UNIT 1

C PROGRAMMING BASICS

Introduction to C

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc

Structure of C Language program

1) Comment line

2) Preprocessor directive

3) Global variable declaration

4) main function( )

```
{

    Local variables; Statements;
}
User defined function
}
}
```

Comment line

It indicates the purpose of the program. It is represented as

/*……..*/

Comment line is used for increasing the readability of the program. It is enclosed within the decimeters. Comment line can be single or multiple line but should not be nested.

Preprocessor Directive:

#include<stdio.h> tells the compiler to include information about the standard input/output library.

Global Declaration:

This is the section where variable are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function .

main()

syntax

```
Main()
{
……..
……..
……..
}
```

Compiling and executing the Programs

A compiler is a software program that analyzes a program developed in a particular computer language and then translates it into a form that is suitable for execution on a particular computer system. Figure below shows the steps that are involved in entering, compiling, and executing a computer program developed in the C programming language and the typical Unix commands that would be entered from the command line.

Step 1: The program that is to be compiled is first typed into a *file* on the computer system.There are various conventions that are used for naming files, typically be any name provided the last two characters are **".c"** or file with extension .c. So, the file name **prog1.c** might be a valid filename for a C program. A text editor is usually used to enter the C program into a file. For example, vi is a popular text editor used on Unix systems. The program that is entered into the file is known as the ***source program*** because it represents the original form of the program expressed in the C language.

Step 2: After the source program has been entered into a file, then proceed to have it compiled. The compilation process is initiated by typing a special command on the system. When this command is entered, the name of the file that contains the source program must also be specified. For example, under Unix, the command to initiate program compilation is called **cc**. If we are using the popular GNU C compiler, the command we use is **gcc**.

Typing the line

gcc prog1.c or cc prog1.c

In the first step of the compilation process, the compiler examines each program statement contained in the source program and checks it to ensure that it conforms to the syntax and semantics of the language. If any mistakes are discovered by the compiler during this phase, they are reported to the user and the compilation process ends right there. The errors then have to be corrected in the source program (with the use of an editor), and the compilation process must be restarted. Typical

errors reported during this phase of compilation might be due to an expression that has unbalanced parentheses (**syntactic error**), or due to the use of a variable that is not "defined" (**semantic error**).

Step 3: When all the syntactic and semantic errors have been removed from the program, the compiler then proceeds to take each statement of the program and translate it into a "lower" form that is equivalent to assembly language program needed to perform the identical task.

Step 4: After the program has been translated the next step in the compilation process is to translate the assembly language statements into actual machine instructions. The assembler takes each assembly language statement and converts it into a binary format known as ***object code***, which is then written into another file on the system. This file has the same name as the source file under Unix, with the last letter an "**o**" (**for *object***) instead of a "**c**".

Step 5: After the program has been translated into object code, it is ready to be ***linked.*** This process is once again performed automatically whenever the cc or gcc command is issued under Unix. The purpose of the linking phase is to get the program into a final form for execution on the computer.

If the program uses other programs that were previously processed by the compiler, then during this phase the programs are linked together. Programs that are used from the system's program ***library*** are also searched and linked together with the object program during this phase.

The process of compiling and linking a program is often called ***building*.**

The final linked file, which is in an executable ***object*** code format, is stored in another file on the system, ready to be run or ***executed*.** Under Unix, this file is called **a.out** by default. Under Windows, the executable file usually has the same name as the source file, with the c extension replaced by an exe extension.

Step 6: To subsequently execute the program, the command **a.out** has the effect of *loading* the program called **a.out** into the computer's memory and initiating its execution.

When the program is executed, each of the statements of the program is sequentially executed in turn. If the program requests any data from the user, known as ***input***, the program temporarily suspends its execution so that the input can be entered. Or, the program might simply wait for an ***event***, such as a mouse being clicked, to occur. Results that are displayed by the program, known as ***output,*** appear in a window, sometimes called the ***console***. If the program does not produce the desired results, it is necessary to go back and reanalyze the program's logic. This is known as the ***debugging phase***, during which an attempt is made to remove all the known problems or ***bugs*** from the program. To do this, it will most likely be necessary to make changes to original source program.

Constants

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

Numeric constant

Character constant

String constant

Numeric constant: Numeric constant consists of digits. It required minimum size of 2 bytes and max 4 bytes. It may be positive or negative but by default sign is always positive. No comma or space is allowed within the numeric constant and it must have at least 1 digit. The allowable range for integer constants is -32768 to 32767. Truly speaking the range of an Integer constant depends upon the compiler. For a 16-bit compiler like Turbo C or Turbo C++ the range is –32768 to 32767.

For a 32-bit compiler the range would be even greater. Mean by a 16-bit or a 32- bit compiler, what range of an Integer constant has to do with the type of compiler.

