UNIT—1

Syllabus:

Introduction: Computer systems, Hardware and software concepts
Problem solving: Algorithm/pseudo code, flowchart,program development steps,
Computer languages: machine,symbolic,and high level languages,creating and running
Programs: writing,editing,compiling,linking and executing.
Basics of C: structure of a c program, identifiers, basic data types and sizes, constants,variables,arithmetic,relational,and logical operator,increment and decrement operators,conditional operator,assignment operators,expressions,type conversions,conditional expressions,precedence and order of evaluation,sample programs.

Introduction: Computer systems, hardware and software concepts:

Computer: It is an electronic machine which accept data as its input, process to it by doing some kind of manipulations and produce the output in a desired format. Techincally a computer is a high speed electronic data processing machine.

Difference between a computer and calculator

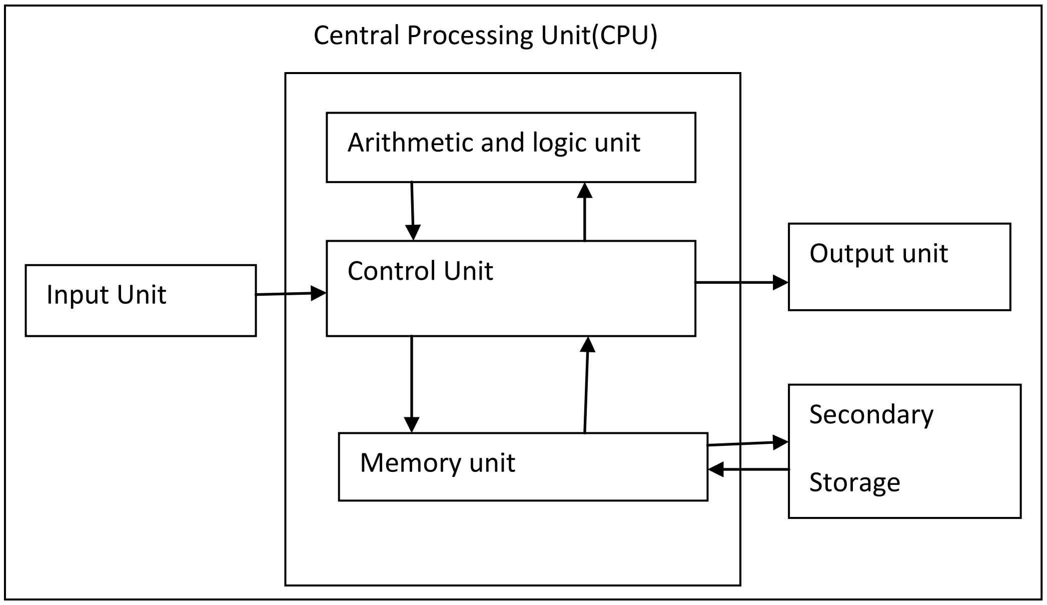
Calculator	Computer
1.works with numeric data	1.Works with alpha numeric data
2.can not store information	2.can store information
3.Input in the form of pressing or push buttons.	3.Input may be different types

Comparison of computer and human

Functions	Computers	Human being
1.Speed	Very fast	Slow
2.decision making	Very poor	Good
3.memory	Very good	Normal
4.accuracy	Very good	Not so good

Today computer systems are found everywhere .computer have become almost as common as televisions.

Block diagram of a computer:



The computer consists of **four** blocks , they are

1. Input unit
2. Central processing Unit(CPU)
3. Output Unit
4. Auxiliary Memory unit

Input unit: Input unit is provided for man-to-machine communication. It accept data in human readable form converts it into machine readable form and sends it to CPU.A computer may have one or more input devices depending upon its type, size and use. Keyboard and mouse are most commonly used input devices. Other input devices are :punch card reader,lightpen etc.

CPU: it consists of three units: Control unit(CU),Arithmetic and logic unit(ALU),and memory unit(MU). The main function of CPU is:

1. Control the sequence of operation as per the stored instructions
2. Issue commands to all parts of the computer
3. Stores data and instructions
4. Process the data and sends results to output.

Control unit: The control unit controls and coordinates all operations of the CPU,input and output devices.

1. It gives commands to transfer data from input unit to memory unit and ALU
2. It transfers the results from ALU to the memory unit and output unit.

ALU: it carries out all arithmetic operations like addition , subtraction and division etc and also it perform all logical operations

Memory unit: It is used to store programs and data.It is mainly two types

1.**Main memory or primary memory or immediate access storage(IAS**),which is part of the CPU

2.Auxilary memory or secondary storage ,which is external to the CPU

Output unit: Output unit is provides for machine to man communication. It receives the information from CPU in machine readable form and presents it to the user in a desired form.A computer may have one or more output devices depending upon use. Printer ,monitor etc.

Input devices:

Keyboard: A keyboard is used to enter data into a computer. The keyboard contains function keys, numeric keys and toggle keys(caps lock,num lock, scroll lock) and so on. The keyboard is the most widely used input device. It has keys similar to a type writer to enter characters and other symbols.

Mouse: A mouse is a picking device used to select a command by moving it any direction on a flat surface. It has two or three buttons to confirm the selection. The cursor is moved to the required icon or menu on the monitor and a button is pressed.

Scanner: Scanners are used to scan a printed page, photograph or an illustration. These data are converted into bit patterns for processing ,storage or output. When a image Is scanned it is converted into light and dark picture elements.

Output devices: The output devices receive information from the CPU and present it to the user in a desired form. Printers and VDU s are most commonly used output devices.

Display screen or visual display or monitor: When a text is keyed in , the screen displays the characters.

Printers: The final output obtained from a printer is often referred to as a printout. Printers are available with a variety of printing mechanism such as speed and varying quality.

Benefits of computer:

The main benefits of using a computer are

Speed Accuracy Diligence Storage Automation

Speed: A computer works at very high speed and is much faster than human beings it can perform hundreds of calculations in less than a few seconds.

Accuracy: In addition to being fast, a computer is very accurate if the data and instructions given to the computer are correct, then the result given by the computer will also be correct.

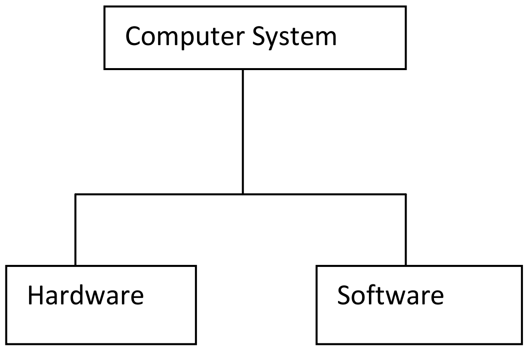
Diligence: Human beings get tired and bored doing the same work again and again .But the computer never gets tired or bored if it has to repeat the same work many times it can continue doing the same job with the same accuracy.

Storage: Computer can store a large amount of information for future use. For example the details about all the students in a college rollno, names, addresses etc can be stored in a computer. This information can be obtained from the computer whenever required.

Automation: a computer can be instructed to do a task automatically for example print thousand copies of the invitation card for the college annual day ,the computer has to be instructed only once and not thousand times.

Components of a computer: A computer is a system made of two major components

a)Hardware b)Software



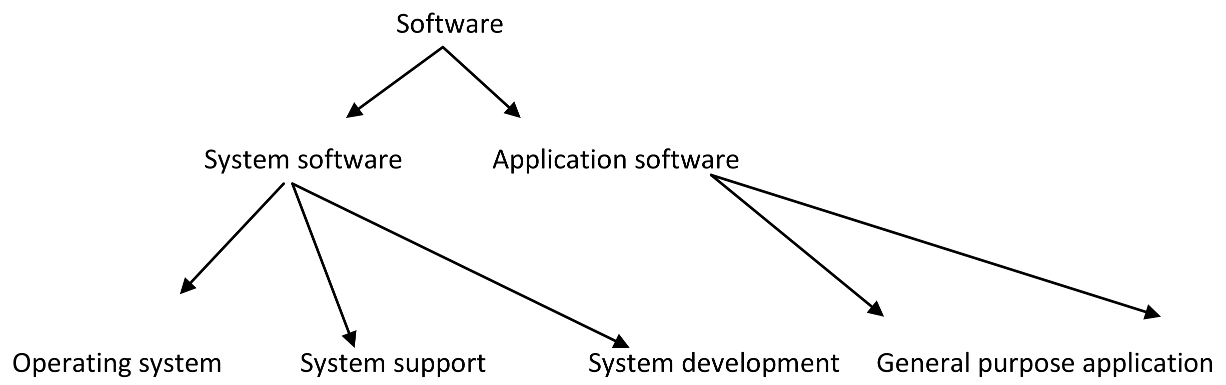
Hardware: Computer hardware is the physical Equipment: It is what which we can touch and feel (viewable components) computer hardware consists of the monitor, CPU,keyboard ,mouse and all other devices connected to the computer either externally or internally.

Software: The physical equipment that makes up computer will not work unless there is instruction to guide the device. These instructions that are grouped as a set is called program. These set of program are called software. The hardware cannot function without software and software is created by people.

Program: Set of instructions which are going to be executed to perform particular task(work).

Process: Program under execution is called as a process.

Types of software: Software can be classified into two types they are system software and application software. Both types of software is necessary for the computers.



System software: It consists of programs that manage the hardware resources of a computer and perform required information processing tasks. These programs are divided into three classes: Operating system, system support and system development.
Operating system: It is an interface between user and computer hardware
System support software: It provides utilities and other operating services. Examples of system utilities are sort programs and disk format programs.
System development software: It includes language translators that convert program into machine language for execution, debugging tools to ensure that the programs are error free.
Appliaction software: It is a set of program that are used to perform particular task. It is divided into two types
General purpose software: It includes word processors, database management system etc
Application specific software: It can be used only for its intended purpose.
Difference between hardware and software

Hardware	**Software**
1.It is the physical components of the computer system	1.It is collection of program to bring the computer hardware system into operation
2.It consists of electronic components like IC's, boards etc.	2.It consist of numbers, alphabets, alphanumeric symbols, identifiers, keywords etc.
3.The design can be modified according to the capacity	3. This should be prepared according to the type of software.

Algorithm: An algorithm is the step by step logical procedure for solving a problem. Each step is called an instruction. An algorithm can be written in English like sentences or in any standard representation.
An algorithm must satisfy the following properties:
1. Input: Zero or more inputs
2. Output: At least one quantity is produced
3. Definiteness: Each instruction is clear and unambiguous
4. Finiteness: The algorithm should terminate after a finite number of steps. It should not enter into infinite loop.
5. Effectiveness: Each operation must be simple and should complete in a finite time.

Qualities of a good algorithm:
The following are the primary factors that are often used to judge the quality of an algorithm.
Time
Memory
Accuracy
Sequence
Generability
Time: The lesser is the time required to execute a program better is the algorithm
Memory: Computer takes some amount of memory to storage to execute a program. The lesser is the memory required the better is the algorithm
Accuracy: Algorithms which provide the accurate results to a given problem should be chosen.
Sequence: The procedure of an algorithm must form in a sequence
Generability: The designed algorithm must solve the problem for a different range of input data.
Algorithm is divided into two sections:
Algorithm header: It consists name of the algorithm, problem description (process), input and output (pre condition and post condition) and return
Algorithm body: It consists of logical body of the algorithm by making use of various programming constructs and assignment statements.
Algorithm Header: The heading consists of keyword algorithm, name of the algorithm and parameter list. The syntax is
Algorithm name (p1, p2, --------------------pm)
Algorithm---the keyword should be written first
Name-here write the name of the algorithm
p1, p2--------------pm----write parameters if any
Purpose---short statement about what the Algorithm does.
Precondition: Input to the program
Post condition: Output from the program
Return: If a value is returned it is identified by return condition
Any algorithm could be written using three programming constructs: sequence, selection, and iteration (loop)
Sequence: A sequence is one or more statements that do not alter the execution path within an algorithm.
Selection: A selection statement evaluates a condition and executes zero or more alternatives. The results of the evaluates determine which alternates are taken.
Iterative: A loop statements iterates a block of statements, first the condition is evaluated before block of statements if it is true then statements are executed.
Examples:
Sequence:
Algorithm to find sum of two integer numbers
Algorithm sum ()
Process: To calculate of sum of two numbers
Pre condition;read two values

Pose condition;sum of two values
Return:nothing
Step 1:start
Step 2:read two integer numbers
Step 3:perform of sum of 2 numbers
Step 4:display the sum:
Step 5:stop

Ex2:Alogrithm to find sum of three integer numbers

Algorithm addition( )
Process:sum of three integers
Pre: read three integer values
Post:addition of three values
Return:nothing
Step 1 :start
Step2:read the three integers
Step 3:peform addition three values
Step 4: display the sum
Step 5;stop

Ex3:Algorithm to find sum of three floating point numbers

Algorithm floatsum( )
Process:addition of three floating point numbers
Pre;read three floating point numbers
Post:addition of three floating point numbers
Return:nothing
Step 1:start
Step 2:read three floating point numbers
Step 3:perform addition of three floating point values
Step 4:display the sum
Step 5:stop

Algorithm to find average of two numbers

Algorithm average( )
Process:average of two numbers
Pre:read two values
Post :average of two values
Return:nothing
Step 1:start
Step2:read first number
Step 3: read second number
Step 4:perform sum of two numbers
Step 5:divide sum with 2
Step 6:print the average Step 7:stop

Selection:

Algorithm to find gretest among two numbers

Algorithm greatest( )
Process:greatest among two numbers

Pre;read two values
Post :large value among two
Return :nothing
Step 1:start
Step 2:read two numbers
Step 3:compare two values(ex a and b)
Step 4:if a is greater than b display a is greater otherwise b is greater
Step 5:stop

Iterative:

Algorithm to find sum of n numbers

Algorithm sumofnnumbers( )
Process:cacluate sum of n numbers
Pre:read n numbers
Post;sum of n numbers
Return:nothing
Step 1:start
Step 2:read numbers upto n(suppose if n=10 read 10 numbers)
Step 3: perform the addition
Step 4:repeat step 2 and step 3 upto n numbers addition
Step 5:display the result
Step 6:stop

Representation of an algorithm:

Algorithm can be represented using

1. flowcharts
2. pseudo code
3. program

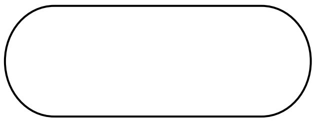
Flowchart: A flowchart is a pictorial representation of an algorithm. It shows the flow of operation in pictorial form and any error in the logic of the problem can be detected very easily. A flowchart uses different boxes and symbols to denote different types of instructions. These symbols are connected by solid lines with arrow marks to indicate the operation flow. Normally an algorithm is represented in the form of a flowchart and the flowchart is then expressed in some programming language to prepare a computer program.

Flow chart symbols:

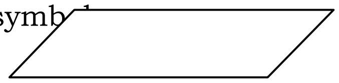
A few symbols are needed to indicate the necessary operations in flowcharts. These symbols have been standardized by the American National Standard Institute (ANSI).These symbols are

Terminal:

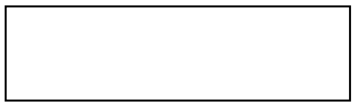
The terminal (oval)symbol as the name implies is used to indicate the beginning(start),ending(stop)and pause in the logic flow. It is the first and last symbol in the programming logic.



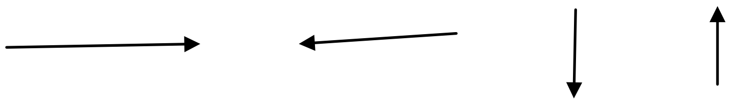
Input/output:
The input/output (parallelogram) symbol is used to denote any function of an input/output device in the program. All input/output instructions in the program are indicated with this symb



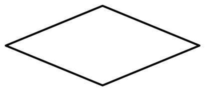
Processing: A processing(rectangle) symbol is used in a flowchart to represent arithmetic and data movement instructions. Thus all arithmetic processes of addition ,subtraction, multiplication etc



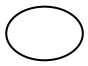
Flow lines: Flow lines with arrow heads are used to indicate the flow of operations i.e. the exact sequence in which the instructions are to be executed. The normal flow of flowchart is from top to bottom and left to right.



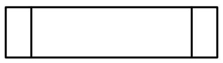
Decision: The decision (diamond) symbol is used in a flowchart to indicate a point at which a decision has to be made and a branch to one of two or more alternate points is possible.



Connectors: If a flowchart becomes very long the flow chart starts criss crossing at many places that causes confusion and reduces understandability of the flowchart. Moreover there are instances when a flowchart becomes too long to fit in a single page and the use of flow lines become impossible. A connector symbol is represented by a circle. A pair of identically labeled connector symbols is commonly used to indicate a continued flow when the use of a line is confusing. So two connectors with identical labels can be used.



Predefined process: The predefined process(double sided rectangle) symbol is used in flowcharts to indicate that modules or subroutines are specified elsewhere.



Annotation: The annotation (bracket with broken line) symbol is used in flowcharts to indicate the descriptive comments or explanation of the instruction.

Examples on flowcharts and algorithms:

1.Write an algorithm for finding the average of three numbers and draw the flowchart for the same.

Step 1: read the value

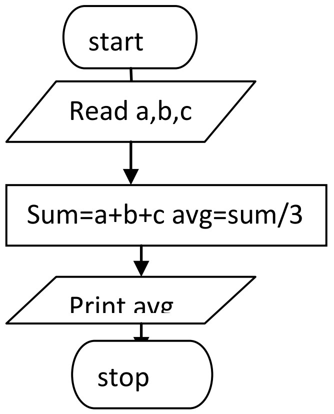
Read a,b and c

Step2:compute sum=a+b+c

Avg=sum/3

Step 3:print avg

Step 4:stop



Example 2.Write an algorithm for finding greatest among two numbers

Step 1:start

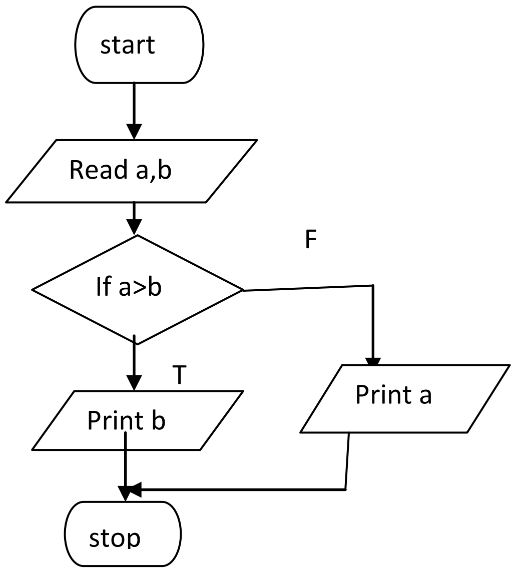
Step 2:read two numbers

Step 3:compare two values(ex a and b)

Step 4:if a is greater than b display

a is greater otherwise b is greater

Step 5:stop



Example 3: write an algorithm and flowchart for finding the sum of n numbers

Step 1:start

Step 2:read numbers upto n

Initialize count =1 sum=0

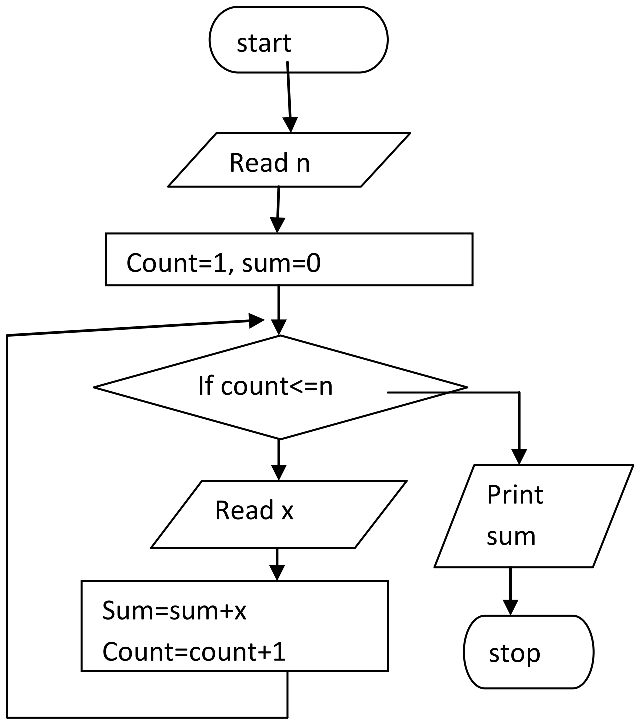
Step 3:repeat through count<=n

Step 4: x=1 sum=sum+x

Count=count+1

Step 5:print sum

Step 6:stop



Advantages of flowchart:
1. A flowchart can easily explain the program logic to the program development team.
2. The flowchart details help prepare efficient program coding.
3. The flowchart helps detect and remove the mistakes in a program
4. A flowchart is useful to test the logic of the program.

Pseudocode:
Pseudocode is a tool used for planning a computer program logic or method. .”pseudo” means imitation, and code refers to the instructions written in a computer language. The pseudocode instructions may be written in English or any vernacular.
Pseudocode is useful to design structured programs. A pseudocode looks similar to the actual coding. Consider the following pesudocode to find the biggest of given two numbers.

```
Start
Read  Number1,Number2
If  Number1>Number2
Print Number1
Else
Print Number2
Endif
End
```

Program development steps:
While developing a program the following steps are required
Problem specification: The problem must be thoroughly understood. The programmer must know exactly what he wants to do before he can begin to do it. If the user does not

fully understand his/her own problem there is little hope that the computer will understand it better.

Outlining the solution: In this step a solution method is to be chosen or developed. Once the problem is cleared then a solution method for solving the problem has to be found, a path leading from what has been given to what is required.

Selecting and representing the algorithm: The solution method is described step by step .when a solution method has been outlined ,it must be translated into an algorithm. An algorithm is a way of representing the solution used to solve the particular problem .Algorithms can be written in an ordinary language or in any other standard notation.

The algorithm is programmed: It will be relatively easy to convert the algorithm into a program. The choice of a programming language depends on the nature of problem and the availability of programming languages. Sometimes programming is also called coding.

Removing errors: Errors are common while developing a program that causes wrong results. The errors are to be corrected after detecting them. Assuring a program's correctness is called program verification and process of detecting and removing the errors is called debugging.

Three types of errors are possible while developing a program

Syntax errors
Runtime errors
Logical errors

Syntax errors: These errors are due to wrongly typed statements, which are not according to the syntax or grammatical rules of language.(the grammatical rules which govern a programming language are called as syntax. When these rules are not followed or adhered to, then that language terns it as an syntax error)

Ex; sum=x+y+z);

In this the error is missing left parenthesis .typing errors are found quickly at the time of compiling the program. Most c implementations will generate diagnostic messages when syntax errors are detected during compilation.

Run time errors: Errors detected at the time of program execution are called run time errors.

Run time errors may occur during the execution of programs even though it is free from syntax errors

Ex. Dividing by zero-10/0
Square root of a negative number.

Logical errors: Syntax and run time errors generally produce error messages when the program is compiled or executed. These are easy to find and they can be corrected.

The third type of error is logical error, which is very difficult to detect logical errors are due to the existence of logically incorrect instructions in the program. These errors can be known only after output is examined.

Testing and validation: Once program is written a program must be tested and then validated .The program always must guarantee to produce correct results.

Documentation and maintenance: Documentation is a process of collecting, organizing and maintaining complete information about the program. Modifications of the program according to changing requirements are called program maintenance. The program must be written in a clear and easy way, so that the other users who use the program may not find any difficulty. In order to achieve this the program should be maintained well by using the following

1. Proper comments
2. Proper indentation for improving readability and clarity
3. Detail description about the different methods used

There are two kinds of documentation that are very much useful in program modification and enhancements.

Operational documentation: It provides the information regarding the input and output formats, operating instructions, different kinds of user information with the program and limitations if any.

Technical documentation: It provides the technical details including the design aspects and brief explanation of all the producers involved, details of the problem logic involved, hardware to be operated etc.

<u>Computer languages:Machine,Symbolic,and High level languages</u>

Computer programming means giving instructions to a computer, in the language which computer understands.

Machine language: In this language all instructions were given in the binary form (ie,0's and 1's only).machine language is also called as low level language. It is very difficult for us to write or read instruction written in binaries. Consider the following instruction written in binaries.

0101   0101101 10101100

Assembly language: Instructions are written with mnemonics to simplify the program. It is also called as a symbolic language. In order to execute these instructions all mnemonics are converted into binaries with the help of a translator known as Assembler. The program written using mnemonics is called source program, the binary form of the source program is called object program.

High level language: Instructions are written using English language with symbols and digits. The commonly used high level languages are FORTAN,BASIC,COBOL,PASCAL,C,C++,etc. The complete instruction set written in one of these languages is called computer program or source program.

In order to execute the instructions the source program is translated into binary form by a compiler or interpreter.

Advantages:

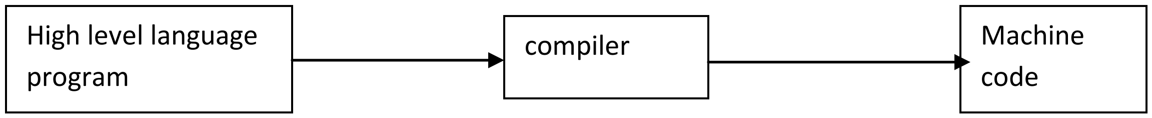
1. Easy to learn and easy to understand than machine or assembly language
2. Takes less time to write
3. Easy to maintain
4. Better documentation

Machine language vs Assembly language:

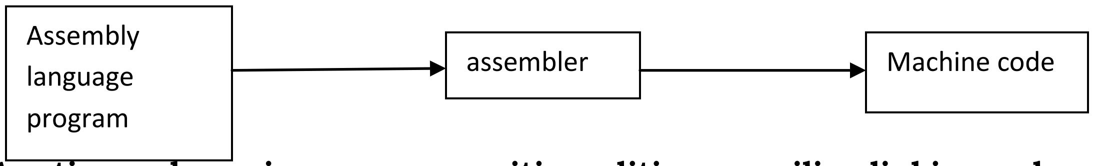
1. Execution of machine language program is faster than the assembly language program,because there is no conversion in machine language.
2. Reading and writing the assembly language programs are easy for programmers it is difficult in machine language.
3. Machine instructions are difficult to remember, where as mnemonics are easy to remember.
4. Introduction of data to program is easier in assembly language ,while it is difficult in machine language.

<u>Translators:</u> Translators are software that accepts one language as input and converts it into another language. The input for any translator is called as source language or source code and output of any translator is called as object language or object code. There are two type of translators ;they are

1.Compiler: It is a software that accepts the high level language program as input and produces the machine code as output.



2. Assembler: It is a software that accepts the assembly language program as input and produces the machine code as output.

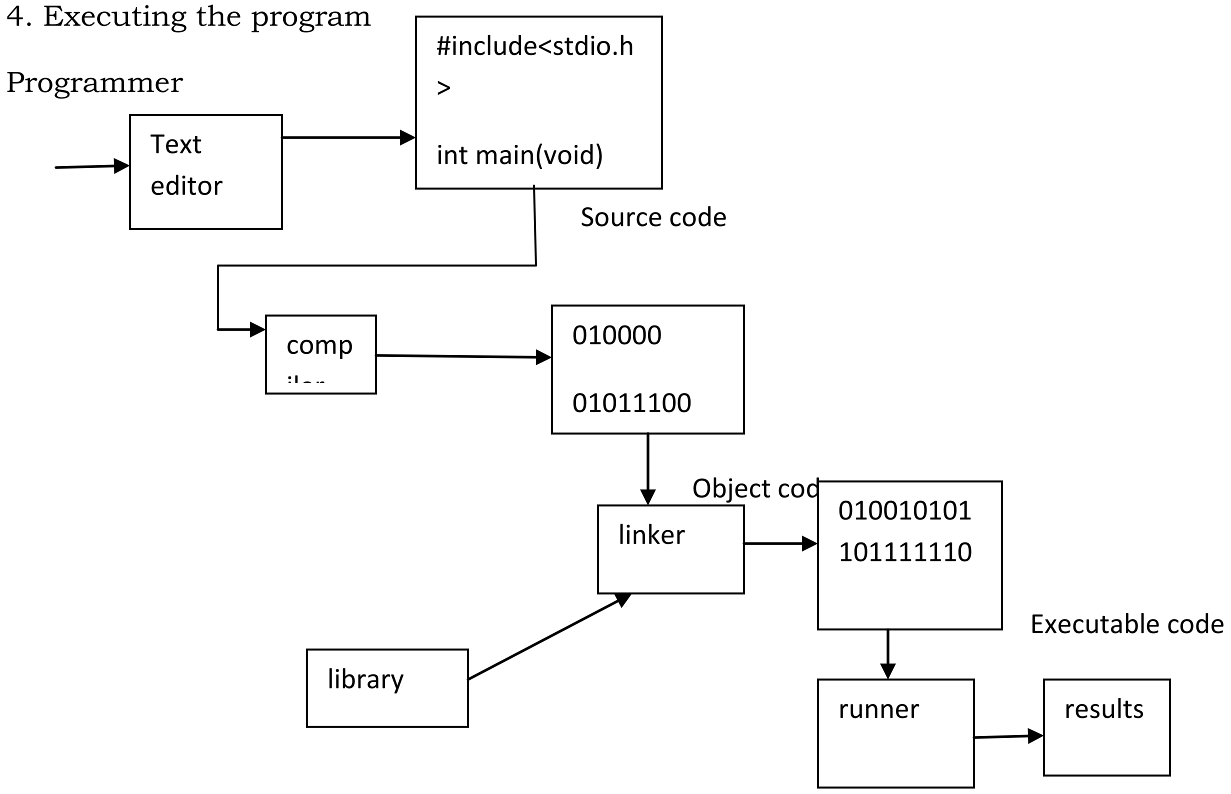


Creating and running programs:writing,editing,compiling,linking,and executing

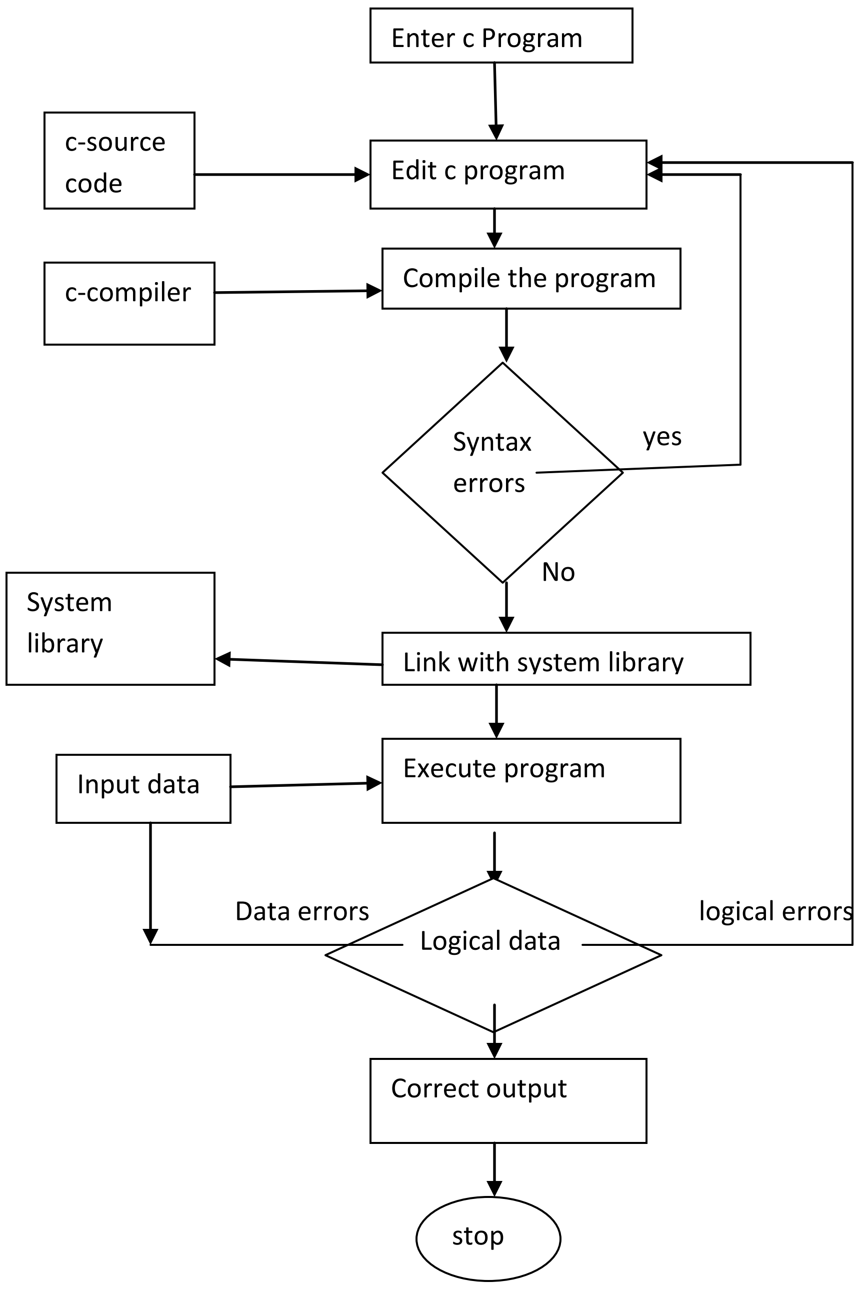
Computer understands a program only if it is coded in its machine language. In this section we explain the procedure for turning a program written in c language into machine language. This process is presented in a straight-forward, linear fashion but you should recognize that these steps are repeated many times during development to correct errors and make improvements to the code.

It is the job of the programmer to write and test the program there are 4 steps in this process.

1. Writing and editing the program
2. Compiling the program
3. Linking the program with the required library modules
4. Executing the program



Process of creating ,compiling,and executing a program



Writing and editing program: The software used to write the programs is known as text editor.

A text editor helps us enter, change and store characters. After we complete a program we save our file to disk in .c(dot c)extension. This file will be input to the compiler. It is known as source code or source file.

Compiling the program: This is the process of converting the high level language program into machine understandable form(machine language).for this purpose compiler is used. Usually this can be done in ‘c’language by pressing ALT+F9(or)choose compile option in the system. Here there is a possibility to show errors ie.syntax errors.

Linking the program with system library: C language program is the collection of predefined functions. These functions are already written some standard ‘c’ header files. Therefore before executing a program we need to link with system library. This can be done automatically at the time of execution.
Executing the program: There is the process of running and testing the program with sample data. At this time is a possibility to show two types of errors given below.
Data errors: There are the errors in which the input data given is not in a proper syntax as specified in input statements.
Logical errors: These are the errors in which the conditional and control statements cannot end their match after some sequential execution.
Executing the program can be done by pressing **CTRL+F9** or choose run option from the menu sys

Basics of c:

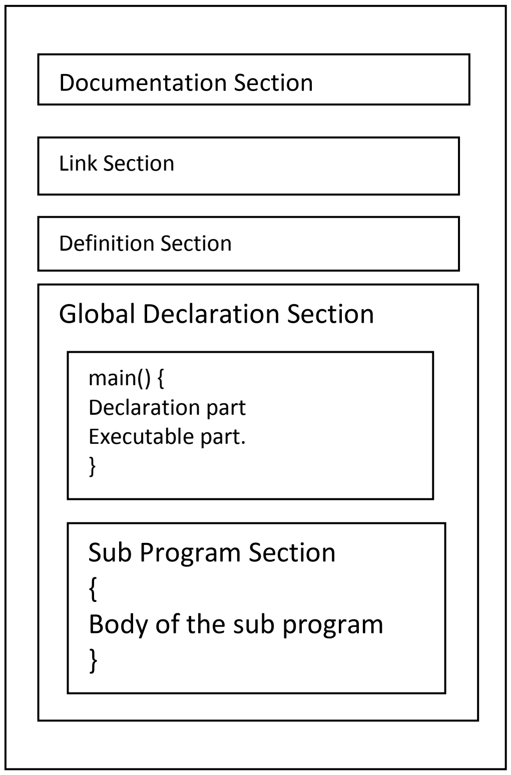
The programming language C was developed in 1972 by Dennis Ritchie at AT&T Bell laboratories USA. This was derived from a language called B, which was developed by Ken Thompson in 1970.the language B was derived from an existing language BCPL (Basic Combined Programming Language)which was developed by Martin Richards in 1967.

$$\begin{tikzcd}
\text{ALGOL (algorithmic language)} \\
\text{(By international committee,1960)} \arrow[d] \\
\text{CPL (combined Programming Language)} \\
\text{(By Cambridge university of London ,1963)} \arrow[d] \\
\text{BCPL (Basic Combined Programming Language)} \\
\text{(By Martin Richards,1967)} \arrow[d] \\
\text{B (Basic programming language)} \\
\text{\{By Ken Thompson,1970)} \arrow[d] \\
\text{C} \\
\text{(By Dennis Ritchie,1972)}
\end{tikzcd}$$

Importance of C:

1. C is often called a middle level language, because it has the feature of both high level languages , such as BASIC,PASCAL etc and low level languages .
2. C is general purpose structured language, thus requiring the user to think of a problem in terms of function modules. A proper collection of these modules would make a complete program. This modular structure makes program debugging, testing and maintenance easier.
3. C has very small instruction set. C provides rich set of data types , both built in as well as user defined data types. And it also provides rich set of operators , many of which are not available in other high level languages.
4. C is highly portable .i.e., software written for one computer can be run on another computer
5. C is powerful efficient, compact and flexible.
6. C has the ability to extend itself. We can continuously add our own functions to the existing C library functions.
7. C is a case sensitive language.
8. C language allows dynamic allocation of memory

Structure of a C program:



Documentation section: It consists a set of comment lines used to specify the name of program ,the author and other details etc.

Comments:

1. Comments are very helpful in identifying the program features
2. The lines begins with '/*' and ending with '*/' are known as comment lines.

3. These comment lines are omitted by the compiler at the time of compilation.

Examples:

/* program1*/
/*addition of 2 values*/

Link section or preprocessor section:

C is rich set of built in functions and they can used to write any complex program.These standard functions are kept in various system libraries such as

stdio.h->standard input-output header file
math.h->mathematical functions header file
conio.h->configuration input-output header file etc

If we want to use the functions of these libraries we have to provide instructions to the compiler to link the functions from the corresponding system library .This can be achieved through link section of the ‘ C’ program structure by using #include directive[#include is a preprocessor directive].

Example:-
```
#include<stdio.h>
#include<conio.h>
```

Definition section: Some problems require constant values to solve the problem

ex:area of the circle =3.14 *r*r
“3.14 can called as PI value it is a constant value”
Such symbolic constants Ex.PI can be defined in definition section using **#define** directive

Example for definition section:

#define pi 3.14

Global declaration section: The variable that is used in more than one function throughout the program are called global variables and declared outside of all the function i.e. before main ()

“Every C program must have the one main() function which specify the starting of C program”.

Main() function section:

Main() function conatains two parts

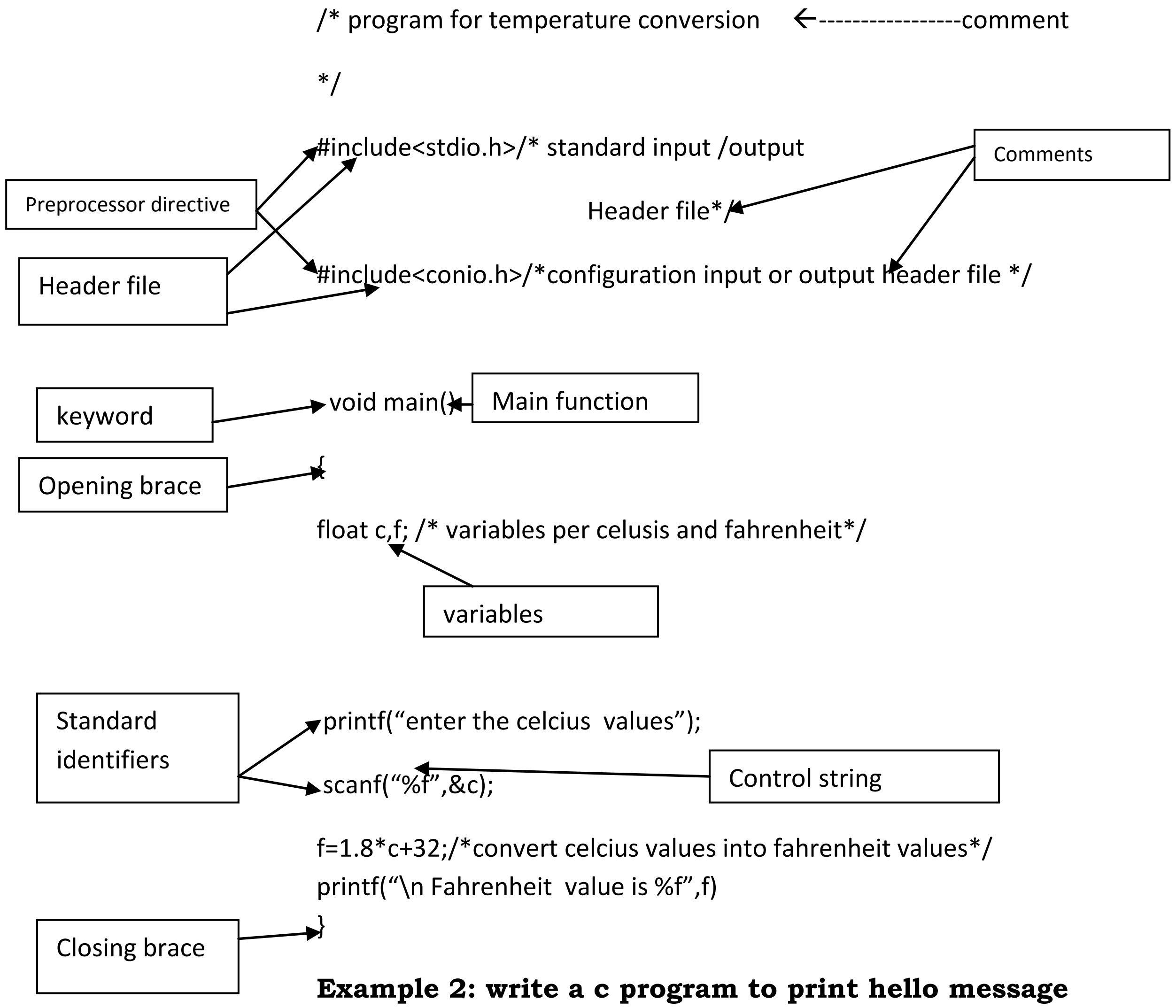
1.Declaration section
2.Executable section

Declaration section: This part is used to declare all the variables that are used in the executable part of the program and the entire code for the program is written in the executable part and these variables are called local variables.

Executable section: It contains at least one valid C statement .The execution of a program begins with opening brace ‘{‘ and ends with closing brace ‘}’.the closing brace of the main function is the logical end of the program.

Note: All the statements in the program ends with a semicolon except conditional and control statements.

