EC 8393 FUNDAMENTALS OF C PROGRAMMING

Unit – I

1.1'C' Programming Basics

Problem Formulation & Problem solving

Problem formulation :- is fundamental to all analyst it involves decomposition of the analytic problem into apropos dimensions such as structures, functions, mission areas.

Problem formulation is an iterative process which is the advantage like

i) Time saving ii) Quality

Problem solving : to know

How to solve the problem

To understand what the problem is.

Method of problem solving

1. Recognize and understand the problem
2. Accumulate facts
3. Select appropriate theory
4. Make necessary assumptions
5. Solve the problem
6. Verify results

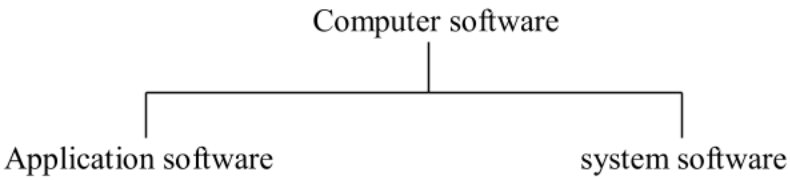
5 steps in using a computer as a problem solving tool

1. Develop an Algorithm and a Flowchart
2. Write the program in a computer language
3. Enter the problem into the computer
4. Test and debug the program
5. Run the program, input data and get the results from the computer

1.2 Introduction to 'C' programming

Introduction

Computer software may be classified into application software and system software.



Application software:

Is the set of programs necessary to carry out operations for specified applications.

System software:

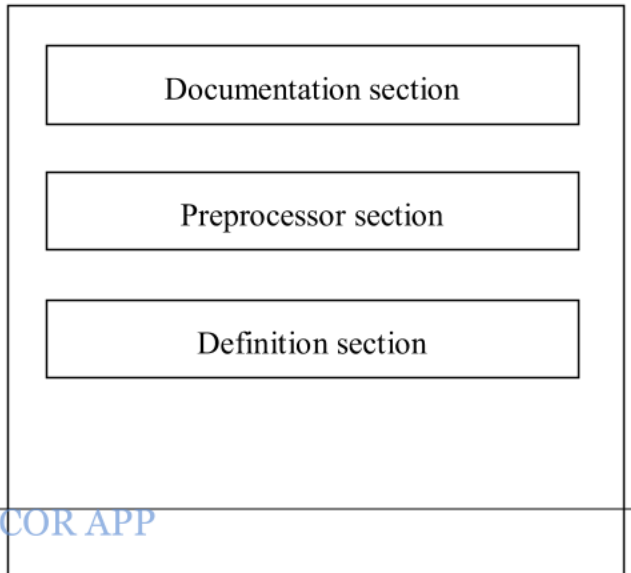
Are the general programs written for the system, which provide the environment to facilitate the writing of application software.

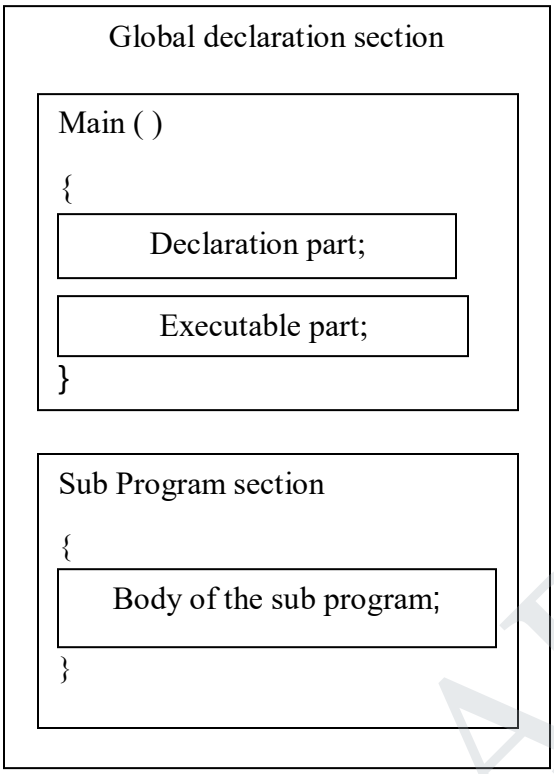
Features and Applications of 'C' language

1. 'C' is a general purpose, structural programming language.
2. 'C' is highly portable.
3. 'C' is a robust language.
4. 'C' is well suited for writing system software and application software.
5. 'C' is a middle level language.
6. 'C' languages allow dynamic memory allocation.
7. 'C' is widely available commercial.
8. 'C' compilers are available on most pc.
9. 'C' has got rich set of operators.
10. 'C' can be applied in system programming areas like compilers, Interpreters and Assemblers etc.

1.3 Structure of a 'C' program & compilation and Linking Parocess

Structure of a 'C' program





i)Documentation section :-

It consists a set of comment line used to specify the name of the program, author name etc.

ii) Comments /* */

Comments are very helpful in identifying the program features and logic of the program.

the lines begins with /* and ending with */ are known as comment lines

iii) Preprocessor section :-

It is used to link system library files, for defining the macros and for defining the conditional inclusion.

eg :- # include<stido.h>

define A 10

iv) Global declaration section :-

The variables that are use in more than one function throughout the program are called global variables & declared outside of all function (i.d) before main ()

Two parts → Declaration part
→ Executable part

v) Declaration part :-

This part is used to declare all the variables that as used in the executable part of the program and these are called local variables.

vi) Executable part :-

It contains at least one valid 'C' Statement. The execute of a program begins with opening brace '{' and ends with closing brace '}'

1.4 Executing a 'C' Program

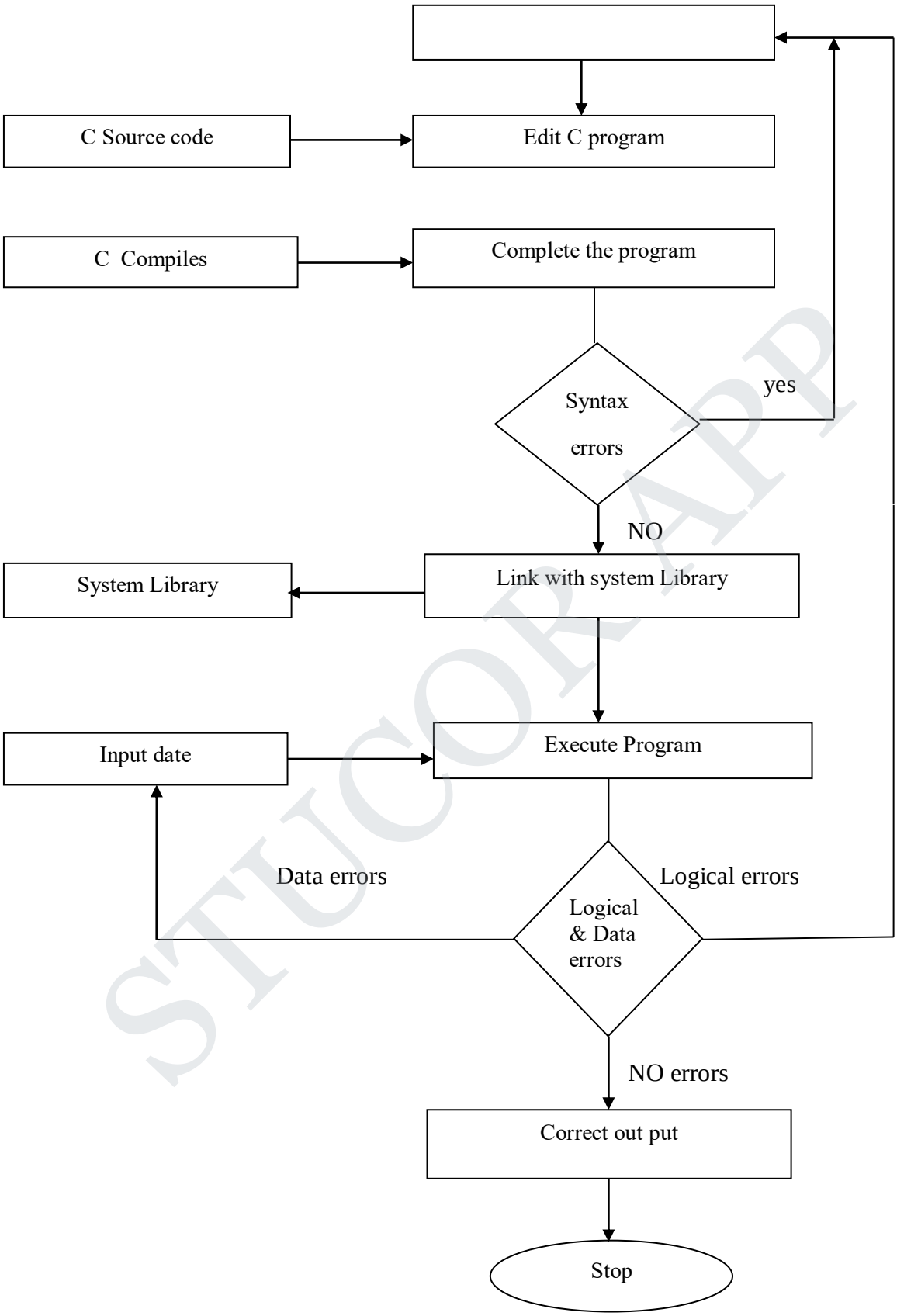
Execution is the process of running the program, to execute a 'C' program, we need to the steps.

1.Creating we need to the steps.

2. Compiling the program

3. Linking the program with system library

4. Executing the program.



Errors : - Two types

i)Logical errors :- These are the errors, in which the conditional and control statement cannot end their match after some sequential execution.

ii) Data errors :- These are the errors, in which the input data given, is not in a proper syntax as specified in input statements.

of a C Program :-

1.Clarity – It refers to the readability of the program

2. Integrity – It refers to the accuracy of the program

3. Simplicity – The clarity and accuracy of a program is enhance by keeping the things as simple as Possible.

4. Efficiency – It refers the execution speed and efficient memory utilization of the program. It enhance the accuracy and clarity of the programs.

5. Modularity – Manu programs can be broken down into a serie of sub-programs. It enhance the accuracy and clarity of the programs.

6. Generality – The program to be as general as possible, with reasonable units.

1.5 Compiling & executing a 'C' program

1)My computer → Local disk(D:) → TC → Bin

↓

tc

2) File New

3) Type the program

4) Compile → ALT+F9

5) Run → ALT+F9

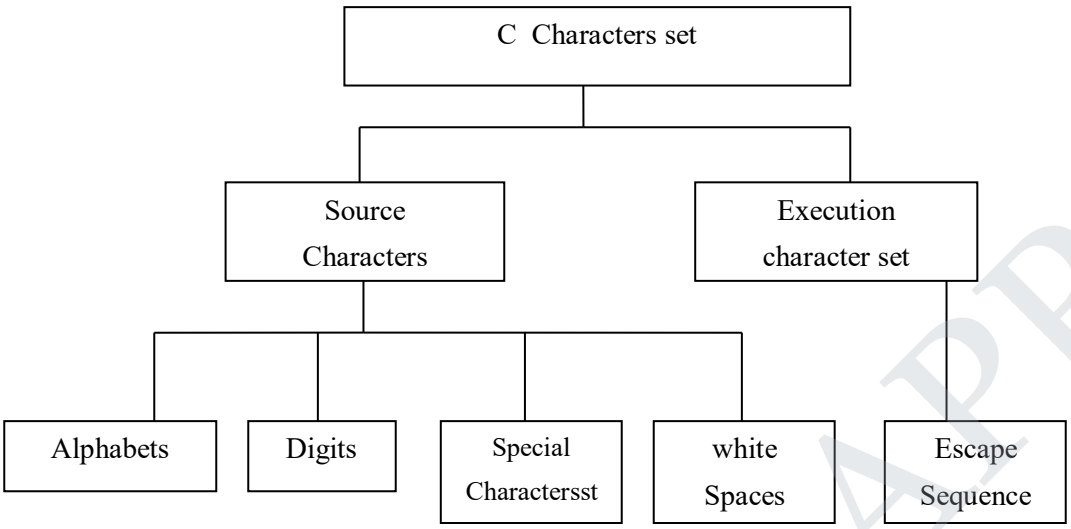
While compiling, any error occur, clear the error. Each error shows line number where the error was detected as well the type of errors.