It is categorized a **integer constant** and **real constant.** An integer constants are whole number which have no decimal point. Types of integer constants are:

- Decimal constant: 0-------9 (base 10)
- Octal constant: 0-------7 (base 8)
- Hexa decimal constant: 0----9, A------F (base 16)

In decimal constant first digit should not be zero unlike octal constant first digit must be zero(as 076, 0127) and in hexadecimal constant first two digit should be 0x/ 0X (such as 0x24, 0x87A). By default type of integer constant is integer but if the value of integer constant is exceeds range then value represented by integer type is taken to be unsigned integer or long integer. It can also be explicitly mention integer and unsigned integer type by suffix l/L and u/U.

Real constant is also called floating point constant. To construct real constant we must follow the rule of ,

-real constant must have at least one digit.

-It must have a decimal point.

-It could be either positive or negative.

-Default sign is positive.

-No commas or blanks are allowed within a real constant. Ex.: +325.34 426.0

-32.76

To express small/large real constant exponent(scientific) form is used where number is written in mantissa and exponent form separated by e/E. Exponent can be positive or negative integer but mantissa can be real/integer type, for example $3.6*10^5$=3.6e+5. By default type of floating point constant is double, it can also be explicitly defined it by suffix of f/F.

Character constant

Character constant represented as a single character enclosed within a single quote. These can be single digit, single special symbol or white spaces such as '9','c','$', ' ' etc. Every character constant has a unique integer like value in machine's character code as if machine using ASCII (American standard code for information interchange). Some numeric value associated with each upper and lower case alphabets and decimal integers are as:

A---------- Z ASCII value (65-90)

a----------z ASCII value (97-122)

0----------9ASCII value (48-59)

; ASCII value (59)

String constant

Set of characters are called string and when sequence of characters are enclosed within a double quote (it may be combination of all kind of symbols) is a string constant. String constant has zero, one or more than one character and at the end of the string null character(\0) is automatically placed by compiler. Some examples are ",sarathina" , "908", "3"," ", "A" etc. In C although same characters are enclosed within single and double quotes it represents different meaning such as "A" and 'A' are different because first one is string attached with null character at the end but second one is character constant with its corresponding ASCII value is 65.

Symbolic constant

Symbolic constant is a name that substitute for a sequence of characters and, characters may be numeric, character or string constant. These constant are generally defined at the beginning of the program as

#define name value , here name generally written in

upper case for example

#define MAX 10

#define CH 'b'

#define NAME "sony"

Variables

Variable is a data name which is used to store some data value or symbolic names for storing program computations and results. The value of the variable can be change during the execution. The rule for naming the variables is same as the naming identifier. Before used in the program it must be declared. Declaration of variables specify its name, data types and range of the value that variables can store depends upon its data types.

Syntax: int a; char c; float f;

Variable initialization

When we assign any initial value to variable during the declaration, is called initialization of variables. When variable is declared but contain undefined value then it is called garbage value. The variable is initialized with the assignment operator such as

Data type variable name=constant;

Example: int a=20;

or int a;

a=20;

statements.

Expressions

An expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical and relational for example:-

```
int z=  x+y       // arithmatic expression
a>b       //relational
a==b          // logical
 func(a, b)    // function call
```

Expressions consisting entirely of constant values are called *constant expressions*. So, the expression

121 + 17 - 110

is a constant expression because each of the terms of the expression is a constant value. But if i were declared to be an integer variable, the expression

180 + 2 – j

would not represent a constant expression.

DATA TYPES

Data types refer to an extensive system used for declaring variables or functions of different types before its use. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. The value of a variable can be changed any time.

C has the following 4 types of data types

1. **Basic built-in data types**: int, float, double, char
2. **Enumeration data type:** enum
3. **Derived data type**: pointer, array, structure, union
4. **Void data type**: void

Basic data type	Data type with type qualifier	Size (byte)	Range
char	char or signed char	1	-128 to 127
	Unsigned char	1	0 to 255
int	int or signed int	2	-32768 to 32767
	unsigned int	2	0 to 65535
	short int or signed short int	1	-128 to 127
	unsigned short int	1	0 to 255
	long int or signed long int	4	-2147483648 to 2147483647
	unsigned long int	4	0 to 4294967295
float	float	4	-3.4E-38 to 3.4E+38
double	double	8	1.7E-308 to 1.7E+308
	Long double	10	3.4E-4932 to 1.1E+4932

DECISION MAKING STATEMENTS

If statement

Statement execute set of command like when condition is true and its syntax is

```
If(condition)
Statement;
```

The statement is executed only when condition is true. If the if statement body is consists of several statement then better to use pair of curly braces. Here in case condition is false then compiler skip the line within the if block.

```
void main()
{
    int n;
     printf (" enter a number:");
      scanf("%d",&n);
          If (n>10)
         Printf(" number is grater");

}
```

Output:Enter
a number:12
Number is
greater

if…..else ... Statement

it is bidirectional conditional control statement that contains one condition & two possible action. Condition may be true or false, where non-zero value regarded as true & zero value regarded as false. If condition are satisfy true, then a single or block of statement executed otherwise another single or block of statement is executed.