Syntax for main function
main()
{
Declaration part;
Executable part;
}
Subprogram section: This section consists of one or more functions which are defined by the user. User defined functions are placed immediately after the main() function and they may appear in any order.
Example program



```
/* program for temperature conversion

*/

#include<stdio.h>/* standard input /output

                                          Header file*/

#include<conio.h>/*configuration input or output header file */


void main()

{

float c,f; /* variables per celusis and fahrenheit*/


printf("enter the celcius  values");

scanf("%f",&c);

f=1.8*c+32;/*convert celcius values into fahrenheit values*/
printf("\n Fahrenheit  value is %f",f)
}
```

Example 2: write a c program to print hello message
#include<stdio.h>
#include<conio.h>
void main()
{
printf("hello");
}

Example 3: write a program to find out the area and circumference of a circle

```
/* to find out the area and circumference of acircle*/
#include<stdio.h>
#include<conio.h>
#define PI 3.14
void main()
{
int r;
float a,c;
clrscr();
printf("enter radius \n");
scanf("%d",&r);
a=PI*r*r;
c=2*PI*r;
printf("area=%f \n",a);
printf("circumference=%f \n",c);
getch();
}
```

Character Set: The set of characters used in a language is known as its character set. These character can be represented in computer. Every language makes use of its character set to form words or symbols that make up the vocabulary of the language. The character set for ANSIC is given below

Alphabets: Uppercase :A,B………,Z

Lowercase :a,b,………………,z

Digits: 0 1 2 3 4 5 6 7 8 9

Special characters:

,comma	<opening angle bracket	>closing angle bracket
.period	_underscore	(opening parenthesis
;semicolon	$dollar sign	)closing parenthesis
:colon	%percent sign	[opening square bracket
?question mark	#number sign	]closing square bracket
'single quote	&ampersand	{opening barace
"double quote	Caret	/slash
!exclamation mark	*asterisk	}closing brace
|vertical bar	-minus sign	\back slash
~tilde	+plus sign	
White Space characters:		
\b blank space	\n new line	\r carriage return
\f form feed	\t horizontal tab	\v vertical tab

C-Tokens:

"Smallest individual unit in C program is known as token". C has 6 types of tokens.

Keywords(ex:int,double,for)

Identifiers(main(),total,avg)

Constants(eg;39,39.77)

Operators(+,*,-)

Special charcters(#,[],{ })
Strings("xyz","lak")
In C language every word is classified into either a keyword or an identifier.
Keywords: keywords are the English words which have fixed meaning ,which cannot be changed .Keyword should be written in lowercase .C supports 32 keywords they are

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

Identifiers: Identifiers are names given to various program elements such as variables, functions and arrays etc.
Rules for naming identifiers:
1. Identifiers consist of letters and digits in any order
2. The First character must be a character or may begin with underscore (_).
3. Both upper/lower cases are permitted although uppercase character is not equivalent to corresponding lowercase character.
4. The underscore ' _' can also be used and is considered as a letter
5. An identifier can be of any length while most of the 'C' compiler recognizes only the first 31 characters
6. No space and special symbols are allowed between the identifier
Examples for identifiers:
tv9
roll_no
name
NAME(name is different from NAME)

Examples for invalid identifiers:
4xy
*%2
roll no
Basic data types and sizes:
A data type defines a set of values and the operation that can be performed on them.
Every data type item(variable, constant ,etc)in a C program has a data type associated with it.
Data given to the system may be different types like integer, real, word etc. Each and every data has to be represented with the appropriate data type. Each data type may have predefined memory requirement and storage representation.
C supports the following classes of data types:

Built in (fundamental)data types
Derived data types
User defined data types
Void data types

C data types			
Primary	**User defined**	**Derived**	**Empty data set**
char int float double	typedef	Array Pointers Structure Union	void

Integer(int): Integers are the whole numbers in the range -32768 to +32767 including 0(zero).An integer can be represented with the keyword int. It is stored 16 bits as its storage location. An integer can be stored as short int, int, long int both in signed and unsigned forms depending upon the value stored.

Type	**Size**	**range**
int or signed int	2 bytes	-32768 to +32767
unsigned int	2 bytes	0 to 65535
short int or signed short int	1 byte	-327682 TO +32767
unsigned shortint	1 byte	0 to 65535
long int or signed long int	4 bytes	-2 14 748 3648 to+2147483647
unsigned long int	4 bytes	0 to 4294967295

Float type: The floating point numbers are generally stored 32 bits with 6 digits of precision. These numbers are defined by using the keywords float and double. The double data types uses the 64 bits.

Type	**Size**	**range**
Float	4 bytes	3.4e-38 to 3.4e+38
Double	8 bytes	1.7e-308 to 1.7e+308
long double	10 bytes	3.4e-4932 to 3.4e+4932

Character: Characters are defined by the data type char. There are usually stored in 8 bits of internal storage.

Type	**Size**	**Range**
signed char	1 byte	-128 to +127
unsigned char	1 byte	0 to 255

Data type	**Description**	**Memory bytes**	**Range**	**Control string**	**example**

int	Integer quantity	2 bytes	-32 768 to +32767	%d or %i	int a=39
char	Single character	1 byte	-128 to +127	%c	char s='n'
float	Floating pointing numbers(number containing a decimal point or an exponent)	4 bytes	3.4e-38 to 3.4e+38	%f or %g	float f=29.77

User defined data types: typedef is a user defined data type used to define an identifier that would represent an existing data type.

Syntax: typedef data type identifier

typedef; it is the user defined type declaration

datatype: It is the existing data type

identifier: It is the identifier refers to the new name given to the data type

Example: typedef int marks;

marks m1,m2,m3

Here marks is the type of int and this can be later used to declare variables. Another user defined datatype available is enumerated data type enum.

Syntax; enum identifier{value 1,value 2,……………………value n}

Description: identifier is the user defined enumerated data type .value 1,value 2,……….value n are the enumeration constants.

Example: enum day{mon,tue,wed,………….sun}

enum day w-st,w-end;

w-st=mon;

w-end=sun;

Derived data types: Derived data types are the data types such as arrays, functions, structures ,pointers.

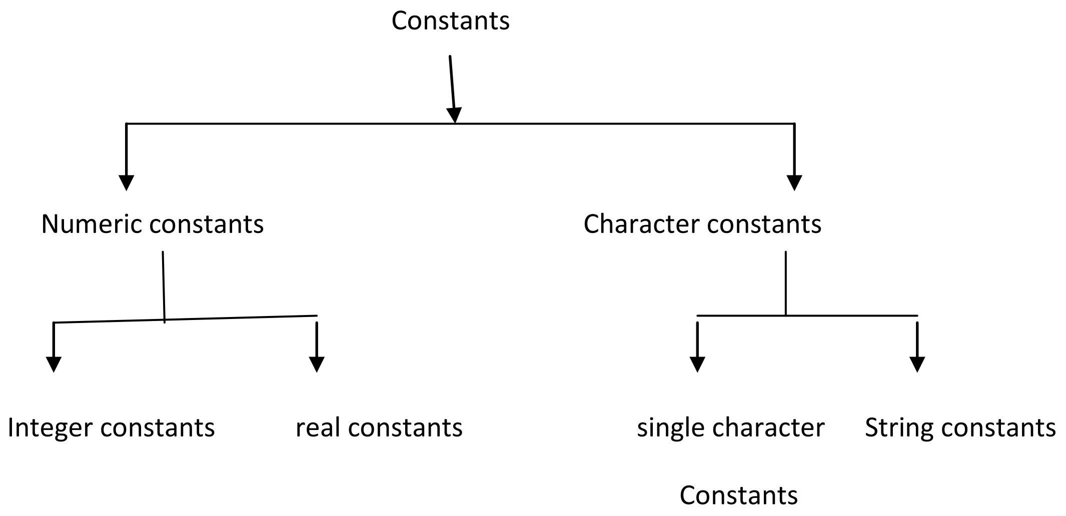
Empty data set: void is the keyword used to specify empty data set.The void type has no values. This is usually used to specify the type of functions this is generally specified with the function which has no arguments.

Constants: Constants are those which do not change during the execution of the program .Constants may be categorized in to:

Numeric constants

Character constants

String constants



Integer constants: An integer constant is a sequence of digits without any fractional part. There are 3 types of integer constants in C language. They are

Decimal integer constants: It is a sequence of one or more digits ([0…9],the symbols of decimal number system). It may have an optional + or – sign .In the absence of sign, the constant is assumed to be positive. Commas and blank spaces are not permitted. It should not have a period as part of it.

Valid decimal constants	**Invalid decimal constants**
48597	13,436 illegal character comma
-433132	10.235 illegal character period
+25	10 455 illegal character space
6	75-976 illegal character dash
857	064368 the first digit cannot be zero

Octal integer constants:

- ➔ It is a sequence of one or more digits([0..7],symbols of octal number system)
- ➔ It may have an optional + or – sign .in the absence of sign, the constant is assumed to be positive
- ➔ It should start with the digit 0
- ➔ Commas and blank spaces are not permitted
- ➔ It should not have a period as part of it

Valid octal constants	**Invalid octal constants**
0	23316 does not begin with zero
0126	04682 illegal digit 8
-045673	07.36 illegal character .

+02345	07,45 illegal character

Hexa decimal integer constants:

- ➔ It is a sequence of one or more symbols ([0..9][a..f][A…F],the symbols of hexadecimal number system)
- ➔ It may have an optional + or – sign .in the absence of sign, the constant is assumed to be positive
- ➔ It should start with the symbol 0X or 0x
- ➔ Commas and blank spaces are not permitted
- ➔ It should not have a period as part of it

Valid constants	invalid constants
0x3b25	0x12.6 iiiegal character .
0XF98	0a236 does not begin with 0x or 0X

Real constants: The real constants also known as floating point constants are written in two forms

1.fractional form

2.exponential form

Fractional form: The real constants in fractional form are characterized by the following characteristics

- ➔ Must have at least one digit
- ➔ Must have a decimal point
- ➔ May be positive or negative and in the absence of sign taken as positive
- ➔ Must not contain blanks or commas in between digits
- ➔ May be representd in exponential form if the value is too high or too low.

Valid real constants	invalid real
456.78	4 56
-123.45	4,56

Exponential form: The exponential form offers a convenient way for writing very large and small real constants. For example 56000000.00,which can be written as 0.56*10 is written 0.56E8 or 0.56e8 in exponential form.

A real constant expressed in exponential form has two parts:1 mantissa part 2.Exponent part .Mantissa is the part of the real constant to the left of E or e, and the exponent of a real constant is to the right of E or e.

Character constants:

Any character enclosed with in single quote(' ') is called character constant. A character constant:

1. May be single alphabet, single digit or single special character placed within the single quotes

2. Has a maximum length of 1 character

Example: ‘C’,’c,’,’:’

<u>**String Constants:**</u>

A string constant is a sequence of alphanumeric characters enclosed in double quotes whose maximum length is 255 characters.

Examples:”my name is computer” ,”programming with c”

Examples: invalid: my name is computer,” my name is computer

Backslash character constants:

C supports some special backslash character constants that are used in output functions.

Code	**Meaning**
‘\a’	Beep
‘\n’	Newline
‘\t’	Horizontal tab
‘\b’	Backspace
‘\r’	Carriage return
‘\f’	Formfeed
‘\v’	Vertical tab
‘\\’	Back slash
‘\”	Single quote
‘\0’	Null
‘\?’	Question mark
‘\N’	Constant(wher N is an octal constant)
‘\xN’	Constant(where N is a hexadecimal constant)

Variable: A variable is used to store a data value. A variable may take different values at different times during program execution

Rules for defining a variable:

1. It consists of letters, digits, underscore character (_).
2. Must begin with a letter or an underscore
3. Maximum distinguishable length should be 8 characters
4. It should not be a keyword
5. White spaces are not allowed Ex:a,x,sum etc

Variable declaration: The variables used in the program should be declared first.

Syntax: data type identifier;

Ex: int a;

char ch;

float f;

<u>**Variables initialization or assign values to variables:**</u>

We can assign variables at the time of declaration or we can assign values separately.

Syntax: data type identifier= value;

Ex. int a=3;

Syntax : datat ype identifier;

identifier=value;

Ex: int a;

a=3;

Declaring a variable as constant:
When the value of some of the variable may remain constantly during the execution of the program in such a situation ,this can be done by using the keyword const.
Syntax: const data type variable=value;
Const-it is the keyword to declare constant
Datatype-it is the type of the data
Variable-it is the name of the variable
Example: const int dob=3977;
Operators: An operator is a symbol that specifies an operation to be performed on the operands. The data items that operators acts upon are called operands
Example: a+b
Where '+' is operator and a,b are operands
The operators tells the computer to perform the specified operation on operands.
Types of operators: 'C' provides a rich set of operators, depending upon their operation they are classified as
Arithmetic operators
Relational operators
Logical operators
Assignment operators
Increment and decrement operators
Conditional operators(ternary operator)
Bitwise operator
Special operator
Arithmetic operators: These operators are used to perform basic arithmetic operations like addition, subtraction, multiplication, division and modulo division. The operators used for performing these arithmetic operations are as follows

Operator	**Meaning**	**Example**
+	Addition	2+9=11
-	Subtraction	9-2=7
*	Multiplication	2*9=18
/	Division	9/3=3
%	Modulo division	9%2=1

The modulo division gives the remainder of an integer division. The modulo division cannot be used with real operands.

The following table shows how the division operator operates on various data types.

Operation	**Result**	**Example**
int/int	int	2/5=0
real/int	real	5.0/2=2.5
int/real	real	5/2.0=2.5
real/real	real	5.0/2.0=2.5

In the above table first example the integer division truncates the fractional part.

Arithmetic operators can be classified as
Unary arithmetic; it requires only one operand example:+x,-y
Binary arithmetic: it requires two operands
Ex: a+b,a-b,a/b,a*b,a%b(2+3,2-3,2/3,2*3,2%3)
Integer arithmetic: It requires both operands are integer values for arithmetic operation
Ex: a=5,b=4

Expression	**result**
a+b	9
a-b	1

Floating point arithmetic: It requires both operands are float type for arithmetic operation.
Ex: a=6.5,b=3.5

Expression	**result**
a+b	10.0
a-b	3.0

Example programs to iilustrate the usage of arithmetic operators
```
#include<stdio.h>
#include<conio.h>
void main()
{
        int I,j,k;
        clrscr();
        i=10;j=20;k=i+j;
        printf("value of k is %d\n",k);
        getch();
}
Output:value of k is30

2.#include<stdio.h>
#include<conio.h>
void main()
{
        int I,j,sum,diff,pro,quo,rem;
        clrscr();
        i=22;
        j=21;
        sum=i+j;
        diff=i-j;
        pro=i*j;
        quo=i/j;
        rem=i%j;
        printf("the sum of i and j is %d",sum);
        printf("the difference between i and j is %d",diff);
        printf("the product of i and j is %d",pro);
        printf("the quotient of i and j is %d",quo);
        printf("the remainder of i and j is %d",rem);
        getch();
}
```

Output: the sum of *i* and j is 43
The difference between *i* and j is 1
The product of *i* and j is 462
The quotient of *i* and j is 1
The remainder of *i* and j is 1

Relational operators: Relational operators are used to compare two or more operands. Operands may be variables, constants or expression. Generally these operators are used in decision making process in conditional or control statements. For example we may compare the age of two persons or the price of two items and so on. These comparisons can be done with the help of relational operators.

Operator	**Meaning**	**Example**	**Return value**
<	Is less than	2<9	1
<=	Is less than or equal to	2<=2	1
>	Is greater than	2>9	0
>=	Is greater than or equal to	3>=2	1
==	Is equal to	2==3	0
!=	Is not equal to	2!=2	0

These operators return two values if the comparison is true then it return 1 or else it returns zero(0)
Syntax: op1 relational operator op2
Op1,op2 are constants or expression variable
Ex: 9>2 this is true so '1' is returned
Note: When both arithmetic operators and relational operators are used, arithmetic calculation is done first and the result is then compare.
Example:program to use various relational operators

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        printf("\n condition:return values\n");
        printf("\n 5!=5:%d",5!=5);
        printf("\n 5==5:%d",5==5);
        printf("\n 5>=5:%d",5>=5);
        printf("\n 5<=50:%d",5<=50);
        printf("\n 5!=3:%d",5!=3);
        getch();
}
Output:
Condition:return values
5!=5:0
5==5:1
5>=5:1
5<=50:1
5!=3:1
```

Logical operators: Logical operators are used to combine the result two or more conditions. The logical operators used in "C" languages are

Operator	**meaning**
&&	logical AND
||	logical OR
!	logical NOT

&&: This operator return true i.e,'1' when both the conditions are true otherwise it returns '0'.
||: This operator returns true if at least one of the conditions combined is true
!: This operator reverses the value of the expression it operates on
Example: (3<7)&&(7<5)->1&&0=0(false) result is 0

Example: program to demonstrate use of logical operators

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int c1,c2,c3;
        clrscr();
        printf("enter values of c1,c2,c3");
        scanf("%d%d%d",&c1,&c2,&c3);
        printf("(c1<c2)&&(c2<c3)",(c1<c2)&&(c2<c3));
        printf("(c1<c2)||(c2<c3) is ",(c1<c2)||(c2<c3));
        printf("!(c1<c2) is ",!(c1<c2));
        getch();
}
Output:
enter  values of c1,c2 c3;
3 1 5
(c1<c2)&&(c2<c3) is 0
(c1<c2)||(c2<c3) is 1
!(c1<c2) is 1
```

Assignment operator: Assignment operator are used to assign a value or an expression or a value of an variable to another variable.
Syntax: variable=expression(or)value;
Example: x=10;

x=a+b;
x=y;

(i)Compound assignment or short hand assignment operator : 'C' provides compound assignment operator to assign value to a variable in order to assign a new value to a variable after performing a specified operation. Some of the compound assignment operators and their meanings are given in the below table.

Operator	**Example**	**Meaning**
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Nested or multiple assignments: 'C' language has got distinct features in assignment called nested or multiple assignments using these feature. We can assign a single value or an expression to multiple variables.

Syntax: var1=var2=var3=……..var n=single variable or expression;

Example: i=j=k=1;
x=y=z=(i+j+k);

Demonstartion of assignment operators:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,j,k;
        clrscr();
        i=4;
        j=5;
        k=j;
        printf("k=%d",k);
        getch();
}
```

Output:k=5

Example 2:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x,y;
        clrscr();
        x=4;
        y=5;
        x+=y;
        printf("x=%d",x);
        getch();
}
```

Output=9

Increment and decrement operators:

"C has two way useful operators such as increment(++) and decrement(--) operators".
"++" adds one to the variable and "--"subtracts one from the variable these operators are called unary operators as they act upon only on a single variable.

Operator	**Meaning**
++x	Pre increment
x++	Post increment
--x	Pre decrement
x--	Post decrement

Where x++;post increment ,first do the operation and then increment
Example:a=x++;
1.a=x (value of expression is x before increment)
2.x=x+1(value of x is incremented by 1)
++x;pre increment ,first increment and then do the the operation
Example;a=++x;
1.x=x+1(value of x is incremented by 1)
2.a=x(value of expression is x after increment)
--x;pre decrement,first decrements and then do the operation
Example:a=--x;
1.x=x-1(value of x is decremented by 1)
2.a=x(value of expression is x after decrement)
x--;post decrement,first do the operation and then decrement
Example:a=x--;
1.a=x(value of expression is x before decrement)
2:x=x-1(value of x is decremented by 1)
Note:We cannot increment or decrement constant and expressions.
Examples:
1.i=5;
i++ or ++i
printf("%d",i);
ans:6
2.i=5
i—or--i
printf("%d",i)
ans:4
3.i=5
J=5+(++i);
Ans :j=11 i=6
4.i=5
J=5+i++;
Ans:j=10 i=6
5.i=5
J=5+(--i);

Ans:j=9 i=4

6.i=5

J=5+i--;

Ans:j=10 i=4

Program using increment and decrement operators

```
#include<stdio.h>
main()
{
        int a=10;
        printf("a++=%d\n",a++);
        printf("++a=%d \n",++a);
        printf("--a=%d\n",--a);
        printf("a--=%d\n",a--);
}
```

```
 Output:a++=10
++a=12
--a=11
a--=11
```

Explanation: In the above program the integer variable 'a' is initialized as value 10.the post increment variable 'a++' produced as output 10 because 'a' does not increased. The pre increment variable '++a' produces the output 12 because 'a' is increase by 1 before this the a value is 11.The predecrement variable '- -a 'produced the output 11 and variable 'a- -' produced the output 11 after this the a value is 10 respectively.

Example 2: #include<stdio.h>

```
#include<conio.h>
void main()
{
        int I,j,k;
        clrscr();
        i=3;
        j=4;
        k=i++ + - -j;
        printf("i=%d,j=%d,k=%d",i,j,k);
        getch();
}
```

```
Output:i=4,j=3,k=6
```

Note:Do not use increment and decrement on floating point variables.

<u>Conditional operator or ternary operator:</u>

Conditional operator itself checks the condition and executes the statement depending on the condition.

Syntax: condition? exp1:exp2;

Description: The?: operator acts as a ternary operator ,if the condition is true then the exp1 is evaluated ,if the condition is false then the exp2 is evaluated.

Example: #include<stdio.h>

void main()

```
{
        int a=5,b=3,big;
        big=a>b?a:b;
        printf("big is %d",big);
}
```

Output:big is 5

It checks the condition 'a>b' if it is true , then the value of 'a ' is stored in 'big' otherwise the value of 'b' is stored in 'big'.

<u>Comma operator:</u>

The comma operator is used to separate two or more expressions. The comma operator has the lowest priority among all the operators. It is not essential to enclose the expressions with comma operators within the parenthesis. For example, the following statements are valid.

Example:a=2,b=3,c=a+b;

Program:

```
void main()
{
        clrscr();
        printf("addition =%d\n subtraction=%d',2+3,5-4);
}
```

Output:

```
addition: 5
subtraction:1
```

<u>Size of () operator:</u> The operator sizeof() returns the byte size of data type or a variable. The general syntax for the operator sizeof() is:

Type 1: sizeof(data type);

Example:

```
int n;
n=sizeof(char);
```

here n gets the size of character type data.

Type 2:sizeof(variable name);

Example: int x;

```
float s;
x=sizeof(s);
```

here s is declared as float type and its size is obtained in integer x.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        printf("no of bytes occupied by various data types \n");
        printf("char=%d \n",sizeof(char));
        printf("int =%d \n",sizeof(int));
        printf("float =%d\n",sizeof(float));
        getch();
}
```

Output:

```
no of bytes occupied by various datataypes
char=1
int=2
float=4
```

<u>Expressions:</u> Expression is a combination of variables, constants and operators. An expression is evaluated using assignment operator.

Syntax : variable=expression;

Example : x=a+b+c;

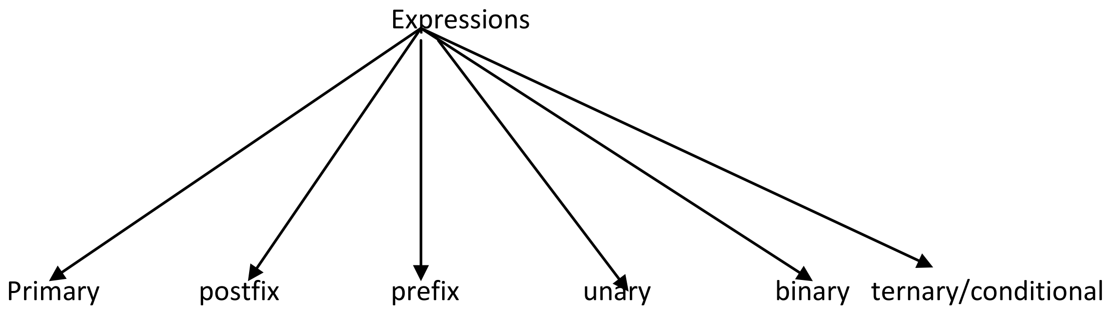
In the above statement the expression evaluated first from left to right. After the evaluation of the expression the final value is assigned to the variable from right to left.

Example: y=(a/b)+c;

Z=(a*b)-d;

Examples of algebraic expressions and C expressions

Algebraic expression	**C expression**
a+bxc	a+b*c
ax2+bx+c	a*x*x+b*x+c
[4ac/2a]	(4*a*c)/(2*a)
[2x2/b]-c	((2*x*x)/b)-c



Primary: Only one operand without any operators

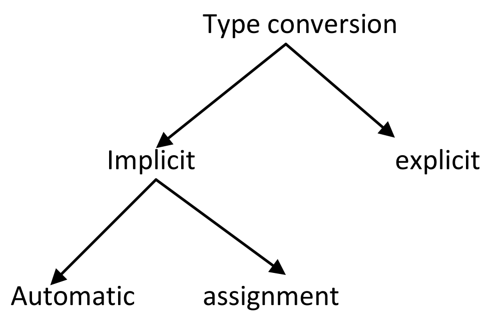
Postfix: Expression which contains one operand followed by an operator Ex.a++

Prefix: Expression which contains one operand which has operator before it Ex:++a

Unary: One operator and one operand .operator comes first. Ex.+a,-x;

Binary: Operand –operator-operand type of expression ex:10*3,a*b,3%2.

<u>Type conversion:</u> C supports the use of mixed mode operations in arithmetic expressions. Normally, before an operation takes place, both the operands must have the same type. C converts one or both the operands to the appropriate data types by type conversion.



Implicit type conversion: In case of implicit type conversion one data type is automatically converted into another data type. There are two types of implicit type conversion. They are automatic and assignment type conversion.
In **automatic type conversion** the data type having lower rank is converted automatically into higher rank data before the operation proceeds. The result is of the data type having higher rank as given in the following table.

Data type	**Rank**
long double	1
double	2
float	3
int	4
shortint	5
char	6

Example:
```
int x=5;
float y=6;
double z;
z=y-x;
```
In the above expression 'x' is first converted to float type before subtraction operation is performed over it. Finally the result so obtained is then converted to higher data type i.e. double and stored in variable 'z'.
Example 2:
```
#include<stdio.h>
#include<conio.h>
Void main()
{
        int a=10;
        float  b=15.5,c;
        c=a+b;/* here implicit type casting takes place */
        printf("C=%f",c);
        getch();
}
```
Output:

C=25.5
In the above example, expression a+b evaluated as
Step 1: a+b addition takes place first because a and b are int data types a+b produces int as result
Step 2: to perform assignment operation c is float data type and a+b is int data type. To perform this operation value of a+b automatically converted into float type and it Is assigned to c.

Example 2: suppose that i is an integer variable whose value is 9 ,f is a floating point variable whose value is 6.5 and c is a character type variable that represent the character 'w'. Several expressions which include the use of these variables are shown below.

Expression	**Value**	**Type**
i + f	15.5	float
i + c	128	integer
i +c –'o'	17	integer
(i+c)-(2*f/5)	125.4	float

Assignment Conversion: if the two operands in an assignment operation are of different data types, the right side operand automatically converted to the data type of the left side. For example, if the left operand is int type and the right operand is float type, any fractional part of the right side value is truncated and its integral part only is assigned. But, when a double type is assigned to a float type, the value may be required or truncated based on implementation.

Example: int x;
float y;
y=x;
In the above statement ,the type of 'x' is converted to type of 'y' automatically .i.e, float. But if the higher type value is assigned to the lower type value then the result so obtained is either truncated or there is loss in precision.

Example :

```
Example:
#include<stdio.h>
#include<conio.h>
Void main()
{
        int a=10,b=20;
        float  c;
        c=a+b;          /* here implicit type casting takes place */
        printf("C=%f",c);
        getch();
}
```

Output:
```
C=30.000000
```

Explanation:
In the above example, expression c= a+b evaluated as

Step 1: a+b addition takes place first because a and b are int data types a+b produces int as result
Step 2: to perform assignment operation c is float data type and a+b is int data type. To perform this operation value of a+b automatically converted into float type and it Is assigned to c.

Explicit type conversion:

Built in data types can be explicitly converted into desired data types. It is accomplished by using the cast operator. The general format of cast operation is given below.
(Data type)expression
Where expression is converted to the target data_type enclosed within the parentheses. The expression may be constant or a variable. The data_type must be any built_in type or void. If it is void, the operand must be other than the void type expression. The name of the data type to which the conversion ts to be made is enclosed in parentheses and placed directly to the left of the expression.

Example:

/*Without explicit type casting*/

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        int a=10,b=4;
        float  c;
        c=a/b;          /* here int/int evaluated as int and stores the result in float*/
        printf("C=%f",c);
        getch();
}
```

Output:

```
C=2.000000
```

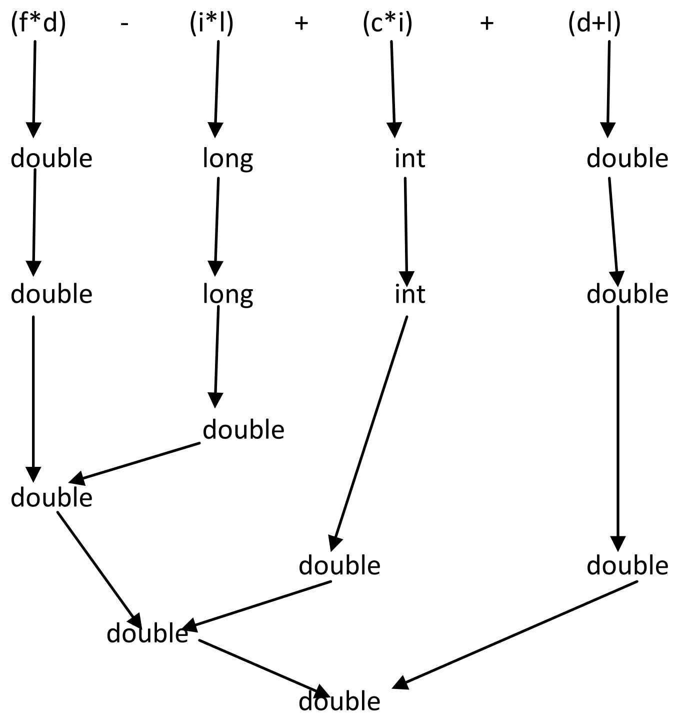
/*With explicit type casting*/

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        int a=10,b=4;
        float  c;
        c=(float)a/b;           /* here int/int evaluated as float and stores the result in
float*/
        printf("C=%f",c);
        getch();
}
```

Output:

```
C=2.500000
```

Type conversion in expression:
When constants and variables of different types are mixed in an expression they are converted to the same type. This conversion is done implicitly by C compiler. The C compiler will convert all operands to the type of the largest operand. For example if an expression contains an operation between an int and a float,the int would be automatically promoted to a float before carrying out the operation. The conversion rules are
1.All the chars and short ints are converted to int. All floats are converted to double.
2.For all operand pairs: If one of the operands is double the other operand is converted to double. If one of the operands is long, the other operand is converted to long. If one of the operand is unsigned the other is converted to unsigned. In an expression, all the types mixed , the rules must be applied in sequence: unsigned, long double.
Example: char c;
int I;
double d;
float f;
long int l;



<u>**Operator precedence and Associativity:**</u>
Precedence: it is one of the most important properties of the operator. Precedence refers to the priority given to the operator for a process. When an expression contains many operators the operations are carried out according to the priority of the operators. The higher priority operations are solved first.
For example, in an arithmetic operators, the operators *,/ and % are highest priority and of similar precedence. The operators + and – having lowest precedence.

Example:
8+9*2-10
The operator * is the highest priority. Hence, the multiplication operation is solved first.
=8+18-10
In the above expression + and – are having the same priority. In such situation ,the left most operation is solved first.
=26-10
=16
Associativity: When an expression has operators with equal precedence, the associativity property decides which operation to be carried out first. Associativity means the direction of execution. Associativity is of two types.
Left to right: In this type evaluation starts form the left to right direction.
Example: 12*4/8%2
In the above expression, all operators have the same precedence so the associativity rule is as follows. Multiplication is performed first, division, and finally modulus operation.
=48/8%2
=6%2
=0
Right to left: in this type, expression evaluation starts from right to left direction
Example: x=8+5%2
In the above expression, assignment operator has right to left associativity , hence the right hand side operator will be executed first(8+5%2) then the result is assigned to x.

Operators	**Operation**	**Associativity**	**Priority**
()	Function call	Left to right	1
[]	Array expression		
2	Structure operator		
.	Structure operator		
+	Unary plus		
-	Unary minus	Right to left	2
++	Increment		
--	Decrement		
!	Not operator		
~	Ones compliment		
*	Pointer operator		
&	Address operator		
size of	Size of an object	Right to left	2
type	Type case		
*	multiplication		
/	Division	Left to right	3
%	modulus		
+	Addition(binary plus)	Left to right	4
-	Subtraction(binary minus)		

<<	Left shift	Left to right	5
>>	Right shift		
<	Less than	Left to right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal		
==	equality	Left to right	7
!=	Not equal to		
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
||	Logical OR	Left to right	12
?:	Conditional operator	Right to left	13
=,*=,/=,%=,+=,-=,&=,^=,|=,<<=,>>=	Assignment operators	Right to left	14
,	Comma operator	Right to left	15

Review Questions:

1. Why to use computer? Brief its benefits.
2. What are functions of interpreter and compiler?
3. What is the difference between compiler and interpreter?
4. Why the c language called middle level language?
5. Explain the block diagram of a computer with a neat sketch?
6. List and explain various input devices and output devices of a computer?
7. What is computer software? Explain types of software?
8. Elaborate different sections of a C program?
9. Explain the various languages of a computer?
10. Write the advantages of C language?
11. What is an algorithm? Write various criteria used for judging an algorithm?
12. What is a flow chart? Explain different symbols used in flowchart?
13. What is the use of flowchart?
14. What are the different steps involved in program development?
15. What are different data types? Explain type conversion with example programs?
16. Explain different types of constants in C?
17. Explain the methods for initialization of variables?
18. Write about the space requirement for variables of different data types?
19. List any four keywords with their use?
20. What is a variable? Explain the rules for variable naming?
21. Explain different types of operators supported by C with example?
22. What is the difference between "=" and "=="?
23. What are the difference between precedence and associativity?

Selsction making decision: two way selection: if-else,null else,nested if ,examples,multiway selection: switch,else if examples
ITERATIVE: loops- while, do-while and for statements , break, continue, initialization and updating, event and counter controlled loops, Looping applications: Summation, powers, smallest and largest.
Arrays : concepts,declarations,definitions,accessing elements,storing elements, string and string manipulations, 1-D arrays, 2-D arrays and character arrays, string manipulations,multi-dimensional arrays,array applications : Matrix operations, checking the symmetricity of a matrix.
Strings: concepts, C strings

SELECTION –MAKING DECESION:-

SELECTION CONTROL STATEMENTS:- Based on the decision it will executes the statements, these are two types:-

1. Two way selection (example :if)
2. Multi way selection (example :else if ladder, switch)

1).TWO-WAY SELECTION:- Selection allows us to choose b/w two (or) more alternatives. It allows us to make decisions. The basic decision statement in the computer is two-way selection.

The decision is described to the computer as a conditional stmt that can be either true / false. If the condition is false, then a different action or set of actions is executed. Regardless of which set of actions is executed, program centimes with the next stmt.

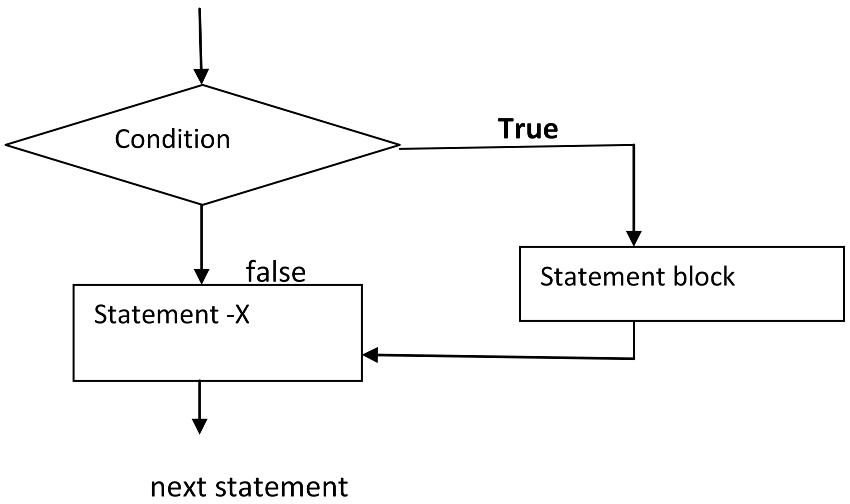
IF STATEMENT:-Generally , the control in a program will be in sequential order. i.e., it execute the statements in top-down approach.

'if' is used control the sequence of statements based on a given condition. 'if' can be used in the following forms.

(i) Simple if:- This 'if' statements contains true part. i.e; Whenever the condition is true, then only the statement(or) statements will get executed.

Syntax:
```
if (condition)
{
        Stmt block;
}
Stmt-X;
```

FlowChart



Ex:- 1. if(a>b)
```
printf("a is big");
2. if (x==10)
{
        x=x+10;
        printf("%d,", x)
}
```
Examples:

Write a program to check whether the entered number is less than 10 if yes display the same?
```
#include<stdio.h>
#include<conio.h>
void main()
{
        int v;
        clrscr();
        printf("enter the number");
        scanf("%d",&v);
        if(v<10)
        printf("\n number entered is less than 10");
        getch();
}
```
Output:
```
Enter the number 9
Number entered is less than 10
```

Write a program to check equivalence of two numbers
```
#include<stdio.h>
#include<conio.h>
void main()
{
        int m,n;
        clrscr();
        printf("enter two numbers");
        scanf("%d%d",&m,&n)
        if(m-n==0)
        printf("\n two numbers are equal");
        getch();
}
```
Output:
```
    enter two numbers 5 5
    two numbers are equal
```

Write a program to check whether the candidate age is greater than 17 or not .if yes display the message “eligible for voting”

```
void main()
{
        int age;
        clrscr();
        printf(“\n enter age”);
        scanf(“%d”,age);
        if(age>17)
        prinft(“eligible for voting”);
        getch();
}
```

Output:

```
        enter age 18
        eligible for voting
```

Write a program to find whether the given number is positive or negative

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n;
        clrscr();
        printf(“enter a number\n”);
        scanf(“%d”,&n);
        if(n>0)
        printf(“the number is positive \n”);
        getch();
}
```

<u>Output:</u>

```
enter a number 10
the number is positive
enter a number -8
```

Write a program for comparision of two numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n1,n2;
        clrscr();
        printf(“enter two numbers”);
        scanf(“%d%d”,&n1,&n2);
        if(n1>n2)
        printf(“%d is greater than %d”,n1,n2);
        if(n1<n2)
        printf(“%d is less than %d”,n1,n2);
        if(n1==n2)
        printf("%d is equal to %d",n1,n2);
        getch();
}
```

Output:

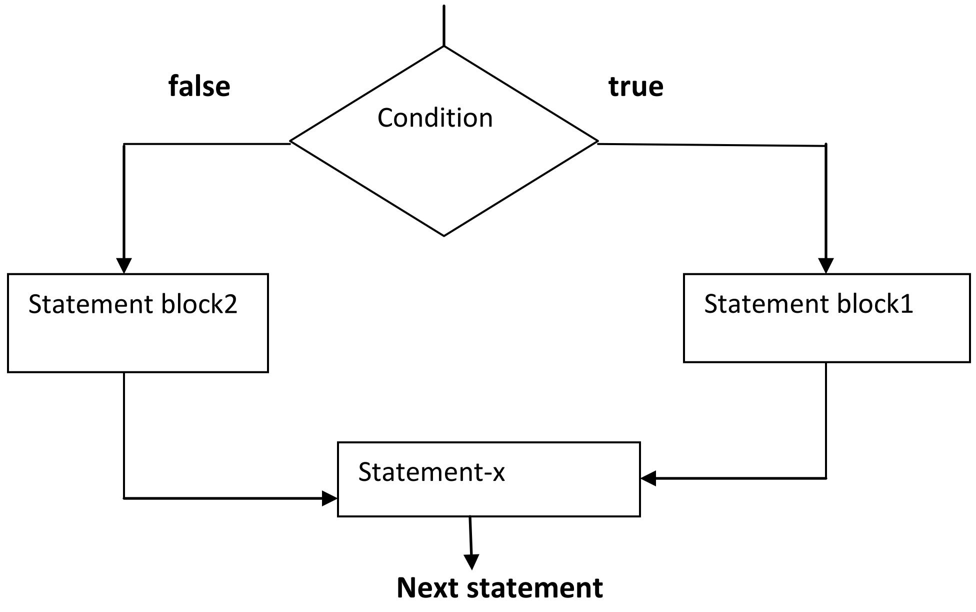
```
        Enter two numbers 10 12
        10 is less than 12
        Enter two numbers 15 11
        15 is greater than 11
```

IF – ELSE STATEMENT:- C implements two-way selection with the if-else statement. An if –else statement is a composite statement used to make a decision b/w two alternatives. if –else structure contains two parts. One is true part & other is the false part but only of these two parts is executed based on the condition tested.

SYNTAX:-

```
if (test condition)
{
        Stmt block- 1;          /* true block*/
}
else
{
        Stmt bock- 2;          /* false block*/
}
Stmt-X;
```

Flowchart:



Ex:-

```
if (a>b)
printf("a is big");
else
```

printf ("b is big");

→ In the above syntax, stmt block 1 will be executed if the condition becomes true. Otherwise stmt block 2 will be executed. Stmt .X will be executed in both cases.

NOTE:- While writing if –else stmt

(i) The expression must be enclosed in paranthesis.
(ii) No semicolon (;) is needed for an if –else stmt:
(iii) The expression can have side effects.
(iv) Both the true & false stmt can be any stmt (even another if-else stmt) or they can be a null stmt.

Examples:

Read the values of a,b,c through the keyboard. Add them and after addition check if it is range of 100 and 200 or not. Print the sepearte message for each.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b,c,d;
        clrscr();
        printf("enter three numbers");
        scanf("%d%d%d",&a,&b,&c);
        d=a+b+c;
        if(d<=200 && d>=100)
        printf("sum is in between 100 and 200");
        else
        printf("out of range");
        getch();
}
```

Output:

```
        Enter three numbers 50 52 54
        Sum is in between 100 and 200
```

Write a program to find the roots of a quadratic equation by using if else condition