'C' Character set

The character set is the fundamental raw material of any language and they are used to represent information. This is used to build the programs.

Two types → Source character set

. → Execution character set



Source character set :-

These are used to construct the statements in the source program.

These are of four types :

Source character set	Notation
1)Alphabets	A to Z and a to z
2) Decimal Digits	0 to 9
3) white spaces	Blank space, Horizontal & vertical tab
4) Special characters	+ plus, * asterisk, - minus , comma, ; semicolon, :colon, % percent, $ dollar, ? question mark, N tilde, ”quotation, <less than >greater than, _underscore, ╱╲caret, =equal to ….etc.

Keyword :-

These are certain reserved words called keywords, that have standard and predefined meaning in 'C' Language. Which cannot be changed and they are the basic building blocks for program statements.

The 'C' keywords are listed below.

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

1.6 CONSTANTS/LITERALS

A constant is a value or an identifier whose value cannot be altered in a program. For example: 1, 2.5, "C programming is easy", etc.

As mentioned, an identifier also can be defined as a constant.

```
const double PI = 3.14
```

Here, PI is a constant. Basically what it means is that, PI and 3.14 is same for this program.

Below are the different types of constants you can use in C.

1. Integer constants

An integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

For example:

```
Decimal constants: 0, -9, 22 etc
Octal constants: 021, 077, 033 etc
Hexadecimal constants: 0x7f, 0x2a, 0x521 etc
```

In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

2. Floating-point constantsA floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

```
-2.0
0.0000234
-0.22E-5
```

Note: E-5 = 10^{-5}

3. Character constants

A character constant is a constant which uses single quotation around characters. For example: 'a', 'l', 'm', 'F'

4. Escape Sequences Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc. In order to use these characters, escape sequence is used.

For example: \n is used for newline. The backslash (\ ) causes "escape" from the normal way the characters are interpreted by the compiler.

Escape Sequences	
Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

5. String constants

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

```
"good"                  //string constant
""                  //null string constant
"      "            //string constant of six white space
"x"                  //string constant having single character.
"Earth is round\n"        //prints string with newline
```

6. Enumeration constants

Keyword enum is used to define enumeration types. For example:

```
enum color {yellow, green, black, white};
```

Here, color is a variable and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively. For more information, visit page: C Enumeration.

1. 7 VARIABLES

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

```
int playerScore = 95;
```

Here, playerScore is a variable of integer type. The variable is assigned value: 95.

The value of a variable can be changed, hence the name 'variable'.

In C programming, you have to declare a variable before you can use it.

Rules for naming a variable in C

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore. However, it is discouraged to start variable name with an underscore. It is because variable name that starts with an underscore can conflict with system name and may cause error.
3. There is no rule on how long a variable can be. However, only the first 31 characters of a variable are checked by the compiler. So, the first 31 letters of two variables in a program should be different.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive. Based on the basic types explained in the previous chapter, there will be the following basic variable types −

Sr.No.	Type & Description
1	**char** Typically a single octet(one byte). This is an integer type.
2	**int** The most natural size of integer for the machine.
3	**float** A single-precision floating point value.

4	**double**	A double-precision floating point value.
5	**void**	Represents the absence of type.

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

Variable Definition in C

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows −

type variable_list;

Here, **type** must be a valid C data type including char, w_char, int, float, double, bool, or any user-defined object; and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here −

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** declares and defines the variables i, j, and k; which instruct the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows −

type variable_name = value;

Some examples are −

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables are undefined.

Variable Declaration in C

A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation without requiring the complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use the keyword **extern** to declare a variable at any place. Though you can declare a variable multiple times in your C program, it can be defined only once in a file, a function, or a block of code.

Example

Try the following example, where variables have been declared at the top, but they have been defined and initialized inside the main function −

Live Demo

```
#include <stdio.h>

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main () {

   /* variable definition: */
   int a, b;
   int c;
   float f;

   /* actual initialization */
   a = 10;
   b = 20;

   c = a + b;
   printf("value of c : %d \n", c);

   f = 70.0/3.0;
   printf("value of f : %f \n", f);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
value of c : 30
value of f : 23.333334
```

The same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. For example −

```
// function declaration
int func();

int main() {

   // function call
   int i = func();
}

// function definition
int func() {
   return 0;
}
```

1.8 DATA TYPES :-

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows −

Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void** The type specifier *void* indicates that no value is available.
4	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges −

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

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions *sizeof(type)* yields the storage size of the object or type in bytes. Given below is an example to get the size of int type on any machine −

Live Demo

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

int main() {
   printf("Storage size for int : %d \n", sizeof(int));

   return 0;
}
```

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision −

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

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. The following example prints the storage space taken by a float type and its range values −

Live Demo

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

int main() {
   printf("Storage size for float : %d \n", sizeof(float));
   printf("Minimum float positive value: %E\n", FLT_MIN );
   printf("Maximum float positive value: %E\n", FLT_MAX );
   printf("Precision value: %d\n", FLT_DIG );

   return 0;
}
```

compile and execute the above program, it produces the following result on Linux −

```
Storage size for float : 4
Minimum float positive value: 1.175494E-38
Maximum float positive value: 3.402823E+38
Precision value: 6
```

1.9 OPERATORS

Definition

i. An operator is a symbol that specifies an operation to be performed on the operands.
ii. The data items that operators acts upon are called operands.
iii. Operators are usually form a part of mathematical (or) logical expression.

Examples: a+b

First Operand : a

Operator : +

Second Operand: b

Types of Operators

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment And Decrement Operators
6. Bitwise Operators
7. Special Operators

1. Arithmetic Operators

'C' Allows us to carryout basic arithmetic operations like addition, subtraction, multiplications& division.

OPERATORS	MEANING	EXAMPLE
+	Addition	2+9=11
-	Subtraction,	9-2=7
*	Multiplications	2*9=18
/	Division.	9/3=3
%	Modulo division	9%2=1

Example

```
main()
{
```

int a=10,b=3,c;

Printf("%d",c);

}

OUTPUT : 1

Arithmetic Operator can be classified as,

1. Unary arithmetic : It requires only one operand

Example : +X,-Y

2. Binary arithmetic : It requires two operands

Example : a+b ,a-b , a/b , a*b , a%b.

3. Integer arithmetic : It requires both operands are integer values for arithmetic operation.

Example : a=5,b=4

Expression	Result
a+b	9
a-b	1

4.Floating point arithmetic : It requires both operands are float type for arithmetic operations.

Example : a= 6.5 , b=3.5

Expression	Result
a+b	10.0
a-b	3.0

Example Program

/*Program to illustrate the usage of arithmetic operator*/

```
#include <stdio.h>
#include<conio.h>
voidmain()
{
int i.j.k;                    /*Local declaration*/
clrscr();
i=10;
j=20;
```

```
k=i+j;
Printf("Value of k is %d", k);
getch();
}
```

2.Relational Operators

Relational operators are used to compare two or more operands

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

Example Program

```
#include <stdio.h>
#include<conio.h>
voidmain()
{
clrscr();
Printf("Condition : return value \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<= 5 : %d ", 5<= 5);
Printf("\n  5!= 3 : %d ", 5!= 3);
}
```

OUTPUT

```
Condition      :      Return value
5!= 5          :      0
```

5== 5 : 1

5>= 5 : 1

5<= 5 : 1

5!= 3 : 1

3.Logical Operators:

Logical operators are used to combine the results of two (or) more conditions

Operator	Meaning	Example	Return Value
&&	Logical AND	(9>2)&&(17>2)	1
||	Logical OR	(9>2)||(17==7)	1
!	Logical NOT	29!=29	0

Example:

(i>=6)&&(c=='w') if i=6,

(6>=6)&&(c=='w')

True && False

1&& 0

Result is 0

Example : (C!='P')||(i<=100) ,if i=50

True || (50<=100)

True || True

1 || 1

Value is 1.

Example Program:

```
#include <stdio.h>
#include<conio.h>
voidmain()
{
int c1,c2,c3;
clrscr();
Printf("Enter values of c1,c2,and c3");
Scanf ("%d%d%d", &c1,&c2 &c3);
```

```
if((c1<c2)&&(c1<c3))
printf("\n c1 is less than c2 and c3") ;
if(!(c1<c2))
printf("\n c1 is greater than c2") ;
if((c1<c2)||(c1<c3))
printf("\n c1 is less than c2 (or) c3 (or) both") ;
getch();
}
```

OUTPUT:

Enter value of c1,c2 and c3 : 43 32 98

c1 is greater than c2

c1 is less than c2 (or) c3 (or) both.

4.Assignment Operator:

Assignment operators are used to assign a value of a variable to another variable.

Syntax : Variable = Expression (or) Value ;

Example : X=10; X=a+b;X=Y

Example Program :

```
/* Program to demonstrate assignment operator*/
 #include <stdio.h>
#include<conio.h>
voidmain()
{
int  i,j,k;
clrscr();
k= (i=4,j=5);
Printf("k=%d",k);
getch();
}
```

OUTPUT: k=5

There are two types of assignment operators, they are

i. Compound Assignment
ii. Nested (or) Multiple Assignment

5.Compound Assignment :

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

ii.Nested (or) Multiple Assignment :

Syntax : Var1=Var2=Var 3=…….Var n= Single Value;

Example : i=j=k=1;

6.Increment And Decrement Operators(Unary Operators)

i. The'++' adds one to the variable,
ii. The'--' subtracts one from the variable,

Operator	Meaning
++X	Pre Increment
--X	Pre Decrement
X++	Post Increment
X--	Post Decrement

X= a++ ,if a=1 /*then a++ = a+1 = 1+1 = 2*/ /* increment 'a ' by 1*/

X = a+1

X=2

Content of 'a' Before	Operator	Value	Content of 'a' after
5	a++	5	6
5	a—	5	4

Example Program:

```
/*Program Using increment and decrement operators*/
#include <stdio.h>
#include<conio.h>
voidmain()
{
int a=10;
Printf ("a++ = %d", a++);
Printf ("++a = %d", ++a);
Printf ("--a = %d", --a);
Printf ("a-- = %d", a--);
}
```

OUTPUT:

a++ : 10

++a : 12

--a : 11

a-- : 11

7.Conditional Operator (or) Ternary Operator:

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

Syntax : Condition? Expression 1 : Expression 2;

Meaning of Syntax :

i. Condition true ,it will executes expression1.
ii. Condition false, it will executes expression2.

Example :

13== 8 ? 100 : 200 , it returns 200, because 13 not equal to 8.

Condition ? Expression 1 : Expression 2;

Executes if Condition True

Executes if Condition False

Example Program:

```
#include <stdio.h>
#include<conio.h>
voidmain()
{
int a=5,b=3,big;
big = a>b?a:b;
Printf (“Big is …….%d”,big);
}
```

OUTPUT : Big is ……..5

8 .Bitwise Operators:

Bitwise operators are used to manipulate the data at bit level . It operates on integer only. it may not be applied to float (or) real(or) character

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	One’s Complement

Bitwise AND (&):

&	0	1
0	0	0
1	0	1

Example X =7 = 0000 0111

Y=8 = 0000 1000

X&Y = 0000 0000

Bitwise OR (|)

|	0	1
0	0	1

1	0	1

Example X =7 = 0000 0111

Y=8 = 0000 1000

X|Y = 0000 1111

Bitwise Exclusive OR(^) :

^	0	1
0	0	1
1	1	0

Example X = 13 = 0000 1101

Y = 08 = 0000 1000

X^Y = 0000 0101

Possible Combinations

a	b	a | b	a &b	a^ b	~a
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

9.The Special Operator:

'C' language supports some of the special operators,

Operators	Meaning
,	Comma Operator
size of	Size of operator
& and *	Pointer Operators
. and →	Member selection operator

a)Comma Operator:

It is used to separate the statement elements such as variable ,constants (or) expression.

