Compares two Strings |
| strcpy( ) | Copies one String Over another |
| strlen( ) | Finds length of String |

- ✓ **strcat() function:**

This function adds two strings together.

Syntax: char *strcat(const char *string1, char *string2);

strcat(string1,string2); string1
= VERY
string2 = FOOLISH
strcat(string1,string2);
string1=VERY FOOLISH
string2 = FOOLISH

**Strncat**: Append n characters from string2 to stringl.
char *strncat(const char *string1, char *string2, size_t n);

**Example Program: CONCATENATE**

```
# include < stdio .h >
# include < string .h >
main ()
{
char str1 [20] = " Aditya ", str2 [20] = " college ";
strcat ( str1 , str2 );
puts ( str1 );
}
```

Output : Adityacollege

✓ **strcmp() function :**

This function compares two strings identified by arguments and has a value 0 if they are equal. If they are not, it has the numeric difference betweenthe first non-matching characters in the Strings.

Syntax: int strcmp (const char *string1,const char *string2);
strcmp(string1,string2);

Ex:- strcmp(name1,name2);
strcmp(name1,"John");
strcmp("ROM","Ram");

**Strncmp:** Compare first n characters of two strings.

int strncmp(const char *string1, char *string2, size_t n);

**Example Program: COMPARE**

```
# include < stdio .h >
# include < string .h >
main ()
{
char str1 [] = " abcd ", str2 [] = " abcd ";
int result ;
result = strcmp ( str1 , str2 );
if( result ==0)
printf ("Two strings are equals ");
else
printf ("Two strings are not equal ");
}
```

**Output:**

```
Two strings are equals
```

✓ **strcpy() function :**

It works almost as a string assignment operators. It takes the formSyntax: char

*strcpy(const char *string1,const char *string2);

strcpy(string1,string2);

string2 can be array or a constant.

**Strncpy:** Copy first n characters of string2 to stringl .

char *strncpy(const char *string1,const char *string2, size_t n);

**Example program : COPY**

```
# include < stdio .h >
# include < string .h >
main ()
{
char str1 [10]= " awesome ";
char str2 [10];
char str3 [10];
strcpy ( str2 , str1 );
strcpy ( str3 , " well ");
puts ( str2 );
puts ( str3 );
}
```

**Output:**

```
awesome
well
```

✓ **strlen() function :**

Counts and returns the number of characters in a string.

Syntax:int strlen(const char *string);

n= strlen(string);

n - integer variable which receives the value of length of string.

**Example program : LENGTH**

```
# include < stdio .h >
# include < string .h >
main ()
{
char a [20]= " Program ";
char b [20]={ ’P’,’r’,’o’,’g’,’r’,’a’,’m’,’\0 ’};
char c [20];
printf (" Enter string : ");
gets ( c );
printf (" Length of string a = %d \n", strlen ( a ));
printf (" Length of string b = %d \n", strlen ( b ));
printf (" Length of string c = %d \n", strlen ( c ));
}
```

**Output:**

```
Enter string : String
Length of string a = 7
Length of string b = 7
Length of string c = 6
```

**Write  C program to perform String manipulations without using library function .**
**a ) copy**
**b ) concatenate**
**c ) length**
**d ) compare**

**( a ) copy one string to other**

```
# include < stdio .h >
main ()
{
char s1 [100] , s2 [100];
int i=0;
printf ("\ nEnter the string :");
gets ( s1 );
while ( s1 [ i ] != '\0 ')
{
s2 [ i] = s1 [ i ];
i ++;
}
s2 [ i] = '\0 ';
printf ("\ nCopied String is %s ", s2 );
}
```

**Output :**

```
Enter the string : Aditya
Copied String is Aditya
```

**( b ) Concatenation of strings**

```
# include < stdio .h >
# include < string .h >
main () {
char s1 [50] , s2 [30];
int i , j , len ;
printf ("\ nEnter String 1 :");
gets ( s1 );
printf ("\ nEnter String 2 :");
gets ( s2 );
for ( i =0; str [ i ] != '\0 '; i ++)
len ++;
for ( j = 0; s2 [ j ] != '\0 '; len ++ , j ++) {
s1 [ len ] = s2 [ j ];
}
s1 [ len ] = '\0 ';
printf (" concated string is :%s", s1 );
}
```

**Output :**

```
Enter String 1 : Aditya
Enter String 2 : college
Concated string is : Aditya college
```

**( c ) Calculating Length of the strings**

```
# include < stdio .h >
main () {
char str [100];
int length , i ;
printf ("\ nEnter the String : ");
gets ( str );
length = 0;
for ( i =0; str [ i ] != ’\0 ’; i ++)
length ++;
printf ("\ nLength of the String is : %d", length );
}
```

**Output:**

```
Enter the string : aditya
Length of the string is :6
```

**( d ) compare two strings**

```
# include < stdio .h >
# include < string .h >
main () {
char s1 [50] , s2 [30];
int i , j , flag =0;
printf ("\ nEnter String 1 :");
gets ( s1 );
printf ("\ nEnter String 2 :");
gets ( s2 );
for ( i =0 , j =0; s1 [ i ]!= \0 && s2 [ j ]!= \0 ; i ++ , j ++)
{
if( s1 [ i ]!= s2 [ j ])
{
flag ++;
break ;
}
}
if( flag ==0)
printf ("\ nTwo strings are equals ");
else
printf ("\ nTwo strings are not equal ");
}
```

**Output :**

```
Enter String 1 : aditya
Enter String 2 : aditya
Two strings are equals
```