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int b,a,c;
        float x1,x2;
        clrscr();
        printf("enter values for a,b,c");
        scanf("%d %d%d",&a,&b,&c);
        if(b*b>4*a*c)
        {
                x1=-b+sqrt(b*b-4*a*c)/2*a;
                x2=-b-sqrt(b*b-4*a*c)/2*a;
                printf("\n x1=%f x2=%f",x1,x2);
        }
        else
                printf("roots are imaginary");
        getch();
}
```

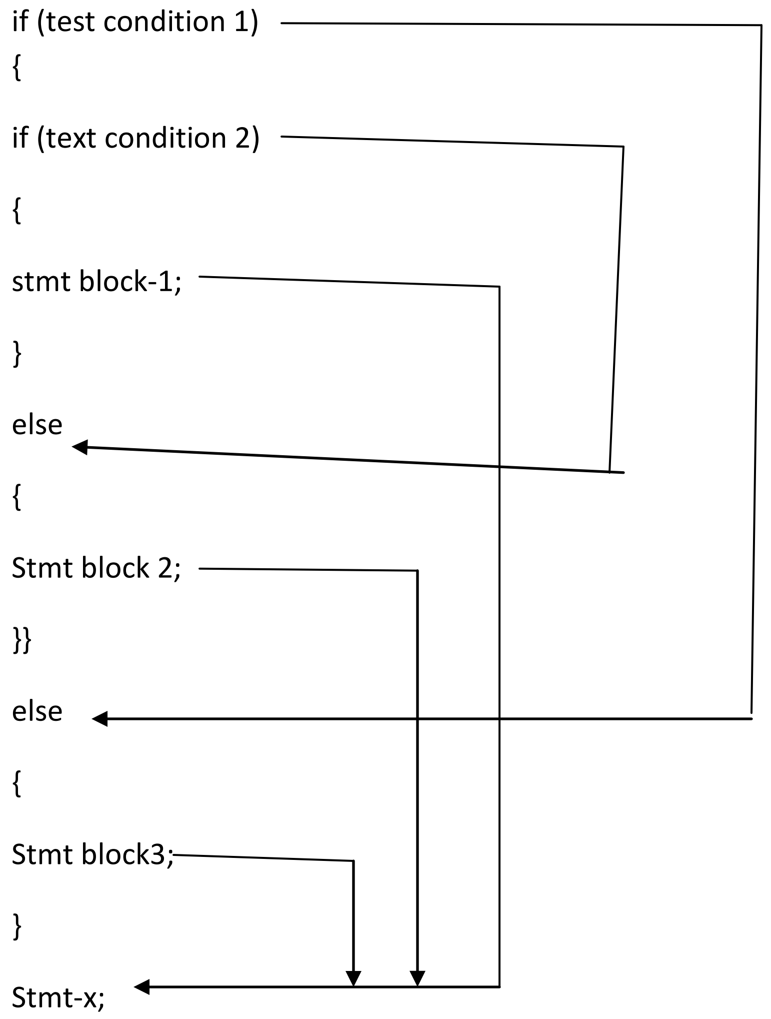
Output:
```
        Enter values for a b c 5 1 5
        Roots are imaginary
```
Write a program to find whether the given number is even or odd
```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n;
        clrscr();
        printf(“enter a number to check);
        scanf(“%d”,&n);
        if(n%2==0)
                printf(“%d is an even number “,n);
        else
                printf(“%d is an odd number”,n);
        getch();
}
```
Output:
```
        Enter a number to check 7
        7 is an odd number
        Enter a number to check 4
        4 is an even number
```
Write a program to find the biggest among two numbers
```
#include<stdio.h>
#include<conio.h>
Void main()
{
        int x,y,big;
        Clrscr();
        Printf(“enter two numbers”);
        scanf(“%d%d”,&x,&y);
        if(x>y)
        big=x;
        else
        big=y;
        printf(“biggest number %d”,big);
        getch();
}
```
Output:
```
        Enter two numbers 5 6
        Biggest number 6
```
Write a program to check whether a given character is vowel or not.
```
#include<stdio.h>
#include<conio.h>
void main()
{
        char ch;
        clrscr();
        printf(“enter any character”);
        scanf(%c”,&ch);
        if(ch==’a’||ch==’e’||ch==’I’||ch==’o’||ch==’u’||ch==’A’||ch==’E’||ch==’I’||ch==’O
        ’||ch==’U’)
        printf(“ you entered vowel”);
        else
        printf(“you didn’t entered vowel”);
        getch();
}
```

Output:

```
        Enter any character e
        You entered vowel
```

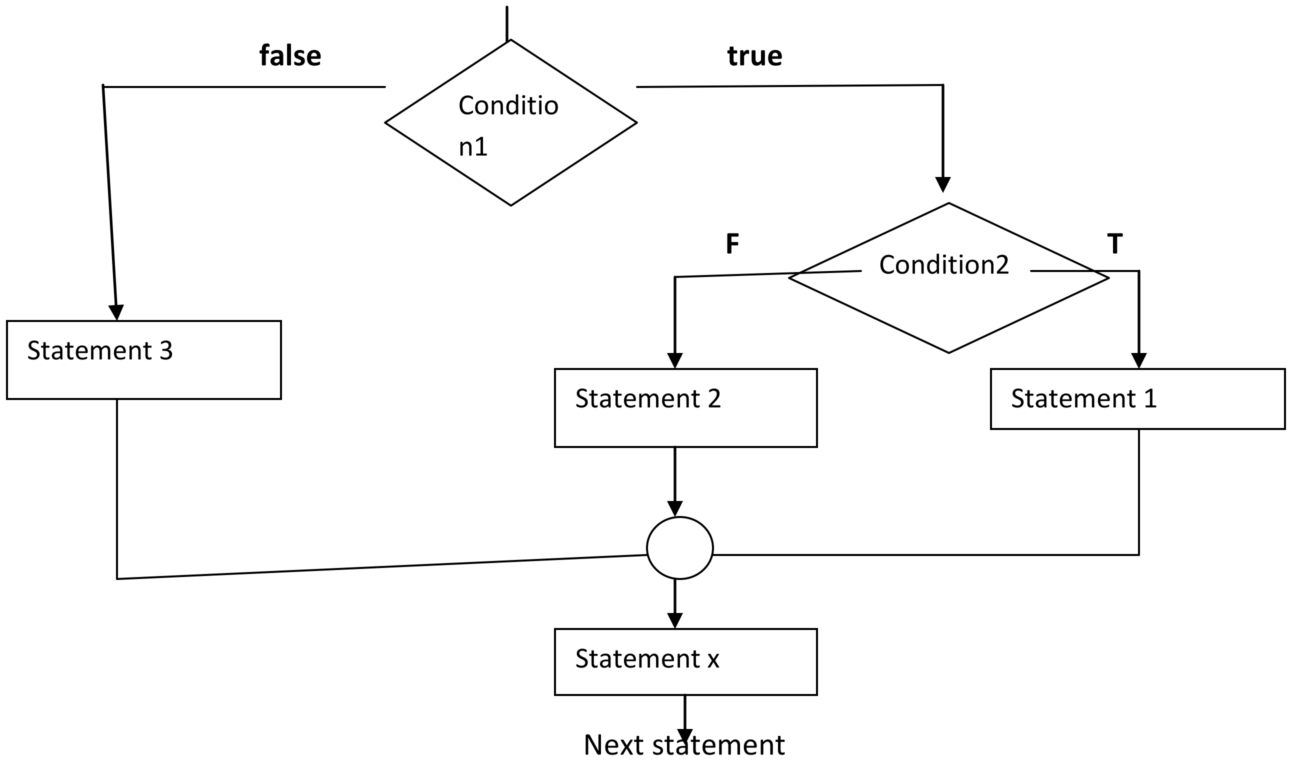
<u>NESTED IF-ELSE</u>:- Whenever a series of decisions are to be made, more than one if-else students are to be used within one another known as nested if-else.

<u>SYNTAX:-</u>



```
if (test condition 1)
{
if (text condition 2)
{
stmt block-1;
}
else
{
Stmt block 2;
}}
else
{
Stmt block3;
}
Stmt-x;
```

Flowchart:



Ex:

```
if(a>b)
{
        if(a>c)
                printf("a is big");
        else
                printf("c is big");
}
else if (b>c)
{
        printf("b is big");
}
else
printf("c is big");
```

Write a program to find the maximum of three numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
        Iint x,y,z,max;
        clrscr();
        printf("enter three numbers");
        scanf("%d%d%d",&x,&y,&z);
        if(x>y)
        {
                if(x>z)
                        max=x;
                else
                        max=z;
        }
        else
        {
                if(y>z)
                        max=y;
                else
                        max=z;
        }
        printf("maximum =%d ",max);
        getch();
}
```

Output:

```
        Enter three numbers 1 2 3
        Maximum=3
```

Write a program for comparison of two numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x,y;
        clrscr();
        printf("enter two numbers");
        scanf("%d%d",&x,&y);
        if(x>=y)
                if(x>y)
                        printf("%d is greater than %d",x,y);
                else
                        printf("%d is equal to %d",x,y);
        else
                printf("%d is less than %d",x,y);
}
```

Output:

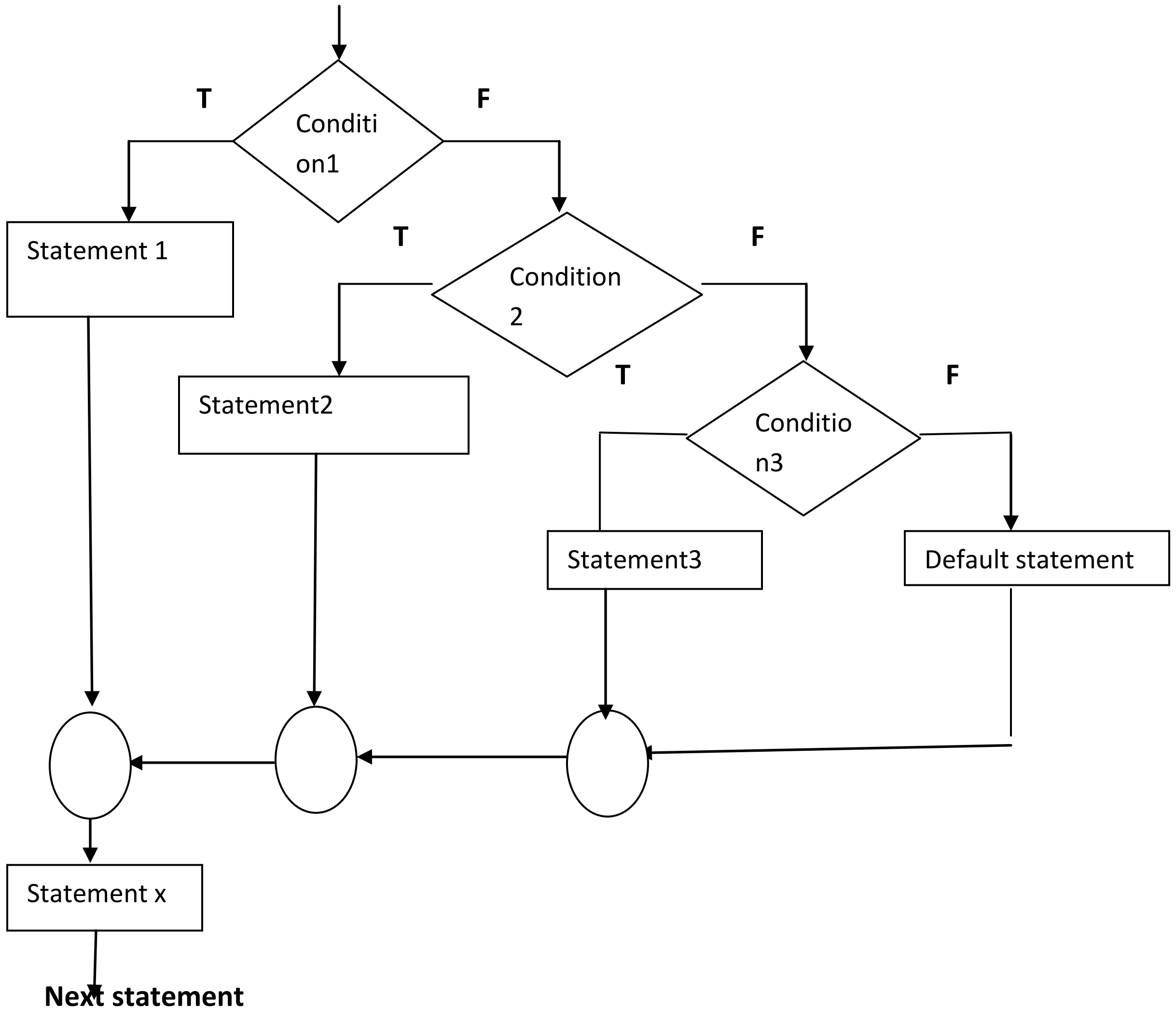
```
        Enter two numbers 10 20
        10 is less than 20
        Enter two numbers 50 40
        50 is greater than 40
```

<u>Multi way selection:</u>
In addition to 2-way selection most programming languages provide another selection concept known as **Multiway Selection.** Multiway Selection chooses among several altermatives.

<u>ELSE-IF LADDER</u>:- Whenever a condition has to be tested based on the falsity of the above condition, else-if ladder can be used.
<u>SYNTAX</u>:-
```
if (test condition1)
{
        Stmt block -1;
}
else if (test condition2)
{
        Stmt block-2;
}
else if (test condition 3)
{
        Stmt block-3;
}
else
{
        Default stmt;
}
Stmt – X;
```

Here test condition 1 ,test condition 2,test condition 3-------------test condition n are mutually exclusive. That is ,only one test condition of all will be true. There are n+1 blocks of statements.
Initially test condition1 is checked if it is true block of statements would get executed,all the other block of statements would be skipped. If test condition 1 is false ,test condition 2 is checked if it is true the block of statements would get executed all other statements would be skipped. If none of the expressions is found to be true the last block of statements would get executed.

Flowchart:



Next statement

Ex:-

```
If (a>b&&a>c&&a>b)
     Printf("a is big");
Else if (b>c&&b>d)
     Printf("b is big");
Else if (c>d)
     Printf ("c is big");
Else
     Printf("d is big");
```

Write a program to find the maximum of three numbers using else-if ladder

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

```
void main()
{
        int x,y,z,max;
        clrscr();
        printf("enter three numbers");
        scanf("%d%d%d",&x,&y,&z);
        if((x>y)&&(x>z))
                max=x;
        else if((y>x)&&(y>z))
                max=y;
        else
                max=z;
        printf("maximum =%d \n",max);
        getch();
}
```

Output:

```
        Enter three numbers
        1 2 3
        Maximum=3
```

Write a program to calculate area of square/rectangle/circle/triangle depending upon user choice.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int choice;
        float area,a,b,c,s;
        printf("main menu \n");
        printf("1. area of square\n");
        printf("2.area of rectangle\n");
        printf("3.area of circle\n");
        printf("4.area of triangle\n");
        printf("enter your choice\n");
        scanf("%d",&choice);
        if(choice==1)
        {
                printf("enter the side of square");
                scanf("%f",&a);
                area=a*a;
                printf("area of square is %f",area);
        }
        else if(choice==2)
        {
                printf("enter the length and breadth of rectangle");
                scanf("%f%f",&a,&b);
```

```
            area=a*b;
            printf("area of rectangle is %f",area);
        }
        else if(choice==3)
        {
            printf("enter the radius of circle");
            scanf("%f",&a);
            area=3.14*a*a;
            printf("area of circle is %f",area);
        }
        else if(choice==4)
        {
            printf('enter three sides of triangle");
            scanf("%f%f%f",&a,&b,&c);
            s=(a+b+c)/2;
            area=sqrt(s*(s-a)*(s-b)*(s-c));
            printf("area of triangle is %f",area);
        }
        else
            printf("wrong choice\n");
        getch();
}
```

Output:

```
        main menu
        1.area of square
        2.area of rectangle
        3.area of circle
        4.area of triangle
        enter your choice 2
        enter the length and breadth of rectangle 4 6
        area of rectangle is 24
```

Write a program to sort six numbers and find the largest one by using else if ladder

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b,c,d,e,f;
        clrscr();
        printf("enter numbers");
        scanf("%d%d%d%d%d%d',&a,&b,&c,&d,&e,&f);
        if((a>b)&&(a>c)&&(a>d)&&(a>e)&&(a>f))
            printf("highest of six numbers is %d",a);
        else if((b>c)&&(b>d)&&(b>e)&&(b>f)
            printf("highest of six numbers is %d",b);
        else if((c>d)&&(c>e)&&(c>f))
```

```
            printf("highest of six numbers is %d',c);
        else if((d>e)&&(d>f))
            printf("highest of six numbers is %d",d);
        else if(e>f)
            printf('highest of six numbers is %d",e);
        else
            printf("highest of six numbers is %d",f);
        getch();
}
```

Output:

```
        enter numbers 12 13 14 15 16 17
        highest of six numbers is 17
```

Write a program that reads marks in three subjects ,calculate average marks and assigns grade as per following specifications

If marks	Then grade
>=90	A
75-90	B
60-75	C
50-60	D
<50	Fail

```
#include<stdio.h>
#include<conio.h>
void main()
{
        float m1,m2,m3,average;
        clrscr();
        printf("enter the marks in three subjects");
        scanf("%f%f%f',&m1,&m2,&m3);
        average=(m1+m2+m3)/3;
        printf("average marks are %f",average);
        if(average>=90)
            printf("grade is A");
        else if(average>75 &&average<90)
            printf("grade is B");
        else if(average>=60 &&average<75)
            printf("grade is C");
        else if(average>=50 &&average<60)
            printf("grade is D");
        else
            printf("fail");
        getch();
}
```

Output:

```
        Enter the marks in three subjects 90 70 82
        Average marks are 80.6666
        Grade is B
```
<u>Switch statement:</u>

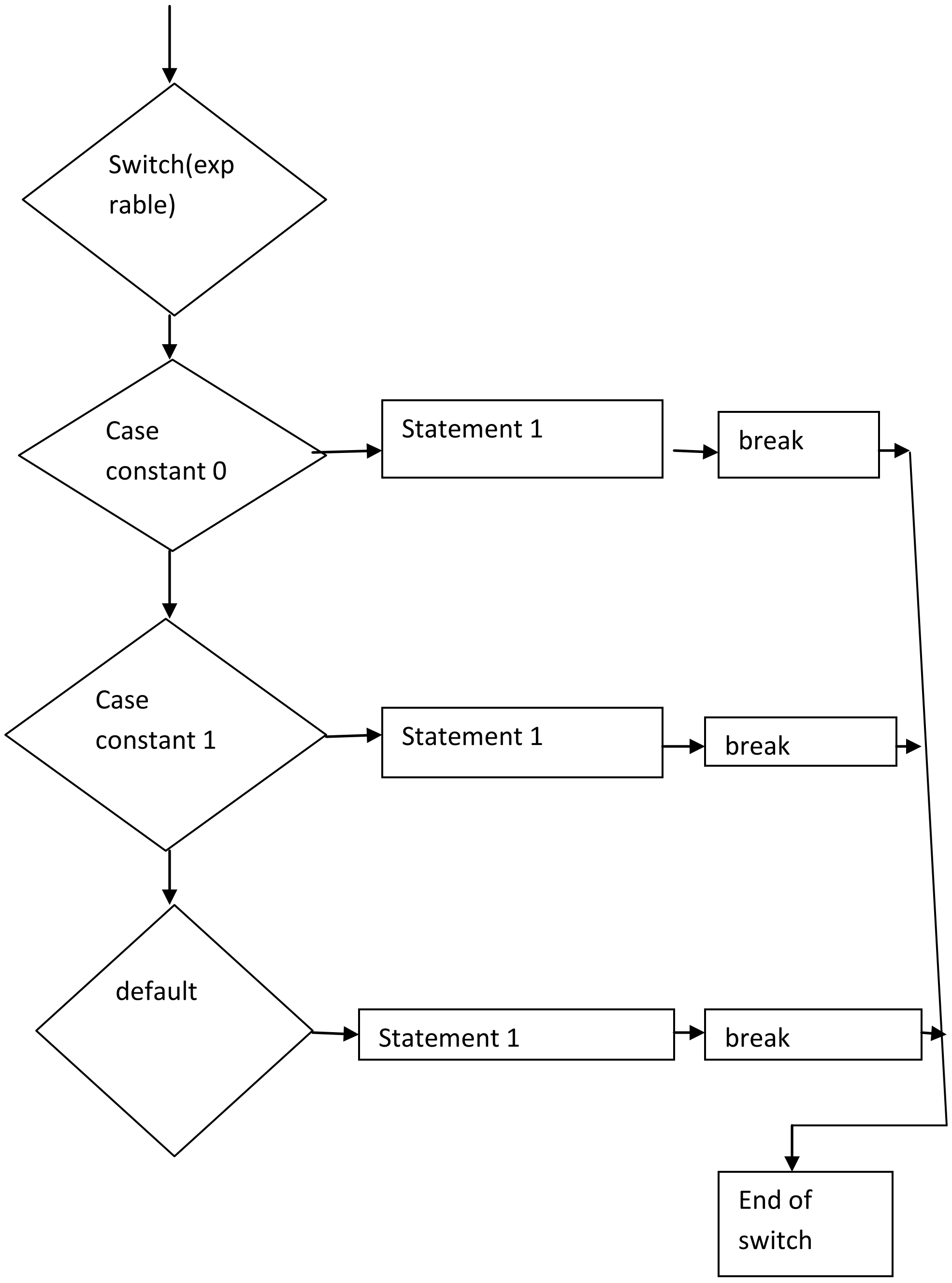
The switch statement is a multi way branch statement. In the program if there is a possibility to make a choice from a number of options, this structured selection is useful. The switch statement requires only one argument of any data type,which is checked with the number of case options .The switch statement evaluates expression and then looks for its value among the case constants. If the value matches with case constant, this particular case statement is executed. If not default is executed. Here switch,case,default are reserved keywords. Every case statements terminates with ':'. The break statement is used to exit from current case structure. The switch statement is useful for writing the menu driven program.

The syntax of the switch case statement is as follows:

```
switch(variable or expression)
{
        case constant a: statement;
        break;
        aase constant b:statement;
        break;
        default: statement;
}
```
The switch expression: in the block ,the variable can be a character or an integer. The integer expression following the keyword switch will yield an integer value only. The integer may be any value 1,2,3 and so on. In case a character constant ,the values may be given in the alphabetic order such 'x','y','z'.

Switch organization: The switch expression should not be terminated with a semicolon and or with any other symbol. The entire case structure following switch() should be enclosed with curly braces. The keyword case is followed by a constant. Every constant terminates with a colon. Each case statement must contain different constant values. Any number of case statements can be provided. If the case structure contains multiple statements, they need not be enclosed with curly braces. Here the keywords case and break perform the job of opening and closing curly braces respectively

Switch execution: When one of the cases satisfies, the statements following it are executed. In case there is no match the default case is executed. The default can be put anywhere in switch() expression. The switch() statement can also be written without the default statement. The break statement used in switch causes the control to go outside the switch block. By mistake if no break statements are given all the cases following it are executed.



Write a program to provide multiple functions such as 1.addition 2.subtraction 3.multiplication 4.division 5.remainder using switch statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b,c,ch;
        clrscr();
        printf("menu");
        printf("1.addition");
        printf("2.subtraction");
        printf("3.multiplication");
        printf("4.division");
        printf("5.remainder");
        printf("6.exit");
        printf("enter your choice");
        scanf("%d",&ch);
        if(ch<6&&ch>=1)
        {
                printf("enter two numbers");
                scanf("%d%d",&a,&b);
        }
        switch(ch)
        {
                case 1: c=a+b;
                        printf("addition %d',c);
                        break;
                case 2:c=a-b;
                        printf("subtraction %d",c);
                        break;
                case 3: c=a*b;
                        printf("multiplication %d",c);
                        break;
                case 4:c=a/b;
                        printf("division %d",c);
                        break;
                case 5:c=a%b;
                        printf("remainder %d',c);
                        break;
                case 6:exit();
                        break;
                default:printf("invalid choice");
        }
}
```

Write a program to read a day number and print corresponding day of the week

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int num;
        clrscr();
        printf("enter any number in between 1 to 7");
        scanf("%d",&num);
        switch(num)
        {
                case 1:printf("it is Sunday");
```

```
                        break;
                case 2:printf("it is Monday");
                        break;
                case 3: printf("it is Tuesday");
                        break;
                case 4: printf( "it is wedensday");
                        break;
                case 5: printf(" it is Thursday");
                        break;
                case 6: printf("it is Friday");
                        break;
                case 7:printf("it is Saturday");
                        break;
                default: print("wrong number ");
        }
        getch();
}
```

output:

```
        enter any number in between 1 to 7 3
        It is Tuesday.
```

Switch statement with out break statement

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x;
        printf("enter any value");
        scanf("%d",&x);
        switch(x)
        {
                case 1: printf("tenali");
                case 2:printf("vizag");
                case 3:printf("Hyderabad");
                        break;
        }
        getch()
}
```

Output:

```
        enter any value 1
        tenali
        vizag
        Hyderabad
```

Nested switch case:

The c supports nested switch statements. The inner switch can be a part of an outer switch. The inner and outer switch case constants may be the same. No conflicts arise even if they are the same. A few examples are given below on the basis of nested switch statements.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x,y;
        clrscr();
        printf("enter a number");
        scanf("%d",x);
        switch(x)
        {
                case 0: printf("the number is even");
                        break;
                case 1: printf("the number is odd");
                        break;
                default:
                        y=x%2;
                switch(y)
                {
                        case 0: printf("number is even");
                                break;
                        default: printf("number is odd");
                }
        }
getch();
}
```

Output:

```
        Enter a number 5
        Number is odd
```

Explanation: in the above given program the first switch statement is used for displaying the message such as even or odd numbers when the entered numbers are 0 and 1 respectively .When the entered number is other than 0 and 1 its remainder is calculated with modulus operator and stored in the variable 'y' is used in the inner switch statement. If the remainder is 0 the message displayed will be number is even otherwise for non zero it will be number is odd.

Example programs:

1.Calculates the area of circle or rectangle.

```
#define PI 3.14
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
        float  n1,n2;
        char ch;
        clrscr();
        printf("press C or R for the area of circle or rectangle respectively");
        scanf("%c',&ch);
        if(ch=='C'||ch=='c')
        {
                printf("enter radius");
                scanf("%f",&n1);
                printf("area of circle is %f',PI*n1*n2);
        }
        if(ch=='R'||ch=='r')
        {
                printf("enter length and width");
                scanf("%f%f",n1,n2);
                printf("area of rectangle is %f",n1*n2);
        }
}
```

Output:

```
        press  C or R for the area of circle or rectangle respectively C
        Enter radius 5
        Area of circle 78.539750
```

2.Program that accepts sales amount and then calculates discount as 10% of sales amount if sales amount is more than 5000 otherwise as 5% of sales amount.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        float sales,discount;
        printf("enter the sales");
        scanf("%f',&sales);
        if(sales>5000)
                discount=.10*sales;
        else
                discount=.05*sales;
        printf("discount is %f",discount);
        getch();
}
```

Output:

```
        Enter the sales 8000
        Discount is 800
```

3.Program to display profit or loss after obtaining cost and selling prices.

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

```
#include<conio.h>
void main()
{
        float sp,cp,lp;
        printf("enter the cost price");
        scanf("%f",&cp);
        printf("enter the selling price");
        scanf("%f',&sp);
        lp=sp-cp;
        if(lp==0)
                printf("no loss and profit");
        else if(lp>0)
                printf("profit has occurred amd amount of profit is %f",lp);
        else
                printf("loss has occurred and amount of loss is %f",abs(lp));
        getch();
}
```

Output:

```
        enter the cost price 90
        enter the selling price 100
        profit has occurred and amount of profit is 10
```

4.Program for temperature conversion as per user choice

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int choice;
        float tempf,tempc;
        clrscr();
        printf("temperature conversion menu\n");
        printf("1. Faherenheit to Celsius \n");
        printf("2. Celsius to faherenheit \n");
        printf("enter your choce");
        scanf("%d',&choice);
        if(choice==1)
        {
                printf("enter the temperature in Fahrenheit");
                scanf("%f",&tempf);
                tempc=(tempf-32)/1.8;
                printf("temperature in Celsius is %f",tempc);
        }
        else if(choice==2)
        {
                printf("enter the temperature in Celsius");
                scanf("%f",&tempc);
                tempf=(tempc*1.8)+32;
                printf("temperature in Fahrenheit %f",tempf);
        }
        else
                printf("wrong choce");
        getch();
}
```
Output:
```
        Temperature conversion menu
        1.fahrenheit to Celsius
        2.celsius to Fahrenheit
        Enter your choice 2
        Enter the temperature in Celsius 37
        Temperature in Fahrenheit 98.59999
```
5.Program to check whether a given year is leap year or not
```
#include<stdio.h>
#include<conio.h>
void main()
{
        int year;
        clrscr();
        printf("enter a year");
        scanf("%d',&year);
        if(year %100==0)
        if(year%400==0)
        printf("it's a centuary leap year");
        else
                printf("it's a centuary year but not a leap year");
        else if(year %4==0)
                printf("leap year");
        else
                printf("not leap year");
        getch();
}
```
Output:
```
        enter a year 2005
        not leap year
```
6.Program to check whether a character is an uppercase or lowercase alphabet or a digit or a special symbol
```
#include<stdio.h>
#include<conio.h>
void main()
{
        char ch;
        clrscr();
        printf("enter any character");
        scanf("%c",&ch);
        if(ch>='A'&&ch<='Z')
        printf("you entered an uppercase alphabet");
        else if(ch>='a'&&ch<='z')
                printf("you entered a lowercase character");
        else if(ch>='0' &&ch<='9')
                printf("you entered a digit");
        else
                printf("you entered a special character");
        getch();
}
```

Output:

```
        enter any character y
        you entered a lowercase character
```

7. Write a program to calculate the square of those numbers only whose least significant digit is 5

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int s,d;
        clrscr();
        printf("enter a number");
        scanf("%d",%s);
        d=s%10;
        if(d==5)
        {
                s=s/10;
                printf("square=%d%d",s*s++,d*d);
        }
        else
                printf("invalid number");
}
```

Output:

```
        enter a number 25
        square=625
```

<u>Assignment programs:</u>

1.Write a program to calculate energy bill. Read the starting and ending meter reading . The charges are as follows

No of units	consumed rates
200-500	3.50
100-200	2.50
Less than 100	1.50

2.Write a program to find average of six subjects and display the results as follows

Average	result
>34 & <50	third division
>49 &<60	second division
>=60 & <75	first division
>=75 &<=100	distinction

If marks in any subject less than 35 fail.
3.Write a program to check whether the blood donor is eligible or not for donating blood. The conditions laid down are as under . Use if statement
1. age should be greater than 18 years but not more than 55 years
2. weight should be more than 45 kg
4.Write a program to enter a character through keyboard. Use switch() case structure and print appropriate message. Recognize the entered character whether it is vowel,consonants or symbol.
5. Write a program to check whether the voter is eligible for voting or not. If his/her age is equal to or greater than 18 display message 'eligible' otherwise 'noneligible'.
6.Write a program to convert decimal to hexadecimal number.
7.Write a program to calculate the gross salary for the condition given below.

Basic salary	da	hra	conveyance
<=5000	110%of basic	20% of basic	500
>=3000 &&<5000	100% of basic	15% of basic	400
<3000	90% of basic	10 % of basic	300

Review questions:

1. What is control structure? Explain various program control structures with examples.
2. Explain if statement with example program
3. Explain about if else and nested if else statements with example programs
4. Explain about else if ladder with simple program
5. Explain about switch statement with program
6. Explain about nested switch statement with example program.

****

ITERATIVE LOOPS

Loops are used to repeat a block of code. Being able to have your program repeatedly execute a block of code is one of the most basic but useful tasks in programming.

There are three basic types of loops which are:

- "while loop"
- "do while loop"
- "for loop"

While loop:

Syntax:

```
while (test expression)
{
        Single statement;
        or
        Block of statements;
}
```

Description:

In the beginning of while loop, test expression is checked. If it is true, codes inside the body of while loop,i.e, code/s inside parentheses are executed and again the test expression is checked and process continues until the test expression becomes false.

Flow chart:

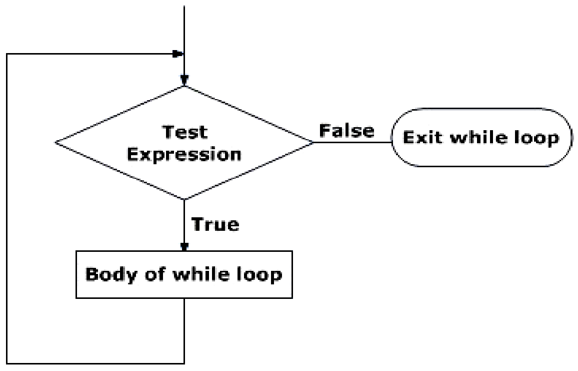


Figure: Flowchart of while loop

Example:

```
/* C program to print first 10  natural numbers*/
#include<stdio.h>
#include<conio.h>
Void main()
{
        int a=0;
        clrscr();
        while(a<10)
        {
                Printf("%d\t",a);
                a++;
        }
}
```
Output:

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

Example:

```
/* C Program to find factorial of a number using while loop*/
#include<stdio.h>
#include<conio.h>
 void main()
 {
         int n,f=1;
         printf("Enter the number:\n");
         scanf("%d",&n);
         while(n>0)
         {
                 printf("%d",n);
                 f=f*n;
                 n--;
         }
        printf("The factorial of the integer is:%d",f);
        getch();
 }
```
Output:

```
        Enter the number:
        5
        The factorial of the integer is:
        120
```
<u>do...while loop:</u>

In C, do...while loop is very similar to while loop. Only difference between these two loops is that, in while loops, test expression is checked at first but, in do...while loop code is executed at first then the condition is checked. So, the code are executed at least once in do...while loops.

Syntax:

```
                do
                {
                      Single statement;
                       or
                       Block of statements;
                }
                while (test expression);
```

Description:

At first codes inside body of do is executed. Then, the test expression is checked. If it is true, code/s inside body of do are executed again and the process continues until test expression becomes false(zero).

Notice, there is semicolon in the end of while (); in do...while loop.

Flow chart:

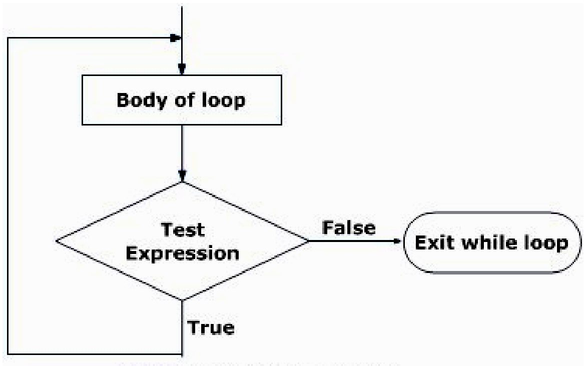


Figure: Flowchart of do...while loop

Example

```
/* C program to print first 10  natural numbers*/
#include<stdio.h>
#include<conio.h>
Void main()
{
        int a=0;
        clrscr();
        do
        {
                Printf("%d\t",a);
                a++;
        } while(a<10);
}
```

Output:

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

Example 2 :

```
/* C Program to find factorial of a number using while loop*/
#include<stdio.h>
#include<conio.h>
 void main()
 {
         int n,f=1;
         printf("Enter the number:\n");
         scanf("%d",&n);
         do
```

```
           {
                  printf("%d",n);
                  f=f*n;
                  n--;
           } while(n>0);
          printf("The factorial of the integer is:%d",f);
          getch();
   }
```

Output:

```
Enter the number:
5
The factorial of the integer is:
120
```

<u>For loop</u>

Syntax:

```
for(initial expression; test expression; update expression)
{
              Single statement;
                 or
                 Block of statements;
}
```

Description:

The initial expression is initialized only once at the beginning of the for loop. Then, the test expression is checked by the program. If the test expression is false, for loop is terminated. But, if test expression is true then, the codes are executed and update expression is updated. Again, the test expression is checked. If it is false, loop is terminated and if it is true, the same process repeats until test expression is false.
This flowchart describes the working of for loop in C programming.

Flow chart:

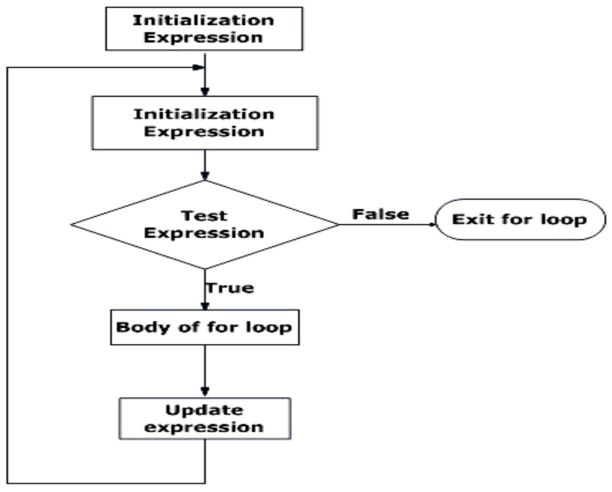


Figure: Flowchart of for loop

```
/* C Program to find factorial of a number using for loop*/
#include<stdio.h>
#include<conio.h>
 void main()
 {
         int n,i,f=1;
         printf("Enter the number:\n");
         scanf("%d",&n);
         for (i=1;i<=n;i++)
         {
                  f=f*i;
          }
        printf("The factorial of the integer is:%d",f);
        getch();
 }
```

Output:

```
        Enter the number:
        5
        The factorial of the integer is:
        120
```

Example

```
/* C program to print first 10  natural numbers using for loop*/
#include<stdio.h>
#include<conio.h>
Void main()
{
        int i;
        clrscr();
        for(i=0;i<=10;i++)
        {
                Printf("%d\t",a);
        }
}
```

Output:

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

Example

```
/* C program to print even numbers between 1 and n*/
#include<stdio.h>
#include<conio.h>
Void main()
{
        int i,n;
        clrscr();
        printf("Enter the number:\n");
          scanf("%d",&n);
        for(i=1;i<=10;i++)
        {
                If(i%2==0)
                        Printf("%d\t",i);
        }
}
```

Output:

```
2       4       6       8       10
```

Example

```
/* C program to print odd numbers between 1 and n*/
#include<stdio.h>
#include<conio.h>
Void main()
{
        int i,n;
        clrscr();
        printf("Enter the number:\n");
          scanf("%d",&n);
        for(i=1;i<=10;i++)
        {
                If(i%2==1)
                        Printf("%d\t",i);
        }
}
```

Output:

```
1       3       5       7       9
```

<u>break and continue Statement</u>

There are two statement built in C, break; and continue; to interrupt the normal flow of control of a program. Loops performs a set of operation repeatedly until certain condition becomes false but, it is sometimes desirable to skip some statements inside loop and terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

<u>break Statement:</u>

In C programming, break is used in terminating the loop immediately after it is encountered. The break statement is used with conditional if statement

Syntax:

```
                    break;
```

The break statement can be used in terminating all three loops for, while and do...while loops.

The figure below explains the working of break statement in all three type of loops.

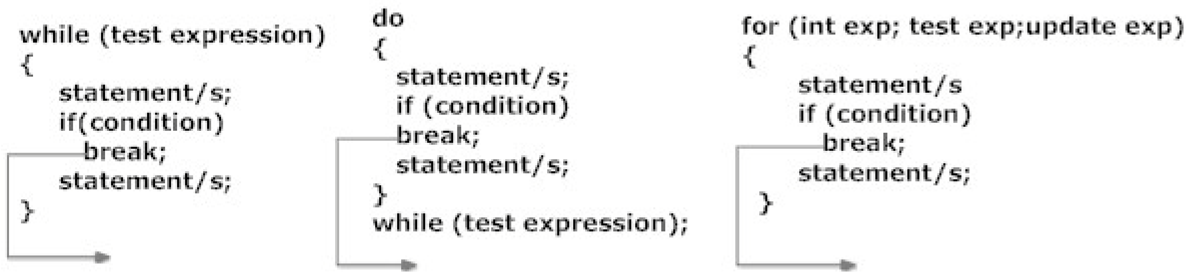


Fig: Working of break statement in different loops

Examples:

```
/* C program to demonstrate the working of break statement by terminating a loop, if
user inputs negative number*/
# include <stdio.h>
#include<conio.h>
void  main()
{
        float num,average,sum;
         int i,n;
         printf("Maximum no. of inputs\n");
         scanf("%d",&n);
         for(i=1;i<=n;++i)
        {
                        printf("Enter n%d: ",i);
                        scanf("%f",&num);
                        if(num<0.0)
                        break;                    //for loop breaks if num<0.0
                        sum=sum+num;
        }
          average=sum/(i-1);
          printf("Average=%.2f",average);
}
```

Output:

```
Maximum no. of inputs
4
Enter n1: 1.5
Enter n2: 12.5
Enter n3: 7.2
Enter n4: -1
Average=7.07
```

continue Statement:

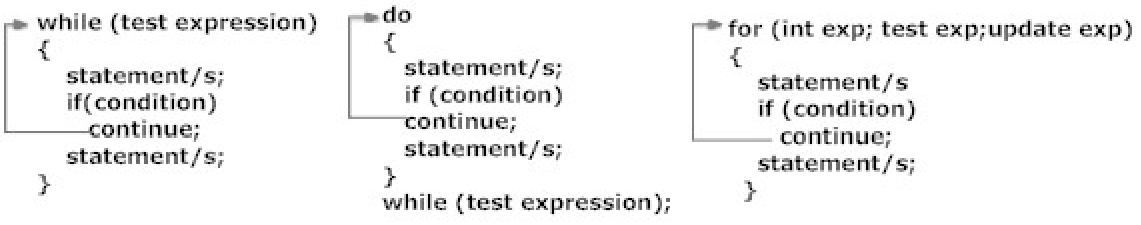
It is sometimes desirable to skip some statements inside the loop. In such cases, continue statements are used.

Syntax:

```
continue;
```

The continue statement can be used in terminating all three loops for, while and do...while loops.

The figure below explains the working of continue statement in all three type of loops.



```
while (test expression)
{
   statement/s;
   if(condition)
     continue;
   statement/s;
}
```

```
do
{
   statement/s;
   if (condition)
   continue;
   statement/s;
}
while (test expression);
```

```
for (int exp; test exp;update exp)
{
   statement/s
   if (condition)
     continue;
   statement/s;
 }
```

Fig: Working of continue statement in different loops

Example:

//program to demonstrate the working of continue statement in C programming

/*Write a C program to find the product of 4 integers entered by a user. If user enters 0 skip it.*/

```
# include <stdio.h>
int main()
{
         int i,num,product;
         for(i=1,product=1;i<=4;++i)
        {
            printf("Enter num%d:",i);
            scanf("%d",&num);
            if(num==0)
                  continue;       / *In this program, when num equals to zero, it skips the
              statement product*=num and continue the loop. */
                   product*=num;
        }
         printf("product=%d",product);
        return 0;
}
```

Output

```
Enter num1:    3
Enter num2:    0
Enter num3:   -5
Enter num4:    2
product=       - 30
```

Loop initialization and updating:

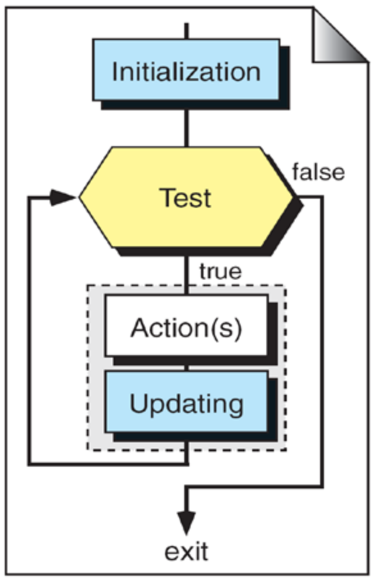
Loop Initialization:

- Programs typically require some type of preparation before executing loops.
- *Loop initialization*, which happens before a loop's first iteration, is a way to prepare the loop. It may be explicit or implicit.
- In explicit initialization, we write code to set the values of key variables used by the loop.