Example :

val=(a=3,b=9,c=77,a+c);

b)size of() operator :

It returns the bytes of the variable.

Syntax : Sizeof (var);

Example:

```
Main()
{
int a;
Printf ("Size of variable a is ……%d ", size of (a));
}
```

OUTPUT :

Size of variable a is …….2

c)Pointer Operator:

& : This Symbol specifies the address of the variable.

* : This Symbol specifies the value of the variable.

d)Member selection operators:

. and → : These symbols used to access the elements from a structure.

Managing I/P and O/P Operation:

- Unformatted I/P and O/P Operation
- Formatted I/P and O/P Operation

Unformatted I/P and O/P Operation

I/P – get	O/P—Put
→getchar	→ gets
→ Putchar	→ puts
➔ getc	→ getch
➔ put c	→ Putch

getch:

get the input directly from the key but show

Formatted I/P and O/P Operation

Printf ()

Scanf ()

1.10. MANAGING INPUT OUTPUT OPERATIOINS

Input, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.

When we say **Output**, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.

The Standard Files

C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

Standard File	File Pointer	Device
Standard input	stdin	Keyboard
Standard output	stdout	Screen
Standard error	stderr	Your screen

The file pointers are the means to access the file for reading and writing purpose. This section explains how to read values from the screen and how to print the result on the screen.

The getchar() and putchar() Functions

The **int getchar(void)** function reads the next available character from the screen and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one character from the screen.

The **int putchar(int c)** function puts the passed character on the screen and returns the same character. This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character on the screen. Check the following example −

```
#include <stdio.h>
int main( ) {

   int c;
   printf( "Enter a value :");
```

```
   c = getchar( );
   printf( "\nYou entered: ");
   putchar( c );
   return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads only a single character and displays it as follows −

```
$./a.out
Enter a value : this is test
You entered: t
```

The gets() and puts() Functions

The **char *gets(char *s)** function reads a line from **stdin** into the buffer pointed to by **s** until either a terminating newline or EOF (End of File).

The **int puts(const char *s)** function writes the string 's' and 'a' trailing newline to **stdout**.

```
#include <stdio.h>
int main( ) {
   char str[100];
   printf( "Enter a value :");
   gets( str );
   printf( "\nYou entered: ");
   puts( str );
   return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads the complete line till end, and displays it as follows −

```
$./a.out
Enter a value : this is test
You entered: this is test
```

The scanf() and printf() Functions

The **int scanf(const char *format, ...)** function reads the input from the standard input stream **stdin** and scans that input according to the **format**provided.

The **int printf(const char *format, ...)** function writes the output to the standard output stream **stdout** and produces the output according to the format provided.

The **format** can be a simple constant string, but you can specify %s, %d, %c, %f, etc., to print or read strings, integer, character or float respectively. There are many other formatting options available which can be used based on requirements. Let us now proceed with a simple example to understand the concepts better −

```
#include <stdio.h>
int main( ) {

   char str[100];
   int i;

   printf( "Enter a value :");
   scanf("%s %d", str, &i);
   printf( "\nYou entered: %s %d ", str, i);
   return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then program proceeds and reads the input and displays it as follows −

```
$./a.out
Enter a value : seven 7
You entered: seven 7
```

Here, it should be noted that scanf() expects input in the same format as you provided %s and %d, which means you have to provide valid inputs like "string integer". If you provide "string string" or "integer integer", then it will be assumed as wrong input. Secondly, while reading a string, scanf() stops reading as soon as it encounters a space, so "this is test" are three strings for scanf().

1.11 Decision making and branching

Control statements – Introduction

In a program all the instructions ate executed sequeally by default. In some situation we may have to change the execution order of statements based on condition until cert. conditions are met. In such situation control statements are very useful.

'C' language provides 4 general categories of control structures.

if the condition true, then the statement i=i+1 will be execute other wise it executes j=j+1

1. Iteration structure, in which statements are repeatedly executed

   example : for (i=1; i<=5;i++)

   {i=i+1:

   }1) Sequential structure :- in which instructions are executed in seq

   example : i=i+1;

   j=j=1;

The above statements are executed one by one.

2. Selection structure :- The sequence of instructions are determines by using the result of the condition.

   example : if(x>y)

   i=i+1;

   else

   j=j+1;

Where the statement i=i+1 will be executed 5 times and value of I will change from 1,2,3,4 and 5

3. Encapsulation structure, in which the others compound statements are included.

   for loop in a if statement

'C' Language provides the following conditional (decision making) Statements.

if statement
if…..else statement
nested if  else statements
if  else ladder

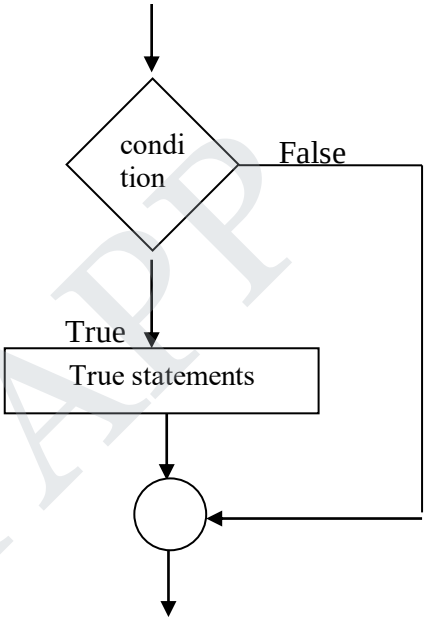
1)The if statement :

The if statement is a decision making statement. It is use ot control the flow of execution of the statements and used to theologically whether the conditions is true or false.

Syntax :

```
if (condition is true)
{
True statements
}
```

Flowchart



Properties of an if statement

1)If the condition is true, than the simple condition statements are executed.

2) If the condition is false, it does not do any thing.

3) The condition is given in parenthesis and must be evaluate as true (non-zero) or false (zero value)

4) If a compound structure is provided. It must be enclosed in opening and closing braces.

example 1 :

```
/* program to check whether the entered number is less than 10*/
#include<stdio.h>
#include<conio.h>
void main ( )
{
int  i  ;
clrscr (  )
print f (" Enter the number") ;
scan f ("%d", &i);
```

output

```
enter the number :5
the number is less than
```

```
if (i<10)
}
Print f (“The number is less than 10”)
}
getch( );
}
```

example 2 :

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
int  n ;
clrscr (  )
print f (“ Enter the number”) ;
scan f (“%d”, &n);
if (n%2 = = 0)
}
Print f (“The number is even”)
getch( );
}
```

output 1

enter the number :6

the number is even

out put 2

enter the number 5

(simply give the blank screen)

example 3 :

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
int  m, n a ;
clrscr (  )
print f (“ Enter two number”) ;
scan f (“%d. %d”, &m, &n);
if (m>n)
}
a=m;
```

output 1

enter the number :6

the number is even

out put 2

enter the number 5

(simply give the blank screen)

```
m=n;
n=a;
}
Print f (“The inter changed value %d %d” m,n);
getch( );
}
```

output

Enter two numbers 34 28

the interchanged values are : 28   34

2) if – else statement

It is two way decision making statements

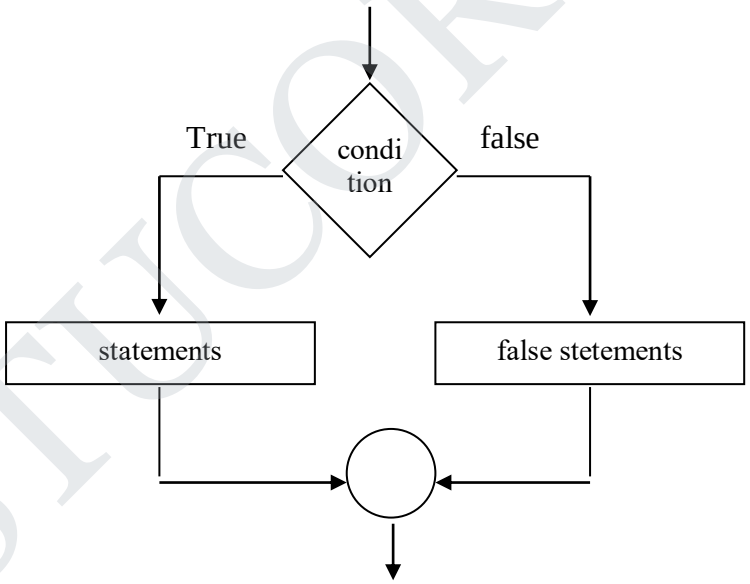
It is used to control the flow of execution and also used to carry out the logical test and pickup nay one of the two possible actions depending on the logical test.

it is used to execute some other statements, when the condition is false.

Syntax

```
if (condition)
{
true statements;
}
else
{
false statements;
}
```

flowchart



<u>example 1 :</u>

```
#include<stdio.h>
#include<conio.h>
{
void main ( )
int  a;
```

<u>output</u>

enter a value : -10

The number is negative

```
print f (“ Enter a value”) ;
scan f (“%d. &a);
if (n>o)
Print f (“The number is positive”);
else
Print f (“The number is negative”);
getch( );
}
```

example 2 :

```
program to determine give number is even or odd */
#include<stdio.h>
#include<conio.h>
void main ( )
{
int  n, rem ;
clrscr (  );
print f (“ Enter your number_”) ;
scan f (“%d” &n);
rem = n% 2;
if (rem = = 0)
Print f (“The number entered number is even”);
else
Print f (“The number is odd”);
getch( );
}
```

output

```
enter your number - 15
The number is odd
```

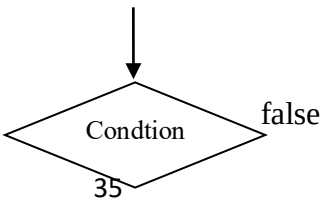
3) Nested if ….else statement

When a series of if…..else statements are occurred in a program, we can write an entire if….else statement in another if…. else statement called nesting, and the statement is called nested if.

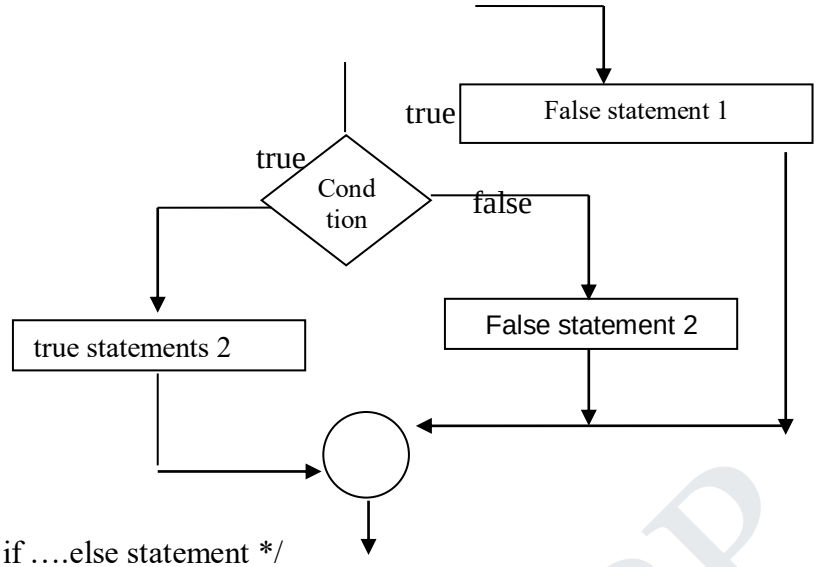
Syntax :

```
if (condition 1)
{
if (condition 2)
```

Flowchart



```
}
true statement 2;
}
else
{
false statement 2;
}
else
{
false statement 1;
}
```



example : -