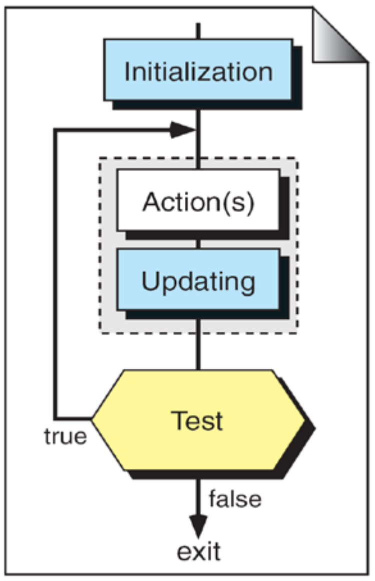
  Implicit initialization relies on a "preexisting situation to control the loop.

Updating a Loop:

- A loop update is what happens inside a loop's block that eventually causes the loop to satisfy the condition, thus ending the loop.
- Updating happens during each loop iteration.
- Without a loop update, the loop would be an infinite loop.



(a) Pretest Loop



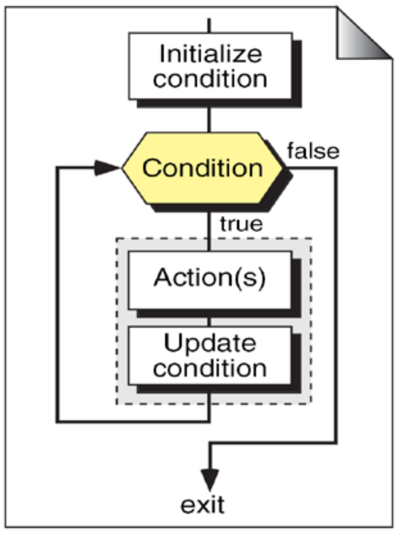
(b) Post-test Loop

-

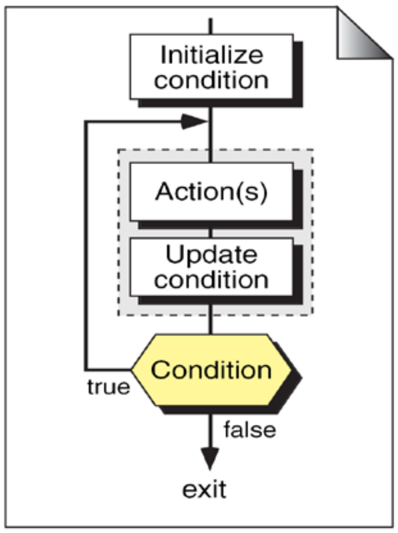
Event and counter controlled loops:

Event-Controlled Loops

- In an *event-controlled loop*, an event is something that happens in the loop's execution block that changes the loop's control expression from true to false.
- The program can update the loop explicitly or implicitly.
- We cannot predict the maximum number of iterations during the run of a program.



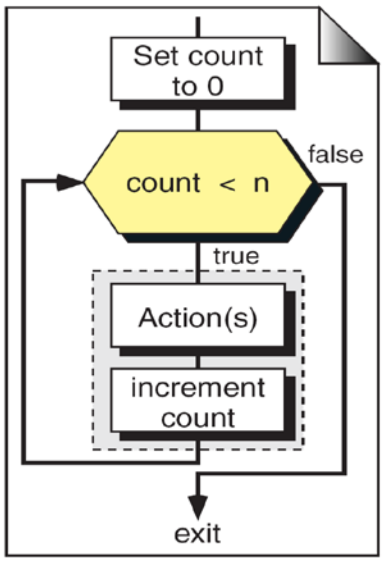
(a) Pretest Loop



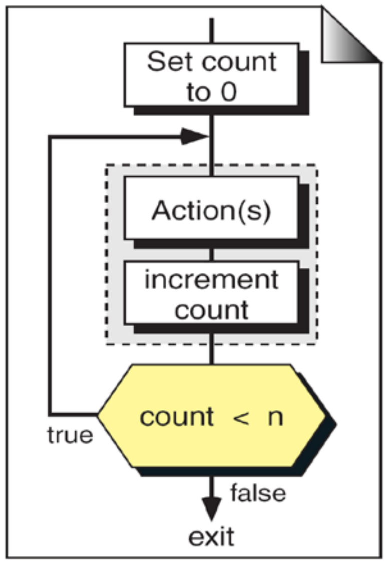
(b) Post-test Loop

Counter controlled loops:

- In a *counter-controlled loop*, we can control the number of loop iterations.
- In such a loop, we use a counter, which we must initialize, update and test.
- The number the loop assigns to the counter doesn't need to be a constant value.
- To update, we can increment or decrement the counter.



(a) Pretest Loop



(b) Post-test Loop

APPLICATIONS OF LOOPS:

```
/*c program to find the sum of N natural numbers*/
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,n,sum=0;
        printf("enter the value of N");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                sum=sum+i;
        }
        printf("the sum is%d",sum);
        getch();
}
```

OUTPUT

```
enter the value of N
5
The sum is 15
```

/*c program to find m power n*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int I,n,m,product=1;
        printf("enter the value of M");
        scanf("%d",&m);
        printf("enter the value of N");
        scanf("%d",&n);

        for(i=1;i<=n;i++)
        {
                product =product*m;
        }
        printf("the result is  %d",product);
        getch();
}
```

OUTPUT

```
enter the value of M
5
enter the value of N
3
The result is 125
```

****

ARRAYS

Introduction:

In C programming for accessing of multiple variables of same datatype the declaration is as follows

Let five integer variables

int a1,a2,a3,a4,a5;

Reading the values for all the five elements the statement can be written as

Scanf(“ %d %d %d %d %d”,&a1, &a2, &a3, &a4, &a5);

What, if the number of values is more than 10, we need to declare all the variables and need to clearly specify all the variables in the corresponding scnaf( ) statement, it will become hard.

Solution for the above problem is the main aim of implementing arrays.

Defnition: An array is a collection of dataitems of same datatypes are sharing a common name and stored in contiguous memory locations.

Types of arrays: Depending on the number of subcripts used, arrays are categorized as follows

1. One-dimensional Arrays
2. Two -dimensional Arrays
3. Multi-dimensional Arrays

One -Dimensional Arrays

An array with only one subscript is called one -dimensional Arrays or 1-D array.

Declaration:

The general form of declaring a one -dimensional Array is

Syntax:

Data_type Array_name[size];

Where,

Data_type refers to any datatype supported by C,

Array_name should be a valid idenfier and

Size indicates the maximum number of elements that can be stored inside the array. it must be a positive integer constant.

Example:

int A[6];

Here, A is declared to be an array of int type and size is 6. Six contiguous memory locations get allocated as shown below to store 6 integer values

element1	element 2	element 3	element 4	element 5	element 6
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]

Each data item in an array A is identified by the array name A followed by a pair of square brackets enclosing a subscript value.the subscript value is indexed from 0 to 5. i.e,

A[0] indicates **element1**

A[1] indicates **element2**

A[2] indicates **element3**

A[3] indicates **element4**

A[4] indicates **element5**

A[5] indicates **element6**

Size of an array:

The number of bytes allocated for an array is calculated using the formula

Total size = (sizeof(datatype)*array size)Bytes

For the above example

Total size = 2*6

= 12 Bytes

i.e every location occupies 2byes of memory.

Initialization:

Syntax: **Data_type Array_name[size]={list of values};**

Ex: int x[6]={1,2,3,4,5,6};

As a result , memory locations of X filledup as follows:

1	2	3	4	5	6
X[0]	X[1]	X[2]	X[3]	X[4]	X[5]

Note:

1.if the number of values in initializer is less than the size of an array, only those many first locations of the array are assigned the value. The remaining locations are assigned as zeros.

Example:

int x[6]={1,2,3};

Size of array = 6,

Values initialized =3.

Then the initialization is as follows

1	2	3	0	0	0
X[0]	X[1]	X[2]	X[3]	X[4]	X[5]

2. if the number of values in initializer is greater than the size of an array , compiler raises an error.

Example:

int x[6]={1,2,3,4,5,6,7,8};

Size of array = 6,

Values initialized =8.

Now, the compiler raises an error.

3. If a static array is declared without initializer list then all locations are set to zeros .

Example:

Static int X[6];

0	0	0	0	0	0
X[0]	X[1]	X[2]	X[3]	X[4]	X[5]

4. If size is ommited in a 1-D array declaration which is initialized, the compiler willl supply this value by examining the number of values in the initializer-list.

Example: 1.

int x[]={1,2,3,4,5,6};

Values initialized =6
Hence, size of array =6
And sizeof(X)=12bytes.

1	2	3	4	5	6
X[0]	X[1]	X[2]	X[3]	X[4]	X[5]

2.

int x[]={1,2,3,4};

Values initialized =4
Hence, size of array =4
And sizeof(X)=8bytes.

1	2	3	4
X[0]	X[1]	X[2]	X[3]

5. Array elements can not be initialized selectivily

Example:

Int X[5]={ , 20}

An attempt to initialize only second location is illegal.

<u>Accesing elements:</u>

The syntax template for accessing an array component is
ArrayName[IndexExpression]

Example:
int x[6]={1,2,3,4,5,6};
in the above statement
x[0]=1
x[1]=2
x[2]=3
x[3]=4
x[4]=5
x[5]=6

example1:
/*example to access 1-D array*/

#include<stdio.h>

```
#include<conio.h>
void main()
{
        int A[5]={1,2,3,4,5},i;        /*compile-time initialization*/
        clrscr();
        for(i=0;i<5;i++)
        {
                printf("A[%d]=%d",i,A[i]);
                printf("\n");
        }
        getch();
}
```

Ouput:

```
A[0]=1
A[1]=2
A[2]=3
A[3]=4
A[4]=5
```

Example2:

/*example to access 1-D array @ runtime initialization*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int A[5],i;                                     /*array declaration*/
        clrscr();
        for(i=0;i<5;i++)                                /*reading values for array*/
        {
                printf("enter A[%d]\n",i);
                scanf("%d",&A[i]);
        }
        printf("entered values are\n");
        for(i=0;i<5;i++)                                /*accessing the values of an
array*/
        {
                printf("A[%d]=%d\n",i,A[i]);
        }

        getch();
}
```

<u>Ouput:</u>

```
Enter A[0]
10
```

```
Enter A[1]
20
Enter A[2]
30
Enter A[3]
40
Enter A[4]
50
entered values are
A[0]=10
A[1]=20
A[2]=30
A[3]=40
A[4]=50
```

Example3:

```
//Sum of the elemets in an array

#include<stdio.h>
#include<conio.h>
void main()
{
        int A[5],i,sum=0;                                   /*array declaration*/
        clrscr();
        for(i=0;i<5;i++)                                /*reading values for array*/
        {
                printf("enter A[%d]\n",i);
                scanf("%d",&A[i]);
        }
        for(i=0;i<5;i++)                                /*accessing the values of an
array*/
        {
           sum=sum+A[i];
        }
        printf("Sum of elemets in array is %d",sum);
        getch();
}
```

<u>Ouput:</u>

```
Enter A[0]
10
Enter A[1]
20
Enter A[2]
30
Enter A[3]
```

```
40
Enter A[4]
50
Sum  of  elements  is  150
```

Example 4:

//minimum and maximum elemets in Array

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int min=0,max=0,i,a[5];
  clrscr();
  for(i=0;i<5;i++)              //reading array elements
  {
    printf("Enter the A[%d] elements ",i);
    scanf("%d",&a[i]);
  }
  for(i=0;i<5;i++)
  {
    if(a[i]<min)              //checking   for   minimum  element
      min=a[i];
    if(a[i]>max)          //checking   for maximum  element
      max=a[i];
  }
  printf("\n Minimum element is %d",min);
  printf("\n Maximum element is %d",max);
  getch();
}
```

<u>Ouput:</u>

```
Enter A[0]
10
Enter A[1]
20
Enter A[2]
30
Enter A[3]
40
Enter A[4]
50
Maximum  element  is  50
Minimum   element  is  10
```

STRINGS:

Introduction:

Consider the statement printf("HELLOW"); the printf() statement displays the message "HELLOW" on the screen. Let us knoww understand how the message "HELLOW" is stored in memory and how th printf() works().

The message "HELLOW" is stored in memory as

H	E	L	L	O	W	\0

Note that the last location in the memory block has the special character '\0' is automatically appended to the end of the messageby the compiler.

DEFNITION:

A String in C is defined as sequence of charecters terminated by a special character '\0' is called null character.

Declaration of Array of char type:

A String variable is a valid C variable name and is always declared an array.

Syntax:

char string_name[size];

the size determines the number of charecters in the string name.

Example:

An array of char typen ot store the above string is to be declared as follows:

char str[7]="HELLOW";

H	E	L	L	O	W	\0
Str[0]	Str[1]	Str[2]	Str[3]	Str[4]	Str[5]	Str[6]

An array of char type also called as string variable.

Initialization of array of char type:

Syntax:

1. **char str1[7]={'h','e','l','l','o','w','\0'};**

   here, str1 is declared to be a string variable with the size 7. It can be stored maximum 7 charecters including the null character.

2. **char str2[7]= "hellow";**

   here, str2 is declared to be a string variable with the size 7. It can be stored maximum 7 charecters including the null character. Though null character is not given in the input it automatically appended to the input.

STRING I/O:

Control string for string is **%s**

scanf() and printf():

scanf()

scanf() is used to attempt a string through keyboard.

Syntax:

```
scanf("%s", str);
```

example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char str1[20];          /* declaration */
        clrscr();
        printf("enter any string\n");
        scanf("%s",str1);               /* reading the string */
        printf("u entered is\n");
        printf("%s",str1);              /*printing the string */
        getch();
}
```

OUTPUT:

```
Enter any string
HI
U entered is
Hi
```

getchar() and putchar():

getchar() is used to read a character from keyboard

syntax:

```
        ch=getchar();
```

putchar() is used to print a character to the output screen

syntax:

```
        putchat(ch);
```

where ch represents a variable of type char or a character constant.

example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char  a;
        clrscr();
        printf("enter a character");
        a=getchar();                    /* reading the charecter */
        printf("u entered is\n");
        putchar(a);                     /*printing the charecter*/

        getch();
}
```

Output:

```
Enetr a character
H
U entered is H
```

Example2:

```
/* accept a line of charecters and display using getchar() & putchar() functions*/
#include<stdio.h>
#include<conio.h>
void main()
{
        char  str[100],ch;
        int  I=0;
        printf("enter a line of charecters\n");
        ch=getchar();
        while(ch != '\n')                                   /*reading and storing*/
        {
                str[i]=ch;
                ch=getchar();

                i++:
        }
        str[i]='\0';
        printf("the line of chrecters u entered is\n");
        for(i=0;str[i]!='\0';i++)                       /* displaying*/
        {
                putchar(str[i]);
        }
        getch();
}
```

OUTPUT:

```
enter a line of charecters
HOW R U..?
the line of chrecters u entered is
HOW R U..?
```

<u>gets() and puts():</u>

gets() is used to accept a string upto newline character. It automatically appends the null character'\0' to the end of the string.

Syntax:

```
        gets(str);
```

the purpose of **puts()** is to display a string contained in a string variable. It automatically appends the new line character'\n' to the end of the string.as a result the cursor is moved down by 1 line afer the string is displayed.

Syntax:

```
        puts(str);
```

exxample:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        char  str[100];
        printf("enter a line of charecters\n");
        gets(str);                              /*reading*/
        printf("the line of chrecters u entered is\n");
        puts(str);                              /* displaying*/
        getch();
}
```

STING MANIPULATIONS:

C supports large set of string handling functions, which are contained in the header file

#include< sting.h>

The most commonly performed operations ver strings are:

1. finding the length of the string (**strlen()**)
2. coping one string to another string (**strcpy()**)
3. comparing twostrings (**strcmp()**)
4. concatenation oftwo strings (**strcat()**)

finding the length of the string:

the function strlen() return the number of charecters in the given string excluding null character ('\0').

Syntax:

```
strlen(str);
```

example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
Void main()
{
        char str[30]="hellow";
        int k;
        clrscr();
        k=strlen(str);
        printf("the length of the string is %d",k);
        getch();
}
```

OUTPUT:

```
 The length of the string is 6.
```

Description:

Here, the string "hellow" stored in memory as follows

H	E	L	L	O	W	\0
Str[0]	Str[1]	Str[2]	Str[3]	Str[4]	Str[5]	Str[6]

Number of charecters in the string “hellow” including null character are 7. The function strlen() return the number of charecters in the given string excluding null character (‘\0’). Hence, for the above example strlen() returns the length as 6.

<u>Coping one string to another string:</u>

The function used to copy from one string to the another is strcpy()

Syntax:

    Strcpy(str2,str1);

Copies string contained in str1 to str2. Str1 can be string constant or a string variable.but str2 should be a string variable and it should be large enough to collect the string from str1.

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char str1[20]="hello",str2[30],str3[20];
        clrscr();
        strcpy(str2,str1);     /* copying from str1 to str2 */
        printf("the string in str2 is %s\n",str2);
        strcpy(str3,"hi");     /* copying a constant “hi” to str3*/
        printf("the string in str3 is %s",str3);
        getch();
}
```

OUTPUT:

```
The string in str 2 is hello
The string in str3 is hi
```

<u>Comparing two strings :</u>

strcmp() is used to compare two strings

syntax:

    strcmp(str1,str2);

description:

str1 and str2 represent two strings compared. Str1 str2 can be string constants or string variables.thefunction returns the numerical difference between the first non-matching pair of charactres of str1 and str2. It returns a positive value when str1 ig alphabetically greater than str2.

It returns a negative value when str1 ig alphabetically lower than str2.it returns zero when str1 and str2 are equal.

Example:

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

```
#include<conio.h>
#include<string.h>
void main()
{
        char str1[20]="hello",str2[30]="hello";
        int i;
        clrscr();
        i=strcmp(str2,str1); /* comparing str1 and str2 */
        printf("comparision of str1 and str2 is %d",i);
        getch();
}
```

OUTPUT:

```
comparision of str1 and str2 is 0
```

Example:

```
        char str1[20]="xyz",str2[30]="xxz";
        int i;
        i=strcmp(str2,str1);
```

OUTPUT:

```
comparision of str1 and str2 is 1
```

Example:

```
        char str1[20]="xxz",str2[30]="xyz";
        int i;
        i=strcmp(str2,str1);
```

OUTPUT:

```
comparision of str1 and str2 is -1.
```

Concatenation oftwo strings :

The built-in functionto perform_concatenation of two strings is strcat()

syntax:

```
        strcat(str1,str2);
```

it appends both str2 to str1. Str2 can be a string variable or a string constant.but str1 should be a string variableand the size of str1 should be large enough to collect even str2 also inaddition to its own string

example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char str1[20]="hello",str2[10]="hi";
        clrscr();
        strcat(str1,str2);
        printf("after concatenating str1 contains %s",str1);
        getch();
}
```

Output:

After concatenating str1 contains **hellohi**

Two-Dimensional arrays:

The array which is used to represent and store data in a tabular form is called as 'two dimensional array.' Such type of array specially used to represent data in a matrix form. The following syntax is used to represent two dimensional array.

Syntax:

<data-type> <array_nm> [row_subscript][column-subscript];

Example:

```
int a[3][3];
```

In above example, a is an array of type integer which has storage size of 3 * 3 matrix. The total size would be 3 * 3 * 2 = 18 bytes.

Memory Allocation :

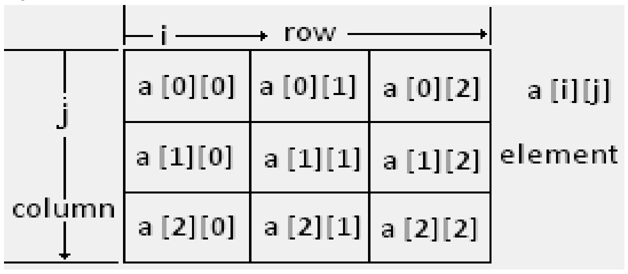


Fig : Memory allocation for two dimensional array

Program :

```
/*  Program to demonstrate two dimensional array.

#include <stdio.h>
#include <conio.h>
void main()
{
        int a[3][3], i, j;
        clrscr();
        printf("\n\t Enter matrix of 3*3 : ");
        for(i=0; i<3; i++)
        {
                for(j=0; j<3; j++)
                {
```

```
                scanf("%d",&a[i][j]);  //read 3*3 array
                }
        }
        printf("\n\t Matrix is : \n");
        for(i=0; i<3; i++)
        {
                for(j=0; j<3; j++)
                {
                printf("\t %d",a[i][j]);  //print 3*3 array
                }
          printf("\n");
        }
        getch();
}
```

Output :

```
        Enter matrix of 3*3 : 3 4 5 6 7 2 1 2 3

        Matrix is :
        3       4       5
        6       7       2
        1       2       3
```

Limitations of two dimensional array :

- We cannot delete any element from an array.
- If we dont know that how many elements have to be stored in a memory in advance, then there will be memory wastage if large array size is specified.

ARRAY APLICATIONS:

Matrix Addition:

```
#include<stdio.h>
#include<conio.h>
void main()
{
       int i,j,n,a[10][10],b[10][10],c[10][10];
       clrscr();
       printf("Enter size of the matrix ");
       scanf("%d",&n);  //reading size of the matrix
       printf("Enter Matrix 1 elements ");
       for(i=0;i<n;i++)  //for reading each row
       {
         for(j=0;j<n;j++)   //for reading columns
         {
           scanf("%d",&a[i][j]);   //reading the elements of matrix 1
         }
       }
       printf("Enter Matrix 2 elements ");
       for(i=0;i<n;i++)   //for reading each row
```

```
        {
          for(j=0;j<n;j++)  //for reading columns
          {
            scanf("%d",&b[i][j]);   //reading the elements of matrix 2
          }
        }
        for(i=0;i<n;i++)
        {
          for(j=0;j<n;j++)
          {
            c[i][j]=a[i][j]+b[i][j]; //adding 1,2 matrices
          }
        }
        printf("Sum of two matrices is  \n");
        for(i=0;i<n;i++)
        {
          for(j=0;j<n;j++)
          {
            printf("%d  ",c[i][j]);   //displaying of matrices
          }
          printf("\n"); //It is for going to next line after each row printing
        }
        getch();
}
```

Matrix Multiplication:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i,j,k,n,a[10][10],b[10][10],c[10][10];
        clrscr();
        printf("Enter size of the matrix ");
        scanf("%d",&n);  //reading size of the matrix
        printf("Enter Matrix 1 elements ");
        for(i=0;i<n;i++)  // for going to each row
        {
          for(j=0;j<n;j++)  //for going to each column
          {
            scanf("%d",&a[i][j]);   //reading the elements of matrix 1
          }
        }
        printf("Enter Matrix 2 elements ");
        for(i=0;i<n;i++)
        {
          for(j=0;j<n;j++)
```

```
            {
               scanf("%d",&b[i][j]);   //reading the elements of matrix 2
            }
         }
         for(i=0;i<n;i++)
         {
            for(j=0;j<n;j++)
            {
            c[i][j]=0;
              for(k=0;k<n;k++)
              {
                   c[i][j]=c[i][j]+a[i][k]*b[k][j]; //multiplying 1,2 matrices
              }
            }
         }
         printf("Sum of two matrices is  \n");
         for(i=0;i<n;i++)
         {
            for(j=0;j<n;j++)
            {
               printf("%d  ",c[i][j]);   //displaying of matrices
            }
            printf("\n"); //It is for going to next line after each row printing
         }
         getch();
}
```

Symetricity Of A Matrix:

```
#include<stdio.h>
#include<conio.h>
void main()
{
         int i,j,n,a[10][10],b[10][10],flag=0;
         clrscr();
         printf("Enter size of the matrix ");
         scanf("%d",&n);  //reading size of the matrix
         printf("Enter Matrix elements ");
         for(i=0;i<n;i++)     //for reading each row
         {
            for(j=0;j<n;j++) //for reading columns
            {
               scanf("%d",&a[i][j]);   //reading the elements of matrix 1
            }
         }
         for(i=0;i<n;i++)
         {
            for(j=0;j<n;j++)
            {
               b[j][i]=a[i][j]; //creating transpose matrix
            }
         }
         for(i=0;i<n;i++)
         {
            for(j=0;j<n;j++)
            {
               if(a[i][j]!=b[j][i])        //It transpose is not equal to the original matrix
               {
                    flag=1;      //set the flag to 0 to print it is not symmentric
                    break;       //break the loop..
                }
            }
         }
         if(flag==1)
                  printf(" Matrices are not Symmetric  ");
         else
                  printf(" Matrices are Symmetric  ");
         getch();
}
```

****

SYLLABUS
FUNCTIONS-MODULAR PROGRAMMING: Functions, basics, parameter passing, storage classesextern,
auto, register, static, scope rules, block structure, user defined functions, standard library functions,
recursive functions, Recursive solutions for Fibonacci series, Towers of Hanoi, header files, C pre-processor,
example c programs. Passing 1-D arrays, 2-D arrays to functions.

Functions in C:
The function is a self contained block of statements which performs a coherent task of a same kind.
C program does not execute the functions directly. It is required to invoke or call that functions. When a function is called in a program then program control goes to the function body. Then, it executes the statements which are involved in a function body. Therefore, it is possible to call function whenever we want to process that functions statements.
Advantages of functions:
1. Many programs require that a specific function is repeated many times instead of writing the function code as many timers as it is required we can write it as a single function and access the same function again and again as many times as it is required.
2. We can avoid writing redundant program code of some instructions again and again.
3. Programs with using functions are compact & easy to understand.
4. Testing and correcting errors is easy because errors are localized and corrected.
5. We can understand the flow of program, and its code easily since the readability is enhanced while using the functions.
6. A single function written in a program can also be used in other programs also.

Types of functions:
There are 2(two) types of functions as:
1. Built in Functions
2. User Defined Functions

1. Built in Functions :
These functions are also called as 'library functions'. These functions are provided by system. These functions are stored in library files.
Examples:
main()
- The execution of every C program starts from this main.
printf()
- prinf() is used for displaying output in C.
scanf()

- scanf() is used for taking input in C.

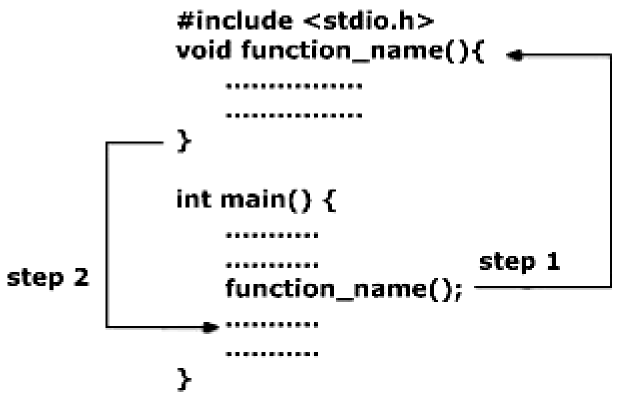
2.User defined function

C provides programmer to define their own function according to their requirement known as user defined functions.

Example of how C function works

```
#include <stdio.h>
void function_name(){
................
................
}
int main(){
...........
...........
function_name();
...........
...........
}
```

As mentioned earlier, every C program begins from main() and program starts executing the codes inside main function. When the control of program reaches to function_name() inside main. The control of program jumps to "void function_name()" and executes the codes inside it. When, all the codes inside that user defined function is executed, control of the program jumps to statement just below it. Analyze the figure below for understanding the concept of function in C.



```
#include <stdio.h>
void function_name(){
    ................
    ................
}

int main() {
    ...........
    ...........
    function_name();
    ...........
    ...........
}
```

Fig: Working of Functions

Remember, the function name is an identifier and should be unique.

Advantages of user defined functions

- User defined functions helps to decompose the large program into small segments which makes programmer easy to understand, maintain and debug.
- If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
- Programmer working on large project can divide the workload by making different functions.

Example of user defined function

1.Write a C program to add two integers. Make a function add() to add integers and display sum in main function.

```
/*Program to demonstrate the working of user defined function*/
#include <stdio.h>
#include <conio.h>
int add(int a, int b);            //function prototype(declaration)
void main(){
    int num1,num2,sum;
    printf("Enters two number to add\n");
    scanf("%d %d",&num1,&num2);
    sum=add(num1,num2);          //function call
    printf("sum=%d",sum);
}
int add(int a,int b)             //function declarator
{
    int add;
    add=a+b;
    return add;                 //return statement of function
}
```

2.Write a program to find biggest of the two numbers. Make a function bigger() to perform this and return the biggest value back to the main().

```
#include <stdio.h>

/* a function taking two double numbers and returning
the larger number of the two */

/* this function will return a double */
double  bigger (double n1, double n2)
{
        double big;

        if (n1>n2)
                big = n1;
        else
                big = n2;

        return (big);
}

void main ()
{
        double x, y, result;
        printf ("Enter a real number: ");
        scanf ("%lf", &x);

        printf ("Enter a real number: ");
        scanf ("%lf", &y);

        /* calling the function (2 arguments) */
        result = bigger (x,y);

        /* printing the report */
        printf ("The bigger number is %lf.\n", result);

}
```

Function prototype(declaration):

Every function in C programming should be declared before they are used. These type of declaration are also called function prototype. Function prototype gives compiler information about function name, type of arguments to be passed and return type.

Syntax of function prototype

return_type function_name(type(1) argument(1),....,type(n) argument(n));

Here, in the above example, function prototype is "int add(int a, int b);" which provides following information to the compiler:

1. name of the function is "add"
2. return type of the function is int.
3. two arguments of type int are passed to function.

Function prototype is not needed, if you write function definition above the main function.

Function call

Control of the program cannot be transferred to user-defined function (function definition) unless it is called (invoked).

Syntax of function call

function_name(argument(1),....argument(n));

In the above example, function call is made using statement "add(num1,num2);"

Function definition

Function definition contains programming codes to perform specific task.

Syntax of function definition

```
return_type function_name(type(1) argument(1),..,type(n) argument(n))
{
            //body of function
}
```

Function definition has two major components:

1. Function declarator

Function declarator is the first line of function definition. When a function is invoked from calling function, control of the program is transferred to function declarator or called function.

Syntax of function declarator

return_type function_name(type(1) argument(1),....,type(n) argument(n))

Syntax of function declaration and declarator are almost same except, there is no semicolon at the end of declarator and function declarator is followed by function body.

In above example, " int add(int a,int b)" in line 12 is function declarator.

2. Function body

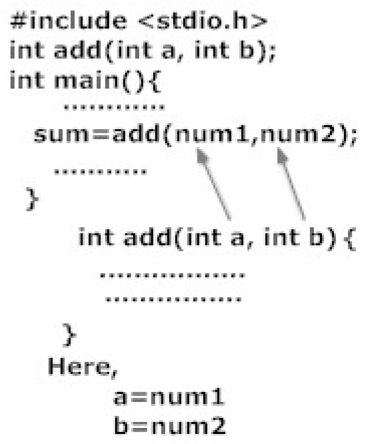
Function declarator is followed by body of function which is composed of statements.

In above example, line 13-19 represents body of a function.

Passing arguments to functions

In programming, argument/parameter is a piece of data(constant or variable) passed from a program to the function.

In above example two variable, num1 and num2 are passed to function during function call and these arguments are accepted by arguments a and b in function definition.



```
#include <stdio.h>
int add(int a, int b);
int main(){
      ............
   sum=add(num1,num2);
     ...........
 }
       int add(int a, int b){
           .................
           ................
     }
    Here,
           a=num1
           b=num2
```

Arguments that are passed in function call and arguments that are accepted in function definition should have same data type. For example:

If argument num1 was of int type and num2 was of float type then, argument a should be of int type and b should be of float type in function definition. i.e. type of argument during function call and function definition should be same.

A function can be called with or without an argument.

Function parameters / Arguments

There are Two types of function parameters. They are

(i) Actual Parameters

The parameters that are mentioned in the function call are said to be actual parameters.

(ii) Formal Parameters

The parameters that are mentioned in the function declaration are said to be formal parameters.

In the above example, num1 and num2 are Actual Parameters.

Where as   a and b are Formal Parameters.

Return Statement

Return statement is used for returning a value from function definition to calling function.

Syntax of return statement

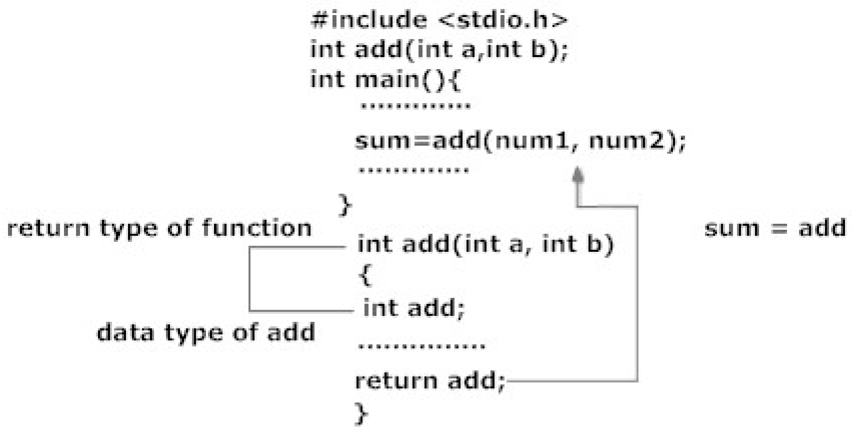
return (expression);

OR

 return;

For example:
return;   // returns a garbage value back to the calling function
return a; // returns value of 'a' back to the calling function

return(a+b); /* returns the value that is produced after evaluating the expression a+b */
Note that, data type of add is int and also the return type of function is int. The data type of expression in return statement should also match the return type of function.



```
#include <stdio.h>
int add(int a,int b);
int main(){
  .............
  sum=add(num1, num2);
  .............
 }
  int add(int a, int b)
  {
   int add;
  ...............
  return add;
  }
```

The Standard Library Functions and Header files
Some of the "commands" in C are not really "commands" at all but are functions. For example, we have been using **printf** and **scanf** to do input and output, and we have used **rand** to generate random numbers - all three are functions.
A list of the most common libraries and a brief description of the most useful functions they contain follows:

- **stdio.h: I/O functions:**
    - **getchar()** returns the next character typed on the keyboard.
    - **putchar()** outputs a single character to the screen.
    - **printf()** as previously described
    - **scanf()** as previously described
- **string.h: String functions**
    - **strcat()** concatenates a copy of str2 to str1
    - **strcmp()** compares two strings
    - **strcpy()** copys contents of str2 to str1
- **ctype.h: Character functions**
    - **isdigit()** returns non-0 if arg is digit 0 to 9
    - **isalpha()** returns non-0 if arg is a letter of the alphabet
    - **isalnum()** returns non-0 if arg is a letter or digit
    - **islower()** returns non-0 if arg is lowercase letter
    - **isupper()** returns non-0 if arg is uppercase letter
- **math.h: Mathematics functions**
    - **acos()** returns arc cosine of arg
    - **asin()** returns arc sine of arg

    - **atan()** returns arc tangent of arg
    - **cos()** returns cosine of arg
    - **exp()** returns natural logarithim e
    - **fabs()** returns absolute value of num
    - **sqrt()** returns square root of num
- **time.h: Time and Date functions**
    - **time()** returns current calender time of system
    - **difftime()** returns difference in secs between two times
    - **clock()** returns number of system clock cycles since program execution
- **stdlib.h:Miscellaneous functions**
    - **malloc()** provides dynamic memory allocation, covered in future sections
    - **rand()** as already described previously
    - **srand()** used to set the starting point for rand()

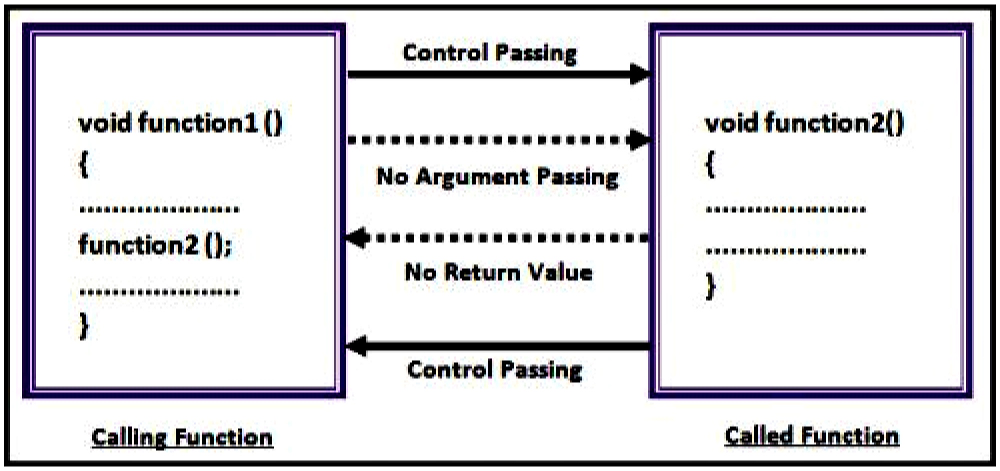
User defined functions are classified into 4 types depending on Arguments and their Return values

1. Functions with no arguments and no return values.
2. Functions with arguments and no return values.
3. Functions with arguments and return values.
4. Functions with no arguments and return values.

1. Function with no arguments and no return value:- When a function has no arguments it does not receive any data from the calling function. Similarly when it does not return a value the calling function does not receive any data from the called function. In affect there is no data transfer between the calling function and the called function.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 void sum();             /*declaration;*/
 clrscr();
 sum();                  /*calling;*/
 getch();
 }
 void sum()              /* definition*/
 {
        int a,b;
        printf("Enter any two values:");
        scanf("%d%d",&a,&b);
        printf("Sum is %d\n",a+b);
  }
```

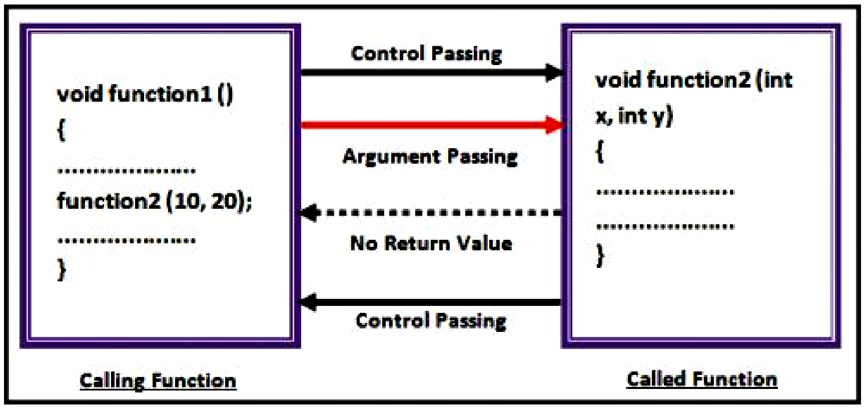


2. Functions with arguments and no return value.

In this type the function has some arguments, it receive data from the calling function but it does not return any value. The calling function does not receive data from the called function. So there is one way data communication calling function and the called function.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 void sum(int, int);                    /*declaration;*/
int  a,b;
clrscr();
printf("Enter any two values:");
scanf("%d%d",&a,&b);
sum(a,b);                              /*calling;*/
getch();
 }
 void sum(int x, int y)                         /* definition*/
 {
         printf("Sum is %d\n",x+y);
  }
```



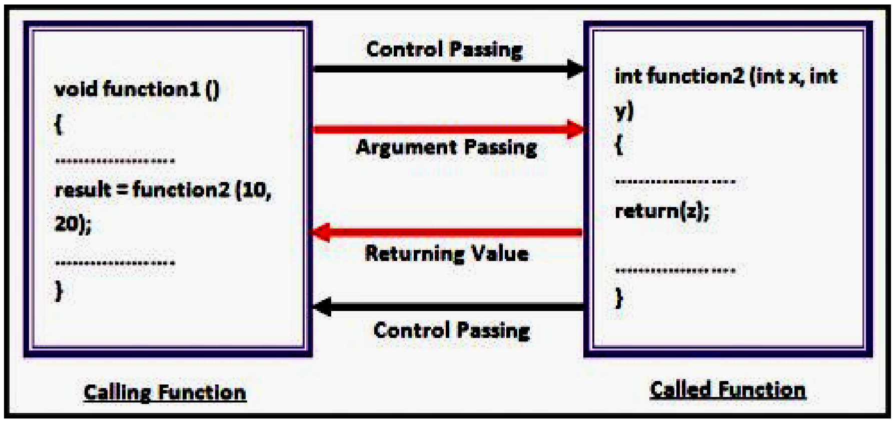
Logic of the function with arguments and no return value.

3. Functions with arguments and return value.

In this type the function has some arguments. It receives data from the calling function. Similarly it returns a value. The calling function receive data from the called function. So it is a two way data communication between the calling function and the called function.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int sum(int,int);
 int a,b,n;
 clrscr();
 printf("Enter any two values:");
 scanf("%d%d",&a,&b);
 n=sum(a,b);
 printf("Sum is %d",n);
 getch();
 }
 int sum(int x,int y)
 {
         return x+y;
  }
```



```
void function1 ()
{
.....................
result = function2 (10,
20);
.....................
}
```

```
int function2 (int x, int
y)
{
.....................
return(z);
.....................
}
```

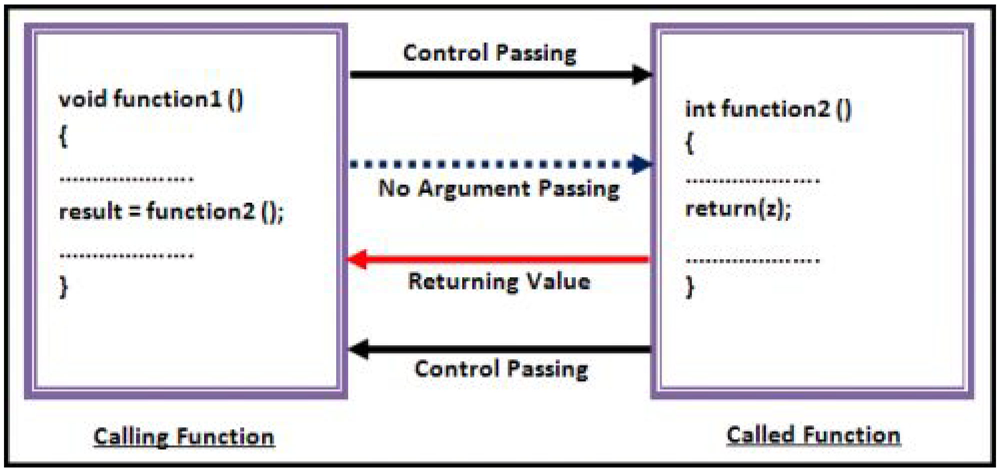
Logic of the function with arguments and return value.

4. Functions with no arguments and returns value.

In this type the function has no arguments. It does not receive data from the calling function. But it returns a value the calling function receive data from the called function. So it is a one way data communication between called function and the calling function.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int sum();                    /*declaration;*/
int s;
clrscr();
s=sum();                    /*calling;*/
printf("Sum is %d",s);
getch();
}
int sum()                   /* definition*/
{
        int a,b;
        printf("Enter any two values:");
        scanf("%d%d",&a,&b);
        return a+b;
 }
```



*****Note that the value return by any function when no format is specified is an integer***

Parameter passing mechanisms

Pass By Value

Passing a variable by value makes a copy of the variable before passing it onto a function. This means that if you try to modify the value inside a function, it will only have the modified value inside that function. One the function returns, the variable you passed it will have the same value it had before you passed it into the function.

```
void swap(int x, int y)
{
  int temp;
  temp = x;
  x = y;
  y = temp;
  printf("Swapped values are a = %d and b = %d", x, y);
}

void main()
{
   int a = 7, b = 4;

   printf("Original values are a = %d and b = %d", a, b);
   swap(a, b);
   printf("The values after swap are a = %d and b = %d", a, b);
}
```

Output:

```
            Original Values are a = 7 and b = 4
            Swapped values are a = 4 and b = 7
            The values after swap are a = 7 and b = 4
```

Pass By Reference

There are two instances where a variable is passed by reference:

1. When you modify the value of the passed variable locally and also the value of the variable in the calling function as well.
2. To avoid making a copy of the variable for efficiency reasons.

```
void swap(int *x, int *y)
{
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
  printf("Swapped values are a = %d and b = %d", *x, *y);
}
void main()
{
  int a = 7, b = 4;
  printf("Original values are a = %d and b = %d",a,b);
  swap(&a, &b);
  printf("The values after swap are a = %d and b = %d",a,b);
}
```

Output:

```
  Original Values are a = 7 and b = 4
  Swapped values are a = 4 and b = 7
  The values after swap are a = 4 and b = 7
```

Storage Classes:

'Storage' refers to the scope of a variable and memory allocated by compiler to store that variable. Scope of a variable is the boundary within which a variable can be used. Storage class defines the the scope and lifetime of a variable.

From the point view of C compiler, a variable name identifies physical location from a computer where variable is stored. There are two memory locations in a computer system where variables are stored as : Memory and CPU Registers.

Functions of storage class :

To determine the location of a variable where it is stored?

Set initial value of a variable or if not specified then setting it to default value.
Defining scope of a variable.
To determine the life of a variable.

Types of Storage Classes :

Storage classes are categorized in 4 (four) types as,

- Automatic Storage Class
- Register Storage Class
- Static Storage Class
- External Storage Class

Automatic Storage Class:

You can go forever without ever declaring a variable automatic, the reason being that all variables are implicitly automatic. The technical meaning of automatic variable means that memory will automatically allocated for it by the compiler and will be destroyed automatically after it is no longer within scope.

Syntax :

```
auto [data_type] [variable_name];
```

Example :

```
auto int a;
```

```
auto int var = 0;
int var = 0;
```

Are exactly the same.

- Keyword : auto
- Storage Location : Main memory
- Initial Value : Garbage Value
- Life time : Control remains in a block where it is defined.
- Scope : Local to the block in which variable is declared.

Program

```
#include <stdio.h>
#include <conio.h>
void main()
{
        auto int i=10;
        clrscr();
        {
                auto int i=20;
                printf("\n\t %d",i);
        }
```

```
            printf("\n\n\t %d",i);
            getch();
}
```

Output :

```
        20
        10
```

Register Storage Class:

When the calculations are done in CPU, then the values of variables are transferred from main memory to CPU. Calculations are done and the final result is sent back to main memory. This leads to slowing down of processes.

Register variables occur in CPU and value of that register variable is stored in a register within that CPU. Thus, it increases the resultant speed of operations. There is no waste of time, getting variables from memory and sending it to back again.

It is not applicable for arrays, structures or pointers.

Unary and address of (&) cannot be used with these variables as explicitly or implicitly.

- Keyword : register
- Storage Location : CPU Register
- Initial Value : Garbage
- Life : Local to the block in which variable is declared.
- Scope : Local to the block.

Syntax :

```
            register [data_type] [variable_name];
```

Example :

```
            register int a;
```

Program

```
#include <stdio.h>
#include <conio.h>
void main()
{
            register int i=10;
            clrscr();
            {
                        register int i=20;
                        printf("\n\t %d",i);
            }
            printf("\n\n\t %d",i);
            getch();
}
```

<u>OUTPUT</u>

```
20
10
```

Static Storage Class :
Static storage class can be used only if we want the value of a variable to persist between different function calls.

- Keyword : static
- Storage Location : Main memory
- Initial Value : Zero and can be initialized once only.
- Life : depends on function calls and the whole application or program.
- Scope : Local to the block.

Syntax :

```
static [data_type] [variable_name];
```

Example :

```
static int a;
```

Program

```
#include<stdio.h>
#include<conio.h>
void Check();
void  main()
{
  Check();
  Check();
  Check();
}
void Check()
{
   static int c=0;
   printf("%d\t",c);
   c+=5;
}
```

Output

```
0     5     10
```

Explanation
During first function call, it will display 0. Then, during second function call, variable c will not be initialized to 0 again, as it is static variable. So, 5 is displayed in second function call and 10 in third call.
If variable c had been automatic variable, the output would have been:

```
0    0    0
```

External storage class
External variable can be accessed by any function. They are also known as global variables. Variables declared outside every function are external variables.
In case of large program, containing more than one file, if the global variable is declared in file 1 and that variable is used in file 2 then, compiler will show error. To solve this

problem, keyword extern is used in file 2 to indicate that, the variable specified is global variable and declared in another file.

- o Keyword : extern
- o Storage Location : Main memory
- o Initial Value : Zero
- o Life : Until the program ends.
- o Scope : Global to the program.

Syntax :

```
extern [data_type] [variable_name];
```

Example :

```
extern int a;
```

Program

```
#include <stdio.h>
void Check();
int a=5;        // a is global variable because it is outside every function
int main()
{
   a+=4;
   Check();
   return 0;
}

Void Check ()
{
++a;   /*   Variable a is not declared in this function but, works in any  function as they
are global variable   */
  printf ("a=%d\n",a);
}
```

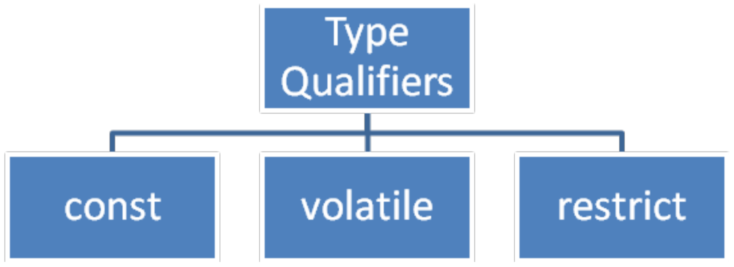
Output

```
a=10
```

SCOPE RULES:

1.The scope of a global variable is the entire program file.
2.The scope of a local variable begins at point of declaration and ends at the end of the block or function in which it is declared.
3.The scope of a formal function argument is its own function.
4.The life time of an auto variable declared in main( ) is the entire program exception time ,although its scope is only the main function .
5.The life of an auto variable declared in a function ends when the function is exited.
6.A static local variable ,although its scope is limited to its function,its life time extends till the end of program execution.
7.All variables have visibility in their scope, provided they are not declared again.
8.If a variable is redeclared within its scope again,it loses its visibility in the scope of the redeclared variable.

Type Qualifiers:



Constants:

The Keyword for the constant type qualifier is const.
A constant object is a read-only object,that is , it can only be used as rvalue.A constant object must be initialized when it is declared because , it cannot be changed later.A simple constant is shown below.
Example: const double pi=3.1414926;
Const char str[]="hello";

Volatile:

The Keyword for the volatile type qualifier is volatile
The volatile qualifier tells the computer that an object value may be changed by entities other than this program.
Example: volatile int x;
volatile int *ptr;

Restricted: The Keyword for the restricted type qualifier is restrict
The restrict qualifier which is used only with pointers, indicates that the pointer is only the initial way to access the dereferenced data.
Example: restrict int *ptr;
int a=0;
ptr=&a;

Block Structure :

C is not a block-structured language in the sense of Pascal or similar languages, because functions may not be defined within other functions. On the other hand, variables can be defined in a block-structured fashion within a function. Declarations of variables (including initializations) may follow the left brace that introduces *any* compound statement, not just the one that begins a function. Variables declared in this way hide any identically named variables in outer blocks, and remain in existence until the matching right brace. For example, in

```
if (n > 0) {
int i; /* declare a new i */
for (i = 0; i < n; i++)
...
}
```

the scope of the variable i is the "true" branch of the if; this i is unrelated to any i outside the block. An automatic variable declared and initialized in a block is initialized each time the block is entered.
Automatic variables, including formal parameters, also hide external variables and functions of the same name. Given the
declarations

```
int x;
int y;
f(double x)
{
double y;
}
```

then within the function f, occurrences of x refer to the parameter, which is a double; outside f, they refer to the external int.
The same is true of the variable y.

RECURSION

A Function calling itself is called recursion. Any function can call any function including itself. In this scenario, if it happens to invoke a function by itself, it is recursion. One of the instructions in the function is a call to the function itself, usually the last statement. In some ways it is similar to looping. When the result of 'one time call of a function is the input for the next time call of the function', recursion is one of the best ways. For example, calculating factorial, in which the product of previous digit factorial is the input for the next digit's factorial.

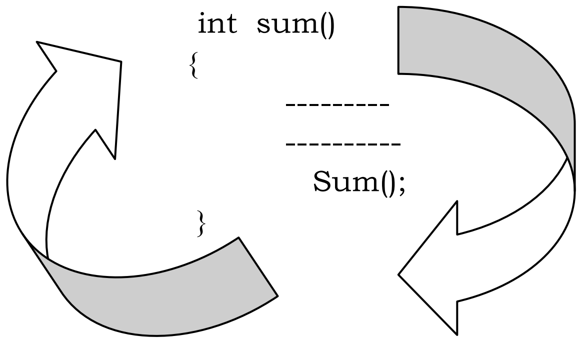
Basic elements of Recursion

- A test to stop or continue the recursion
- An end case that terminates the recursion
- A recursive call(s) that continues the recursion

Types of recursion

1) Direct recursion occurs when a method invokes itself, such as when sum calls sum.

Ex:



```
int  sum()
{
        ---------
        ----------
           Sum();
}
```

2) Indirect recursion occurs when a method invokes another method, eventually resulting in the original method being invoked again.

```
Int sum()
{
---------
---------
 Calc();
}

Int calc()
{
-------
      Sum();
}
```
Examples of Recursion

1)Ex : Calculation of factorial

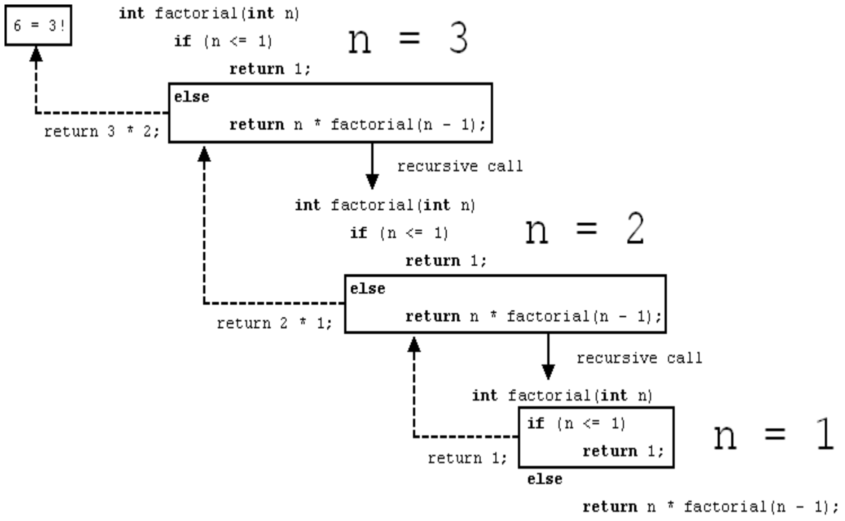
Factorial of n *(*denoted *n!)* is a product of integer numbers from 1 to n. For instance, 5! = 1 * 2 * 3 * 4 * 5 = 120.

Recursion is one of techniques to calculate factorial. Indeed, 5! = 4! * 5. To calculate factorial of *n*, we should calculate it for *(n-1)*. To calculate factorial of *(n-1)* algorithm should find *(n-2)!* and so on.

```
int factorial(int n) {
      if (n <= 1)
            return 1;
      else
            return n * factorial(n - 1);
}
```
The above program works as follows,

fact(3) = {if(3= =1) is false thus return 3* fact (2)}
fact(2) = {if(2= =1) is false thus return 2* fact (1)}
fact(1) = {if(1= =1) is true thus return 1}

Calculation of 3! in details



```
/* Write C programs that use both recursive and non-recursive functions
  To find the factorial of a given integer.*/
#include<stdio.h>
```

```
#include<conio.h>
unsigned int recr_factorial(int n);
unsigned int iter_factorial(int n);
void main()
{
  int n,i;
  long fact;
  clrscr();
  printf("Enter the number: ");
  scanf("%d",&n);
  if(n==0)
    printf("Factorial of 0 is 1\n");
  else
  {
    printf("Factorial of %d Using Recursive Function is %d\n",n,recr_factorial(n));
    printf("Factorial of %d Using Non-Recursive Function is %d\n",n,iter_factorial(n));
   }
   getch();
}
/* Recursive Function*/
unsigned int recr_factorial(int n) {
    return n>=1 ? n * recr_factorial(n-1) : 1;
}
/* Non-Recursive Function*/
unsigned int iter_factorial(int n) {
    int accu = 1;
    int i;
    for(i = 1; i <= n; i++) {
accu *= i;
    }
    return accu;
}
```

output:-

```
Enter the number  6 /* here given 6 as input */

Factorial of 6 using recursive function is 720

Factorial of 6 using non recursive function is 720
```

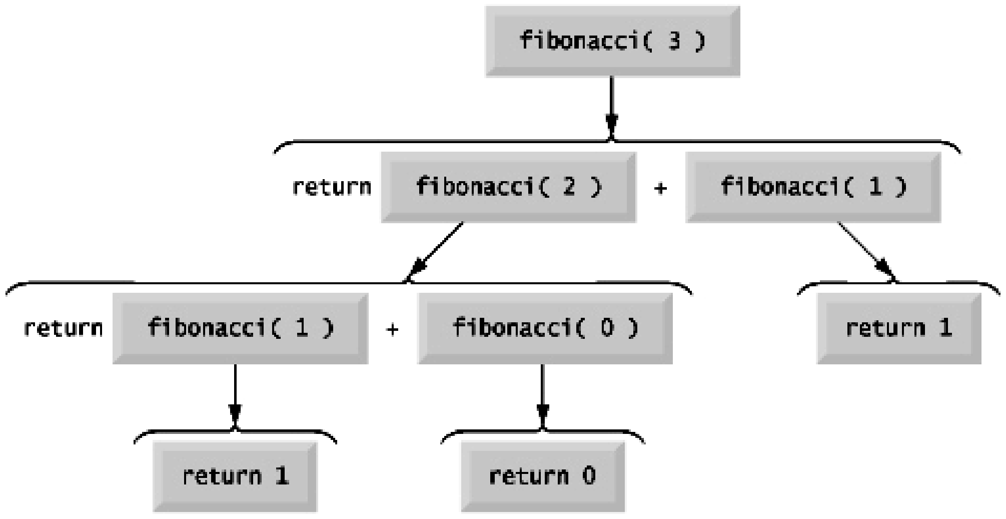
2. Example Using Recursion: Fibonacci Series

The Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

begins with 0 and 1 and has the property that each subsequent Fibonacci number is the sum of the previous two Fibonacci numbers.

The Fibonacci series can be defined recursively as follows:
fibonacci(0) = 0
fibonacci(1) = 1
fibonacci(n) = fibonacci(n – 1) + fibonacci(n – 2)



PROGRAM

```
#include<stdio.h>

int Fibonacci(int);

main()
{
  int n, i = 0, c;

  scanf("%d",&n);

  printf("Fibonacci series\n");

  for ( c = 1 ; c <= n ; c++ )
  {
    printf("%d\n", Fibonacci(i));
    i++;
  }

  return 0;
}

int Fibonacci(int n)
{
  if ( n == 0 )
    return 0;
```

```
   else if ( n == 1 )
      return 1;
   else
      return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

c program to find gcd of given two numbers :

```
/* Write C programs that use both recursive and non-recursive functions
To find the GCD (greatest common divisor) of two given integers */
#include<stdio.h>
#include<conio.h>
#include<math.h>
unsigned int GcdRecursive(unsigned m, unsigned n);
unsigned int GcdNonRecursive(unsigned p,unsigned q);
void main(void)
{
  int a,b;
  clrscr();
  printf("Enter the two numbers whose GCD is to be found: ");
  scanf("%d%d",&a,&b);
  printf("GCD of %d and %d Using Recursive Function is %d\n",a,b,GcdRecursive(a,b));
  printf("GCD of %d and %d Using Non-Recursive Function is
%d\n",a,b,GcdNonRecursive(a,b));
  getch();
}
/* Recursive Function*/
unsigned int GcdRecursive(unsigned m, unsigned n)
{
 if(n>m)
 return GcdRecursive(n,m);
 if(n==0)
return m;
 else
 return GcdRecursive(n,m%n);
}
/* Non-Recursive Function*/
unsigned int GcdNonRecursive(unsigned p,unsigned q)
{
 unsigned remainder;
 remainder = p-(p/q*q);
 if(remainder==0)
 return q;
 else
 GcdRecursive(q,remainder);
}
```

Write a C program to find the highest common factor using recursion.

```
#include <stdio.h>
int hcf(int x, int y);
int main(){
    int n1,n2;
    printf("Enter two numbers to calculate hcf\n");
    scanf("%d%d",&n1,&n2);
    printf("Highest common factor=%d",hcf(n1,n2));
    return 0;
}
int hcf(int x, int y)
{
    if(y==0)
       return x;
    else
       return hcf(y,x%y);
}
```

Write a C program to reverse a sentence using recursive function.

```
#include <stdio.h>
void Reverse(char c);
int main(){
   char c;
   printf("Enter sentence to reverse: \n");
   Reverse(c);
   return 0;
}
void Reverse(char c){
    scanf("%c",&c);
    if(c!='\n'){
        Reverse(c);
        printf("%c",c);
      }
}
```

Write a C program to reverse a number entered by user. Make function Reverse() to reverse the numbers.

```
#include<stdio.h>
int Reverse(int n);
int main()
{
    int num,i;
    printf("Enter number to reverse\n");
    scanf("%d",&num);
    printf("Reverse of %d=%d",num,Reverse(num));
    return 0;
}
int Reverse(int n)
{
    int rem,rev=0;
    while(n!=0){
       rem=(n%10);
       n=(n/10);
       rev=rev*10+rem;
    }
    return rev;
}
```

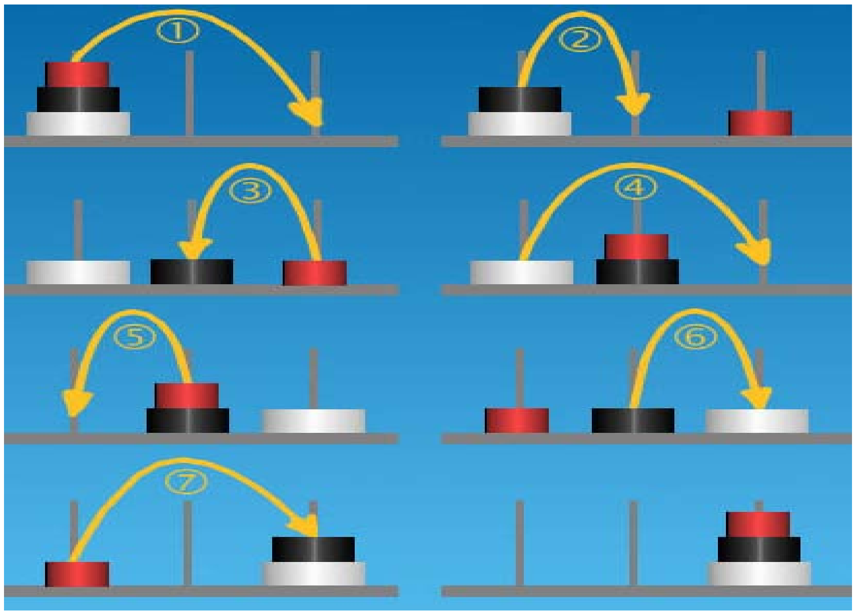
Towers of Hanoi - Problem and solution

The **Tower of Hanoi** is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

Solution



c program to solve towers of hanoi problem

```
/* Write C programs that use both recursive and non-recursive functions
  To solve Towers of Hanoi problem.*/
```

```
#include<conio.h>
#include<stdio.h>
/* Non-Recursive Function*/
void hanoiNonRecursion(int num,char sndl,char indl,char dndl)
{
  char stkn[100],stksndl[100],stkindl[100],stkdndl[100],stkadd[100],temp;
  int top,add;
  top=NULL;
  one:
if(num==1)
{
 printf("\nMove top disk from needle %c to needle %c ",sndl,dndl);
 goto four;
}
  two:
top=top+1;
stkn[top]=num;
stksndl[top]=sndl;
stkindl[top]=indl;
stkdndl[top]=dndl;
stkadd[top]=3;
num=num-1;
sndl=sndl;
temp=indl;
indl=dndl;
dndl=temp;
goto one;
  three:
printf("\nMove top disk from needle %c to needle %c ",sndl,dndl);
top=top+1;
stkn[top]=num;
stksndl[top]=sndl;
stkindl[top]=indl;
stkdndl[top]=dndl;
stkadd[top]=5;
num=num-1;
temp=sndl;
sndl=indl;
indl=temp;
dndl=dndl;
goto one;
  four:
if(top==NULL)
 return;
num=stkn[top];
sndl=stksndl[top];
indl=stkindl[top];
dndl=stkdndl[top];
add=stkadd[top];
top=top-1;
if(add==3)
 goto three;
else if(add==5)
 goto four;
}
/* Recursive Function*/
void  hanoiRecursion( int num,char ndl1, char ndl2, char ndl3)
{
    if ( num == 1 ) {
printf( "Move top disk from needle %c to needle %c.", ndl1, ndl2 );
return;
     }
     hanoiRecursion( num - 1,ndl1, ndl3, ndl2 );
     printf( "Move top disk from needle %c to needle %c.", ndl1, ndl2 );
     hanoiRecursion( num - 1,ndl3, ndl2, ndl1 );
}
void main()
{
  int no;
  clrscr();
  printf("Enter the no. of disks to be transferred: ");
  scanf("%d",&no);
  if(no<1)
     printf("\nThere's nothing to move.");
  else
     printf("Non-Recursive");
     hanoiNonRecursion(no,'A','B','C');
     printf("\nRecursive");
     hanoiRecursion(no,'A','B','C');
  getch();
}
```

The C Preprocessor Directives: A unique feature of c language is the preprocessor. A program can use the tools provided by preprocessor to make the program easy to read, modify, portable and more efficient.

The preprocessor more or less provides its own language which can be a very powerful tool to the programmer. Recall that all preprocessor directives or commands begin with a #.

Preprocessor is a program that processes the code before it passes through the compiler. It operates under the control of preprocessor command lines and directives. Preprocessor directives are placed in the source program before the main line before the source code passes through the compiler it is examined by the preprocessor for any preprocessor directives.

Preprocessor directives:

Directive	Function
#define	Defines a macro substitution
#undef	Undefines a macro
#include	Specifies a file to be included
#ifdef	Tests for macro definition
#endif	Specifies the end of # if
#ifndef	Tests whether the macro is not def
#if	Tests a compile time condition
#else	Specifies alternatives when # if test fails

The preprocessor directives can be divided into three categories

1. Macro substitution division
2. File inclusion division
3. Compiler control division

Macros: *Macro substitution is a process where an identifier in a program is replaced by a pre defined string composed of one or more tokens we can use the #define statement for the task. It has the following form*

#define identifier string

The preprocessor replaces every occurrence of the identifier int the source code by a string. The definition should start with the keyword #define and should follow on identifier and a string with at least one blank space between them. The string may be any text and identifier must be a valid c name.

There are different forms of macro substitution. The most common form is

1. Simple macro substitution
2. Argument macro substitution
3. Nested macro substitution

Simple macro substitution: ***Simple string replacement is commonly used to define constants example:***

#define pi 3.1415926

Writing macro definition in capitals is a convention not a rule a macro definition can include more than a simple constant value it can include expressions as well. Following are valid examples:

#define AREA 12.36

Macros as arguments: ***The preprocessor permits us to define more complex and more useful form of replacements it takes the following form.***

# define identifier(f1,f2,f3…..fn) string.

Notice that there is no space between identifier and left parentheses and the identifier f1,f2,f3 …. Fn is analogous to formal arguments in a function definition.

There is a basic difference between simple replacement discussed above and replacement of macro arguments is known as a macro call

A simple example of a macro with arguments is

```
#define CUBE(x) (x*x*x)
```

If the following statements appears later in the program,

```
volume=CUBE(side);
```

The preprocessor would expand the statement to

```
volume =(side*side*side)
```

Nesting of macros: ***We can also use one macro in the definition of another macro. That is macro definitions may be nested. Consider the following macro definitions***

```
#define SQUARE(x)  ((x)*(x))
```

Undefining a macro: *A defined macro can be undefined using the statement*

```
#undef identifier.
```

This is useful when we want to restrict the definition only to a particular part of the program.

File inclusion: *The preprocessor directive "#include file name" can be used to include any file in to your program if the function s or macro definitions are present in an external file they can be included in your file*

In the directive the filename is the name of the file containing the required definitions or functions alternatively the this directive can take the form

```
#include< filename >
```

Without double quotation marks. In this format the file will be searched in only standard directories. The c preprocessor also supports a more general form of test condition #if directive. This takes the following form

```
#if constant expression
{
statement-1;
statemet2'
....
....
}
#endif
```

the constant expression can be a logical expression such as test < = 3 etc

Passing entire one-dimensional array to a function

Arrays are the contiguous block of area. So, if the area of first element is known, then other element of arrays can also be accessed. While passing arrays to the argument, the name of the array is passed as an argument(,i.e, starting address of memory area is passed as argument)

passing an array to function, as an actual argument only the name must be passed; without the square brackets and the subscripts.

The corresponding formal argument is written in the same way but it must be declare inside the function. The formal declaration of the array should not contain the array dimension. In the

declaration of function using arrays as arguments, a pair of empty square brackets must follow the data type of array argument (or name of array if given).

PASSING AN ENTIRE 1D ARRAY TO A FUNCTION

Write a C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.

```
#include <stdio.h>
float average(float a[]);
int main(){
     float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
     avg=average(c);   // Only name of array is passed as argument.
     printf("Average age=%.2f",avg);
     return 0;
   }
float average(float a[]){
     int i;
     float avg, sum=0.0;
     for(i=0;i<6;++i){
       sum+=a[i];
     }
     avg =(sum/6);
     return avg;
}
```

Output

```
Average age=27.08
```

C program to pass a single element of an array to function

```
#include <stdio.h>
void display(int a)
   {
   printf("%d",a);
   }
int main(){
   int c[]={2,3,4};
   display(c[2]);  //Passing array element c[2] only.
   return 0;
}
```

Output

```
4
```

Single element of an array can be passed to function as passing variable to a function.

C Program to Pass 1-D Array to Function Element by Element

```
#include<stdio.h>

void show(int b[3]);

void main()
{
int arr[3] = {1,2,3};
int i;
for(i=0;i<3;i++)
  show(arr[i]);
}

void show(int x)
{
printf("%d\t ",x);
}
```

Output

1 2 3

Write a C program to display all prime numbers within the range specified by user using functions.

```
#include<stdio.h>
int check_prime(int num);
int main(){
   int n1,n2,i,flag;
   printf("Enter the range:\n");
   scanf("%d%d",&n1,&n2);
   for(i=n1+1;i<n2;++i){
      flag=check_prime(i);
      if(flag==1)
         printf("%d ",i);
   }
   return 0;
}
int check_prime(int num){
   int j,temp=1;
   for(j=2;j<=num/2;++j){
        if(num%j==0){
            temp=0;
            break;
        }
   }
   return temp;
}
```

Passing 2D array to a function

Code for Program of matrix addition using function in C .

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

void read_arr(int a[10][10],int row,int col)
{
    int i,j;
    for(i=1;i<=row;i++)
    {
    for(j=1;j<=col;j++)
    {
        printf("Enter Element %d %d : ",i,j);
        scanf("%d",&a[i][j]);
            }
    }
}
void add_arr(int m1[10][10],int m2[10][10],int m3[10][10],int row,int col)
{
    int i,j;
    for(i=1;i<=row;i++)
    {
    for(j=1;j<=col;j++)
    {
    m3[i][j] = m1[i][j] + m2[i][j];
    }
    }
}
void print_arr(int m[10][10],int row,int col)
{
    int i,j;
    for(i=1;i<=row;i++)
        {
        for(j=1;j<=col;j++)
        {
            printf("%d ",m[i][j]);
         }
        printf("\n");
        }
}
main()
{
    int m1[10][10],m2[10][10],m3[10][10],row,col;
    clrscr();
    printf("Enter number of rows :");
    scanf("%d",&row);
    printf("Enter number of colomns :");
    scanf("%d",&col);
    read_arr(m1,row,col);
    read_arr(m2,row,col);
    add_arr(m1,m2,m3,row,col);
    print_arr(m3,row,col);
    getch();
}
```

UNIT IV

➢ POINTERS

POINTERS: Pointers- concepts, initialization of pointer variables, pointers and function arguments, passing by address –dangling memory, address arithmetic, Character pointers and functions, pointers to pointers, pointers and multidimensional arrays, dynamic memory management functions, command line arguments.

Introduction to pointers

C provides the important feature of data manipulations with the address of the variables, the execution time is very much reduced such concept is possible with the special data type called Pointers.

A pointer is a variable that represents the location (rather than the value) of a data item, such as a variable of an array.

Pointers can be used to pass information back and forth between a function and its reference point.In particular, pointers provide a way to return multiple data items from a function via function arguments.

Pointers are also closely associated with arrays and therefore provide an alternative way to access individual array elements.

The computer memory is a sequential collection of storage cells.

This statement instructs the system to find a location for the integer variable 'a' and puts the value 50 in that location. Let us assume that the system has chosen the address location 2000 for a.

```
printf("a=%d address=%u\n"a, &a);
```

output:

```
    a=50   address=2000
```

Declaration:

```
        datatype  *ptname;
```

This tells the compiler three things about the variable ptname.

1. The * tells that the variable ptname is a pointer variable.
2. ptname needs a memory location.
3. ptname points to a variable of type datatype.

Ex:

```
    int *p;
```

Declares the variable p as a pointer variable, that points to an integer data type.
Remember that the type int refers to the datatype of the variable being pointed to by p and not the type of the pointer.
Similarly

```
            float  *x;
```

declares x as a pointer to a floating point variable.

Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement such as

P = &a;

which causes p to point to a.

i.e. p now contains the address of 'a'. This is known as pointer initialization. Before a pointer is initialized, it should not be used.

Pointer:- A pointer is a variable which stores the address of another variable.

Declaration:- Pointer declaration is similar to normal variable declaration but preceded by a *

Syntax:- data type *identifier;

Example:- int *p;

Initialization:- datatype *identifier=address;

Example:- int n;
int *p=&n;

NULL:-NULL pointer value(empty address)

Any type of pointer allocates two bytes of memory because it stores address of memory allocation.In c language the programme keep ([memory]) size is 64 kilo bytes.It is in unsigned integer range.

NOTE:- Any type of pointer it allocates 2 bytes memory. Because it stores address of memory location.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int *p1;
        char *p2;
        float *p3;
        double *p4;
        clrscr();
        printf("\n Size of int pointer:%d bytes", sizeof(p1));  //size of(*)
        printf("\n Size of char pointer:%d bytes", sizeof(p2));  //size of(*)
        printf("\n Size of float pointer:%d bytes", sizeof(p3));  //size of(*)
        printf("\n Size of double pointer:%d bytes", sizeof(p4));  //size of(*)
        getch();
}
```

FUNCTIONS AND POINTERS

Call by Reference:- The process of calling a function using pointers to pass the address of the variables is known as call by reference.

Program:

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

```
void main()
{
	void disp(int *);
	int n;
	clrscr();
	disp(&n);
	printf("%5d",n);
	getch();
}
void disp( int *x)
{
	printf("Enter any value into n:");
	scanf("%d",x);
}
```

**Program:	Swapping two values**

```
#include<stdio.h>
#include<conio.h>
void main()
{
	void swap(int *, int *);
	int a,b;
	clrscr();
	printf("Enter any two values:");
	scanf("%d%d",&a,&b);
	printf("\nBefore swaping:");
	printf("\na=%d",a);
	printf("\nb=%d",b);
	swap(&a,&b);
	printf("\nAfter swaping:");
	printf("\na=%d",a);
	printf("\nb=%d",b);
	getch();
}
void swap(int *x, int *y)
{
	int t;
	t=*x;
	*x=*y;
	*y=t;
}
```

Dangling Memory / Dangling Pointers

Dangling pointers arise when an object is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory. The pointer still points to the same location in memory even though the reference has since been deleted and may now be used for other purposes.

A straightforward example is shown below:

```
{
   char *dp = NULL;
   /* ... */
   {
       char c;
       dp = &c;
   } /* c falls out of scope */
     /* dp is now a dangling pointer */
}
```

If the operating system is able to detect run-time references to null pointers, a solution to the above is to assign 0 (null) to dp immediately before the inner block is exited. Another solution would be to somehow guarantee dp is not used again without further initialization.

Another frequent source of dangling pointers is a jumbled combination of malloc() and free() library calls: a pointer becomes dangling when the block of memory it points to is freed. As with the previous example one way to avoid this is to make sure to reset the pointer to null after freeing its reference—as demonstrated below.

```
#include <stdlib.h>

void func()
{
    char *dp = malloc(A_CONST);
    /* ... */
    free(dp);         /* dp now becomes a dangling pointer */
    dp = NULL;        /* dp is no longer dangling */
    /* ... */
}
```

Pointer arithmetic

Pointer arithmetic actually move the pointer reference by an arithmetic operation. i.e addition and subtraction.

For example:

```
 int x = 5, *ip;
ip = &x;
ip++;
```

From the above mentioned code, ip is an integer pointer that points address of integer variable x.

ip++ represents an increment operation performed on a pointer variable. it will increment by 2 bytes (since size of int is 2) and points to the next memory location.

Pointer arithmetic is very useful when dealing with arrays. Consider the following example:

```
#include <stdio.h>
void  main ()
{
   int array[] = { 45, 67, 89 };
   int * array_ptr;
   array_ptr = array;
   printf(" first element: %i\n", *(array_ptr++));
   printf("second element: %i\n", *(array_ptr++));
   printf(" third element: %i\n", *array_ptr);
   getch();
}
```

OUTPUT

```
first element: 45
second element: 67
 third element: 89
```

Similarly, decrement operation will move the pointer to previous memory cell.

ip= ip+n is equivalent to **ip= ip+ n * sizeof(data type(ip))**

ip= ip-n is equivalent to **ip= ip - n * sizeof(data type(ip))**

However, following operations are invalid upon pointers
Invalid pointer arithmetic include:
(i) Adding, dividing and multiplying two pointers
(ii) Adding double or float to pointer
(iii) Masking or shifting pointer
(iv) Assigning a pointer of one type to another type of pointer.

Character Pointers and Functions

A *string constant*, written as

"I am a string"

is an array of characters. In the internal representation, the array is terminated with the null character '\0' so that programs can find the end. The length in storage is thus one more than the number of characters between the double quotes.
Perhaps the most common occurrence of string constants is as arguments to functions, as in **printf("hello, world\n");**
When a character string like this appears in a program, access to it is through a character pointer; printf receives a pointer to the beginning of the character array. That is, a string constant is accessed by a pointer to its first element.
String constants need not be function arguments. If pmessage is declared as

char *pmessage;
pmessage = "now is the time";

assigns to pmessage a pointer to the character array. This is *not* a string copy; only pointers are involved. C does not provide any operators for processing an entire string of characters as a unit. There is an important difference between these definitions:

```
char amessage[] = "now is the time"; /* an array */
char *pmessage = "now is the time"; /* a pointer */
```

amessage is an array, just big enough to hold the sequence of characters and ’\0’ that initializes it. Individual characters within the array may be changed but amessage will always refer to the same storage. On the other hand, pmessage is a pointer, initialized to point to a string constant; the pointer may subsequently be modified to point elsewhere, but the result is undefined if you try to modify the string contents.

```
/* strcpy: copy t to s; pointer version 2 */
void strcpy(char *s, char *t)
{
while ((*s++ = *t++) != ’\0’)
;
}
```

This moves the increment of s and t into the test part of the loop. The value of *t++ is the character that t pointed to before t was incremented; the postfix ++ doesn’t change t until after this character has been fetched. In the same way, the character is stored into the old s position before s is incremented. This character is also the value that is compared against ’\0’ to control the loop. The net effect is that characters are copied from t to s, up and including the terminating ’\0’.

POINTERS AND ARRAYS

When an array is declared, the compiler allocates a BASE address and sufficient amount of storage and contain all the elements of array in continuous memory allocation. The base address is the location of the first element (index 0 of the array).The compiler also defines the array name as a constant pointer pointed to the first element.

suppose we declare an array 'a' as follows.

Example:- int a[5]={1,2,3,4,5};

Suppose the base address of a is 1000 and assuming that each integer requires 2 bytes. Then the 5 elements will be stored as follows.

```
 Elements------>  a[0] a[1] a[2] a[3] a[4]
                  -----------------------------
 values------->      1    2    3    4     5
                  ----------------------------
 address------>   1000 1002 1004 1006 1008
                   |
                   v
             base address
```

Here 'a' is a pointer that points to the first element. so the value of a is 1000 there fore a=&a[0]=1000

If we declare 'p' is an integer pointer, then we can make the pointer p to point to the array 'a' by the following assignment.

```
int *p;
p=&a[0]=a;
```

Now we can access every value of a using p++(or)p+1 to move one element to another.

The relationship between p and a is shown below.

p+0 = &a[0]=1000
p+1 = &a[1]=1002
p+2 = &a[2]=1004
p+3 = &a[3]=1006
p+4 = &a[4]=1008

When handling arrays instead of using array indexing, we can use pointer to access array elements. Note that *(p+k) gives the value of a[k]. Pointer accessing method is much faster than array indexing.

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n[5];
        int *p=n;
        clrscr();
        printf("n[0]=%u",&n[0]);
        printf("\nn=%u",n);
        printf("\nn[0]=%u",p);
        getch();
}
```

Program: Write a program to accept and display array elements using pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[20],n,i;
        int *p=&a[0];
        clrscr();
        printf("Enter no of elements:");
        scanf("%d",&n);
        printf("Enter array elements:");
        for(i=0;i<n;i++)
        {
                scanf("%5d",p+i);
        }
        printf("Given array elements:");
        for(i=0;i<n;i++)
        {
                printf("%5d",*(p+i));
        }
        getch();
}
```

POINTERS & 2D ARRAYS

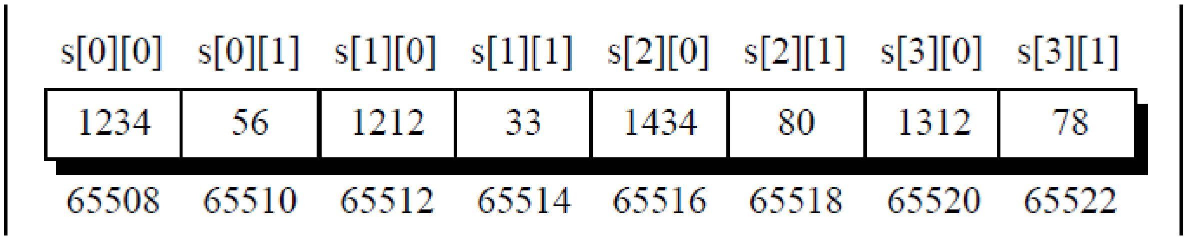
Each row of a two-dimensional array can be thought of as a one-dimensional array. This is a very important fact if we wish to access array elements of a two-dimensional array using pointers.

Suppose we want to refer to the element **s[2][1]** using pointers. We know that **s[2]** would give the address 65516, the address of the second one-dimensional array. Obviously (65516 + 1) would give the address 65518. Or **(s[2] + 1)** would give the address 65518. And the value at this address can be obtained by using the value at address operator, saying ***(s[2] + 1)**. But, we have already studied while learning one-dimensional arrays that **num[i]** is same as ***(num + i)**. Similarly, ***(s[2] + 1)** is same as, ***(*(s + 2) + 1)**. Thus, all the following expressions refer to the same element,

s [2][1]

* (s [2] + 1)

* (* (s + 2) + 1)



As an example,

&a[1][2] = *(a+1)+2

a[1][2] = *(*(a+1)+2)

Therefore, **& a[i][j] is equivalent to *(a+i)+j**

a[i][j] is equivalent to *(*(a+i)+j)

```
/* Pointer notation to access 2-D array elements */
 main( )
{

 int s[4][2] = { { 1234, 56 }, { 1212, 33 }, { 1434, 80 }, { 1312, 78 } } ;
int i, j ;
for ( i = 0 ; i <4 ; i++ )
{
 printf ( "\n" ) ;
for ( j = 0 ; j < 2  ; j++ )
 printf ( "%d ", *( *( s + i ) + j ) ) ;
}

}
```

And here is the output...

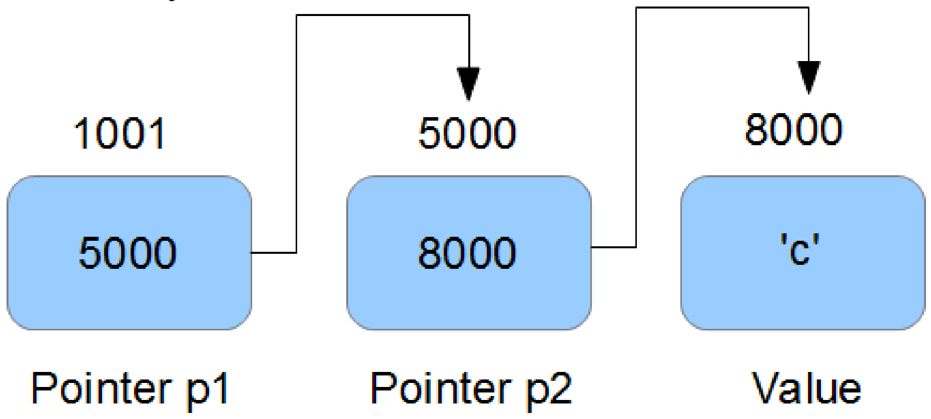
```
1234   56

1212   33
```

```
1434   80
1312   78
```

```
/*pointer notation to read and write a 2d array */
main()
{
  int a[3][3],i,j;
printf("Enter a 3X3 Matrix\n");
for(i=0 ; i<3 ; i++)
for(j=0 ; j<3; j++)
scanf("%d", *(a+i)+j);
printf("Displaying elements of 3X3 Matrix\n");
for(i=0 ; i<3 ; i++)
for(j=0 ; j<3 ; j++)
printf("%d", *(*(a+i)+j));
}
```

POINTER TO A POINTER

As the definition of pointer says that its a special variable that can store the address of an other variable. Then the other variable can very well be a pointer. This means that its perfectly legal for a pointer to be pointing to another pointer.
Lets suppose we have a pointer 'p1' that points to yet another pointer 'p2' that points to a character 'ch'. In memory, the three variables can be visualized as :



So we can see that in memory, pointer p1 holds the address of pointer p2. Pointer p2 holds the address of character 'ch'.
So 'p2' is pointer to character 'ch', while 'p1' is pointer to 'p2' or we can also say that 'p2' is a pointer to pointer to character 'ch'.
Now, in code 'p2' can be declared as :
char *p2 = &ch;
But 'p1' is declared as :
char **p1 = &p2;
So we see that 'p1' is a double pointer (ie pointer to a pointer to a character) and hence the two *s in declaration.

Now,

- 'p1' is the address of 'p2' ie 5000
- '*p1' is the value held by 'p2' ie 8000
- '**p1' is the value at 8000 ie 'c'

example :

```
#include<stdio.h>

int main(void)
{
   char **ptr = NULL;

   char *p = NULL;

   char c = 'd';

   p = &c;
   ptr = &p;

   printf("\n c = [%c]\n",c);
   printf("\n *p = [%c]\n",*p);
   printf("\n **ptr = [%c]\n",**ptr);

   return 0;
}
```

Pointers and Strings

Suppose we wish to store "Varma". We may either store it in a string or we may ask the C compiler to store it at some location in memory and assign the address of the string in a **char** pointer. This is shown below:

```
char str[ ] = "Varma" ;
char *p = "Varma" ;
```

There is a difference in usage of these two forms. For example, we cannot assign a string to another, whereas, we can assign a **char** pointer to another **char** pointer. This is shown in the following program.