```
/*program using nested if ….else statement */
#include<stdio.h>
#include<conio.h>
void main ( )
{
int  n,
print f (“ Enter the number ”) ;
scan f (“%d” &n);
if (n= = 15)
Print f (“play foot ball”);
else
{
if (n==10)
print f (“play cricket”);
else
print f (“don’t play”);
}
getch( );
}
```

output

```
enter the number - 10
Play cricket
```

4) The if …..else ladder

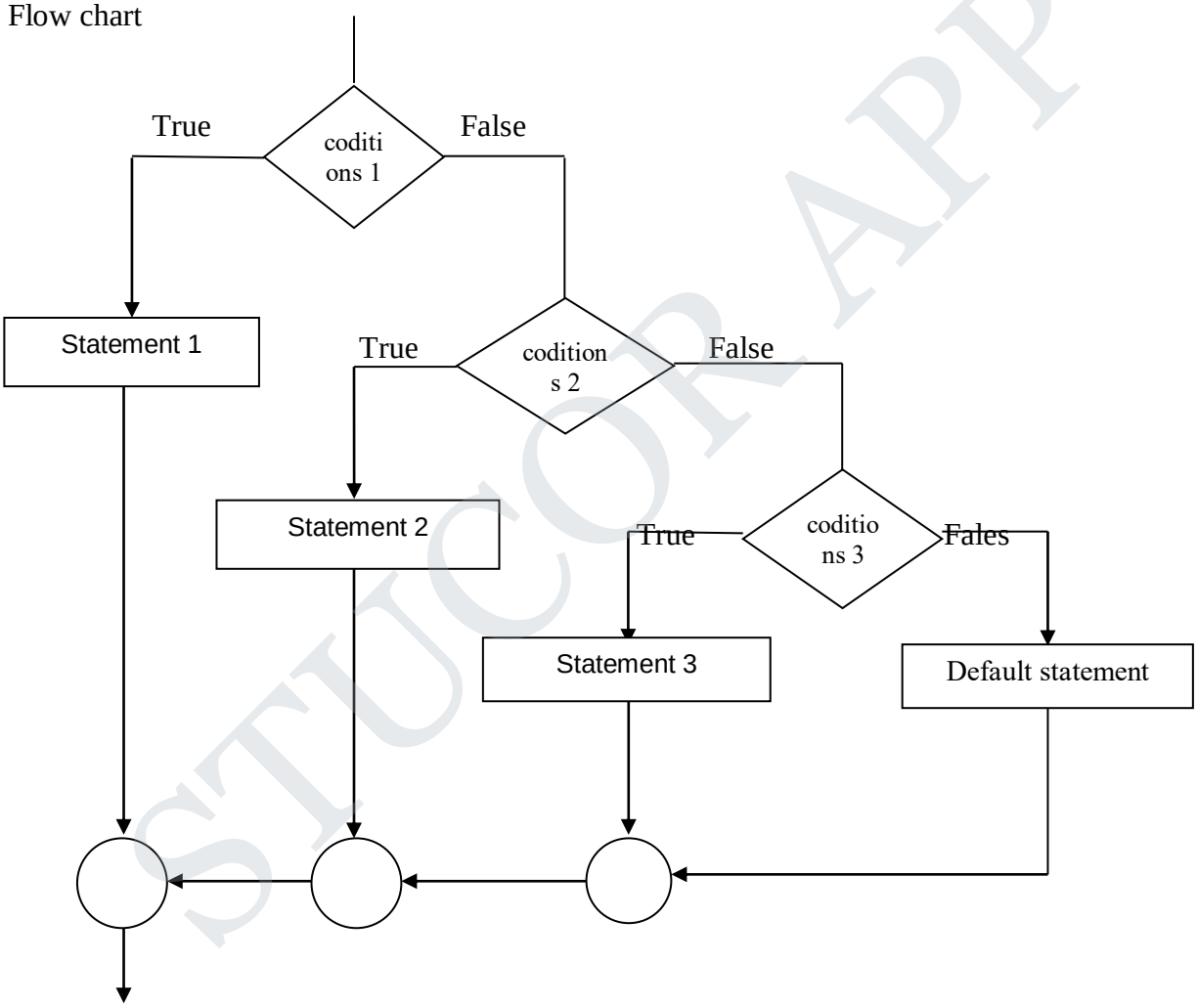
Nested if statements can become quit complex. If there are more than 3 alternatives, is not consistent. In situal you can see the nested if as the else if ladder.

Syntax :

```
if (Condition 1)
{
statement 1 ;
}
else (condition 2)
```

```
{
statement 2
}
else if (condition 3)
{
statement 3 ;
}
else
{
default statement ;
}
```

Flow chart



Example : -

/* program for if…..else ladder */

include<stdio.h>

```
# include<conio.h>
void main ( )
{
int  ch. exp ;
clrscr ( )
print f("1-principal  2.Professor ");
print f ("Enter year of experience ");
scan f ("%d" , & exp) ;
if (ch = = 1)
}
if (exp >=10)
print f ("salary os 40000");
else
print f ("salary is 30000");
}
if (ch==2)
{
if (exp> = 6)
print f ("salary is 25000");
print f ("salary is 20000☺;
}
else
print f ("choice is invalid ");
getch ( ) ;
}
```

output

```
1.Pincipal  2. professor
 Enter  your designation
 1
 Enter your experience
 15
 salary is 40000
```

4. The Switch statement :

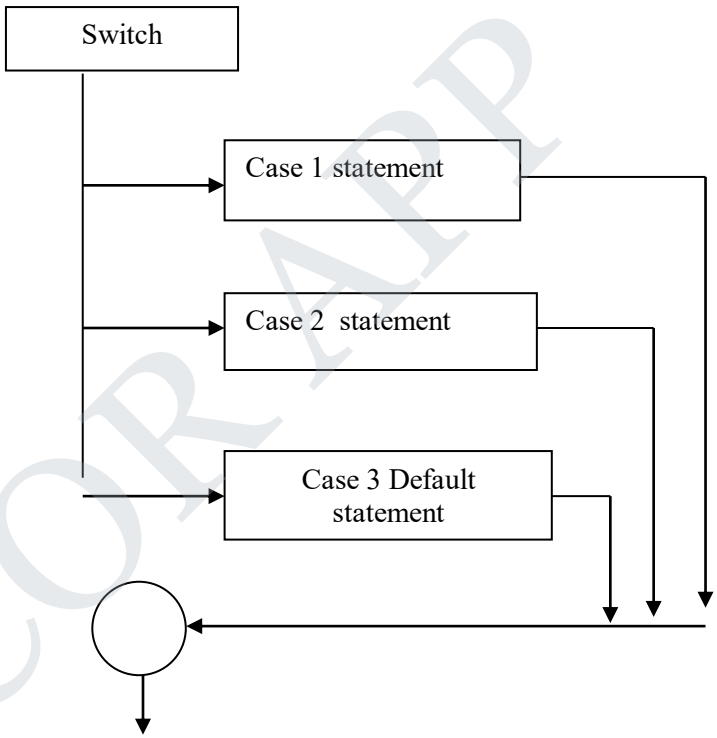
The switch statement is used to pick up or execute a particular group of statements from several group of statements It always us to make a decision from the number of choice.

it is a multi way decision statement.

Syntax :

```
Switch (expression)
{
case constant 1 ;
block 1;
break ;
case contant 2 ;
block 1;
break ;
:
:
default :
default block ;
break ;
}
```

Flowchart



Rules for writing switch ( ) statement :

1. The expression in switch statement must be an integer value or character constant
2. No real number are used in an expression.
3. Each case block and default blocks must be terminated with break statements.
4. The default is optional and can be placed anywhere, but use placed at end.
5. The case keyword must terminate with colon(:).
6. No two case constants are identical.
7. The case labels must be constants.
8. The switch can be nested.

9. The value of switch statement expression is compared with the case constant expression in the order specified, (i.e.) for top to bottom

10. In the absence of break statement, all statements that are followed by matched cases are executed.

example

```
#include<stdio.h>
#include<conio.h>
#void main ()
{
int a,b,c=0;
char op;
clrscr();
printf("calculation code");
print ('+ADD-SUB*MUL /DIV;
printf ("Enter code….."");
Scanf ("%c", & op);
printf ("Enter values");
Scanf ("%d %d", &a,&b);
switch (op)
{
case '+':
c=a+b;
break;
case '-':
c=a-b;
break;
case '*';
c=a*b;
break;
case '/';
c=a/b;
```

```
break;
{
printf ("Result is %d",c);
getch();
}
```

Output

calculation code

+ADD-SUB*MUL /DIV

Enter code….+

Enter values 10 20

Result is 30.

5. Nested switch

'C' Supports the nested switch statements. The inner switch ( ) Statements can be a part of outer switch ( ) statement . The inner and outer switch ( ) case constants may be same no conflicts arises even if the are same.

example :

```
#include<stdio.h>
#include<conio.h>
#void main ()
{
int a,b,c=0;
clrscr();
printf("Enter a number");
Scanf ("%d, &a);
switch (a)
{
case o ;
print f ("The number is even ");
break ;
case 1:
print f ("The number is odd");
break ;
default :
```

out put

Enter a number : 89

The number is odd

```
b=a%2;
switch (b)
{
case o:
printf ("The number is even");
default :
print f ("The number is odd");
}
getch ( );
}
```

Comparison of switch ( ) case and nested if

S.No.	Switch case	Nested if
1.	The switch ( ) can test only constant values.	The if can evaluate relations and logical expressions.
2.	No two case statements have identical constants in the same switch.	same conditions may be repeated for number of times
3.	Character constants are automatically converted to integers.	Character constants are automatically converted to integers.
4.	In switch ( ) case statement nested if can be used.	In nested if statements, switch () case can be used.

IX Solving simple scientific & statistical problems :-

1)write a program to check, whether the given number is prime or not ?

```
# include <stdio.h>
# include<conio.h>
# void main ( )
{
int  n,I,r;
clrscr ( ) ;
print f ("Enter the number");
```

```
scanf (“%d”, & n);
i=2;
Step 1 :                                    output
if (i<=sqrt (n))                            enter the number : 7
{                                           7 is prime
r= n % I;
if (r= =o)
{
print f (“%d is not a prime”,n);
}
}
else
{
i++;
goto step 1 ;
}
print f (“%d is prime”, n);
end :
print f( “   “) ;
getch(  ) ;
}
```

example : 2

```
/* program to convert integer to character using if condition
# include <stdio.h>
# include <conio.h>
void main (  )
{
int  a ;
```