```
main( )
{
char str1[ ] = "Hello" ;
char str2[10] ;
char *s = "Good Morning" ;
char *q ;
str2 = str1 ; /* error */
q = s ; /* works */
}
```

Also, once a string has been defined it cannot be initialized to another set of characters. Unlike strings, such an operation is perfectly valid with **char** pointers.

```
main( )
{
char str1[ ] = "Hello" ;
char *p = "Hello" ;
str1 = "Bye" ; /* error */
p = "Bye" ; /* works */
}
```

DYNAMIC MEMORY ALLOCATION

C language requires the no of elements in an array to be specified at compile time.But we may not be able to do so always our initial judgement of size,if it is wrong,it make cause failure of the program (or) wastage of the memory space.In this situation we use Dynamic Memory allocation.

Definition:-The process of allocating memory at run time is known as Dynamic memory allocation.

Malloc():- **<alloc.h>**

It is a function which is used to allocating memory at run time.

Syntax:- void *malloc(size_t size);

size_t:- unsigned integer. this is used for memory object sizes.

Pointer variable=(type casting)malloc(memory size);

Example:- int *p;

```
p=(int *)malloc(sizeof(int));  //for 1 location
p=(int *)malloc(n*sizeof(int));  //for n locations
```

Calloc():- **<alloc.h>**

This is also used for allocating memory at run time.

Syntax: void *calloc(size_t nitems, size_t size);

NOTE:- Calloc allocates a block (n times*size)bytes and clears into 0.

Difference between malloc and calloc:

1. Malloc defaultly store garbage value where as calloc defaultly stores zero
2. In malloc only one argurment we will pass where as in calloc we will pass two arguments

Program: *Write a program to create a dynamic array (vector) store values from keyboard display*

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
void main()
{
        int *a,n,i;
```

```
	clrscr();
	printf("Enter no of elements:");
	scanf("%d",&n);
	a=(int *)malloc(n* sizeof(int));
	//a=(int *)calloc(n,sizeof(int));
	printf("Enter array elements:");
	for(i=0;i<n;i++)
	{
		scanf("%d",a+i);
	}
	printf("Given array elements:");
	for(i=0;i<n;i++)
	{
		printf("%d\t",*(a+i));
	}
	getch();
}
```

Program: W*rite a program to create a dynamic memory allocation for two dimensional array and read values from keyboard and display*

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
void main()
{
	int **a,r,c,i,j;
	clrscr();
	printf("Enter no of rows and columns:");
	scanf("%d%d",&r,&c);
	a=(int **)malloc(r* sizeof(int *));
	//a=(int *)calloc(n,sizeof(int));
	for(i=0;i<r;i++)
	{
		*(a+i)=(int *)malloc(c*sizeof(int));
	}
	printf("enter array elements:\n");
	for(i=0;i<r;i++)
	{
	for(j=0;j<c;j++)
	{
		scanf("%d",*(a+i)+j);
	}
	}
	printf("Given array elements:");
	for(i=0;i<r;i++)
	{
```

```
                for(j=0;j<c;j++)
                {
                        printf("%3d\t",*(*(a+i)+j));
                }
                printf("\n");
        }
        free(a);
        getch();
}
```

Program: Write a program to create a variable to column size array in read values

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
void main()
{
        int **a,r,*c,i,j;
        clrscr();
        printf("Enter no of rows:");
        scanf("%d",&r);
        c=(int *)malloc(r* sizeof(int));
        //a=(int *)calloc(n,sizeof(int));
        for(i=0;i<r;i++)
        {
              printf("enter no of cols for row %d:",i+1);
              scanf("%d",c+i);
        }
        a=(int **)malloc(r* sizeof(int *));
        for(i=0;i<r;i++)
        {
                *(a+i)=(int *)malloc(*(c+i)*sizeof(int));
        }
        printf("enter array elements:\n");
        for(i=0;i<r;i++)
        {
        for(j=0;j<*(c+i);j++)
        {
                scanf("%d",*(a+i)+j);
        }
        }
        printf("Given array elements:\n\n");
        for(i=0;i<r;i++)
        {
                for(j=0;j<*(c+i);j++)
                {
                    printf("%3d\t",*(*(a+i)+j));
                }
                printf("\n");
        }
        free(a);
        getch();
}
```

Realloc():- It reallocates Main memory. ***Syntax:-*** void *realloc(void *block, size_t size);

free():- It deallocates the allocated memory. ***Syntax:-*** void free(void *block);

Program: *Write a program to demonstrate reaolloc*

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<string.h>
void main()
{
char *str;
clrscr();
str=(char *)malloc(8);
strcpy(str,"Hello");
printf("String is %s",str);
str=(char *)realloc(str,25);
strcat(str," demonstration of realloc");
printf("\n New string is %s",str);
free(str);
getch();
}
```

Write a C program to sort an numbers in ascending order using dynamic memory allocation.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *num,n,i,step,temp;
    printf("Enter total numeber.\n");
    scanf("%d",&n);
    num=(int*)malloc(n*sizeof(int));
    for(i=0;i<n;++i)
   {
      printf("Enter data%d:\n",i+1);
```

```
        scanf("%d",(num+i));
    }


for(step=0;step<n-1;++step)
        for(i=0;i<n-step;++i)
        {
          if(*(num+i)>*(num+i+1))
          {
              temp=*(num+i);
             *(num+i)=*(num+i+1);
             *(num+i+1)=temp;
          }
        }
    printf("In ascending order:\n");
    for(i=0;i<n;++i)
        printf("%d \t",*(num+i));
    return  0;
}
```

Command-line arguments

The C language provides a method to pass parameters to the main() function. This is typically accomplished by specifying arguments on the operating system command line (console).

The prototype for main() looks like:

```
int main(int argc, char *argv[])
{
…
}
```

There are two parameters passed to main(). The first parameter is the number of items on the command line (int argc). Each argument on the command line is separated by one or more spaces, and the operating system places each argument directly into its own null-terminated string. The second parameter passed to main() is an array of pointers to the character strings containing each argument (char *argv[]).

For example, at the command prompt:

```
test_prog 1 apple orange 4096.0
```

There are 5 items on the command line, so the operating system will set argc=5 . The parameter argv is a pointer to an array of pointers to strings of characters, such that:

argv[0] is a pointer to the string "test_prog"
argv[1] is a pointer to the string "1"
argv[2] is a pointer to the string "apple"
argv[3] is a pointer to the string "orange"
argv[4] is a pointer to the string "4096.0"

Example program for Command line arguments

```
1.  #include <stdio.h>
2.
3.  int main(int argc, char *argv[]) {
4.     if ( argc != 3) {
5.          printf("Usage:\n %s Integer1 Integer2\n",argv[0]);
6.     }
7.     else {
8.          printf("%s + %s = %d\n",argv[1],argv[2], atoi(argv[1])+atoi(argv[2]));
9.     }
10.   return 0;
11. }
```

Explanation

line 4 : we check if the user passed two arguments to the program. We actually need two arguments but in C the first argument (argv[0]) is the name of our program, so we need two more.

line 5 : If the user didn't pass two arguments we print the usage of our program and exit

line 8 : Using atoi() function we convert pointers to char (string) to decimal numbers and display their sum.

C program prints the number and all arguments which are passed to it.

```
#include <stdio.h>
void main(int argc, char *argv[])
{
 int c;
 printf("Number of command line arguments passed: %d\n", argc);

 for ( c = 0 ; c < argc ; c++)
printf("%d. Command line argument passed is %s\n", c+1,argv[c]);

}
```

/*program to find large and small from an array using a function*/

```
#include<stdio.h>
find(x,m,l,s)
int x[];
int m;
int *l,*s;
{
  int i;
  *l=*s=x[0];
  for(i=1;i<m;i++)
   {
     if (x[i]>*l)
          *l=x[i];
     if (x[i]<*s)
          *s=x[i];
   }
}

main()
{
   int a[10],n,i,lar,sma;
   clrscr();
   printf("enter n");
   scanf("%d",&n);
   printf("enter array elements\n");
   for(i=0;i<n;i++)
       scanf("%d",&a[i]);
   find(a,n,&lar,&sma);
   printf("largest=%d  smallest=%d\n",lar,sma);
   getch();
 }
```

/*program to interchange values of two variables using call by reference function call*/

```
#include<stdio.h>
interchange(x,y)
int *x;
int *y;
{
  int temp;
  temp=*x;
  *x=*y;
  *y=temp;
}
main()
{
   int a,b;
   clrscr();
   printf("enter a and b");
   scanf("%d%d",&a,&b);
   printf("before interchanging   a=%d  b=%d\n",a,b);
   interchange(&a,&b);  /*call by reference*/
   printf("After interchanging    a=%d  b=%d\n",a,b);
   getch();
 }
```

/*program to find no of words, characters and vowels in a line of text using a call by reference function call*/

```
#include<stdio.h>
void find(str,nw,nc,nv)
```

```
char str[];
int *nw,*nc,*nv;
{
  int i=0;
  *nw=0;
  *nc=0;
  *nv=0;
  while (str[i]!='\0')
  {
     putchar(str[i]);
     if (str[i]==' ')
            (*nw)++;
     else
       if ((str[i]=='a')||(str[i]=='e')||(str[i]=='i')
                                ||(str[i]=='o')||(str[i]=='u'))
               (*nv)++;
       else
               (*nc)++;
      i++;
  }
     (*nw)++;
}
main()
{
   char line[80],ch;
   int i,n,nw,nc,nv;
   clrscr();
   printf("enter a line of text\n");
   i=0;
   do
   {
      ch=getchar();
      line[i]=ch;
      i++;
   } while(ch!='\n');
   i--;
   line[i]='\0';
   printf("the line of text = %s\n",line);

       find(line,&nw,&nc,&nv);
       printf("The number of words       = %d\n",nw);
       printf("The number of characters = %d\n",nc);
       printf("The number of vowels      = %d\n",nv);
       getch();
   }
```

```
/*---------------------- program to find length of a string */
#include<stdio.h>
main()
{
   char str[20],ch;
   int n;
   char *p;
       clrscr();
       printf("enter a string");
       scanf("%s",str);
       printf("the string = %s\n",str);
       p=str;
       while(*p!='\0')
             p++;
       n=p-str;
       printf("The length of %s = %d\n",str,n);
       getch();
   }

/*-------------------------------------------------------
program to access elements of a single dimensional array
using a pointer variable
-------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
main()
{
  static int a[5] = { 1,2,3,4,5 };
  int i,*p;
  clrscr();

  printf("The array elements are (normal accessing)\n");
  for(i=0;i<5;i++)
    printf("%d is stored at %u\n",a[i],&a[i]);
  p=&a[0];      /*..........pointer initialization*/
  printf("The array elements are(pointer accessing)\n");
  for(i=0;i<5;i++)
    printf("%d is stored at %u\n",*(p+i),(p+i));
  getch();
}

/*-------------------------------------------------------
program to access elements of a single dimensional array
using a pointer variable
```

```
------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void display(x,n)
int *x,n;
{
  int i;
  for(i=0;i<n;i++)
    printf("%d is stored at %u\n",*(x+i),(x+i));
}
main()
{
  static int a[5] = { 1,2,3,4,5 };
  int i,*p;
  clrscr();
  printf("the array elements are\n");
  display(a,5);
  getch();
}

/*------------------------------------------------------
program to find largest and second largest from an array
using pointer variables
------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
void findlsl(x,n,l,sl)
int *x,n,*l,*sl;
{
  int i;
  *l = 0;
  *sl = 0;
  for(i=0;i<n;i++)
    {
          if ( *(x+i) > *l)
          {
            *sl = *l;
            *l = *(x+i);
          }
          else
          if ( *(x+i) > *sl )
             *sl = *(x+i);
    }
}
main()
```

```
{
  int a[10];
  int n,i,large,slarge;
  clrscr();
  printf("enter n");
  scanf("%d",&n);
  printf("enter array elements\n");
  for(i=0;i<n;i++)
     scanf("%d",&a[i]);
  findlsl(a,n,&large,&slarge);
  printf("the largest number  =%d\n",large);
  printf("the second largest number =%d\n",slarge);
  getch();
}

/*------------------------------------------------------
           program to find length of a string
-------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
main()
{
  char *name;
  char *p;
  int n;
  clrscr();
  name="VARMA";
  p=name;
  while( *p !='\0')
  {
     printf("%c is stored at position%u\n",*p,p);
     p++;
  }
  n=p-name;
  printf("the length of the string is %d\n",n);
  getch();
}

/*-------------------------------------------------------
           program to find sum of elements of a given array
           using a pointer variable
------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
addition(x,n,s)
```

```
int *x,n,*s;
{
  int i;
  for (i=0;i<n;i++)
     *s = *s + *(x+i);
}

main()
{
  static int a[5]={10,20,30,40,50};
  int sum=0;

  clrscr();
  addition(a,5,&sum);
  printf("the sum = %d\n",sum);
  getch();
}

/*-------------------------------------------------------
           program to find no of vowels in a line of text
           using a pointer variable
-------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
countvowels(str,n,nv)
char *str;
int n,*nv;
{
   int i;
   i=0;
   while( *str!='\0')
   {
     if((*str=='a')||(*str=='e')||(*str=='i')||(*str=='o')||(*str=='u'))
          (*nv)++;

          str++;
   }
}
main()
{
  char ch,line[80];
  int i,nv;
  clrscr();
  printf("enter a line of text\n");
  i=0;
  do
  {
    ch=getchar();
    line[i]=ch;
    i++;
  }while(ch!='\n');
  i--;
  line[i]='\0';
  printf("line of text=\n");
  printf("%s\n",line);
  nv=0;
  countvowels(line,i,&nv);
  printf("no of vowels = %d\n",nv);
  getch();
}

/*-------------------------------------------------------------
          program to find no of tabs,
          number of lines, uppercase characters,
          lowercase characters and blank spaces in a file
-------------------------------------------------------------*/
#include<stdio.h>
#include<conio.h>
count(str,nol,nouc,nolc,nobl,notb)
char *str;
int *nol,*nouc,*nolc,*nobl,*notb;
{
   while( *str!='\0')
   {
     if(*str>64 && *str<91)
          (*nouc)++;
     else
     if(*str>96 && *str< 123)
          (*nolc)++;
     else
     if(*str==' ')
          (*nobl)++;
     else
     if(*str=='\n')
          (*nol)++;
     else
     if(*str=='\t')
          (*notb)++;
   str++;
   }
```

```
}
main()
{
  char str[80];
  char ch;
  int i,nol,nouc,nolc,nobl,notb;
  clrscr();
  printf("enter text\n");
  i=0;
  do
  {
    ch=getchar();
    str[i]=ch;
    i++;
  }while(ch!=EOF);
  i--;
  str[i]='\0';
  printf("line of text=\n");
  printf("%s\n",str);
  nol=nouc=nolc=nobl=notb=0;
  count(str,&nol,&nouc,&nolc,&nobl,&notb);
  printf("no of lines                  = %d\n",nol);
  printf("no of upper case characters = %d\n",nouc);
  printf("no of lower case characters = %d\n",nolc);
  printf("no of blank characters       = %d\n",nobl);
  printf("no of tabs                   = %d\n",notb);
  getch();
}

/*---------------------------------------------
 program to pass an array to a function
 sequential search using pointers
---------------------------------------------*/
#include <stdio.h>
#include<conio.h>
void sequentialsearch(b,m,t,found,pos)
int b[];
int m,t;
int *found,*pos;
{
   int i=0;
   while(i<m)
   {
     if ( b[i] == t)
       {
           *found=1;
           *pos= i;
           break;
          }
      i++;
    }
}

main()
{
  int n,a[10],i,tar,found,pos;

  clrscr();
  printf("enter n");
  scanf("%d",&n);
  printf("enter array elements \n");
  for(i=0;i<n;i++)
    scanf("%d",&a[i]);

  printf("enter target ");
  scanf("%d",&tar);
  found=0;
  sequentialsearch(a,n,tar,&found,&pos);
  if(found)
     printf("the target found at position %d\n",pos);
  else
     printf("target not found\n");
  getch();
}


/*---------------------------------------------
 program to pass an array to a
function binary search
 --------------------------------------------*/
#include <stdio.h>
#include<conio.h>
void binsearch(b,m,t,found,pos)
int b[];
int m,t;
int *found,*pos;
{
   int low,high,mid;
   low=0;
   high=m-1;
   while(low<=high)
   {
     mid=(low+high)/2;
     if ( b[mid] == t)
       {
           *found=1;
           *pos=mid;
           break;
          }
           else
             if(t>b[mid])
               low=mid+1;
               else
                 high=mid-1;
   }
}
main()
{
  int n,a[10],i,tar,found,pos;
  clrscr();
  printf("enter n");
  scanf("%d",&n);
  printf("enter array elements in ascending order\n");
  for(i=0;i<n;i++)
    scanf("%d",&a[i]);
  printf("enter target ");
  scanf("%d",&tar);
  found=0;
  binsearch(a,n,tar,&found,&pos);
  if(found)
     printf("the target found at position %d\n",pos);
  else
     printf("target not found\n");
  getch();
}
```

POINTERS AND STRUCTURES

Arrow operator

```
#include <stdio.h>
#include <conio.h>
struct student
{
   int stno;
   char stname[20];
   int m1,m2,m3;
   int tot;
```

```
  float avg;
  char result[10];
};
typedef struct student stype;
void process(s)
stype *s;
{
    s->tot= s->m1 + s->m2 + s->m3;
    s->avg = (float)s->tot / 3;
    if ((s->m1 <40)|| (s->m2<40)|| (s->m3<40))
         strcpy (s->result,"fail");
    else
         strcpy (s->result,"pass");
}
main()
{
   stype st;
   clrscr();
   printf("enter stno name, m1,m2 m3");
   scanf("%d%s%d%d%d",&st.stno,st.stname, &st.m1,&st.m2,&st.m3);
   process(&st);
   printf("Total   = %d\n",st.tot);
   printf("Average = %f\n",st.avg);
   printf("Result  = %s\n",st.result);
   getch();
 }
```

****

UNIT V

ENUMERATED, STRUCTURE AND UNION TYPES:

Derived types- structures- declaration, definition and initialization of structures, accessing structures, nested structures, arrays of structures, structures and functions, pointers to structures, self referential structures, unions, typedef, bit-fields, program applications
BIT-WISE OPERATORS: logical, shift, rotation, masks.

Structure is user defined data type which is used to store heterogeneous data under unique name. Keyword 'struct' is used to declare structure.
The variables which are declared inside the structure are called as 'members of structure'.

Syntax:

```
struct structure_nm
{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;
}struct_var;
```

Example :

```
struct  employee
{
        char eid[10];
        char name[100];
        float sal;
}emp;
```

Note :
1. Structure is always terminated with semicolon (;).
2. Structure name as employee can be later used to declare structure variables of its type in a program.

Instances of Structure :

Instances of structure can be created in two ways as,

Instance 1:

```
struct employee
{
        char eid[10];
        char name[100];
        float sal;
}emp;
```

Instance 2:

```
struct employee
{
        char eid[10];
        char name[100];
        float sal;
};
struct employee emp;
```

In above example, employee is a simple structure which consists of structure members as Employee ID(eid), Employee Name(name), Employee Salary(sal).

Initializing structure

C programming language treats a structure as a custom data type therefore you can initialize a structure like a variable. Here is an example of initialize *product* structure:

```
struct product
{
char name[50];
char author[30];
 double price;
};
struct  product  book = { "C programming language",”surendra”,400.00};
```

In above example, we defined *product* structure, then we declared and initialized book as a product structure with its *name* and *price*.

Example :

```
void main( )
{
struct employee
{
char name[10] ;
int age ;
float salary ;
} ;
struct employee e1 = { "Sanjay", 30, 5500.50 } ;
struct employee e2, e3 ;
strcpy ( e2.name, e1.name ) ;
e2.age = e1.age ;
e2.salary = e1.salary ;
/* copying all elements at one go */
e3 = e2 ;
printf ( "\n%s %d %f", e1.name, e1.age, e1.salary ) ;
printf ( "\n%s %d %f", e2.name, e2.age, e2.salary ) ;
printf ( "\n%s %d %f", e3.name, e3.age, e3.salary ) ;
}
```

The output of the program would be...

```
Sanjay 30 5500.500000
Sanjay 30 5500.500000
Sanjay 30 5500.500000
```

Accessing of Structure Members:

Structure members can be accessed using member operator '**.**' . It is also called as '**dot operator**' or '**period operator**'.

structure_var.member;

Example: 1

```
#include<stdio.h>
#include<conio.h>
struct  emp
{
        int eno;
        char ename[20];
        float sal;
};
void main()
{
     struct emp e;
     clrscr();
     printf("Size of structre      :%d",sizeof(struct emp));
     printf("\nEnter employee number:");
     scanf("%d",&e.eno);
     printf("Enter employee name:");
```

```
	gets(e.ename);
	printf("Enter employee salary:");
	scanf("%f",&e.sal);
	printf("\nGiven employee number:%d",e.eno);
	printf("\nGiven employee name  :%s",e.ename);
	printf("\nGiven employee salary:%f",e.sal);
	getch();
}
```

Example-2:

```
/*PROGRAM TO EXPLAIN THE USE OF structures */
#include<stdio.h>
struct student
{
int rollno;
char name[10];
int m1;
int m2;
int m3;
};
void main()
{
struct student s;
int total;
clrscr();
printf("Enter the rollno");
scanf("%d",&s.rollno);
printf("Enter the name");
scanf("%s",s.name);
printf("Enter marks in m1");
scanf("%d",&s.m1);
printf("Enter marks in m2");
scanf("%d",&s.m2);
printf("Enter marks in m3");
scanf("%d",&s.m3);
total=s.m1+s.m2+s.m3;
printf("Student No:%d\n",s.rollno);
printf("Student Name:%s\n",s.name);
printf("Marks(M1):%d\n",s.m1);
printf("Marks(M2):%d\n",s.m2);
printf("Marks(M3):%d\n",s.m3);
printf("Total:%d",total);
getch();
}
```

Structures within Structures (Nested Structures) :

Structures can be used as structures within structures. It is also called as 'nesting of structures'.

Syntax:

```
struct structure_nm
{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;

        struct structure_nm
        {
                <data-type> element 1;
                <data-type> element 2;
                - - - - - - - - - - -
                - - - - - - - - - - -
                <data-type> element n;
        }inner_struct_var;
}outer_struct_var;
```

Example :

```
struct stud_Res
{
        int rno;
        char nm[50];
        char std[10];

        struct stud_subj
        {
                char subjnm[30];
                int marks;
        }subj;
}result;
```

In above example, the structure stud_Res consists of stud_subj which itself is a structure with two members. Structure stud_Res is called as 'outer structure' while stud_subj is called as 'inner structure.' The members which are inside the inner structure can be accessed as follow :

```
result.subj.subjnm
result.subj.marks
```

Example-1 :
/* Program to demonstrate nested structures. */

```
main( )
{
struct address
{
char phone[15] ;
char city[25] ;
int pin ;
} ;
struct emp
{
char name[25] ;
struct address a ;
} ;
struct emp e = { "varma", "99999", "vijayawada", 12 };
printf ( "\nname = %s phone = %s", e.name, e.a.phone ) ;
printf ( "\ncity = %s pin = %d", e.a.city, e.a.pin ) ;
}
```

And here is the output...

```
name = varma phone = 99999
city = vijayawada pin = 12
```

Notice the method used to access the element of a structure that is part of another structure. For this the dot operator is used twice, as in the expression,

e.a.pin or e.a.city

Example-2:
/* Program to demonstrate nested structures. */

```
#include <stdio.h>
#include <conio.h>
struct stud_Res
{
        int rno;
        char std[10];
        struct stud_Marks
        {
                char subj_nm[30];
                int subj_mark;
        }marks;
}result;
void main()
{
        clrscr();
```

```
	printf("\n\t Enter Roll Number : ");
	scanf("%d",&result.rno);
	printf("\n\t Enter Standard : ");
	scanf("%s",result.std);
	printf("\n\t Enter Subject Code : ");
	scanf("%s",result.marks.subj_nm);
	printf("\n\t Enter Marks : ");
	scanf("%d",&result.marks.subj_mark);
	printf("\n\n\t Roll Number : %d",result.rno);
	printf("\n\n\t Standard : %s",result.std);
	printf("\nSubject Code : %s",result.marks.subj_nm);
	printf("\n\n\t Marks : %d",result.marks.subj_mark);
	getch();
}
```

Array of Structures

Like any other data type structure arrays can be defined. So that each array element can be of structure data type. In our sample program, to store data of 10 employees we would be required to use 10 different structure variables from **e1** to **e10**, which is definitely not very convenient. A better approach would be to use an array of structures. Following program shows how to use an array of structures.

Without using Array of Structures	**Using Array of structures**

struct emp
{
int eno;
char ename[20];
float sal;
};
struct emp e1,e2,e3,e4,e5,e6,e7,e8,e9,e10;
	struct emp
	{
	int eno;
	char ename[20];
	float sal;
	};
	struct emp e[10];

Example:

/* Program to demonstrate array of structures. */

```
#include <stdio.h>
#include <conio.h>
struct  emp
{
	int eno;
	char ename[20];
```

```
        float sal;
};

void main()
{
struct emp e[10];
int  i:
printf("enter each employee details\n");
for(i=0;i<10;i++)
scanf("%d %s %f", &e[i].eno,e[i].ename,&e[i].sal);

printf("Displaying each employee details\n");
for(i=0;i<10;i++)
printf("%d %s %f", e[i].eno,e[i].ename,e[i].sal);
getch();
}
```

Array within Structures

Sometimes, it is necessary to use structure members with array.For example, consider student structure need an integer array to store the marks obtained by he/she in different subjects.

Consider the following example which illustrates Array with in structures

Example:

```
#include <stdio.h>
#include <conio.h>
struct  student
{
        int rno;
        int  marks[5];
}res;
void main()
{
        int i,total;
        clrscr();
        total = 0;
        printf("\n\t Enter Roll Number : ");
        scanf("%d",&res.rno);
        printf("\n\t Enter Marks of 3 Subjects : ");
        for(i=0;i<3;i++)
        {
                scanf("%d",&res.marks[i]);
                total = total + res.marks[i];
        }
        printf("\n\n\t Roll Number : %d",res.rno);
        printf("\n\n\t Marks are :");
        for(i=0;i<3;i++)
        {
                printf(" %d",res.marks[i]);
        }
        printf("\n\n\t Total is : %d",total);
        getch();
}
```

Structures and Functions

There are three methods by which the values of structure can be transferred from one function to another:

1. Passing each member of the structure as an actual argument of the function call.
2. Passing of a copy of the entire structure to the called function
3. Passing address of a structure as an argument to the function

1. Passing each member of the structure as an actual argument of the function

Like an ordinary variable, a structure variable can also be passed to a function.

Example:

```
void main()
{
struct book
{
char name[25] ;
char author[25] ;
float price ;
} ;
struct book b1 = { "C coding", "varma", 400.00 } ;
display ( b1.name, b1.author, b1.price ) ;
}
display ( char *s, char *t, float  n )
{
printf ( "\n%s %s %f", s, t, n ) ;
}
```

<u>OUTPUT</u>

```
C coding varma 400.00
```

2. Passing of a copy of the entire structure to the function

A better way would be to pass the entire structure variable at a time. This method is shown in the following program.

```
struct book
{
```

```
char name[25] ;
char author[25] ;
float price;
} ;

void main( )
{
struct book b1 = { "Compilers", "surendra", 500.00 } ;
display ( b1 ) ;
}
display ( struct book b )
{
printf ( "\n%s %s %f", b.name, b.author, b.price ) ;
}
```

OUTPUT

```
Compilers surendra 500.00
```

3. Passing address of a structure as an argument to the function

In this case, the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it.

```
/* Passing address of a structure variable */
struct book
{
char name[25] ;
char author[25] ;
float price;
} ;
main( )
{
struct book b1 = { "Operating Systems", "Surendra varma", 600.00 } ;
display ( &b1 ) ;
}
display ( struct book *b )
{
printf ( "\n%s %s %f", b->name, b->author, b->price ) ;
}
```

And here is the output...

```
Operating Sytems Surendra varma 600.00
```

Again note that to access the structure elements using pointer to a structure we have to use the **'->'** operator.

Also, the structure **struct book** should be declared outside **main ( )** such that this data type is available to **display ( )** while declaring pointer to the structure.

Pointers to structures

The way we can have a pointer pointing to an **int**, or a pointer pointing to a **char**, similarly we can have a pointer pointing to a **struct**. Such pointers are known as 'structure pointers'. Let us look at a program that demonstrates the usage of a structure pointer.

```
void main( )
{
struct book
{
char name[25] ;
char author[25] ;
float price ;
} ;
struct book b1 = { "Theory of Computation", "varma", 700.00 } ;
struct book *ptr ;
ptr = &b1 ;
printf ( "\n%s %s %f", b1.name, b1.author, b1.price ) ;
printf ( "\n%s %s %f", ptr->name, ptr->author, ptr->price ) ;
}
```

OUTPUT

```
Theory of Computation varma 700.00
Theory of Computation varma 700.00
```

The first **printf( )** is as usual. The second **printf( )** however is peculiar. We can't use **ptr.name** or **ptr.callno** because **ptr** is not a structure variable but a pointer to a structure, and the dot operator requires a structure variable on its left. In such cases C provides an operator **->**, called an arrow operator to refer to the structure elements. Remember that on the left hand side of the **'.'** structure operator, there must always be a structure variable, whereas on the left hand side of the **'->'** operator there must always be a pointer to a structure.

Self-Referential Structure

A self-referential structure contains a member which is a pointer to the same structure type.

Self-referential Structure is used in data structure such as binary tree, linked list, stack, Queue etc.

Example:

```
typedef struct node
 {
```

```
int data ;
struct node *next ;
} node, *list ;
```

Node is a type name equivalent to struct node. Similarly list is also a type name, equivalent to struct node *.

UNION

A Union is similar to a struct, except it allows you define variables that shares common storage space.

```
union union_nm
{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;
}union_var;
```

Example :

```
Union   employee
{
        char eid[10];
        char name[100];
        float sal;
}emp;
```

Program:

```
#include<stdio.h>
#include<conio.h>
struct  test1
{
int  n;
char ch;
float ft;
}t1;
union test2
{
int n;
char ch;
float ft;
}t2;
void main()
{
clrscr();
printf("\nSize of struct:%d Bytes",sizeof(t1));
printf("\nSize of Union:%d Bytes",sizeof(t2));
getch();
}
```

OUTPUT

```
Size of struct: 7 Bytes
Size of Union : 4 Bytes
```

Difference between Structure and Union:

Structure	Union
i. Access Members	
We can access all the members of structure at anytime.	Only one member of union can be accessed at anytime.
ii. Memory Allocation	
Memory is allocated for all variables.	Allocates memory for variable which variable require more memory.
iii. Initialization	
All members of structure can be initialized	Only the first member of a union can be initialized.
iv. Keyword	
'struct' keyword is used to declare structure.	'union' keyword is used to declare union.
v. Syntax	

struct struct_name
{
    structure element 1;
    structure element 2;
	union union_name
	{
	    union element 1;
	    union element 2;

---------- 	
---------- 	
structure element n;
}struct_var_nm;
	---------- 	
	---------- 	
	union element n;
	}union_var_nm;

vi. Example	

struct item_mst
{
int rno;
char nm[50];
}it;
	union item_mst
	{
	int rno;
	char nm[50];
	}it;

/* illustrating unions in C */

```
#include<stdio.h>
#include<conio.h>
union emp
{
        int eno;
        char ename[20];
        float sal;
}e;
void main()
{
     // union emp s;
     clrscr();
     printf("\nEnter employee number:");
     scanf("%d",&e.eno);
     printf("\n Employee number:%d",e.eno);
      printf("Enter employee name:");
     fflush(stdin);
     gets(e.ename);
     printf("\n Employee name  :%s",e.ename);
     printf("Enter employee salary:");
     scanf("%f",&e.sal);
     printf("\n Employee salaray:%f",e.sal);
     getch();
}
```

Typedef

A typedef declaration lets you define your own identifiers that can be used in place of type specifiers such as int, float, and double. A typedef declaration does not reserve storage. The names you define using typedef are not new data types, but synonyms for the data types or combinations of data types they represent.

Example 1

```
Typedef int number;
number length, width, height;
```

Example 2

typedef can also be used to define a struct type.

```
typedef struct
{
            int length;
            int width;
            int height;
} vehicle;

//The structure vehicle can then be used in the following declarations:

vehicle  car,bike,truck,bus;
```

Enumerated Datatype

Enumerated datatype offers us a way of creating own datatype. The created new datatype enables to assign symbolic names to integer constants.Variables of enumerated datatype are very often used in relation with structures and unions.

Syntax:

```
          enum identifier {value1,value2,.....,valuen};
```

enum is a keyword.

Identifier is any user-defined name.it is better if the identifier happens to a collective name set of values. Value-1,value-2,etc by default.

However, the default integer constants of the values can be overridden by assigning our own values.

Example:

```
enum Boolean {true,false};

enum Boolean a,b;

a=true;

b=false;
```

The compiler automatically assigns integer digits beginning with 0 to all enumeration constants. That is the enumeration constant true is assigned 0, false is assigned 1.

Program:

```
#include<stdio.h>

#include<conio.h>

void main()

{

enum colors{red,green,blue,black,white};

enum colors fav_col,luk_col;

clrscr();

fav_col=black;

luk_col=green;

printf("black=%d\n",fav_col);

printf("green =%d\n",luk_col);

getch();

}
```

Reflecting the

BIT-FIELDS:

C permits to use small bit fields to hold data.We have been using integer field of size 16 bits to store data. If a data item requires much less than 16 bits of memory space.In such situations there is a wastage of memory space. For this purpose we use bit fields in structures.

- The bit field data type is either int Or Unsigned int.
- The largest value that can store integer bit field is 2power(n-1).

- The largest value that can store unsigned integer bit field is 2power n -1.

Declaration:-
```
struct struct_name
{
  int (or) unsigned  identifier-1:bitlength;
  int (or) unsigned  identifier-2:bitlength;
  ----------------------------------------
  ----------------------------------------
  int (or) unsigned  identifier-n:bitlength;
  declaration of normal data items;
};
```

Example:-
```
struct emp
{
  unsigned eno:7;
  unsigned age:6;
  char ename[80];
  float sal;
};
```

NOTE:-The scanf function cannot read small bit fields. Because the scanf function scans and formats data into 2 bytes of address of the integer field.

Program:
```
#include<stdio.h>
#include<conio.h>
struct emp
{
        unsigned int eno:7;
        char ename[20];
        float sal;
};
void main()
{
        struct emp e;
        int n;
        clrscr();
        printf("Enter any emp no:");
        scanf("%d",&n);
        e.eno=n;
        printf("Enter any emp name:");
        gets(e.ename);
        printf("Enter emp salary:");
        scanf("%f",&e.sal);
        printf("Given emp no:%d",e.eno);
        printf("\nGiven emp name:%s",e.ename);
        printf("\nGiven emp salary:%f",e.sal);
        getch();
}
```

Bitwise operators:
Logical ,shift,rotation,masks.
BITWISE OPERATORS :- "These operators are used for manipulation of data at bit level". These operators interpret operands as sating of bits. Bits operations are performed on this data to get the bit strings. These bit strings are these interpreted according to data type. It operates only on integers. It may not be applied to float (or) real.
The bit wise operators are :-
1. LOGICAL OPERATORS

&	-	Bitwise	AND
|	-	Bitwise	OR
(Caret)^	-	Bitwise	XOR
(Tilde)~	-	Bitwise	NOT

LOGICAL OPERATIONS

A	B	A&B	A B	A^B	~A
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

Bitwise and (&):- This operator is represented as `&'. & operates on two operands of integer type. It takes two binary representations of equal length & performs logical AND operation on each pain of corresponding bits. It results `1' only if both are` 1'. Else will get `O'

Ex:-

```
1)      X = 7  =     0 0 0 0            0 1 1 1
        Y=8    =     0 0 0 0            1 0 0 0
        X&Y    =     0 0 0 0            0 0 0 0
2)      X      =     0 1 0 1
        Y      =     0 0 1 1
        X&Y    =     0 0 0 1
```

BIT WISE OR(|):-Takes two bit patterns of equal length, and produces another one of the same length by matching up corresponding bits and performing the logical inclusive OR operation on each pain of corresponding bits. The result will be `1' either if any one value is `1' or if both the values are `1'. Else result will be `O'. Otherwise the result is `O'.

Ex:-

```
1)      X = 7  =     0 0 0 0            0 1 1 1
        Y=8    =     0 0 0 0            1 0 0 0
        X|Y    =     0 0 0 0            1 1 1 1
2)      X      =     0 1 0 1
        Y      =     0 0 1 1
        X|Y    =     0 1  1 1
```

<u>Bitwise Exclusive OR (^)</u>:- The result in each position is `1' if the two bits are different, and O if they same & taken in the form as A + B.

Ex:-

1)	X = 13=	0 0 0 0	1 1 0 0
	Y=8 =	0 0 0 0	1 0 0 0
	X^Y =	0 0 0 0	0 1 0 1
2)	X =	0 1 0 1	
	Y =	0 0 1 1	
	X^Y =	0 1 1 0	

<u>Bitwise NOT (~)</u>:- (or) Compliment, is a unary operation that performs logical negation on each bit, forming the ones complement of the given binary value. Digits which were 0 becomes 1 and vice versa.

Ex:-

1)	X = 7 =	0 0 0 0	0 1 1 1
	Y=8 =	0 0 0 0	1 0 0 0
	~X,~Y =	1 1 1 1	1 0 0 0
	~X, ~Y=	1 1 1 1	0 1 1 1

2) **<u>Shift Operators</u>:-** In this, they operate on the binary representation of an integer instead of its numerical value. In this operation , the digits are moved, or shifted , to the left or right. Registers in a computer processor have a fixed number of available bits for storing numerals, so same bits will be "Shifted out" of register at one end, while the same number of bits are "shifted in" from the other end; the differences b/w bit shift operators lie in how they compute the values of those shifted in bits.

Ex:-

<u>Bitwise Left Shift (<<)</u>

		7 6 5 4 3 2 1 0
A (43):	0 0 0 0 0 0 0 0	0 0 1 0 1 0 1 1
A <<2:	0 0 0 0 0 0 0 0	1 0 1 0 1 1 0 0

<u>Bitwise Right Shift (>>)</u>

(+ ve Values)		7 6 5 4 3 2 1 0
A (43):	0 0 0 0 0 0 0 0	0 0 1 0 1 0 1 1
A >>2:	0 0 0 0 0 0 0 0	0 0 0 0 1 0 1 0

<u>Bitwise Right shif (>>)</u>

(- ve Values)			7 6 5 4 3 2 1 0
A (-44):	1 1 1 1	1 1 1 1	1 1 0 1 0 1 0 0
A >>2:	1 1 1 1	1 1 1 1	1 1 1 1 0 1 0 1

<u>Note</u>:- Right shift operator fills the left vacant fields with 'zeros' for (+ve) nos, with 'ones' for (-ve) nos.

3). **<u>ROTATION</u>:-**

<u>Rotate no Carry:</u>

<u>Circular shift</u>:-

Another form of shift is the circular shift (or) bit rotation. In this operation, the bits are "rotated" as if the left and right ends of the register were joined. The value that is shifted in on the right during a left – shift is whatever value was shifted out on the left, & vice-versa.

→ This operation is useful if it is necessary to retain all the existing bits, & is frequently used in digital cryptography.

Ex: Right circular shift / rotate

MSB LSB

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1

1	0	0	0	1	0	1	1

Left circular shift / rotate

MSB LSB

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1

0	0	1	0	1	1	1	0

ROTATE THROUGH CARRY:-

Is similar of the rotate no carry operation, but the two ends of the register are considered to be separated by the carry flag. The bit that is shifted in (on either end) is the old value of the carry flog, & the bit that is shifted out (on the other end) becomes the new value of the carry flog.

→ A single rotate through carry can simulate a logical (or) arithmetic shift of one position by setting up the carry flag beforehand.
FOR EX:- If the carry flag contains 0, then X Right - Rotate - Through - Carry- By - One is a logical right-shift, & if the carry flag contains a copy of the sign bit, then X- Right – Rotate – Through-Carry-By-One is an arithmetic right shift.

MSB LSB

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1

1	0	0	0	1	0	1	1

Right – Rotate through – carry

MSB LSB

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1

0	0	1	0	1	1	1	1

Left- Rotate-through-carry

4).**MASKS:-** A 'mask' is data that is used for bitwise operations.

→ Using a mask, multiple bits in a byte, nibble, word etc., can be set either on, off (or) inverted from on to off (or vice-versa) in a single bitwise operation

COMMON BIT MASK FUNCTIONS:-

MASKIN BITS TO I:- To turn certain bits on, the bitwise OR operation can be used, following the principle that

Y OR 1 = 1 & Y OR 0 = Y

To leave a bit unchanged , OR is used with a '0'.

Ex:- Turning on the 4th bit.

```
        1 0 0 1 1 1 0 1          1 0 0 1 0 1 0 1
   OR   0 0 0 0 1 0 0 0          0 0 0 0 0 0 0 0
   =    1 0 0 1 1 1 0 1          1 0 0 1 0 1 0 1
```

<u>MAKSING BITS TO 0</u>:-

These is no way to change a bit from on to off using the OR operation Instead, bitwise AND is used. When a value is ANDed with a '1', the result is simply the original value, as in :Y & 1 =Y. However, ANDing a value with 0 is guaranteed to return a 0, so it is possible to turn a bit off by ANDing with 0: Y AND 0=0. To leave the other bits alone, ANDing them with a 1 can be done.