```
clrscr ( ) ;
print f (“Enter a number”);
scan f (“%d”, & a);
if (a = = ‘p’)
print f (“%c”, a);
else if (a = = ‘G’)
print f (“% c”, a);
getch ( ) ;
}
```

output 1

Enter a number 80

P

output 2

Enter a number 65

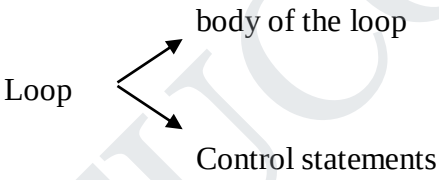
A

1.11 Looping statements :

In some situations there is a need to repeat a set of instructions in specified number of times or unit a particular condition is being satisfied. This repetitive operations are done through a loop control structure.

The loop is defined as the block of statements which are repeatedly executed for certain number of times.

The loop in a program consists two parts.



Control statement is used to test the conditions and directs the repeated execution in body of the loop.

Looping statement would include the following steps :-

1)Initialization of a condition variable

2) Test the control statement

3) Executing the body of the loop depending on the condition.

4) Updating the condition variable

Loop structures available in ‘C’

while…
do…..while
for……

The while loop :

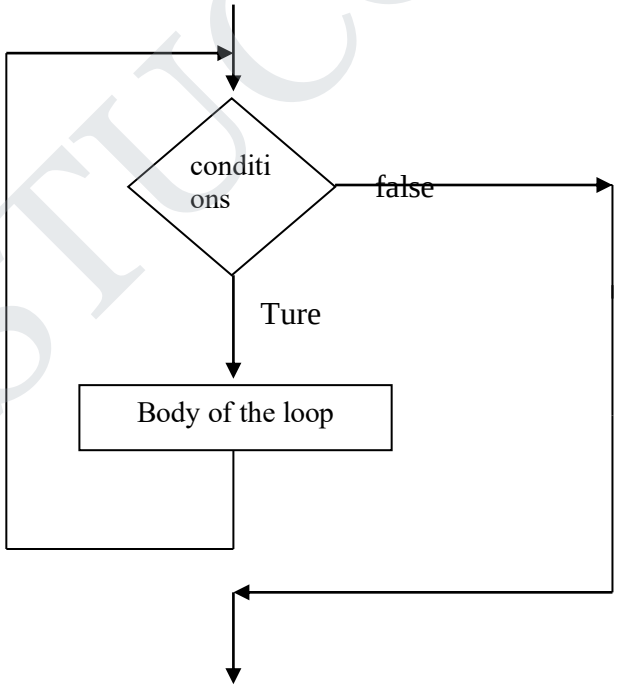
It is a repetitive control structure, used to execute the statements within the body until the condition becomes false the condition is evaluated first and it is true, the body of the loop is executed.

After executing the body of the loop, condition is once age evaluated and if it is true, the body is executed once again, the process of repeated. execution of the loop continue until the condition becomes false.

Syntax :

```
while (condition)
{……..
body of the loop ;
……
}
```

Flow chart



Example programs

/*adding of numbers upto 10 by using while loop */

```
# include <stdio.h>
# include <conio.h>
void main ( )
{
int  I = 1,  sum = 0 ;
while (i<=10)
{
sum = sum=i
i++;
}
printf (“The sum of numbers upto is %d”, sum);
getch (  ) ;
}
```

output :-

The sum of numbers up to 10 is 55

Print-the given number using while loop*/

```
#include <stdio.h>
#include <conio.h>
void main()
{
int number, digit, rev=0;
clrscr();
printf (“Enter the number”);
Scanf (“%d”,&number);
while (number!=0)
{
digit= number% 10;
rev=rev*10+digit;
number=number/10;
```

```
}
printf ("%d",rev);
getch();
```

Working

Enter number =654

1)while(654!=0)

{

digit=654%10=4

rev=0*10+4=4

number=654/10=65

}

2)while (65!=0)

{

digit=65%10=5

rev=4*10+5=45

number=65/10=6

}

3)while (6!=0)

{

digit=6%10=6

rev=45*10+6=456

number=6/10=0

}

4)while(0!=0)

condition false

The do.. While loop:-

The while loop makes a test of condition before the loop is executed, therefore, the body of the loop may not be executed at all, if the condition is not satisfied at first attempt.

It is also repetitive control structure and executes the body of the loop, then it checks the condition and continues the execution until the condition becomes false.

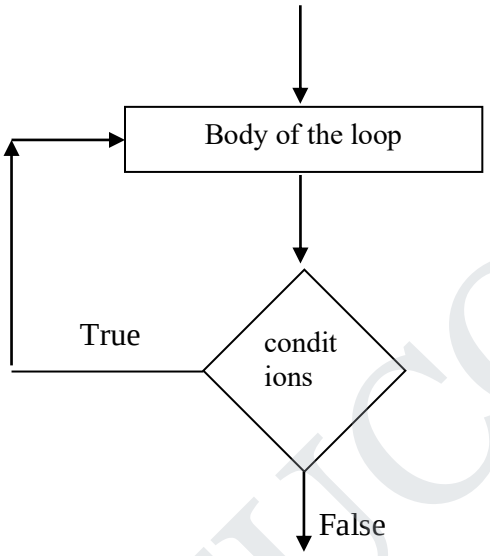
Syntax:- do

{…..

body of the loop:

……

}while (condition);



Example programs :-

/* Addition of numbers uptp 10 by using do …….while loop */

#include <stdio.h>

include<conio.h>

void main ( )

{ int i=1, sum = 0 ;

clrscr ( )

do {

sum = sum+I ;

```
i++ ;
}
while (i<=10);
print f ("sum of numbers upto 10 is …..%d", sum);
getch ( );
}
```

working :

i=1 sum = D+1 = 1	i=6 sum=15+6=21
i= 2 sum = 1+2=3	i=7 sum = 21+7=28
i=3 sum = 3+3=6	i=8 sum = 28+8=36
i=4 sum=6+4=10	i=9 sum = 36+9=45
i=5 sum=10+5=15	i=10 sum = 45+10=55

while (11<=10) condition 1 false

out put :

Sum of numbers upto 10 is……..55

example 2 :

```
/* program to print n numbers using do….while loop */
# include<stdio.h>
# include<conio.h>
void main ( )
{
int i, n ;
clrscr ( );
print f ("Enter the number") ;
scan f ("%d", & n);
i=0;
do
{
```

```
print f (‘the numbers are %d”, i=i+1;
}
while (i<n);
getch ( )
}
```

Enter the number :6

The numbers are 0

The numbers are 1

The numbers are 2

The numbers are 3

The numbers are 4

The numbers are 5

Comparison between while and do… while

S.NO.	while	do…. while
1.	This is the tested loop.	This is the bottom tested loop..
2.	The condition is first tested, if the condition is true then the block is executed until the condition becomes false	It executes the body once, after it checks the condition, if it is true the body is executed until the condition becomes false.
3.	Loop will not be executed if the condition is false	Loop is executed at least once even though the condition is false.

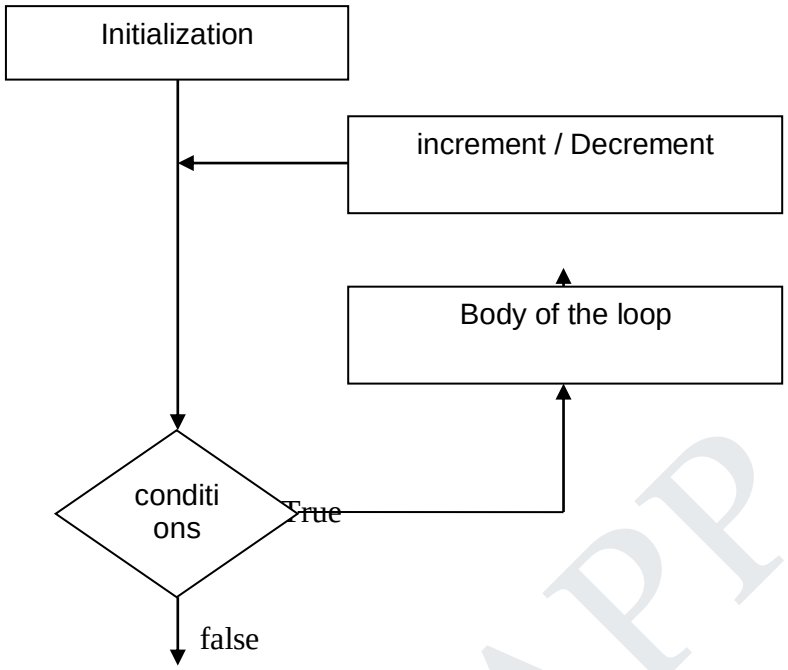
The “for” loop

The foe loop is another repetitive control structure, and used to execute set of instructions repeatedly until the condition becomes false.

The assignment, incrementation (or) decrementation and condition checking is done in for statement only. Other structures does not offered all these features in one statement.

```
for (initialize counter; test condition ; increment / decrement
{ ……..counter)
body of the loop ;
}
```

Flow chart



for loop has three parts :

1.Initialise counter : is used to initialize counter variable

2. Test condition : is used to test the condition

3. Increment / decrement counter is used to increment or decrement counter variable.

example programs :

/* program for addingof numbers using for loop*/

```
#include <stdio.h>
# include <conio.h>
void main  (  )                                    output
{                                                  The addition is : 55
int  I,   sum  = 0
for ( i=1; i<=10; i++)
[
sum = sum=i ;
}
printf (“The addition is % d”, sum);
getch ( );
}
#include<stdio.h>
```

```
#include<conio.h>
void main()
{
int I,n;
clrscr()
printf (‘Enter a number”;
scanf (“%d”,&n);
for (i=o; i<n; i==)
{
printf (The numbers are %d”,I);
}
Getch();
}
```

out put

```
Enter a number: 5
The numbers are: 0
The numbers are: 1
The numbers are: 2
The numbers are: 3
The numbers are: 4
```

Nesting of for loops

The loop within the loop is called nested loop. In nested for loops, two or more for statements are include in the body of the loop.

Example /* program using nesting of for loops*/

```
#include<strudio.h>
void main()
{
int I,j;
clrscr ();
{
for (i=1; i<=3; i++)
```

```
{
printf("in");
for (j=1; j<=3; i++)
printf ("%d\t",j);
```

Output

1 2 3

1 2 3

1 2 3

i) We can initialize more than one variable at a time.

example: for (i=1: j=1; i<=10; i++)

ii) We can increment or decrement more than one part in the incrementation or decrementation section of the for loop.

example: for (i=1; j=1; i<=10; i++,j++0

iii) The test condition may have any compound relation

```
for (j=1; j<10&& sum<100; j++)
{
sum=sum+j;
}
```

iv) The expression in the assignment statement as well as incrementation ot decrementation section.

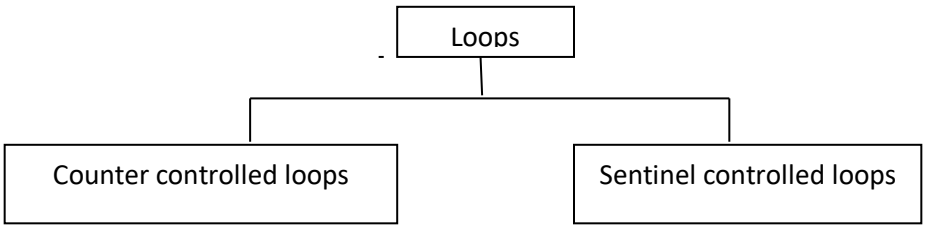
example: for (a=i+j); a>0; a=a/2)

v) One or more section of the loop may be omitted, if necessary.

example: for (;i>100;)

Types of loops

The loops can be divided into two types based on natural of loop control variable assigned for testing the control expression of loop.



Counter controlled loops:-

There are used in a situation when the programmer knows in perior exactly how many times the loops "repetation loop".

example:-

```
i=1;

while 9i<=10)

{

sum=sum+I;

i=i+1;

}
```

Sentinel Controlled loop

The sentinel controlled loops are used in a situalior when the programmer not known in prior, exactly the number of times the loop will be executed. This loop is called a "indefinite repetition loop".

example

```
while (i!=-999)

{

scanf ("%d", &i);

sum=sum+I;)
```

The break statement

The break statement is used to terminate the loop when the keyword "break" is used inside any 'c' loop, contrc automatically transferred to the first statement after the loop.

A break is usually associated with any if statement.

When the break statement is encountered inside a loop the is immediately exited and the program continues with the statement immediately following the loop.

Syntax

```
break;
```

```
/* program using break statement*/

# include<stdio.h>

#include<conio.h>

void main ()

{
```

```
int I;
clrscr ();
for (i=1; i<=10; i++)
{
if (i==6)
break;
printf  ("%d", i);
}
getch ();
}
```

Terminating loop with break statement

```
while (condition)
{….
if (condition)
break;
….
}
```

```
do
{…
if (condition)
break;
…….
}While (Condition)
```

```
for(intializine;condition;incremental/decrement)
{
……
if(condition)
break;
…….
}
```

The Continue statement

When the statement "continue " is encountered in any 'C' loop control automatically passes to the beginning the loop.

Syntax: Continue;

Example

/* program to calculate the sum of the positive numbers*/

```
#include <stdio.h>
#include<conio.h>
voidmain()
{
int I,n,sum=0;
clrscr();
for(i=1;i<=5;i++)
{
Printf("Enter any number");
Scanf("%d",&n);
if(n<0)
continue;
else
sum=sum+n;
}
Printf("Sum is ….%d",sum);
getch();
}
```

Output

Enter any number : 10

Enter any number : 05

Enter any number : 15

Enter any number : 25

Enter any number : -1

Enter any number : 5

Sum is …….105

Continue loop with Continue Statement

While (condition)	do	for (int; cond; increment)
{	{	{
……	……	…….

```
if(condition)          if(condition)          if(condition)
continue ;             continue ;             continue ;
……                     ……                     …….
}                      while (condition)      }
```

S.NO	BREAK	CONTINUE
1	It takes the control to the outside of the loop	It takes the control to the beginning of the loop
2	It is also used in switch statement	This can be used only in the loop statement
3	Always associated with if condition in loops	This is also associated with if condition

The goto Statement

'C' provides the goto statement to transfer control unconditionally from one place to another place in the program.

The goto statement requires a label to identify the places to move the execution. A label is a valid variable name and must be ended with colon (;)

Syntax

```
goto label ;           label;
……                     …….
……                     …….
label;                 goto label;
```

Example

```
/*Program using goto statement */
#include <stdio.h>
#include<conio.h>
voidmain()
{
int a,b;
clrscr();
Printf("Enter any number");
Scanf("%d %d",&a,&b);
```

```
if(a==b)
goto equal;
else
{
Printf (“A&B are not equal”);
exit(0);
}
equal;
Printf (“A & B are equal”);
getch();
}
```

OUTPUT

Enter the numbers 3,3

A& B are equal.

1.12 ARRAY:

Array is a collection of data items of similar that are stored under the common name

Array types:

i) one dimensional array
ii) two dimensional array
iii) multi dimensional array

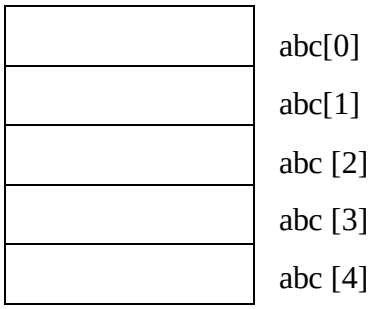
Array declaration:

Syntax:

Data type array type [size or subscript of the array];

Example:

Int abc[5];



Example:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Int a[5], sum=0;
Clrscr;
Printf("enter 5 numbers");
For(i=1;i<=5;i++)
{
Scanf("%d",& a[i]);
Sum=sum+a[i];
}
Printf(sum is .....%d", sum);
```

```
Getch();
}
```

Array initialization of one dimensional array:

⇒ Compile time
⇒ Run time

Compile time:

Syntax:

```
Data_type array name[size]={list of values};
```

Example:

```
Int mark[3]=(70, 80, 90};
```

Runtime:

The array can be initialized at runtime #Example:

```
While(i<=10)
{
If(i<5)
Sum[i]=0;
Else
Sum[i]=sum[i]+o;
}
```

Initialising a two dimensional array;

Syntax:

Data_type array_name [row_size] [column_size;

Example:

```
Int a[3][3];
```

A[0][0]	A[0][1]	A[0][3]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]

```
Ints[4][2]
```

S[0][0]	S[0][1]	S[0][2]	S[0][3]
S[1][0]	S[1][1]	S[1][2]	S[1][3]

Example:

Int student[4][2]={{660,80}, {661,81},{662,82},{663,83}}

Int student[4][2]={660,80,661,81,662,82,663,83}

1.13 TWO DIMENSIONAL ARRAY

Example:

```
#include<conio.h>
#include<stdio,h>
Voidmail()
{
Inta[5][5],b[5][5],c[5][5], r1,r2,c1,c2,i,j,k;
Clrscr();
Step 1;
Printf("/n enter the tize of the matrix a.....")
Scanf("%d,%d", &r1,&c1");
Printf("/nEnter the size of the matrix b..")
Scanf("%d,%d,7r2,&c2");
If(c2==r2)
Goto step2;
Else
Printf("/n Multiplication is not possible");
Goto step1;
Step2;
Printf("/n Enter the matrix A elements../n");
For(i=0;i<r1;i++)
{
For(j=0;j<c1;j++)
```

```
Scanf("%d", &a[i][j]);
}
Printf("/n Enter the matrix B elements../n");
For(i=0;i<r2;i++)
{
For(j=0;j<c2;j++)
Scanf("%d", &b[i][j]);
}
For(i=0;i<r1;i++)
{
Forj-0;j<c1;j==)
{
C[i][j[]=0;
For(k=0;k<c1;k++)
C[i][j]=c[i][j]+a[i][k]*b[i][j];
}
}
Printf("/n The resultant matrix is .../n")
For(i=0;i<r1;i++)
{
For(j=0;j<c1;j++)
Printf("%d,/t", c[i][j]);
Printf("/n");
}
Getch();
}
```

Output:

Enter the size of the matrix A.....33

Enter the size of the matrix 22

Multipilaction is not possible

Enter the size of the matrix A.....3

Enter the size of the matrix B.....3

Enter the matrix A elements

2 2 2 2 2 2 2 2 2

Enter the matrix B elements

3 3 3 3 3 3 3 3 3

The resultant matrix is....

18 18 18

18 18 18

18 18 18

Two dimentional array:

Matrix multiplication:

```
#include<conio.h>
#include<stdio,h>
Voidmail()
{
Inta[3][3],b[3][3],c[3][3],
Int i,j,k;
Clrscr();
Printf("Enter the first matrix");
For(i=0;i<3;i++)
{
For(j=0;j<3;j++)
{
Scanf("%d", &a[i][j]);
}
}
Printf("Enter the Second matrix");
```

```
For(i=0;i<3;i++)
{
For(j=0;j<3;j++)
{
Scanf("%d", &b[i][j]);
}
}
For(i=0;i<3;i++)
{
For(j=0;j<3;j++)

{
C[i][j[]=0;
For(k=0;k<3;k++)
C[i][j]=c[i][j]+a[i][k]*b[i][j];
}
}
}
Printf("Matrix multiplication");
For(i=0;i<3;i++)
{
For(j=0;j<3;j++)
{
Printf("/t%d", c[i][j]);
}
Printf("/n");
}
Getch();
}
```

Output:

Enter the first matrix

1 2 3

4 5 6

7 8 9

Enter the second matrix

7 8 9

4 5 6

1 2 3

Matrix multiplication

18 24 33

30 69 90

90 114 147

Array of characters(string)

A string s a collection of characters

Example:

```
#include<conio.h>
#include<stdio,h>
Voidmail()
{
Char name [10] =”ASHWIN”;
Int i=0;
While(i<=5)
{
Printf (“%c”, name [i]);
I++;
}
Getch();
}
```

Multi dimensional Array:

Syntax: data type array name [size1] [size 2].....[size n];

Example:

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

float b[4][4][4][4];
```

1.14 String standard functions;

1. **Sterlin()** → used to find the length of the string
2. **Strcpy()** → used to copy one string to another
3. **Strcap()** → used to combine two strings
4. **Strcmp()** → used to character of two string.
5. **Strlwr()** → used to convert string into lower case
6. **Strupr()** → used to convert string into upper case
7. **Strdup()** → used to duplicate a string
8. **Strrev()** → used to reverse a string
9. **Strcpy()** → used to copy first n characters of one string to another
10. **Strcmp()** → used to compare first n characters of two strings
11. **Strcmpi()** → used to compare two strings without regarding the case
12. **Strnicmp()** → used to first n characters of two strings without regarding the case
13. **Stricmp()** → used to compare two strings
14. **Strchr()** → determines first occurrence of a given character to a string
15. **Strrchr()** → determines last of a given character to a string
16. **Strstr()** → determines first occurrence of a given string in another string
17. **Strncat()** → appends source string to destination string upto specifies length
18. **Strnset()** →set specified number of characters of a string within a given argument
19. **Strspn()** →finds up to what length two strings are identical
20. **Strpbrk()** → searches the first occurrence of the character in a given string and then it displays the strings starting from that character

1. The strlen function:

This function is used to count and return the number of characters present in the string.

Syntax:

```
Var=stolen(string);
```

Example:

Var→ is the integer variable, which accepts the the length of a string

String→is the constant or string variable in which the length is going to be found the counting ends with the first null(10) characters.

Example;

```
#include<conio.h>

#include<stdio,h>
```

```
Voidmail()

{

Char name[]=”PRIYA”;
Int len1, len2,;
Len 1 =strlen(name);
Len 2 = strlen(“DARSHINI”);
Printf(“string length of %s is “d”,name, len1”);
Printf(“string length of %s is % d”,”darshini”, len2);
Getch();
}
```

String length of PRIYA is 5
String length of DARSHINI is 9

2. The stocpy() function:

This function is used to copy the contents of one string to another & it almost works like string assignment operator

Syntax:

Strcpy(string1, string2);

Description:

String1→ is the destination string

String 2 → is the source string. (i.e) the contents of string 2 assigned to the contents of the string 1 where string 2 may be character array variable (or) string constant.

Example program:

```
#include<stdio,h>

#inclue<conio.h>

Void main()

{

Char source:”ANBU”

Char target(#[10];

Strcpy(target;source);

Print(“source string is %s”, source);

Printf(“target string is %s”,target);

Getch();

}
```

Output:

Source string is “ANBU”

Target string is “ANBU”

3. **The strcat() function:**

This function is used to concatenate or combine two strings to gather & forms a new concatenated string

Syntax: strcat(string 1, string 2);

Description:

String 1 and atring 2 are character type arrays or string constants

When the above strcat() function is executed string 2 combined with string 1 2 it removes the null character () of the string and places string 2 from there

Example program:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Char source=”VINU”
Char target=”DEEPTHI”
Strcat(source,target);
Printf(“source string is %s”, source);
Printf (“target string is %s”, target);
Getch();
}
```

Output:

Source string is VINUDEEPTHI

Target string is DEEPTHI

4. **The strmp() function:**

This is the function which compares two things to find out whether they same or different. If the strings are identical stamp() returns the value zero. If they are not equal it retunes the numeric difference between the first matching characters

Syntax: strcat(string 1, string 2);

Description:

String 1 and airing 2 are character type arrays or string constants

Example program:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
Char source=”V#KALAI”
Char target=”MALAI”
int  i,j
i=strmp(name:”KALAI”0;
j=strcmp(name 1, name);
printf(“/n%d%d”,i.j”);
```

1.15 Array Sorting Methods

The most common applications in computer science is sorting and searching.

The sorting is the process of arranging a set of similar information into an increasing (or) decreasing order.

If the data were not order, we would spend hours trying to find a single piece of information.

The process used to find such a difficult thing is called a search.

There are five commonly used sorting methods

1.Bubble sort(Exchange sort)

2.Selection sort

3.Insertion sort

4.Heap sort

5.Quick sort

Bubble sort

The basic idea in bubble sort is to pass through the array sequentially several times. Each pass consists of comparing each element in the array with its, successor (X[i] with (X[i+1]) and interchanging the two element if they are not in proper order.

Consider the following array

25	**57**	**48**	**37**	**12**	**92**	**86**	**33**
X [0]	with	X[1]	(i.e.)	25	with	57	no exchange
X [1]	with	X[2]	(i.e.)	57	with	48	exchange
X [2]	with	X[3]	(i.e.)	57	with	37	exchange
X [3]	with	X[4]	(i.e.)	57	with	12	exchange
X [4]	with	X[5]	(i.e.)	57	with	92	no exchange
X [5]	with	X[6]	(i.e.)	92	with	86	exchange
X [6]	with	X[7]	(i.e.)	92	with	33	exchange

After the first, the array is in the order

25	**48**	**37**	**12**	**57**	**86**	**33**	**92**
Initial	25	57	48	37	12	92	86	33
pass1	25	48	37	12	57	86	33	92
pass2	25	37	12	48	57	33	86	92
pass3	25	12	37	48	33	57	86	92
pass4	12	25	37	33	48	57	86	92
pass5	12	25	37	37	48	57	86	92
pass6	12	25	33	37	48	57	86	92
pass7	12	25	33	37	48	57	86	92