Ex: Turning off the. 4th bit.

```
        1 0 0 1 1 1 0 1             1 0 0 1 0 1 0 1
AND     1 1 1 1 0 1 1 1             1 1 1 1 0 1 1 1
=       1 0 0 1 0 1 0 1             1 0 0 1 0 1 0 1
```

****

UNIT-VI

FILE HANDLING

Syllabus:
File handling: Input and output-concept of file, text files and binary files, formatted i/o, file i/o operations, example programs.

Introduction:
A computer stores program and data in secondary memory .Storing programs and data permanently in primary memory is not preferred due to the following reasons:
1. Primary memory is a volatile storage device, which losses its contents when the power goes off
2. Primary memory is usually too small to permanently store all the needed programs and data.

File:
A file is defined to be a collection of related data stored on secondary device like disk. Generally files represent programs and data. Data may be numeric , alphabetic or alphanumeric.
In general a file is a collection of bits, bytes, lines, or records whose meaning is defined by its creator and user. A file is named and is referred to by its name.
The operating system provides most of the common file manipulation services such as create,open,write,read, and close.

Need of a file:
Each program revolves around some data. It requires some inputs; performs manipulations over them ;produces required outputs.
The inputs originated from the standard input device, keyboard with the help of scanf(), getchar(),and gets.We displayed the outputs produced ,through the standard output device, screen by the use of I/O functions printf(),putchar(), and puts().

Character oriented I/O functions:
getchar()-The function getchar() is used to read a single character from the terminal. This function returns the integer value of the character in the machine's character code. If the system uses the ASCII character code, then the ASCII value of the character is returned.
variable=getchar();
putchar(): this function transmits a single character to a standard output device(monitor).The general form of putchar is
putchar(variable);

Example:
```
#include<stdio.h>
#include<conio.h>
void main()
{
char ch;
clrscr();
ch=getchar();
putchar();
getch();
}
```
Input-output:
a

Formatted I/O functions:
In our earlier program we use these two functions
scanf();
printf();

gets and puts functions:
gets() and puts() functions are used in strings.

The above type of I/O operations suffers from the following drawbacks.
1. The data involved in the I/O operations are lost when the program is excited or when the power goes off.
2. This scheme is definitely not ideal for programs, which involve voluminous data.
But, many real life programming circumstances require to deal with the large amount of data and require the data to be permanently available even when the program is excited or when power is switched off. This is where the concept of files comes to notice.

Types of files:
There are two types of files. Sequential file and random access file.
1.**Sequential file:** In this type data are kept sequentially. If we want to read the last record of the file it is expected to read all the records before it. It takes more time for accessing the records or if we desire to access the 10th record then the first nine records should be read sequentially for reading the 10th record.
2.Random access file: In this type data can be read and modified randomly. If the user desires to read the last records of a file, directly the same records can be read. Due to random access of data it takes access time less as compared to the sequential file.

Text files and binary files:
Text file: Text files are those files contain letters, digits, and symbols. In C default the file stream operations are performed in text mode.
Binary files: It is a sequential access file in which data are stored and read back one after another in the binary format instead ASCII format. When numerical data is to be transferred to disk from RAM the data occupies more memory space on disk. For example a number 234567 needs 3 bytes memory space in RAM and when transferred to disk requires 6 bytes memory space. For each numerical digit one byte space is needed. Hence total requirement for the number 234567 would be 6 bytes. Thus text mode is inefficient for storing large amount of numerical data because space occupation by ot is large. Only solution to this inefficient memory use is to open a file in binary mode, which takes lesser space than text mode
File pointer: The structure FILE which is declared in stdio.h acts like link between OS and the program and it is used to define a file pointer for use in file operations. Therefore a pointer to a FILE type is required to work with files on disk.
The syntax of declaring a file pointer is

FILE *fptr;
Here fptr is a file pointer of data type FILE.C is case sensitive , the word FILE should be typed in capital letters.
Operations on FILES: The commonly performed operations over files are the following:
1.Naming of file
2.Opening a file
3. Reading data from a file
4.Writing data to a file and
5.Closing a file
1.Naming of file: The file name must be specified for particular file. The file name is typically a string of charactes. The file names can be "data.txt","add.c" or "program.h" and so on.
The extension can be given depending upon the usage of files such as
1. If the file is used for storing text we use '.txt' extension.
2. If the file stores C program we give '.C' extension.
3. If it is a header file, then we give '.h' extension and so on.
Opening of file:
A file has to be opened before read and write operations. Opening of a file creates a link between the operating system and the file functions. The name of the file and its mode of operation are to be indicated to the operating system. The syntax is as follows:
fptr=fopen("filename","mode of opening");
Here file name is the name of the file being opened. Mode of opening can be any one of the following:

Text files	
r	Opens file(text)for reading only
w	Opens file(text) for writing only
a	Opens file(text)for appending only
r+	Opens file(text) for reading and writing
w+	Opens file(text) for reading and writing
a+	Append a text file for read/write

Binary files	
rb	Opens a binary file for reading only
rb+	Opens a binary file for reading and writing
wb	Opens a binary file for writing only
wb+	Opens a binary file for reading and writing

ab	Opens a binary file for appending only
ab+	Opens a binary file for read/write

Example: FILE *fp;
fp=fopen("data.txt","r");
Here fp is a pointer variable contains the address of the structure FILE that has been defined in the header file stdio.h. It is necessary to write FILE uppercase . The function fopen() will open a file "data.txt" in read mode. The computer reads the contents of the file because it finds the read mode ("r"), Here "r" is a string and not a character. Hence it is enclosed with double quotes and not with single quotes.
The fopen() performs the following important tasks

1. It searches the disk for opening the file.
2. In case the file exists, it loads the file from the disk into memory . If the file is found with huge contents then it loads the file part by part.
3. If the file not existing this function returns a NULL. NULL is a macro defined in the header file stdio.h

<u>Closing of file:</u>
The function fclose() is used to close the file. It is the counterpart of fopen(). Closing file means de linking the file from the program and saving the contents of the file.
The syntax is as follows:
fclose(fp);
Where fp is a pointer to FILE type and represents a file. The file represented by fp is closed. If an error occur in closing a file then fclose() returns EOF. For successful close operation it returns 0.
Example: FILE *fp;
fptr=fopen("data.txt","r");
:
:/* perform operations on file*/
fclose(fp);/* disassociate the file with the stream*/

<u>FILE I/O functions:</u>
After a file is opened ,we can read data stored in the file or write new data onto it depending on the mode of opening .C standard library supports a good number of functions which can be used for performing I/O operations.
<u>Unformatted file I/O functions</u>:
fputc() and fgetc() are character oriented file I/O functions.
fputs() and fgets() are string oriented file I/O functions.
<u>Formatted file I/O functions:</u>
fprintf() and fscanf() are mixed data oriented file I/O functions.
<u>Character oriented I/O functions:</u>
fputc() is to write a character onto a file. The syntax is as follows.
fputc(ch,fptr);
Where ch represents a character and fptr, apointer to FILE ,represents a file. The function writes the content of ch onto the file repreented by fptr.

fgetc() is to read a character from a file. The syntax is as follows:

ch=fgetc(fptr);

ch is a variable of char type and fptr is pointer to FILE type. The function reads a character from the file denoted by fptr and returns the character value, which is collected by the variable ch.

The file pointer moves by one character position for every operation of fgetc() or Fputc(). The fgetc() will return an end of file marker EOF, when end of the file has been reached. Therefore the reading should be terminated when EOF is encountered.

Examples:

To create a file consisting of characters using fputc()

```
#include<stdio.h>
#include<conio.h>
void main();
{
char ch;
FILE *fptr;
clrscr();
fptr=fopen("text.dat","w");
printf("input data \n");
ch=getchar();
while(ch!=EOF)
{
fputc(ch,fptr);
ch=getchar();
}
fclose(fptr);
getch();
}
```

Input-output:

```
Input data
My name is computer
```

Explanation: fptr is declared to be a pointer variable to FILE type and ch is declared to be a variable of cahr type. The file pointer variable is to represent the file to be created by the program and the variable ch of char type is to collect characters one at a time, entered through the standard input device,keyboard. Note that the external file "text.dat" is opened in "w" mode.

Till we type an EOF character (which is control-Z) all the characters are written into the external file "text.dat" denoted by fptr in the program. Once the character EOF is typed,the loop terminates. Note that the file is then closed by invoking fclose().

2. To read a file consisting of characters using fgetc()

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fptr;
char ch;
clrscr();
printf("the contents of th file 'text .dat' are :\n");
fptr=fopen("text.dat","r");
while(!feof(fptr))
{
ch=fgetc(fptr);
putchar(ch);
}
fclose(fptr);
getch();
}
```

Input-output:

```
The contents of the file 'text.dat' are:
My name is computer
```

Explanation:fptr is declared to be a pointer variable to FILE type and ch is declared to be a variable of char type. The file pointer variable is to represent the file("text.dat") to be read by the program and the variable ch of char type is to collect characters one at a time, read from the file. Note that the external file "text.dat" is now opened in "r" mode.
The function feof() is to check for the end of a file being read. It takes the file pointer to the file as the argument and returns 1. If the end is reached . Otherwise it returns 0. In the above looping segment, till the end of the file "text.dat" is reached all the characters are read and displayed. The file is then closed.

3 . Write a program to write data to text file and read it.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
FILE *fp;
char c=' ';
clrscr();
fp=fopen("data.txt","w");
if(fp==NULL)
{
printf("can not open");
exit(1);
}
printf("write data & to stop press '.':");
while(c!='.')
{
c=getchar();
fputc(c,fp);
}
fclose(fp);
printf("\n contents read:");
fp=fopen("data.txt","r");
while(!feof(fp))
printf("%c",getc(fp));
}
```

Input-output:

```
Write data & to stop press '.': ABCDEFGHIJK
Contents read: ABCDEFGHIJK
```

4.Write a program to open a pre existing file and add information at the end of file .Display the contents of the file before and after appending.

```
 #include<stdio.h>
#include<conio.h>
#include<process.h.
void main()
{
FILE *fp;
char c;
clrscr();
printf("contents of file before appending:\n");
fp=fopen("data.txt","r");
while(!feof(fp))
{
c=fgetc(fp);
printf("%c",c);
}
fp=fopen("data.txt","a");
if(fp==NULL)
{
printf("file can not appended");
exit(1);
}
printf("\n enter string to append:");
while(c!='.')
{
c=getchar();
fputc(c,fp);
}
fclose(fp);
printf("\n contents of file after appending:\n");
fp=fopen("data.txt","r");
while(!feof(fp))
{
c=fgetc(fp);
printf("%c",c);
}
}
```
Input-output:
```
Contents of file before appending:
Computer
Enter string to append:
System
Contents of file after appending
Computer system
```
5.Write a program to use w+ mode for writing and reading of file
```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
FILE *fp;
char c='.';
clrscr();
fp=fopen("data.txt","w+");
if(fp==NULL)
{
printf("can not open file");
exit(1);
}
printf("write data & to stop press '.':");
while(c!='.')
{
c=getchar();
fputc(c,fp)
}
rewind(fp);
printf("\n contents read:");
while(!feof(fp))
printf("%c",getc(fp));
}
```
Input-output:
```
Write data & to stop '.':ABCDE
Contents read: ABCDE
```
Explanation: Instead of using separate read and write modes, one can use w+ mode to perform both the operations. Hence, in the above program w+ is used . At first writing operation is done . It is not essential to close the file for reading the contents of it. We need to set the character pointer at the beginning . Hence rewind() function is used. The advantage of using w+ is to reduce the number of statements.

6. Write a program to open a file in append mode and add new record in it.
```
#include<stdio.h>
#include<conio.h>
```

```
#include<process.h>
void main()
{
FILE * fp;
char c='.';
clrscr();
fp=fopen("data.txt","a+");
if(fp==NULL)
{
printf("can not open file");
exit(1);
}
printf("write data & to stop press '.':");
while(c!='.')
{
c=getchar();
fputc(c,fp);
}
printf("\n contents read:");
rewind(fp);
while(!feof(fp))
printf("%c",getc(fp));
}
```

Input-output:

```
Write data &/ to stop press '.': this is append and read mode
Contents read: this is append and read mode
```

7. Write a program to open a file in read/write mode in it. Read and write new information in the file.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
FILE * fp;
char c='.';
clrscr();
fp=fopen("data.txt","r+");
if(fp==NULL)
{
printf("can not open file");
exit(1);
}
printf("\n contents read:");
while(!feof(fp))
printf("%c',getc(fp));
printf("write data & to stop press '.':");
while(c!='.')
{
c=getchar();
fputc(c,fp);
}
}
```

Input-output:

```
Contents read: computer
Write data & to stop press '.': system
```

8. Write a program to open a file for read/write operation in binary mode. Read and write new information in the file.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
FILE * fp;
char c='.';
clrscr();
fp=fopen("data.txt","wb");
if(fp==NULL)
{
printf("can not open file");
exit(1);
}
printf("write data & to stop press '.':");
while(c!='.')
{
c=getchar();
fputc(c,fp);
}
fclose(fp);
fp=fopen("data.dat","rb");
printf("\n contents read");
while(!feof(fp))
printf("%c",getc(fp));
}
```

String oriented functions:

The fputs() is to write a string onto a file. The syntax is as follows

fputs(buffer,fptr);

Buffer is the name of a character array and fptr is a pinter to FILE type. The function writes the string represented by buffer to the file pointed to by fptr.

fgets(buffer,size,fptr)

Buffer is the name of a character array, size is an integer value, fptr is a pointer to FILE type . The function reads a string of maximum size-1 characters from the file pointed to by fptr and copies it to the memory area denoted by buffer.

Example: To create a file consisting of strings using fputs()

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str[10];
FILE * fptr;
int I,n;
clrscr();
fptr=fopen("student.dat","w");
printf("enter number of students\n");
scanf("%d",&n);
printf("enter %d students \n",n);
fflush(stdin);
for(i=1;i<=n;i++)
{
gets(str);
fputs(str,fptr);
}
fclose(fptr);
getch();
}
```

Input-output:

```
Enter number of students
2
Enter  2 students
Chandu
Saaketh
```

2.To read a file consisting of strings using fgets()

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE * fptr;
char str[10];
clrscr();
printf("strings are \n");
fptr=fopen("student.dat","r");
while(!feof(fptr))
{
fgets(str,10,fptr);
puts(str);
printf("\n");
}
fclose(fptr);
getch();
}
```

Input-output:

```
Strings are:
Chandu
Saaketh
```

Mixed data oriented functions:

fprintf() is to write multiple data items which may(or not) be of different types to a file. The syntax is as follows:

fprintf(fptr,"control string",argumentslist);

It can be observed that syntax of fprintf() is similar to that of printf() except the presence of an extra parameter fptr, a pointer to FILE type. The parameter fptr associated with a file that has been opened for writing . Control string specifies the format specifiers . Arguments list contains comma separated variables, the value of which are to be written to the file.

fscanf() is to read multiple data items which may be of different types from a file. The syntax is as follows

fscanf(fptr,"control string",argumentlist);

It can be observed that syntax of fscanf() is similar to that of scanf() except the presence of an extra parameter fptr, a pointer to FILE type. The parameter fptr associated with a file that has been opened for reading. Control string specifies the format specifiers ,argument list contains comma separated variables preceded by address of operator &(except in the case of string type) into which data read from the file are to be opened.

Examples:

To create a file consisting of students details using fprintf()

```
#include<stdio.h>
#include<conio.h>
struct student
{
int rollno;
char name[10];
float per;
};
void main()
{
FILE *fptr;
struct student x;
int I,n;
fptr=fopen("student.dat","w");
clrscr();
printf("enter number of students \n");
scanf("%d",&n);
printf("enter %d students details \n",n);
for(i=1;i<=n;i++)
{
scanf("%d %s %f",&x.rollno,x.name,&x.per);
fprintf(fptr,"%d %s%f",x.rollno,x.name,x.per);
}
fclose(fptr);
getch();
}
```

Input-output:

```
Enter number of students
2
Enter 2 student details
1 chandu 90
2 saaketh 80
```

Explanation: The structure struct student is defined with the fields rollno,name and percent. In the main() ,variable fptr is declared to be a pointer to FILE type and it is to denote the file student.dat to be created by the program . x is declared to be a variable of struct student type, which is to collect the student details accepted through the keyboard. The statement fptr=fopen("student.dat","w"); opens the file student.dat in write mode and assigns reference to the file to the file variable fptr. The for loop segment enables us to accept n students details and write them onto the file student.dat.

2.To read a file consisting of students details using fscanf()

```
#include<stdio.h>
#include<conio.h>
struct student
{
int rollno;
char name[10];
float per;
};
void main()
{
FILE *fptr;
struct student x;
clrscr();
printf("contents of the file student.dat \n");
fptr=fopen("student.dat","r");
while(!feof(fptr))
{
fscanf(fptr,"%d %s %f",&x.rollno,x.name,&x.per);
printf("%3d %20s%9.2 f \n",x.rollno,x.name,x.per);
}
fclose(fptr);
getch();
}
```

Input-output:

Contents of the file student.dat

```
1 chandu 90.00
2 saaketh 80.00
```

3.Write a program to open a text file and write some text using fprintf() function. Open the file and verify the contents,

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE * fp;
char text[30];
fp=fopen(“text.txt”,”w”);
clrscr();
printf(“enter the text here:”);
gets(text);
fprintf(fp,”%s”,text);
fclose(fp);
}
```

Output:

```
Enter text here : have a nice day
```

4.Write a program to enter data into the text file and read the same . Use “w+” file mode.Use scanf() to read the contents of the file.

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
char text[15];
int age;
fp=fopen(“text.txt”,”w+”);
clrscr();
printf(“name \t age \n”);
scanf(“%s %d”,text,&age);
fprintf(fp,”%s %d’,text,age);
printf(“name \t age \n”);
fscanf(fp,”%s %d”,text,&age);
printf(“%s \t %d \n”,text,age);
fclose(fp);
}
```

Output:

```
Name  age
Chandu        2
Saaketh       2
```

Reading and writing structures:

The main disadvantage of fprintf() is all data type are treated as characters , for example consider the data 1234 ,occupies two bytes in memory but when it is transferred to the disk file using fprintf() function it would occupy 4 bytes of memory. For each character one byte would be required. Even for float also each digit including dot(.) requires one byte. Thus large amount of integers or float data requires large space on the disk. Hence in text mode storing of numerical data on the disk turns out to be inefficient.

To overcome this problem the file should be read and write in binary mode, using the functions fread() and fwrite()

fwrite() function is used for writing a structure block to a given filr. The syntax is given below.

fwrite(& structure_variable,sizeof(structure),integer,filepointer);

The first argument is the address of the structure to be written to the disk, the second argument is the size of the structure in bytes, the third argument is the number of such structure(records) that the user wants to write at one time and the last argument is the file pointer.

fread() function is used for reading an entire structure block from a given file. The syntax is same as fwrite() function.

fread(&structure_variable,sizeof(structure),integer,filepointer);

Examples:

1.To create a file consisting of student details using fwrite()

```
#include<stdio.h>
#include<conio.h>
struct student
{
int rollno;
char name[10];
float per;
};
void main()
{
FILE *fptr;
struct student x;
int I,n;
clrscr();
fptr=fopen(“student.dat”,”wb”);
printf(“enter number of students \n”);
scanf(“%d”,&n);
printf(“enter %d students details \n”,n);
for(i=1;i<=n;i++)
{
```

```
scanf("%d %s %f",&x.rollno,x.name,&x.per);
fwrite(&x,sizeof(x),1,fptr);
}
fclose(fptr);
getch();
}
```

Input-output:

```
Enter number of students
2
Enter 2 student details
1 chandu 90
2 saaketh 80
```

2.To read from a file consisting of students details using fread()

```
#include<stdio.h>
#include<conio.h>
struct student
{
int rollno;
char name[10];
float per;
};
void main()
{
FILE *fptr;
struct student x;
clrscr();
printf("contents of the file student.dat \n");
fptr=fopen("student.dat","r");
while( fread(&x,sizeof(x),1,fptr))
{
printf("%d%s%f \n",x.rollno,x.name,x.per);
}
fclose(fptr);
getch();
}
```

Input-output:

```
Contents of the file student.dat
1 chandu 90.00
2 saaketh 80.00
```

Random accessing of files:

All the programs, which have been written so far reading the contents of files, employed sequential access mode. That is the contents were read from the beginning of the files till their end is reached in a serial manner. Many a time we need to access the contents at particular positions in the files not necessarily from the beginning till the end. This is refered to as random accessing of files.

For accessing data randomly from a file basically ftell(),fseek(),and rewind() functions are used.

fseek(): The function fseek() is used for setting the pointer position in the file at the specified position. The syntax of it

Usage is as follows:

fseek(fptr,displacement,pointer_position);

Where fptr is the pointer which points to the file. Displacement is positive or negative. This is te number of bytes which are skipped backward (if negative) or forward (if positive) from the current pointer position. The third parameter pointer_position can take one of the following three values 0,1 and 2. The meaning of the values are given in the following table:

Pointer_position	Macro name	Meaning
0	SEEK_SET	Beginning of file
1	SEEK_CUR	Current position of file
2	SEEK_END	End of file

The displacement is attached with L because this is a long integer. The following illustrate the operation of the fseek function:

Function	Meaning
fseek(fptr,0L,0)	Goto the beginning
fseek(fptr,0L,1)	Stay at the current position
fseek(fptr,0L,2)	Go to the end of the file,past the last character of the file
fseek(fptr,n,0)	Move to (n+1)th byte in the file
fseek(fptr,n,1)	Go forward by n bytes
fseek(fptr,-n,1)	Go backward by n bytes from the current position
fseek(fptr,-n,2)	Go backward by n bytes from the end

ftell(): the function ftell() returns the value of the current pointer position in the file. The value is count from the beginning of the file. The syntax of its usage is as follows:

position=ftell(fptr);

Where fptr is the pointer which points to the file

rewind(): the function rewind() is used to move the file pointer to the beginning of a file. The syntax of its usage is as follows:

rewind(ptr);

Examples:

1.Write a program to print the current position of the file pointer in the file using ftell() function

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
char ch;
fp=fopen("text.txt","r");
fseek(fp,21,SEEK_SET);
ch=fgetc(fp);
clrscr();
while(!feof(fp))
{
printf("%c\t",ch);
printf("%d\n",ftell(fp));
ch=fgetc(fp);
}
fclose(fp);
}
```

Output:

```
w 22
o 23
r 24
```

Explanation: In the above fseek() function sets the cursor position on byte 21. The fgetc() function in the while loop reads the character after 21st character. The ftell() function prints the current pointer position in the file.When feof() function is found at the end of file,program terminates.

2.Write a program to show how rewind() function works.

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
char c;
fp=fopen("text.txt","r");
clrscr();
fseek(fp,12,SEEK_SET);
printf("pointer is at %d\n",ftell(fp));
printf("before rewind():");
while(!feof(fp))
{
```

```
c=fgetc(fp);
printf("%c",c);
}
printf("\b after rewind():");
rewind(fp);
while(!feof(fp))
{
c=fgetc(fp);
printf("%c",c);
}
}
```

3.To count the number of records in student file

```
#include<stdio.h>
#include<conio.h>
struct student
{
int rollno;
char name[10];
float per;
};
void main()
{
FILE *fptr;
int no_rec,last_byte;
clrscr();
fptr=fopen("student.dat","r");
fseek(fptr,0,SEEK_END);
last_byte=ftell(fptr);
no_rec=last_byte/sizeof(struct student);
printf("no of records=%d",no_rec);
fclose(fptr);
getch()
}
```

Input-output:

```
No of records=2
```

Error handling during file I/O operations:

The file I/O operations can not always be expected to be smooth sailing. During the course of I/O operations some errors may be occurred . As a consequence of the error conditions the underlying program may prematurely terminate or it may produce erroneous results.

Following are the situations under which the file I/O operations fail:

1.Trying to use a file that has not been opened

2. Trying to open a file, which does not exist for reading purpose.

3. Trying to open a file for writing purpose when there is no disk space

4 , Trying to write a read only file
5. Trying to perform an operation on a file when the file has been opened for some other purpose.

Example: To illustrate error handling during file I/O operations

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
FILE *fptr;
clrscr();
fptr=fopen("employees.dat","r");
if(fptr==NULL)
{
printf("the file employees.dat does not exist");
getch();
fptr=fopen("students.dat","w");
if(fptr==NULL)
{
printf("the read only file students.dat can not be opened in write mode");
getch();
}
fptr=fopen("characters.dat","r");
fputc('a',fptr);
if(ferror(fptr))
{
printf("the file characters.dat has been opened in read mode");
printf("but you are writing to itr \n");
getch();
}
getch();
}
```

Input-output:

```
The file employees.dat does not exist
The readonly file students.dat can not be opened in write mode
The file characters.dat has been opened in read mode but you are writing to it.
```

<u>Command line parameters(arguments):</u>

We know that the parameters play a vital role of establishing data communication between a calling function and a called function . It is through parameters a calling function passes inputs to a called function, which in turn performs required manipulations over them. So far we have discussed passing parameters to functions other than main(). It is important to note that we can even pass parameters to main() also. If the main() is to be defined to accept parameters the actual parameters should be provided at the command prompt itself along with the name of the executable file. Hence these parameters are called command line arguments.

For example consider COPY command of DOS, which copies the contents of source file to the destination file.
To copy the contents of emp1.dat to emp2.dat,we use the following command:
c:\>copy emp1.dat emp2.dat
Here copy is the program file name(executable) and emp1.dat is the source file and emp2.dat is the destination file.
To make main() of a program take command line parameters the function header will have the following form:
void main(int argc,char *argv[ ])
Here argc and argv are the formal parameters which provide mechanism for collecting the parameters given at command line when the program is launched for execution.
The argc is to collect the number of parameters passed and the array of pointers to char type argv is to collect the parameters themselves.
argv[0] will always represent the program file name(executable) and
argv[1],argv[2]…………..represent parameters passed to the main().
<u>Example:</u>
Write a program for copying one file to another file using command line arguments

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main(int argc, char *argv[])
{
FILE *fptr1,*fptr2;
char ch;
clrscr();
if(argc!=3)
{
printf("invalid number of parameters");
exit(1);
}
fptr1=fopen(argv[1],"r");
fptr2=fopen(argv[2],"w");
while(!feof(fptr1)
{
ch=fgetc(fptr1);
fputc(ch,fptr2);
}
fclose(fptr1);
fclose(fptr2);
getch();
}
```

Input-output
The program is executed as follows:
C:\>copy emp1.dat emp2.dat
As a result the contents of emp1.dat are copied to emp2.dat.

Example programs:

1.Write a C program to reverse the first n characters in a file(note: the file name and n are specified on the command line)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
void main(int argc,char *argv[ ])
{
char a[25];
char s[25];
char n;
int k;
int j=0;
int i;
int len;
FILE *fptr;
clrscr();
if(argc!=3)
{
puts("improper number of arguments.\n");
exit(0);
}
fptr=fopen(argv[1],"r");
if(fptr==NULL)
{
puts("file cannot be opened \n");
exit(0);
}
k=*argv[2]-48;
n=fread(a,1,k,fptr);
a[n]='\0';
len=strlen(a);
for(i=len-1;i>=0;i--)
{
s[j]=a[i];
printf("%c",s[j]);
j=j+1;
}
s[j+1]='\0';
getch();
}
```

2. Write a C program to read a text file and to count
a. number of characters b. number of words
c. number of sentences and write in an output file.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char ch;
int nos,noupc,noloc;
FILE *fptr1;
clrscr();
now=nos=noupc=noloc=0;
fptr1=fopen("input.txt","r");
if(fptr1==NULL)
printf("file doesnot exists \n");
else
{
while(1)
{
ch=getc(fptr1);
if(ch==EOF)
break;
if(ch>64 && ch<91)
noupc++;
else if(ch>96 && ch<123)
noloc++;
else if(ch==' ')
now++;
else if(ch=='\n' &&(isupper(ch)||islower(ch))
nos++;
}
}
fptr2=fopen("output.txt","w");
fprintf(fptr2,"no of upper and lower case character s:%d",noupc,noloc);
fprintf(fptr2,"no of words : %d",now);
fprintf(fptr2,"no of sentences : %d",nos);
fclose(fptr1);
fclose(fptr2);
getch();
}
```

3.Write a c program to read last 'n' characters of the file using appropriate file function

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
FILE *fptr;
char ch;
int n;
clrscr();
fptr=fopen("characters.txt","r");
printf("enter no of characters");
scanf("%d",&n);
fseek(fptr,n,2);
if(fptr==NULL)
{
printf("can not open this file");
exit(0);
}
ch=getc(fptr);
while(ch!=EOF)
printf("%c",ch);
fclose(fptr);
getch();
}
```

4.Write a C program to read a text file and convert the file contents in capital and write the contents in a output file

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fptr1,*fptr2;
char ch;
clrscr();
fptr1=fopen("input.txt","r");
if(fptr1==NULL)
{
puts("can not open this file \n");
exit(1);
}
fptr2=fopen("output.txt","w");
if(fptr2==NULL)
{
puts("can not open this file \n");
fclose(fptr1);
exit(1);
}
while(1)
{
ch=fgetc(fptr1);
if(ch==EOF)
break;
else
{
c=toupper(ch);
fputc(c,fptr2);
}
}
fcloseall();
getch();
}
```

5.Write a program to open a preexisting file and add information at the end of file. Display the contents of the file before and after appending.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
FILE *fptr;
char c;
clrscr();
fptr=fopen("data.txt","a");
if(fptr==NULL)
{
printf("file cannot appended \n");
exit(1);
}
printf("contents of file before appending:\n");
while(!feof("fptr"))
{
c=fgetc(fptr);
printf("%c",c);
}
printf("enter string to be appended:\n");
while(c!="\0")
{
c=getchar();
fputc(c,fptr);
}
fclose(fptr);
printf("contents of the file after appending:\n");
rewind(fptr);
while(!feof(fptr))
{
c=fgetc(fptr);
printf("%c",c);
}
getch();
}
```

Assignment programs:

1.Write a C program to read text file containing some paragraph . use fseek() function and erad the text after skipping n characters from beginning of the file.

2.Write a program to copy up to 100 characters from a file to an output array.

3. Write a program for sequential file for the employee database for the following operations.

a. add record b. delete record c. search record

4.Write a program to show how rewind() function works.

5. Write a program to write a block of structure elements to the given file using fwrite() function.

Important questions:

1.Write the syntax for opening a file with various modes and closing a file.

2. Difference between text mode and binary mode operation of a file.

3.What is the significance of end of file(EOF).

4. Define streams. Explain different types of files.

5. Explain the following operations

a. fseek() b. ftell() c. rewind()

6. Explain the following

1 standard I/O 2. Formatted I/O

7.Explain about the file handling functions

8. Difference between getchar() and scanf() functions for reading strings.

9.Difference between feof() and ferror().

10. Explain command line arguments with example program.

****

ASSIGNMENT QUESTIONS

UNIT-1

1. Explain block diagram of a computer with a neat sketch.
2. List and explain various input devices and output devices used in a computer system.
3. What is a computer software? Explain types of software.
4. What is an algorithm? Write the various criteria used for judging an algorithm?
5. What is a flow chart? uses of flowchart? Explain the different symbols used in a flow chart.
6. What are the different steps followed in the program development.
7. Write about space requirements for variables of different types.
8. What is the purpose of type declarations? What are the components of type declaration?
9. What are different types of integer constants,character constant,string constant ,backslash character constant?
10. What is variable? How can variables be characterized? Give the rules for variable declaration?
11. What is the associativity of various operators?
12. Explain the working of ternary operator with example?
13. What is meant by operator precedence?
14. State the rules that are applied while evaluating in automatic type expression?
15. Expain the following functions with examples getchar(),putchar(),scanf(),printf(),gets() and puts()
16. Write about C tokens with examples.
17. Explain about three generation languages in C
18. What is software engineering.

UNIT-2

16. What are various control structures present in c language with examples?
17. Explain about switch statement with program?
18. What is meant by looping? Describe any two different forms of looping with examples?
19. What is the difference between break and continue statements? Explain with examples?
20. What is the purpose of goto statement?How is the associated target statement identified?
21. In what way array is different from an ordinary variable?
22. What conditions must be satisified by the entire elements of any given array?

23. what is an advantage of using arrays?
24. Define an array? What are the different types of arrays?Explain?
25. Write the syntax for declaring two dimensional array with examples?
26. What is a string? Explain declaration and initialization of arrays of char type.
27. Explain various string I/O functions with simple programs?
28. List and explain various string handling functions with suitable examples.

UNIT-3

29. What do you mean by functions? Give the structure of the functions and explain about the arguments and their return values?
30. What is meant by function prototype? Give an example of function prototype.
31. what are the different types of functions? Explain function with no argument and no return type with example.
32. What is the main() in C? Why is it necessary in each program?
33. Explain in detail about call-by-value and call-by-reference . Explain with a simple program?
34. What are the different storage classes in C ? Explain?
35. Write about block structure and scope of variable.Illustrate with an example?
36. Distinguish between user defined and built in functions?
37. what is preprocessor directive?
38. Distinguish between function and preprocessor directive?
39. How does the undefining of a predefined macro done?

UNIT-4

40. Explain the process of declaring and initialization pointers. Give an example?
41. List out the reasons for using pointers?
42. Explain the process of accessing a variable through its pointer. Give an example.
43. Write a C program that uses a pointer as a function argument?
44. Mention the difference between malloc() and calloc() functions?
45. Explain the command line arguments with example
46. Explain the concept of dynamic memory allocation with example program?

UNIT-5

47. Define structure and write the general format for declaring and accessing members?
48. How are structured elements stored in memory?
49. How to copy one structure to another structure of a same data type,give an example
50. How is structure different from an array?Explain.
51. Describe nested structure. Draw diagrams to explain nested structure.

52. How to compare structure variables? Give an example.
53. How are structure elements accessed using pointers? Which operator is used? Give an example.
54. What is the general format of a union? Decalre a union and assign values to it. Explain the process of accessing the union members.
55. Differentiate between a structure and union with respective allocation of memory by the compiler. Give an example of each.
56. Compare arrays,structures and unions.
57. Explain various bit wise operators present in C language with examples.

UNIT-6

58. Write the syntax for opening a file with various modes and closing a file
59. Distinguish between text mode and binary mode operation of a file.
60. What is the significance of EOF
61. Define streams . Explain different types of files.
62. Explain about file handling functions with examples.

Tutorial Programs:

Tutorial-1

1. Write a program to find the maximum of three numbers using if
2. Write a program to find whether the given number is positive or negative
3. Calculates the area of circle or rectangle
4. To find the maximum of three numbers using if….else
5. To find whether the given number is even or odd
6. To find the biggest among two numbers
7. Program that accepts sales amount and then calculates discount as 10% of sales amount if sales amount is more than 5000 otherwise as 5% of sales amount
8. Program that checks whether a given number is vowel or not
9. To find maximum of three numbers using nested if…..

Tutorial-2

10. Program to check whether a character is an uppercase or lowercase alphabet or a digit or a special symbol
11. Program to read three integers and display the largest of these numbers
12. Program that reads marks in three subjects,calculate average marks and assign grade as per the following specifications:

If marks	then grade
>=90	A
75-90	B

60-75	C
50-60	D
<50	fail

13. To calculate area of square/rectangle/circle/triangle depending upon user's choice
14. Program for temperature conversion depending upon user's choice
15. Program to find the roots of quadratic equation
16. Program to read two integers and check whether first number is divisible by second number
17. Using switch statement to read a day-number and print corresponding day of the week

Tutorial-3

18. To find the sum of first n natural numbers using while loop
19. Program to find the series sum for 2+4+6+.....+n
20. Program to check whether a given number is armstrong number or not
21. Program to find length of a given positive integer i.e. number of digits in it. Program should also display sum of digits of the given number.
22. Program to check whether a given number is palindrome or not
23. To find the sum of first n natural using loops
24. To find sum of N odd numbers

Tutorial-4

25. To find the factorial of a number
26. To generate Fibonacci sequence
27. To check whether a number is prime or not
28. Program to print prime numbers between a range
29. Program to display a pattern as shown below

```
*
* *
* * *
* * * *
```

30. Program to display following pattern

```
ABCDE
ABCD
ABC
AB
A
```

The above pattern also display with numbers

31. Program to print multiplication table using nested for loop

32.To find the maximum of N values

Tutorial -5

33. To accept a list of numbers into a 1-d array ,find their sum and average
34. Program to find maximum and minimum element from 1-d array
35. To search for a number in a list of numbers
36. To sort a list of numbers
37. To search for a number in a list of numbers
38. Write a program to perform addition of two matrices
39. Write a program to perform multiplication of two matrices

Tutorial-6

40. Program to print transpose of a matrix
41. Program to check a square matrix if it is symmetrical
42. Program to print row sum and column sum of a matrix
43. Program to find the sum of even numbers using arrays
44. Program to count number of even and odd numbers in a 1-d integer array
45. Program to find the number of occurrences of a character in a string
46. Program to find the length of a string

Tutorial-7

47. Program to find the copy one string to another string
48. Program to compare two strings
49. Program to concatenate two strings
50. Program to check whether the given string is palindrome or not
51. Write a program sorting of N names using string function
52. Write a program to counting characters,words,and lines from the text
53. to find out whether the given string is palindrome or not using the library function

Tutorial-8

54. Program to count number of occurrences of a given character in a string
55. Program to count number of uppercase,lowercase characters,digits,and special characters in a string.
56. Program to delete a substring from a string
57. Program to replace a substring with another substring in a given string
58. Program to sort a list of names alphabetically in the increasing order
59. Program for tax calculation without using function
60. Program for tax calculation using function
61. Write a program to find sum of given series by using function with argument and return value

    $e=2+3/1!-6/2!+9/3!-12/4!+\ldots\ldots\ldots\ldots$

Tutorial-9

62. Swap two numbers using pointers
63. Program to matrix multiplication using pointers
64. Program to illustrate malloc() : creation of an array dynamically

Tutorial-10

65. Write a program using structures to display the following information for each customer name,account number,street,city,old balance,current payment,new balance,account status.
66. Write a C program to calculate student wise total for three students using array of structures.
67. Write a C program to print maximum marks in each subject along with the student name by using structures. Take 3 subjects and 3 students records.
68. Write a C program to compute the monthly pay of 50 employees using each employee's name,basicpay. The DA is computed as 72% of basic pay. Gross salary (basic pay+DA). Print the employee's name and gross salary.

Tutorial-12

69. Write a C program to copy the one file data into another file.
70. Write a C program to merge the content of two files into a third file
71. Write a program to reverse the first n characters in a file.
72. Write a C program to count the number of words in a given file
73. Write a C program to count characters and lines in a given file.

Experiment No : 1.a

Write a C Program to calculate the area of triangle using the formula

area = $(s (s-a) (s-b)(s-c))^{1/2}$ where s= (a+b+c)/2.

Aim: Program to calculate the area of triangle using the given formula.

Source Code:

```
/******************PROGRAM TO CALCULATE THE AREA OF TRIANGLE***************/

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
      int a,b,c;
      double s,area;
      clrscr();
      printf("\n Enter the values of a,b and c:");
      scanf("%d%d%d",&a,&b,&c);
      s=(a+b+c)/2.0;
      area=(s*(s-a)*(s-b)*(s-c));
      printf("\n The Area is: %lf",sqrt(area));
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the values of a,b and c:1 2 3
```

Output:

```
The Area is:0.000000
```

Experiment No : 1.b

Write a C program to find the largest of three numbers using ternary operator.

Aim: Program to find the largest of three numbers using ternary operator.

Source Code:

```
/*****PROGRAM TO FIND THE LARGEST OF THREE NUMBERS USING TERNARY OPERATOR***/

#include<stdio.h>
#include<conio.h>
void main()
{
      int a,b,c,l;
      clrscr();
      printf("\n Enter the values of a,b and c:");
      scanf("%d%d%d",&a,&b,&c);
      l=(a>b)?(a>c)?a:(b>c)?b:c:(b>c)?b:(a>c)?a:c;
      printf("\n The largest is: %d",l);
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the values of a,b and c:3 4 1
```

Output:

```
The largest is:4
```

Experiment No : 1.c

Program to swap two numbers without using temporary variable

Aim: Program to swap two numbers without using temporary variable.

Source Code:

```
/*******PROGRAM TO SWAP TWO NUMBERS WITHOUT USING TEMPORARY VARIABLE********/

/* */
#include<stdio.h>
#include<conio.h>
void main()
{
      int a,b;
      clrscr();
      printf("\n Enter the values of a and b:");
      scanf("%d%d",&a,&b);
      printf("\n Before Swapping a: %d\t b: %d",a,b);
      a=a+b;
      b=a-b;
      a=a-b;
      printf("\n After Swapping a: %d\t b: %d",a,b);
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the values of a and b: 10 20
```

Output:

```
Before Swapping a: 10    b: 20

After Swapping a: 20    b: 10
```

Experiment No : 2.a

2’s complement of a number is obtained by scanning it from right to left and complementing all the bits after the first appearance of a 1. Thus 2’s complement of 11100 is 00100. Write a C program to find the 2’s complement of a binary number.

Aim: Program to find 2’s complement of a binary number.

Source Code:

```
/*********************PROGRAM TO FIND 2’s COMPLEMENT**********************/

#include<stdio.h>
#include<conio.h>
void main()
{
      char bi[20],res;
      int flag=0,i;
      clrscr();
      printf("\n Enter Binary no : ");
      scanf("%s",bi);
      for(i=strlen(bi)-1;i>=0;i--)
      {
            if(flag==0)
            {
                  if(bi[i]==’0’)
                        continue;
                  else
                        flag=1;
            }
            else
            {
                  if(bi[i]==’0’)
                        bi[i]=’1’;
                  else
                        bi[i]=’0’;
            }
      }
      printf(“\n 2’s complement for the given binary no is: %s”,bi);
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
            Enter Binary no: 11100
```

Output:

```
            2’s complement for the given binary no is: 00100
```

Experiment No : 2.b

Write a C program to find the roots of a quadratic equation.

Aim: Program to find the roots of a quadratic equation.

Source Code:

```
/************PROGRAM TO FIND THE ROOTS OF A QUADRATIC EQUATION**************/

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
      int a,b,c,d;
      float r1,r2;
      clrscr();
      printf("\n Enter the values of a,b,c: \n");
      scanf("%d %d %d",&a,&b,&c);
      d=((b*b)-4*a*c);
      if(d==0)
      {
            printf("\n Roots are equal \n");
            r1=(-b)/2*a;
            r2=(-b)/2*a;
            printf("\n The roots of quadratic equation are %f %f",r1,r2);
      }
      else if(d>=0)
      {
            printf("\n Roots are real \n");
            r1=(-b+sqrt(b*b-4*a*c))/2*a;
            r2=(-b-sqrt(b*b-4*a*c))/2*a;
            printf("\n The roots of quadratic equation are %f %f",r1,r2);
      }
      else
      {
            printf("\n Roots are imaginary");
      }
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the values of a,b,c: 1 -5 6
```

Output:

```
The roots of quadratic equation are 3.000000 2.000000
```

Experiment No : 2.c

Write a C program, which takes two integer operands and one operator from the user, performs the operation and then prints the result.

(Consider the operators +,-,*, /, % and use Switch Statement)

Aim: Program which takes two integer operands and one operator from the user, perform the operation.

Source Code:

```
/**********PROGRAM PERFORMS OPERATION BY BASED ON USERS CHOICE**************/

#include<stdio.h>
#include<conio.h>
void main()
{
      int a,b,n;
      clrscr();
      printf("Enter your choice\n 1.)addition \n
      2.)substraction \n 3.)multiplication \n 4.)division \n 5.)modulus");
      Scanf(“&d”,&n);
      printf("\n Enter any two integer values : ");
      scanf("%d%d",&a,&b);
      switch(n)
      {
            case 1: printf("\n Addition of %d and %d is %d\n",a,b,a+b);
                    break;
            case 2: printf("\n Subtraction of %d and %d is %d\n",a,b,a-b);
                    break;
            case 3: printf("\n Multiplication of %d and %d %d\n",a,b,a*b);
                    break;
            case 4: printf("\n Division of %d and %d is %d\n",a,b,a/b);
                    break;
            case 5: printf("\n Modules of %d and %d is %d",a,b,a%b);
                    break;
            default : printf(“\n please enter correct choice”);
      }
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
            Enter your choice
                  1.)addition
                  2.)subtraction
                  3.)multiplication
                  4.)division
                  5.)modulus
            5
            Enter any two integer values : 2 5
```

Output:

```
            Modules of 2 and 5 is 2
```

Experiment No : 3.a

Write a C program to find the sum of individual digits of a positive integer

Aim: Program to find the sum of individual digits of a positive integer.

Source Code:

```
/*****PROGRAM TO FIND THE SUM OF INDIVIDUAL DIGITS OF A POSITIVE INTEGER****/

#include<stdio.h>
#include<conio.h>
void main()
{
      unsigned int n;
      int s=0;
      clrscr();
      printf("\n Enter positive integer: ");
      scanf("%u",&n);
      while(n!=0)
      {
            s=s+n%10;
            n=n/10;
      }
      printf("\n The sum of digits of givem number = %d",s);
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter positive integer:345
```

Output:

```
The sum of digits of givem number =12
```

Experiment No : 3.b.i

A Fibonacci sequence is defined as follows: the first and second terms in the sequence are 0 and 1. Subsequent terms are found by adding the preceding two terms in the sequence. Write a C program to generate the first n terms of the sequence.

Aim: Program to generate the first n terms in the Fibonacci sequence.

Source Code:

```
/******PROGRAM TO GENERATE THE FIRST N TERMS IN THE FIBONACCI SEQUENCE******/

#include<stdio.h>
#include<conio.h>
void main()
{
      int a=0,b=1,c,n,i;
      clrscr();
      printf("\n Enter the number of elements : ");
      scanf("%d",&n);
      printf("\n The fibonacci series is : \n%d\t%d",a,b);
      for(i=2;i<n;i++)
      {
            c=a+b;
            printf("\t%d",c);
            a=b;
            b=c;
      }
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the number of elements :5
```

Output:

```
The fibonacci series is :
0     1     1     2     3
```

Experiment No : 3.b.ii

Use summing series method to compute the value of SIN(x).

Aim: Program to find the value SIN(x) using summing series method.

Source Code:

```
/*********************PROGRAM TO FIND THE VALUE OF SIN(x) *****************/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>

void main()
{
      int n, x1;
      float acc, term, den, x, sinx=0, sinval;
      clrscr();
      printf("\n Enter the value of x (in degrees):");
      scanf("%f",&x);
      x1 = x;
      x = x*(3.142/180.0);
      sinval = sin(x);
      printf("\n Enter the accuary for the result:");
      scanf("%f", &acc);
      term = x;
      sinx = term;
      n = 1;
      do
      {
            den = 2*n*(2*n+1);
            term = -term * x * x / den;
            sinx = sinx + term;
            n = n + 1;
      } while(acc <= fabs(sinval - sinx));
      printf("\n Sum of the sine series = %f", sinx);
      printf("\n Using Library function sin(%d) = %f", x1,sin(x));
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the value of x (in degrees): 30
Enter the accuary for the result: 0.000001
```

Output:

```
Sum of the sine series = 0.500059
Using Library function sin(30) = 0.500059
```

Experiment No : 3.b.iii

Use summing series method to compute the value of COS(x).

Aim: Program to find the value COS(x) using summing series method.

Source Code:

```
/*********************PROGRAM TO FIND THE VALUE OF COS(x) *****************/

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

void main()
{
      int n,x1,i,j;
      float x,sign,cosx,fact;
      printf("Enter the number of the terms in aseries\n");
      scanf("%d",&n);
      printf("Enter the value of x(in degrees)\n");
      scanf("%f",&x);
      x1=x;
      x=x*(3.142/180.0);
      cosx=1;
      sign=-1;
      for(i=2;i<=n;i=i+2)
      {
            fact=1;
            for(j=1;j<=i;j++)
            {
                  fact=fact*j;
            }
      cosx=cosx+(pow(x,i)/fact)*sign;
      sign=sign*(-1);
      }
      printf("Sum of the cosine series = %7.2f\n",cosx);
      printf("The value of cos(%d) using library function = %f\n",x1,cos(x));
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the number of the terms in aseries:5
Enter the value of x(in degrees):60
```

Output:

```
Sum of the cosine series = 0.50
The value of cos(60) using library function = 0.499882
```

Experiment No : 3.b.iv

Use summing series method to compute the value of $\mathbf{e}^{x}$.

Aim: Program to find the value $\mathbf{e}^{x}$ using summing series method.

Source Code:

```
/**********************PROGRAM TO FIND THE VALUE OF e^x *****************/

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
      int n,x,i,j;
      double s=1.0,f=1.0;
      clrscr();
      printf("\n Enter the value of x:");
      scanf("%d",&x);
      printf("\n Enter the accuracy(No.of terms):");
      scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
            for(j=1;j<=i;j++)
                  f=f*j;
            s=s+pow(x,i)/f;
      }
      printf("\n The sum of the series is(e pow x):%lf",s);
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the value of x: 2

Enter the accuracy (No. of terms):5
```

Output:

```
The sum of the series is (e pow x):5.723148
```

Experiment No : 3.C

Write a C program to generate all the prime numbers between 1 and n, where n is a value supplied by the user.

Aim: Program to generate all the prime numbers between 1 and n.

Source Code:

```
/**********PROGRAM TO GENERATE ALL THE PRIME NUMBERS BETWEEN 1 AND N********/

#include<stdio.h>
#include<conio.h>
void main()
{
      int n,i,j;
      clrscr();
      printf("\n Enter the range : ");
      scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
            for(j=2;j<i;j++)
            {
                  if(i%j==0)
                  {
                        break;
                  }
            }
            if(i==j)
            {
                  printf("\t%d",i);
            }
      }
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
            Enter the range : 10
```

Output:

```
            2     3     5     7
```

Experiment No : 4.a

Write a C Program to print the multiplication table of a given number n up to a given value. Where n is entered by the user.

Aim: Program to print the multiplication table of a given number n up to a given value.

Source Code:

```
/*******PROGRAM TO PRINT THE MULTIPLICATION TABLE OF A GIVEN NUMBER*********/

#include<stdio.h>
#include<conio.h>
void main()
{
      int n,u,i;
      clrscr();
      printf("\n Enter the value of n:");
      scanf("%d",&n);
      printf("\n Enter the table range:");
      scanf("%d",&u)
      printf("\n The multiplication Table for %d up to %d is \n",n,u);
      for(i=1;i<=u;i++)
      {
            printf("\n %2d * %2d = %2d",n,i,n*i);
      }
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the value of n:5
Enter the table range:11
```

Output:

```
The multiplication Table for 5 up to 11 is
5 * 1   = 5
5 * 2   = 10
5 * 3   = 15
5 * 4   = 20
5 * 5   = 25
5 * 6   = 30
5 * 7   = 35
5 * 8   = 40
5 * 9   = 45
5 * 10  = 50
5 * 11  = 55
```

Experiment No : 4.b

Write a C Program to enter a decimal number, and calculate and display the binary equivalent of that number.

Aim: Program to convert decimal number into binary equivalent .

Source Code:

```
/*PROGRAM TO ENTER A DECIMAL NO AND CALCULATE AND DISPLAY BINARY EQUIVALENT*/

#include<stdio.h>
#include<conio.h>
void main()
{
      int n,c,k;
      clrscr();
      printf("\n Enter the value of n:");
      scanf("%d",&n);
      printf("\n The Decimal number is: %d",n);
      printf("\n %d in binary number system are:", n);
      for (c = 15; c >= 0; c--)
      {
            k = n >> c;
            if (k & 1)
                  printf("1");
            else
                  printf("0");
      }
      getch();

}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the value of n: 32767
```

Output:

```
32767 in binary number system are: 0111111111111111
```

Experiment No : 4.c

Write a C program to check whether the given number is Armstrong number or not.

Aim: Program to check whether the given number is Armstrong number or not.

Source Code:

```
/***PROGRAM TO CHECK WHETHER THE GIVEN NUMBER IS ARMSTRONG NUMBER OR NOT***/

#include<stdio.h>
#include<conio.h>
void main()
{
      int n,s=0,i,x;
      clrscr();
      printf("\n Enter the value of n:");
      scanf("%d",&n);
      x=n;
      for(;n>0;n=n/10)
      {
            i=n%10;
            s=s+i*i*i;
      }
      if(x==s)
            printf("\n Armstrong Number");
      else
            printf("\n Not a Armstrong Number");
      getch();

}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the value of n: 153
```

Output:

```
Armstrong Number
```

Experiment No : 5.a

Write a C program to interchange the largest and smallest numbers in the array.

Aim: Program to interchange the largest and smallest numbers in the array.

Source Code:

```
/***PROGRAM TO INTERCHANGE THE LARGEST AND SMALLEST NUMBERS IN THE ARRAY****/

#include<stdio.h>
#include<conio.h>
void main()
{
      int a[20],n,i,l=0,s=32767,li,si,t;
      clrscr();
      printf("\n Enter the size of an array:");
      scanf("%d",&n);
      printf("\n Enter array elements :");
      for(i=0;i<n;i++)
      {
            scanf("%d",&a[i]);
            printf("%3d",a[i]);
      }
      for(i=0;i<n;i++){
            if(l<a[i]){
                  l=a[i];
                  li=i;
            }
      }
      for(i=0;i<n;i++){
            if(s>a[i]){
                  s=a[i];
                  si=i;
            }
      }
      t=a[li];    a[li]=a[si];      a[si]=t;
      printf("\n After interchanging the Array Status is:");
      for(i=0;i<n;i++)
            printf("%3d",a[i]);
      getch();

}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the size of an array: 5
Enter array elements: 5 2 1 3 4
```

Output:

```
5 2 1 3 4
After interchanging the Array Status is:  1 2 5 3 4
```

Experiment No : 5.b

Write a C program to implement a liner search.

Aim: Program to implement a linear search.

Source Code:

```
/*******************PROGRAM TO IMPLEMENT A LINEAR SEARCH *******************/

#include<stdio.h>
#include<conio.h>
void main()
{
      int a[20],n,i,x,flag=0;
      clrscr();
      printf("\n Enter the size of an array:");
      scanf("%d",&n);
      printf("\n Enter array elements:");
      for(i=0;i<n;i++)
      {
            scanf("%d",&a[i]);
      }
      printf("\n Enter the search element:");
      scanf("%d",&x);
      for(i=0;i<n;i++)
      {
            if(a[i]==x)
            {
                  flag=1;
                  break;
            }
      }
      if(flag==1)
            printf("\n Element found (position:%d)",++i);
      else
            printf("\n Element Not Found");
      getch();

}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the size of an array: 5
Enter array elements: 4 3 1 6 5
Enter the search element: 6
```

Output:

```
Element found (position: 4)
```

Experiment No : 5.c

Write a C program to implement a binary search.

Aim: Program to implement a binary search.

Source Code:

```
/*******************PROGRAM TO IMPLEMENT A BINARY SEARCH *******************/

#include<stdio.h>
#include<conio.h>
void main()
{
      int a[20],n,i,x,flag=0,l,h,m;
      clrscr();
      printf("\n Enter the size of an array:");
      scanf("%d",&n);
      printf("\n Enter array elements (In Sorted Order):");
      for(i=0;i<n;i++)
      {
            scanf("%d",&a[i]);
      }
      printf("\n Enter the search element:");
      scanf("%d",&x);
      l=0;h=n-1;
      while(l<=h)
      {
            m=(l+h)/2;
            if(a[m]==x)
            {
                  flag=1;
                  break;
            }
            else if(a[m]>x)
                  h=m-1;
            else
                  l=m+1;
      }
      if(flag==1)
            printf("\n Element found (position:%d)",++m);
      else
            printf("\n Element Not Found");
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
            Enter the size of an array: 5
            Enter array elements (In Sorted Order): 1 2 3 4 5
            Enter the search element: 6
```

Output:

```
            Element Not Found
```

Experiment No : 6.a

Write a C program to implement sorting of an array of elements.

Aim: Program to implement sorting of an array of elements.

Source Code:

```
/************PROGRAM TO IMPLEMENT SORTING OF AN ARRAY OF ELEMENTS***********/
#include<stdio.h>
#include<conio.h>
void main()
{
      int a[20],n,i,j,t;
      clrscr();
      printf("\n Enter the size of an array:");
      scanf("%d",&n);
      printf("\n Enter array elements:");
      for(i=0;i<n;i++)
            scanf("%d",&a[i]);
      printf("\n Array elements before sorting:");
      for(i=0;i<n;i++)
            printf("%3d",a[i]);
      for(i=0;i<n;i++){
            for(j=i+1;j<n;j++)
            {
                  if(a[i]>a[j])
                  {
                        t=a[i];
                        a[i]=a[j];
                        a[j]=t;
                  }
            }
      }
      printf("\n Array elements after sorting:");
      for(i=0;i<n;i++)
            printf("\%3d",a[i]);
      getch();

}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the size of an array: 5
Enter array elements (In Sorted Order): 3 2 1 5 4
```

Output:

```
Array elements before sorting: 3 2 1 5 4

Array elements after sorting: 1 2 3 4 5
```

Experiment No : 6.b.i

Write a C program to input two m x n matrices, check the compatibility and perform addition

Aim: Program to find the addition of two matrices.

Source Code:

```
/****************PROGRAM TO FIND THE ADDITION OF TWO MATRICES**************/

#include<stdio.h>
#include<conio.h>
void main()
{
      int a[20][20],b[20][20],c[20][20],i,j,ra,ca,rb,cb;
      clrscr();
      printf("\n Enter order of matrix a:");
      scanf("%d %d",&ra,&ca);
      printf("\n Enter order of matrix b:");
      scanf("%d %d",&rb,&cb);
      if((ra==rb)&&(ca==cb))
      {
            printf("\n \t Enter matrix a\n");
            for(i=0;i<ra;i++)
            {
                  for(j=0;j<ca;j++)
                  {
                        scanf("%d",&a[i][j]);
                  }
            }
            printf("\n\t Enter matrix b\n");
            for(i=0;i<rb;i++)
            {
                  for(j=0;j<cb;j++)
                  {
                        scanf("%d",&b[i][j]);
                  }
            }
            printf("\n MATRIX ADDITION IS\n");
            for(i=0;i<ra;i++)
            {
                  for(j=0;j<cb;j++)
                  {
                        c[i][j]=a[i][j]+b[i][j];
                        printf("%3d",c[i][j]);
                  }
                  printf("\n");
            }
```

```
	}
	else
	{
		printf("\n Matrix Addition is Not Possible");
	}
	getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:
```
Enter order of matrix a: 3 2
Enter order of matrix b: 2 3
```

Output:
```
Matrix Addition is Not Possible
```

Experiment No : 6.b.i

Write a C program to input two m x n matrices, check the compatibility and perform multiplication.

Aim: Program to find the multiplication of two matrices.

Source Code:

```
/*************PROGRAM TO FIND THE MULTIPLICATION OF TWO MATRICES************/

#include<stdio.h>
#include<conio.h>
void main()
{
      int a[20][20],b[20][20],c[20][20],r1,c1,r2,c2,i,j,k,ta,tb;
      clrscr();
      printf("\n Enter number of rows and columns of first matrix :");
      scanf("%d%d",&r1,&c1);
      printf("\n Enter number of rows and columns of second matrix :");
      scanf("%d%d",&r2,&c2);
      if(r2==c1)
      {
            ta=r1*c1;
            printf("\n Enter %d elements of First matrix :",ta);
            for(i=0;i<r1;i++)
            {
                  for(j=0;j<c1;j++)
                  {
                        scanf("%d",&a[i][j]);
                  }
            }
            printf("\n First Matrix is :\n");
            for(i=0;i<r1;i++)
            {
                  for(j=0;j<c1;j++)
                  {
                        printf("%d\t",a[i][j]);
                  }
                  printf("\n");
            }
            tb=r2*c2;
            printf("\n Enter the %d elments of second matrix :",tb);
            for(i=0;i<r2;i++)
            {
                  for(j=0;j<c2;j++)
                  {
                        scanf("%d",&b[i][j]);
                  }
            }
            printf("\n Second Matrix is:\n");
            for(i=0;i<r2;i++)
            {
                  for(j=0;j<c2;j++)
                  {
                        printf("%d\t",b[i][j]);
                  }
                  printf("\n");
            }
            printf("\n Multiplication of the Matrices:\n");
            for(i=0;i<r1;i++)
            {
                  for(j=0;j<c2;j++)
                  {
                        c[i][j]=0;
                        for(k=0;k<r2;k++)
                        {
                              c[i][j]=c[i][j]+a[i][k]*b[k][j];
                        }
                  }
             }
             for(i=0;i<r1;i++)
             {
                  for(j=0;j<c2;j++)
                  {
                        printf("%d\t",c[i][j]);
                  }
                  printf("\n");
             }
      }
             else
             {
                   printf("\n Matrix multiplication is not possible");
             }
             getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
            Enter number of rows and columns of first matrix :2 3
            Enter number of rows and columns of second matrix :2 3
```

Output:

```
            Matrix multiplication is not possible
```

Experiment No : 7.a

Write a C program that uses functions to perform the following operations:
i. To insert a sub-string in to given main string from a given position.

Aim: Program that uses function to insert a sub-string in to given main string from a given position.

Source Code:

```
/**********PROGRAM TO INSERT A SUB STRING INTO A GIVEN MAIN STRING**********/
#include<stdio.h>
#include<conio.h>
void main()
{
      void insert(char s[],char ss[],int n);
      int n;
      char s[80],ss[80];
      clrscr();
      printf("\n Enter the main string :");
      gets(s);
      printf("\n The given main string is :%s",s);
      printf("\n Enter the sub string :");
      gets(ss);
      printf("\n The given sub string is :%s",ss);
      printf("\n Enter the position :");
      scanf("%d",&n);
      insert(s,ss,n);
      printf("\n String after insertion :");
      puts(s);
      getch();
}
void insert(char s[80],char ss[80],int n)
{
      int i,j=0;
      char temp[80];
      for(i=n;s[i]!=NULL;i++)
      {
            temp[j++]=s[i];
            temp[j]=NULL;
      }
      for(i=0;ss[i] != NULL;i++)
      {
            s[n++]=ss[i];
      }
      for(i=0;temp[i] != NULL;i++)
      {
            s[n++]=temp[i];
      }
      s[n]='\0';
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the main string: Hi this is x
The given main string is: Hi this is x
Enter the sub string: Hello
The given sub string is: Hello
Enter the position: 2
```

Output:

```
String after insertion: HiHello this is x
```

Experiment No : 7.b

Write a C program that uses functions to perform the following operations:
ii. To delete n Characters from a given position in a given string.

Aim: Program that uses function to delete n characters from a given position in a given string.

Source Code:

```
/*PROGRAM TO DELETE GIVEN NO OF CHARACTERS FROM THE GIVEN POSITION IN A STR*/

#include<stdio.h>
#include<conio.h>
void main()
{
      void delete(char s[],int p,int n);
      int p,n;
      char s[80];
      clrscr();
      printf("\n Enter the main string :");
      gets(s);
      printf("\n The given main string is :%s",s);
      printf("\n Enter the no.of characters :");
      scanf("%d",&n);
      printf("\n Enter the position :");
      scanf("%d",&p);
      delete(s,p,n);
      printf("\n String after deletion :");
      puts(s);
      getch();
}
void delete(char s[80],int p,int n)
{
      int i,j;
      for(i=p,j=p+n;s[j] != NULL;i++,j++)
      {
            s[i]=s[j];
      }
      s[i]=NULL;
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
            Enter the main string: Hi Hello
            The given main string is: Hi Hello
            Enter the no. of characters: 2
            Enter the position: 2
```

Output:

```
            String after deletion: Hiello
```

Experiment No : 7.c

Write a C program that uses functions to perform the following operations:
iii. To replace a character of string either from beginning or ending or at a specified location

Aim: Program that uses function to replace a character of string from beginning or ending or at a specified position.

Source Code:

```
/*****PROGRAM TO REPLACE A CHARACTER OF STRING AT A SPECIFIED POSITION******/

#include<stdio.h>
#include<conio.h>
void main()
{
      void replace(char s[],char e,char r,int p);
      int p;
      char s[80],e,r;
      clrscr();
      printf("\n Enter the main string :");
      gets(s);
      printf("\n The given main string is :%s",s);
      printf("\n Enter the present character :");
      scanf("%c",&e);
      printf("\n Enter the replace character :");
      r=getche();
      printf("\n Enter the position :");
      scanf("%d",&p);
      replace(s,e,r,p);
      printf("\n String after replace :");
      puts(s);
      getch();
}
void replace(char s[80],char e,char r,int p)
{
      if(s[p]==e)
            s[p]=r;
      else
            printf("\n specified character is not existed");
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the main string: Hi Hello
The given main string is: Hi Hello
Enter the present character: H
Enter the replace character: h
Enter the position: 3
```

Output:

```
String after replace: Hi hello
```

Experiment No : 8

Write a C program that uses functions to perform the following operations using Structure:

i) Reading a complex number iii) Addition of two complex numbers

ii) Writing a complex number iv) Multiplication of two complex numbers

Aim: Program that uses functions to perform operations on complex numbers.

Source Code:

```
/***PROGRAM THAT USES FUNCTIONS TO PERFORM OPERATIONS ON COMPLEX NUMBERS***/

#include<stdio.h>
#include<conio.h>
struct complex
{
      int re;
      int im;
};
typedef struct complex complex;
void main()
{
      complex Read();
      void Write(complex);
      complex Add(complex,complex);
      complex Mul(complex,complex);
      complex c1,c2,a,m;
      clrscr();
      c1=Read();
      Write(c1);
      c2=Read();
      Write(c2);
      printf("\n Addition of two complex numbers\n\t");
      a=Add(c1,c2);
      Write(a);
      printf("\n Multiplication of two complex numbers\n\t");
      m=Mul(c1,c2);
      Write(m);
      getch();
}
complex Read()
{
      complex c;
      printf("\n Enter Real part and Imaginary part of a Complex No:");
      scanf("%d%d",&c.re,&c.im);
      return c;
}
void Write(complex c)
{
      printf("\n The Complex No is :%d+i%d",c.re,c.im);
}
complex Add(complex x,complex y)
{
      complex z;
      z.re=x.re+y.re;
      z.im=x.im+y.im;
      return z;
}
complex Mul(complex x,complex y)
{
      complex z;
      z.re=(x.re*y.re-x.im*y.im);
      z.im=(x.re*y.im+x.im*y.re);
      return z;
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter Real part and Imaginary part of a Complex No:2 3
The Complex No is: 2+i3
Enter Real part and Imaginary part of a Complex No:1 2
The Complex No is: 1+i2
```

Output:

```
Addition of two complex numbers
The Complex No is: 3+i5
Multiplication of two complex numbers
The Complex No is:-4+i7
```

Experiment No : 9

Write C Programs for the following string operations without using the built in functions

- to concatenate two strings

Aim: Write C Programs for the following string operations without using the built in functions
- to concatenate two strings

Source Code:

```
/******************PROGRAM TO CONCATENATE TWO STRINGS ***************/

#include<stdio.h>
#include<conio.h>
void main()
{
   char str1[25],str2[25];
   int i=0,j=0;
   clrscr();
   printf("\nEnter First String:");
   gets(str1);
   printf("\nEnter Second String:");
   gets(str2);
   while(str1[i]!='\0')
   i++;
   while(str2[j]!='\0')
   {
      str1[i]=str2[j];
      j++;
      i++;
   }
   str1[i]='\0';
   printf("\nConcatenated String is %s",str1);
   getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter First String:surya
Enter Second String:kiran
```

Output:

```
Concatenated String is suryakiran
```

Experiment No : 9

Write C Programs for the following string operations without using the built in functions

- to append a string to another string.

Aim: Write C Programs for the following string operations without using the built in functions
to append a string to another string.

Source Code:

```
/*****************PROGRAM TO APPEND A STRING TO ANOTHER STRING ************/

#include<stdio.h>
#include<conio.h>
void main()
{
   char str1[25],str2[25],str3[25];
   int i=0,j=0;
   clrscr();
   printf("\nEnter First String:");
   gets(str1);
   printf("\nEnter Second String:");
   gets(str2);
   while(str1[i]!='\0')
   {
      str3[i]=str1[i];
      i++;
    }
   while(str2[j]!='\0')
   {
     str3[i]=str2[j];
     j++;
     i++;
   }
   str3[i]='\0';
   printf("\nappended String is %s",str3);
   getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter First String:surya
Enter Second String:kiran
```

Output:

```
appended String is suryakiran
```

Experiment No : 9

Write C Programs for the following string operations without using the built in functions - to compare two strings

Aim: Write C Programs for the following string operations without using the built in functions
- to compare two strings
Source Code:

```
/** Program to Compare Two Strings Without using strcmp() **/
#include<stdio.h>
void main()
{
        char string1[25],string2[25];
        int i,temp = 0;
        printf("Enter the string1: \n");
        gets(string1);
        printf("\nEnter the String2: \n");
        gets(string2);
        for(i=0; string1[i]!='\0'; i++)
        {
                if(string1[i] == string2[i])
                temp = 1;
                else
                temp = 0;
        }
        if(temp == 1)
        printf("Both strings are same.");
        else
        printf("Both strings are not same.");
        getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
                Enter the string1: never
                Enter the String2: quit
```

Output:

```
        Both strings are not same
```

Experiment No : 10

Write C Programs for the following string operations without using the built in functions - to find the length of a string

Aim: Write C Programs for the following string operations without using the built in functions - to find the length of a string

Source Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
  char str1[25];
  int len=0;
  clrscr();
  printf("\nEnter String whose length is to be found:");
  gets(str1);
  while(str1[len]!='\0')
  len++;
  printf("\nLengt of the String %s is %d",str1,len);
  getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter string whose length is to be found: surya kiran
```

Output:

```
Length of the string surya kiran is 11
```

Experiment No : 10

Write C Programs for the following string operations without using the built in functions - to find whether a given string is palindrome or not

AIM: To find whether a given string is palindrome or not

Source code:

```
#include <stdio.h>
#include <conio.h>
void main()
{
   char text[100];
   int begin, middle, end, length = 0;
   clrscr();
   printf("Enter any string:" );
   gets(text);
   while ( text[length] != '\0' )
      length++;
   end = length - 1;
   middle = length/2;
   for( begin = 0 ; begin < middle ; begin++ )
   {
      if ( text[begin] != text[end] )
      {
         printf("Not a palindrome.\n");
       break;
      }
      end--;
   }
   if( begin == middle )
      printf("Palindrome.\n");
getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

Enter any string: madam

Output:

```
Palindrome.
```

Experiment No : 11.a

Write a C functions to find both the largest and smallest number of an array of integers

Aim: Program to find both the largest and smallest number of an array of integers.

Source Code:

```
/******* PROGRAM TO FIND BOTH THE LARGEST AND SMALLEST NUMBER OF AN ARRAY OF
INTEGERS ********/

#include<stdio.h>
#include<conio.h>
void main()
{

      int n,i,a[100],max=0,min=0;
      clrscr();
      printf("\n Enter size of array");
      scanf("%d",&n);
      printf("\n Enter elements of array\n");
      for(i=0;i<n;i++)
      {
      scanf("%d",&a[i]);
      }
      max=fmax(a,n);
      min=fmin(a,n);
      printf("\n %d is the max no of array\n",max);
      printf("\n %d is the min no of array",min);
      getch();
}
int fmax(int a[],int n)
{
      int max=a[0],i;
      for(i=1;i<n;i++)
      {
      if(max<a[i])
      {
      max=a[i];
      }
}
return (max);
}
int fmin(int a[],int n)
{
      int min=a[0],i;
      for(i=1;i<n;i++)
      {
      if(min>a[i])
```

```
            {
      min=a[i];
            }
      }
      return (min);
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter size of array: 4
Enter elements of array
4 1 3 8
```

Output:

```
8 is the max no of array
1 is the min no of array
```

Experiment No : 11.b

Write C programs illustrating call by value and call by reference concepts.

AIM: Write C programs illustrating call by value

Source code:
```
#include<stdio.h>
#include<conio.h>
void interchange(int number1,int number2)
{
    int temp;
    temp = number1;
    number1 = number2;
    number2 = temp;
}

void main()
{
    int num1,num2;
    clrscr();
    printf("Enter num1 and num2 values");
    scanf("%d%d",&num1,&num2);
    interchange(num1,num2);
    printf("\nNumber 1 : %d",num1);
    printf("\nNumber 2 : %d",num2);
    getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter num1 and num2 values:
 5      10
```

Output:

```
Number 1 : 5
Number 2 : 10
```

Experiment No : 11.b

Write C programs illustrating call by value and call by reference concepts.

AIM: Write C programs illustrating call by Reference

Source code:

```
#include<stdio.h>
#include<conio.h>
void interchange(int *num1,int *num2)
{
    int temp;
    temp  = *num1;
    *num1 = *num2;
    *num2 = temp;
}
void main()
{
    int num1,num2;
    clrscr();
    printf("Enter num1 and num2 values");
    scanf("%d%d",&num1,&num2);
    interchange(&num1,&num2);
    printf("\nNumber 1 : %d",num1);
    printf("\nNumber 2 : %d",num2);
    getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter num1 and num2 values:
 5      10
```

Output:

```
Number 1 : 10
Number 2 : 5
```

Experiment No : 12.a

Write C programs that use both recursive and non-recursive functions for the following
i) To find the factorial of a given integer.

AIM: Write C programs that use both recursive and non-recursive functions for the following
i) To find the factorial of a given integer.

Source code:

```
/*FACTORIAL IN BOTH RECURSIVE AND NON RECURSIVE*/
#include<stdio.h>
#include<conio.h>
main()
{
        int n,ans1=0,ans2=0;
        clrscr();
        printf("enter a number");
        scanf("%d",&n);
        ans1=factR(n);
        ans2=factNR(n);
        printf("factorial of %d using recursive is %d\n",n,ans1);
        printf("factorial of %d using non
        recursive is %d",n,ans2);
        getch();
}

int factR(int n)
{
if(n==0)
return 1;
else
return (n*factR(n-1));
}
int factNR(int n)
{
int i,fac=1;
for(i=1;i<=n;i++)
fac=fac*i;
return fac;
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter a number: 5
```

Output:

```
Factorial of 5 using recursive is 120
Factorial of 5 using non recursive is 120
```

Experiment No : 12.a

Write C programs that use both recursive and non-recursive functions for the following
ii) To find the GCD (greatest common divisor) of two given integers.

Aim: To find the GCD (greatest common divisor) of two given integers.

Source code:

```
/*G.C.D IN BOTH RECURSIVE AND NON RECURSIVE*/
#include<stdio.h>
#include<conio.h>
main()
{
int a,b,ans1=0,ans2=0;
clrscr();
printf("enter a,b");
scanf("%d %d",&a,&b);
ans1=GCDR(a,b);
ans2=GCDNR(a,b);
printf("GCD of  %d & %d using recursive is %d\n",a,b,ans1);
printf("GCD of  %d & %d using non recursive is %d",a,b,ans2);
getch();
}
int GCDR(int a,int b)
{
if(a%b==0)
return b;
else
return GCDR(b,a%b);
}

int GCDNR(int a,int b)
{
while(a!=b)
        {
        if(a>b)
                a=a-b;
        else
                b=b-a;
        }
return a;
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter a, b      12      16
```

Output:

```
GCD of 12 & 16 using recursive is 4
GCD of 12 & 16 using non recursive is 4
```

Experiment No : 12.a

Write C programs that use both recursive and non-recursive functions for the following

iii) To find Fibonacci sequence

Aim: To find Fibonacci sequence

Source code:

```
/* Fibonacci Series non recursive */
#include<stdio.h>
#include<conio.h>
void main()
{
  int n,c,i=0;
  clrscr();
  printf("Enter the number of terms\n");
  scanf("%d",&n);
  printf("First %d terms of Fibonacci series are :-\n",n);
  printf("result by non recursive function is:\n");
  nrfib(n);
  printf("result by recursive function is:\n");
  for ( c = 1 ; c <= n ; c++ )
  {
    printf("%d\n", rfib(i));
    i++;
  }

  getch();
}

  nrfib(int n)
  {
  int first=0,second=1,next,c=0;
  for(c=0;c<n;c++)
  {
    if ( c <= 1 )
      next = c;
    else
    {
      next = first + second;
      first = second;
      second = next;
    }
    printf("%d\n",next);
  }
}

int rfib(int n)
{
  if ( n == 0 )
    return 0;
  else if ( n == 1 )
    return 1;
  else
    return ( rfib(n-1) + rfib(n-2) );
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the number of terms        5
```

Output:

```
First 5 terms of Fibonacci series are:-
Result by non recursive function is: 0  1  1  2  3
Result by recursive function is: 0  1  1  2  3
```

Experiment No : 13.a

a) Write C Program to reverse a string using pointers

Aim: Write C Program to reverse a string using pointers

Source code:

```
#include<stdio.h>
#include<conio.h>
void rev(char *);
void main()
{
    char string[50];
    clrscr();
    printf("Enter a string to be reversed:");
    gets(string);
    printf("Before reversing the string %s\t",string);
    rev(string);
    printf("\nReverse String is %s",string);
    getch();
}
void rev(char *str)
{
    char *str1,temp;
    str1=str;
    while(*str1!='\0')
    str1++;
    str1--;
    while(str<str1)
    {
      temp=*str;
      *str=*str1;
      *str1=temp;
      str++;
      str1--;
    }
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter a string to be reversed: SURYA
```

Output:

```
Before reversing the string SURYA
Reverse string is AYRUS
```

Aim: Write a C Program to compare two arrays using pointers

Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
      int a[10],b[10],x,y,i,flag=0,*p,*q;
      clrscr();
      printf("enter size of array A and B : ");
      scanf("%d%d",&x,&y);
      if(x==y)
      {
                  printf("enter %d array elemnts of A: ",x);
            for(i=0;i<x;i++)
            scanf("%d",&a[i]);
            printf("\n enter %d array elemnts of b: ",y);
            for(i=0;i<y;i++)
            scanf("%d",&b[i]);
            p=a;
            q=b;
            for(i=0;i<x;i++)
            {
             if(*p++!=*q++)
             {
              flag=1;
             }
       }
      if(flag==0)
      printf("equal");
      else
      printf("not equal");
}
else
{
      printf("number of array elements are not equal");
}
getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter size of array A and B :   3   3
Enter 3 array elements of A :   1   2   3
Enter 3 array elements of B :   1   4   3
```

Output:

```
Not Equal
```

Experiment No : 14.a

a) Write a C program consisting of Pointer based function to exchange value of two integers using passing by address.

Aim: Write a C program consisting of Pointer based function to exchange value of two integers using passing by address.

Source code:

```
#include<stdio.h>
#include<conio.h>
main()
{
int x,y;
void swap(int *a,int *b);
clrscr();
printf("enter value of x,y");
scanf("%d %d",&x,&y);
swap(&x,&y);
printf("the value of x,y in main are %d %d",x,y);
getch();
}
void swap(int *a,int *b)
{
int *temp=0;
*temp=*a;
*a=*b;
*b=*temp;
printf("the value of x,y in functions are %d %d\n",*a,*b);
getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter value of x, y: 4      5
```

Output:

```
The value of x, y in functions are 5         4
The value of x, y in main are 5    4
```

Experiment No : 14.b

a) Write a C program to swap two numbers using pointers

.

Aim: Write a C program to swap two numbers using pointers

Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int x,y,*a,*b,temp;
    clrscr();
    printf("Enter the value of x and y\n");
    scanf("%d%d",&x,&y);
    printf("Before Swapping\nx = %d\ny = %d\n", x, y);
    a = &x;
    b = &y;
    temp = *b;
   *b   = *a;
   *a   = temp;
    printf("After Swapping\nx = %d\ny = %d\n", x, y);
    getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the value of x and y: 2       3
```

Output:

```
Before swapping
X=2
Y=3
After swapping
X=3
Y=2
```

Experiment No : 15

Examples which explores the use of structures, union and other user defined variables.

Aim Program to explain use of structures

Source Code:

```
/*PROGRAM TO EXPLAIN THE USE OF structures */
#include<stdio.h>
struct student
{
int rollno;
char name[10];
int m1;
int m2;
int m3;
};
void main()
{
struct student s;
int total;
clrscr();
printf("Enter the rollno");
scanf("%d",&s.rollno);
printf("Enter the name");
scanf("%s",s.name);
printf("Enter marks in m1");
scanf("%d",&s.m1);
printf("Enter marks in m2");
scanf("%d",&s.m2);
printf("Enter marks in m3");
scanf("%d",&s.m3);
total=s.m1+s.m2+s.m3;
printf("Student No:%d\n",s.rollno);
printf("Student Name:%s\n",s.name);
printf("Marks(M1):%d\n",s.m1);
printf("Marks(M2):%d\n",s.m2);
printf("Marks(M3):%d\n",s.m3);
printf("Total:%d",total);
getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the Rollno: 201
Enter the Name: krishna
Enter marks in m1: 55
Enter marks in m2: 45
Enter marks in m3: 67
```

Output:

```
Student Rollno: 201
Student Name: krishna
Marks(M1): 55
Marks(M2): 45
Marks(M3): 67
Total: 167
```

Aim: Program to explain use of union

Source Code:

```
                    /*PROGRAM TO EXPLAIN THE USE OF union */

#include<stdio.h>
union employee
{
float  empid;
char ename[10];
float basic;
};
void main()
{
union employee e;
clrscr();
printf("Enter the Empid:");
scanf("%f",&e.empid);
printf("Employee Id:%f\n",e.empid);
printf("Enter the Employee name:");
scanf("%s",e.ename);
printf("Employee Name:%s\n",e.ename);
printf("Enter basic pay:");
scanf("%f",&e.basic);
printf("Basic Pay:%f\n",e.basic);
getch();
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter the Empid: 21
Enter the Employee name: Krishna
Enter basic pay: 2344
```

Output:

```
Employee Id: 21
Employee Name: Krishna
Basic Pay: 2344.00
```

Aim: Program to explain use of enum

Source Code:

```
                     /*PROGRAM TO EXPLAIN THE USE OF enum */

#include<stdio.h>
void main()
{   enum{monday, tuesday, wednesday, thursday, friday, saturday, sunday} day;
      day = saturday;
         if(day == saturday || day == sunday)
                 printf( "Day is a weekend day");
         else if(day == wednesday)
                 printf("Day is hump day - middle of the work week");
      getch();
}
```

Result : The program executed successfully and it gives a right output.

Output:

```
Day is a weekend day
```

Experiment No : 16.a

Write a C program which copies one file to another.

Aim: Program to copy one file to another using files.

Source Code:

```
                    /*PROGRAM TO COPY ONE FILE TO ANOTHER */
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main(int arg,char *arr[])
{
      FILE *fs,*ft;
      char ch;
      clrscr();
      if(arg!=3)
      {
      printf("Argument Missing ! Press key to exit.");
      getch();
      exit(0);
}
fs = fopen(arr[1],"r");
if(fs==NULL)
{
      printf("Cannot open source file ! Press key to exit.");
      getch();
      exit(0);
}

ft = fopen(arr[2],"w");
if(ft==NULL)
{
      printf("Cannot copy file ! Press key to exit.");
      fclose(fs);
      getch();
      exit(0);
}
while(1)
{
      ch = getc(fs);
      if(ch==EOF)
      {
      break;
      }
      else
      putc(ch,ft);
      }
      printf("File copied succesfully!");
      fclose(fs);
      fclose(ft);
}
```

Result : The program executed successfully and it gives a right output.

Output:

```
d:\> filecopy file1.txt file2.txt
```

Experiment No : 16.b

b) Write a C program to count the number of characters and number of lines in a file.

Aim: Program to count the number of characters and number of lines in a file.

Source Code:

```
/*********PROGRAM TO FIND SUBSTRING IN MAIN STRING USING FUNCTIONS**********/

#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
      int linescount(char);
      int wordscount(char);
      int charcount(char)
      char text[100];
      int l=0,w=0,c=0;
      clrscr();
      printf("\n Enter a text:\n");
      scanf("%[^z]s",text);
      l=linescount(text);
      w=wordscount(text);
      c=charcount(text);
      printf("%d are no of lines\n%d are no of words\n %d are no of
      characters\n",l,w,c);
      getch();
}
int linescount(char text[])
{
      int l=0,i;
      char ch;
      for(i=0;text[i]!='\0';i++)
      {
            ch=text[i];
            if(ch=='\n')
            {
                  l++;
            }
      }
      return(l);
}
int wordscount(char text[])
{
      int w=0,i;
      char ch;
      for(i=0;text[i]!='\0';i++)
      {
            ch=text[i];
            if((ch==' ')||(ch=='\n'))
            {
            w++;
            }
      }
      return(w);
}
int charcount(char text[])
{
      int i;
      char ch;
      for(i=0;text[i]!='\0';i++)
      {
            ch=text[i];
      }
      return(i);
}
```

Result : The program executed successfully and it gives a right output.

Input:

```
Enter a text: this is some thing
Hi
Hello z
```

Output:

```
no of lines 3
no of words 6
no of characters 28
```

Experiment No : 16.c

c) Write a C Program to merge two files into a third file. The names of the files must be entered using command line arguments.

Aim: To merge two files into a third file. The names of the files must be entered using command line arguments.

Source code:

```
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
void main(int arg,char *arr[])
{
                FILE *fs1, *fs2, *ft;
                 char ch;
               clrscr();
               if(arg!=4)
               {
               printf("argument missing ! press key to exit.");
               getch();
               exit(0);
}
 fs1 = fopen(arr[1],"r");
 fs2 = fopen(arr[2],"r");
 if( fs1 == NULL || fs2 == NULL )
 {
               perror("Error ");
               printf("Press any key to exit...\n");
                getch();
               exit(EXIT_FAILURE);
  }
 ft = fopen(arr[3],"w");
 if( ft == NULL )
 {
      perror("Error ");
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
 }
 while( ( ch = fgetc(fs1) ) != EOF )
 fputc(ch,ft);
 while( ( ch = fgetc(fs2) ) != EOF )
 fputc(ch,ft);
 printf("Two files were merged into %s file successfully.\n",file3);
 fclose(fs1);
   fclose(fs2);
   fclose(ft);
 }
```

Result : The program executed successfully and it gives a right output.

Input: open command prompt.. move to your program location..

```
C:\software\TC>PROGNAME File1.txt File2.txt File3.txt
```

Output:

```
Two files were merged into File3.txt file successfully
```