```
/* Program sorts the array in the ascending order using bubble sort method */
#include<stdio.h>
#include <conio.h>
void main ()
{
int I,j, temp;
int x[8]={25,57,48,37,12,92,86,33};
for (i=0; i<8; i++)
```

```
{
for (j=0;j<8;j++)
{
if (x[j]>x[j++])
{
temp=x[j];
x[j]=x[j+1];
x[j+1]=temp
}
printf (“The  sorted array is…”);
for (i=0;i<8;j++)
{
printf (“%d”, x[i]);
}
getch ()
 }
```

Output

The sorted carry is 12 25 33 37 48 57 86 92

Selection sort

A selection sort selects the elements with the lowest value and exchanges it with the first element.Then, from the remaining n-1 elements, the elements with the smallest key is found and exchange with the second element and so forth.

example:-	**58**	**42**	**12**	**38**	**15**
pass1	12	42	58	38	15
pass2	12	15	58	38	42
pass3	12	15	38	58	42
pass4	12	15	38	42	58

```
/*program sorts the array in the ascending order using selection sort*/
# include<stdio.h>
#include<conio.h>
```

```
void main ()
{
int j,j,k, data, count, exchange;
int [x[5]={58,42,12,38,15};
for (i=0; i<count-1; i++)
{
exchange=0;
k=i;
data=x[i];
for(j=i+1; j<5; j++)
{
k=j;
data=x[j];
exchange=1;
}
}
if (exchange)
{
x[k]=x[i];
x[i]=data;
}
printf ("The sorted array is….")
for (i=0;i<5;i++)
{
printf ("\t %d", x[i]);
}getch
```

Insertion sort

Insertion sort initially sorts first two elements of the array. Next the algorithm sorts the third element into its sorted position in relation to the first two elements. Then it inserts the fourth element into the list of three elements.

example	**58**	**42**	**12**	**38**	**15**
pass1	42	58	12	38	15
pass2	12	42	58	38	15
pass3	12	38	42	58	15
pass4	12	15	38	42	58

```
/*program sorts array in ascending order using insertion sort method*/
#include<stdio.h>
#include<conio.h>]void main ()
{
int I,j;
int x[5]={58,42,12,38,15};
int data;
for (i=1; i<5; i++)
{
data=x[i];
for (j=i-1; (j>=0&& (data<x[j]);]..)
{
x[j+1]=x[j];
x[j+1]=data;
}
printf (“The sorted array is….”);
for (i=0; i<5; i++)
{
printf (“\t %d”, x[i]);
}
getch ();
}
```

Output

The sorted array is….

12 15 38 42 58

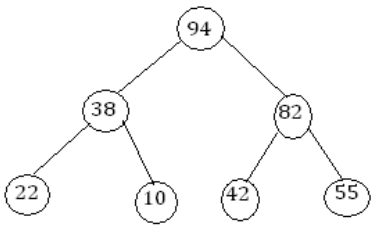
Heap sort

A heap is a binary tree with a specific relation between its each father and child. There are two types of heaps.

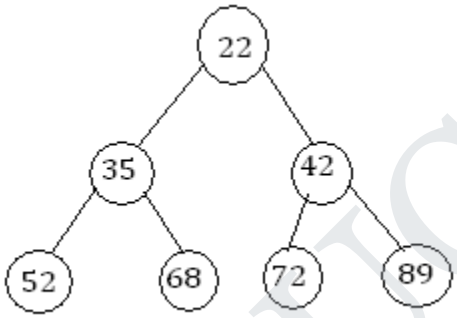
1. Descending heap (also called max heap)
2. Ascending heap (also called min heap)

Descending heap is almost complete binary tree of nodes such that the contents of each node is less than or equal to the contents of its father. In such tree, the root contains the biggest element of the heap, and any path from the root to leaf is an descending order list.

example



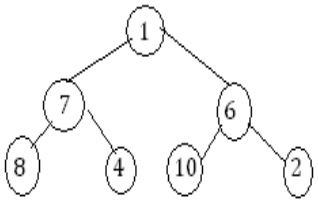
An ascending heap is an almost complete binary tree of n nodes such that the contents of each node is greater than or equal to the contents of its father. In an ascending heap, the root contains the smallest element of the heap, and any path from the root is a leaf is an ascending order list.

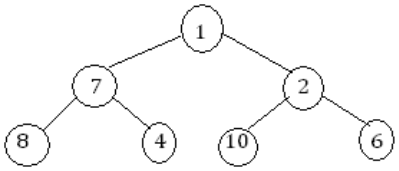


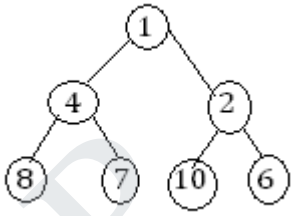
example

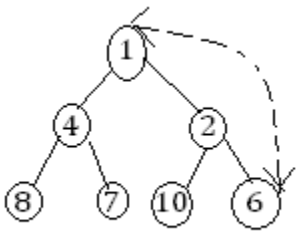
1,7,6,8,4,10,2

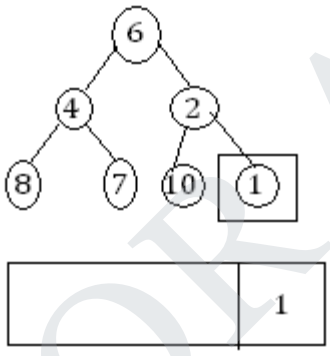
1	**7**	**6**	**8**	**4**	**10**	**2**

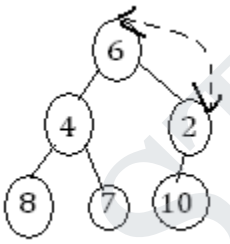


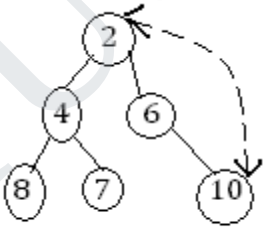


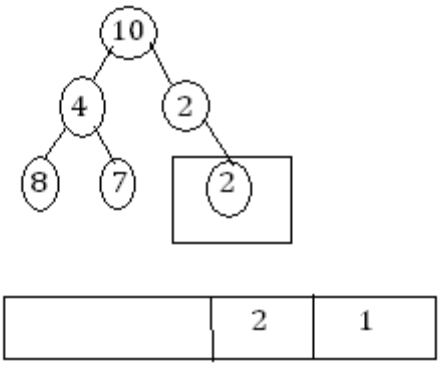


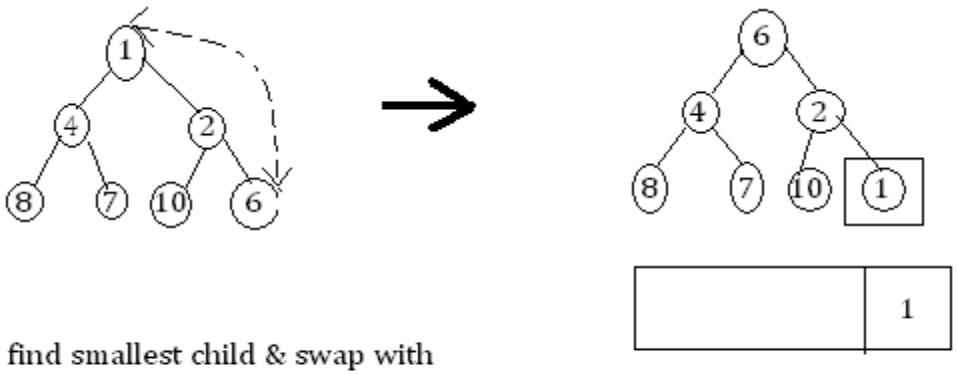




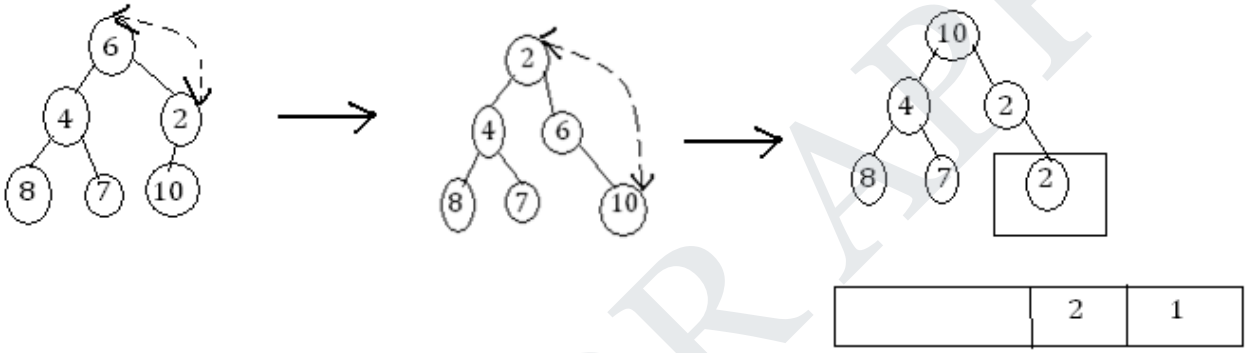


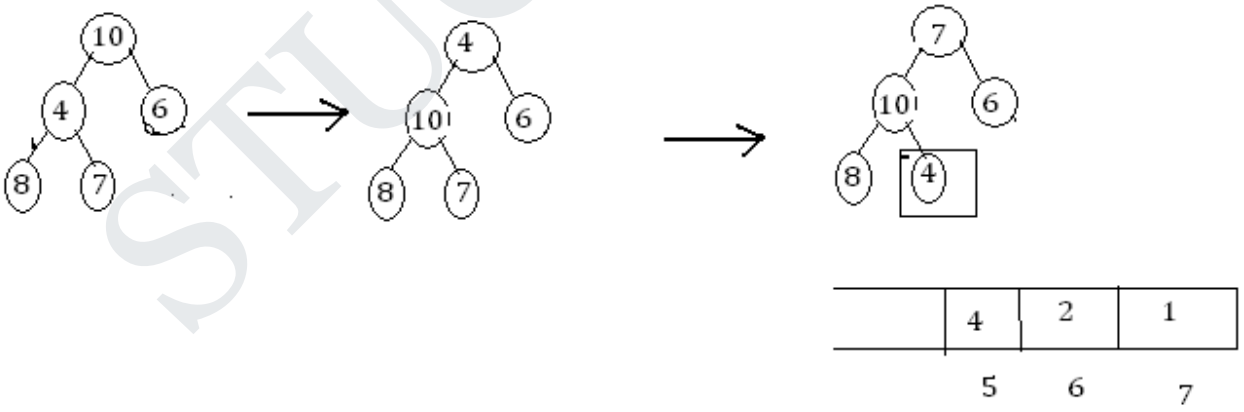


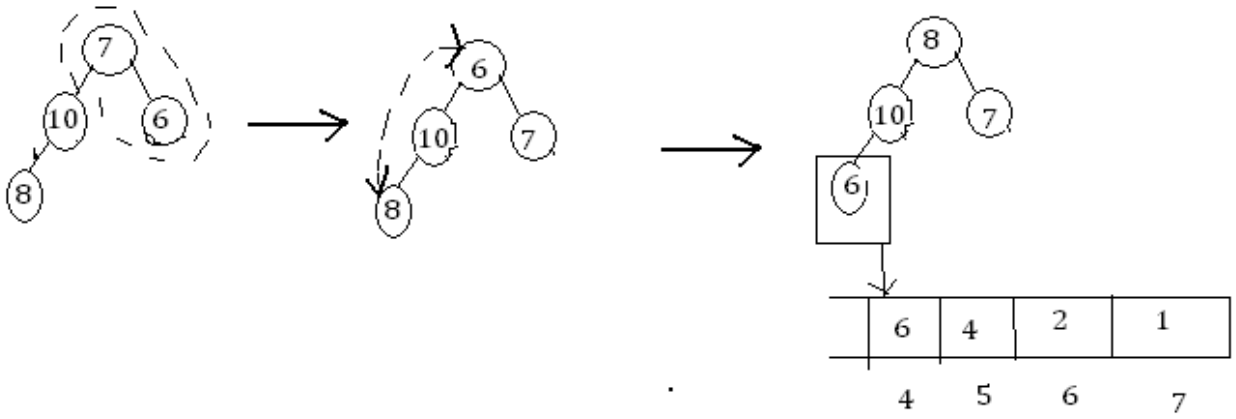


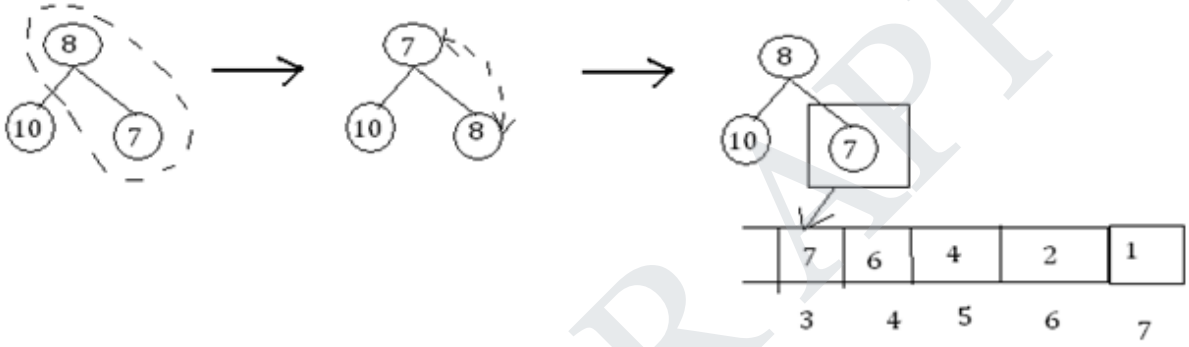


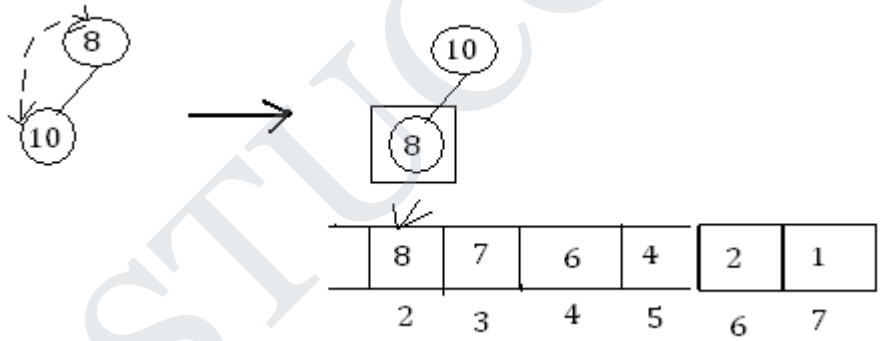
find smallest child & swap with the root

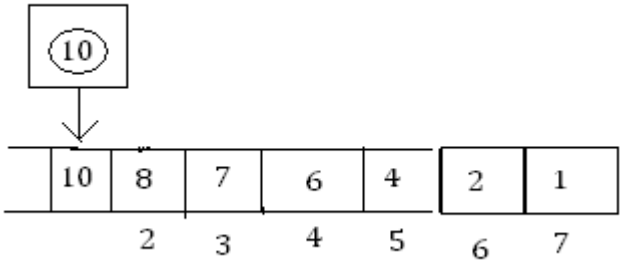












Finally array sorted in descending order using heapsort

example program

```
/*This program sorts array in the ascending order using heap sort menthod*/
#include<stdio.h>
#include<conic.h>
#define SIZE 100
int A[SIZE];
void heapsort (int [], int);
void main ()]
{
int no, I;
int 1,r,x;
clrscr ();
printf (“\ Enter number of elements”);
scanf (“ %d”, & no);
for (i=0; i<no;i++)
{
printf (“Enter the element %d; “,i);
scanf (“%d”,  A[i]);
}
printf (“Creating and sorting of heap:”);
heapsort (A,no);
printf (“The sorted attay is :”);
for (i=0; i<no; i++)
printf (“%d”, A[i]);
getch ();
return 0;
}
void heapsort (int arr [], int N)
```

```
{
int n=N, i=n>>L,j,w,t,p;
for (;;)
{
if (i>0)
t=arr[--i]
else
{
if (--n<=0) return ;
t=arr[n];
arr[n]=arr[0];
}
j=i;
w=(i<< 1)+1;
while (w<n)
{
if (w+1<n && arr[w+1] .> arr[W])
++w;
if (arr[[w]>t)
{
arr[j]=arr[[w];
j=w;
w=cj<<1)+1;
}
else
break;
}
arr[j]=t;
for(p=0; p<N; p++)
```

```
{
printf ("% d", arr[p]);
}
```

output

```
Enter the number of elements
Enter the element 0:  1
Enter the element 1:  7
Enter the element 2:  6
Enter the element 3:  8
Enter the element 4:  4
Enter the element 5:  10
Enter the element 6:  2
The sorted array is  1  2  4  6  7  8  10
```

Quick sort

Quick sort is a well-known sorting algorithm developed by C.A.R. Hoare. The algorithm sorts the list of employing a divide and conquer stratergy to divide a list into two sub-lists. This algorithm is also called partition exchange sort algorithm.

The quick sort works by partitioning the array A[1], A[2],…A[n] by picking some keyvalue in the array as a provide element.

Pivot element is used to rearrange the elements in the array pivot, can be the first element of the array and rest of the elements are moved so that the elements an left side of the pivot are lesed than the pivot, where as those on the right side are greater than the pivot.

example

Consider an unsorted array: 40 20 70 14 60 61 97 30

pivot=40

i=20, j=30

conditions

The value of I is incremented till a[i] <_pivot & the value of j is decremented till a [j]>pivot, this process is repeated until<,

if a [i]>pivot & a[j] <pivot & if i<j then swap a[i] &a[j]

if i>j then swap a[j] & a[pivot]

passes of quick sort

Pivot 40	20/i	70	14	60	61	97	30/j
Pivot 40	20	70/I	14	60	61	97	30/j
Pivot 40	20	30/I	14	60	61	97	70/j
Pivot 40	20	30	14/I	60	61	97/j	70
Pivot 40	20	30	14	60/I	61	97/j	70
Pivot 40	20	30	14	60/I	61/j	97/g	70
Pivot 40	20	30	14	60/I,j	61	97	70
Pivot 40	20	30	14/j	60/I	61	97	70
{14	20	30}	**40**	{60	61	97	70}
14	20	30	40	60	61	70	97

The sorted array is

14 20 30 40 60 61 70 97

program

```
# include <stdio.h>
#include<conio.h>
void quick sort (int [], int, int);
void main ()
{
int A[20];
int I,n;
printf (“Enter the number of elements”);
scanf (“%d”, &n);
for (i=0; i<n;i++
{
printf (“Enter element %d”, i);
```

```
Scanf (" %d, & a [i]);
}
quick sort (A, 0,n-1);
printf ("The sorted array is ;");
for (i=0i<n; i++)
printf ("%d", a[i]);
getch ();
}
void quick sort (int A[], int left, int right)
{
int I,j, pivot, temp;
if (left <right)
{
pivot=left;
i=left+1;
j=right;
while(i<j)
{
while (A[PIVOT] >=A[i]
i=i+1;
while (A[PIVOT] >=A[i]
j=j-1;
if (i<j)
{
temp=A[i];
A[i] =A[j];
A[j]= temp;
}
temp=A[PIVOT];
```

output

```
Enter the number of elements:8
Enter the element 0: 40
Enter the element 1: 20
Enter the element 2: 70
Enter the element 3: 14
Enter the element 4: 60
Enter the element 5: 61
Enter the element 6: 97
Enter the element 7: 30
sorted array is :  14  20  30  40  60  61  70  97
```

```
A[PIVOT]=A[j];

A[j]=temp;

quicksort (A, left, j-1);

quicksort (A,j+1, right);
```

1.16 Array Searching Methods

Searching is the process used to find the location of a target element among a list of elements. There is one method of finding target element is as unsorted array & another for a sorted array.

Finding a target element in a unsorted array requires **sequential search** starting at the first element & stopping either when a match is found or at the end of the array. This method is also know as **linear search**.

If the data has been sorted, we can use a **binary search**, which helps us to located the data more quickly.

Sequential search/Linear search

In sequential search, a target element is comp areal will each of the array element starting form first element in the array up to the last element in the array or up to the match found.]

example sequential sort

/*This program searches a target element & if found, displays its location in the array; otherwise, it displays message that target element is not within the array*/

```
# include<stdio.h>

#include<conio.h>

void main ()

{

int I, target, flag=100;

int x[5]={10,20,50,40,60};

printf (“Enter any integer as a target”);

scanf (“ %d”, & target);

for (i=0; i<5; i++)

{

if (target= =x[i])

{

printf (“Location target is %d”, i);
```

```
}
if (flag= =100)
printf("Target element is not in an array");
getch ();
}
```

Output

```
Enter any integer number as a target
40
Location of target is-3
```

Binary search

If the data to be searched is sorted, we can use most superior method to find a match. It is binary search. It uses the divided and conquer approach. To implement this method, the middle element is tested.

If the middle element is larger than the target, the first half is tested. Otherwise, the second half is tested.

This process is repeated until a match is found (or) there are no more elements to test.

example

```
/* This program searches a target element within the sorted array using binary search*/
# include <stdio.h>
#include <conio.h>
void main ()
{
int x[9]= {10,20,30,40,50,60,70,80,90};
int target;
int low, high, mid;
printf ("Enter any integer number as a target;");
scanf ("%d", & target);
low =0;
high = 8;
while (low<= high)
{
mid = (low +high)/2;
if (target < x[mid])
```

```
high = mid-1;
else if (target> x[mid])
Low=mid+1;
else
{
printf (" Location of target is %d", mid);
break;
}
getch ()
}
```

Output

Enter any integer number as a target: 60

Location of target is 5.

1.16.MATRIX OPERATIONS

Matrix multiplication:

```
#include<conio.h>
#include<stdio,h>
Voidmail()
{
Inta[3][3],b[3][3],c[3][3],
Int i,j,k;
Clrscr();
Printf("Enter the first matrix");
For(i=0;i<3;i++)
{
For(j=0;j<3;j++)
{
Scanf("%d", &a[i][j]);
}
```

```
}
Printf("Enter the Second matrix");
For(i=0;i<3;i++)
{
For(j=0;j<3;j++)
{
Scanf("%d", &b[i][j]);
}
}
For(i=0;i<3;i++)
{
For(j=0;j<3;j++)

{
C[i][j[]=0;
For(k=0;k<3;k++)
C[i][j]=c[i][j]+a[i][k]*b[i][j];
}
}
}
Printf("Matrix multiplication");
For(i=0;i<3;i++)
{
For(j=0;j<3;j++)
{
Printf("/t%d", c[i][j]);
}
Printf("/n");
}
```

```
Getch();
}
```

Output:

```
Enter the first matrix
1  2  3
4  5  6
7  8  9
Enter the second matrix
7  8  9
4  5  6
1  2  3
```

Matrix multiplication

```
18  24  33
30  69  90
90  114 147
```

Array of characters(string)

A string s a collection of characters

Example:

```
#include<conio.h>
#include<stdio,h>
Voidmail()
{
Char name [10] =”ASHWIN”;
Int i=0;
While(i<=5)
{
Printf (“%c”, name [i]);
I++;
}
Getch();
}
```