Its syntax is:-

```
if (condition)
 {
 Statement1;
  Statement2;
 }

    else
      {
       Statement1;
       Statement2;
        }
```

Else statement cannot be used without if or no multiple else statement are allowed within one if statement. It means there must be a if statement with in an else statement.

Example:-

```
/* To check a number is eve or odd */
 void main()
{
  int n;
      printf (“enter a number:”);
     sacnf (“%d”, &n);
      If (n%2==0)
         printf (“even number”);
```

```
else
}
  printf("odd number");
```

Output: enter a number:121 odd number

Nesting of if …else

When there are another if else statement in if-block or else-block, then it is called nesting of if-else statement.

Syntax is :-

```
        if (condition)
         {
              If     (condition)
               Statement1;
    else
                statement2;

    }
                Statement3;
```

If….else LADDER

In this type of nesting there is an if else statement in every else part except the last part. If condition is false control pass to block where condition is again checked with its if statement.

Syntax is :-

```
    if    (condition)
      Statement1;
else   if   (condition)
       statement2;
else   if   (condition)
      statement3;
 else

          statement4;
```

This process continue until there is no if statement in the last block. if one of the condition is satisfy the condition other nested "else if" would not executed.

But it has disadvantage over if else statement that, in if else statement whenever the condition is true, other condition are not checked. While in this case, all condition are checked.

Loops in C

Loop:-it is a block of statement that performs set of instructions. In loops

Repeating particular portion of the program either a specified number of time or until a particular no of condition is being satisfied.

There are three types of loops in c

1.While loop

2.do while loop

3.for loop

Whileloop

Syntax:-

While()
{
Condition;

}

The test condition may be any expression .when we want to do something a fixed no of times but not known about the number of iteration, in a program then while loop is used.

Here first condition is checked if, it is true body of the loop is executed else, If condition is false control will be come out of loop.

Example:-

```
/* wap to print 5 times welcome to C" */
#include<stdio.h>
void main()
{
int p=1;
While(p<=5)
{
printf("Welcome to C\n");
P=p+1;
}
}
```

Output: Welcome to C

Welcome to C

Welcome to C

Welcome to C

Welcome to C

So as long as condition remains true statements within the body of while loop will get executed repeatedly.

do while loop

This (do while loop) statement is also used for looping. The body of this loop may contain single statement or block of statement. The syntax for writing this statement is:

Syntax:-

```
Do
{
Statement;
}
while(condition);
```

Example:-

```
#include<stdio.h>
void main()
{
int   X=4;
 do
{
 Printf("%d",X);
X=X+1;
```

```
}whie(X<=10);
  Printf(" ");
}
```

Output: 4 5 6 7 8 9 10

Here firstly statement inside body is executed then condition is checked. If the condition is true again body of loop is executed and this process continue until the condition becomes false. Unlike while loop semicolon is placed at the end of while.

There is minor difference between while and do while loop, while loop test the condition before executing any of the statement of loop. Whereas do while loop test condition after having executed the statement at least one within the loop.

If initial condition is false while loop would not executed it's statement on other hand do while loop executed it's statement at least once even If condition fails for first time. It means do while loop always executes at least once.

for loop

In a program, for loop is generally used when number of iteration are known in advance. The body of the loop can be single statement or multiple statements. Its syntax for writing is:

Syntax:-

```
for(exp1;exp2;exp3)
{
Statement;
}
```

Or

```
for(initialized counter; test counter; update counter)
{
Statement;
}
```

Here exp1 is an initialization expression, exp2 is test expression or condition and exp3 is an update expression. Expression 1 is executed only once when loop started and used to initialize the loop variables. Condition expression generally uses relational and logical operators. And updation part executed only when after body of the loop is executed.

Example:-

```
void main()
{
int i;
 for(i=1;i<10;i++)
{
Printf(" %d ", i);
}

}
```

Output:-1 2 3 4 5 6 7 8 9

Nesting of loop

When a loop written inside the body of another loop then, it is known as nesting of loop. Any type of loop can be nested in any type such as while, do while, for. For example nesting of for loop can be represented as :

```
void main()
{
```

```
int  i,j;  for(i=0;i<2;i++)
for(j=0;j<5;          j++)
printf("%d %d", i, j);
 }
```

Output: i=0

j=0 1 2 3 4

i=1

j=0 1 2 3 4

Break statement(break)

Sometimes it becomes necessary to come out of the loop even before loop condition becomes false then break statement is used. Break statement is used inside loop and switch statements. It cause immediate exit from that loop in which it appears and it is generally written with condition. It is written with the keyword as **break.** When break statement is encountered loop is terminated and control is transferred to the statement, immediately after loop or situation where we want to jump out of the loop instantly without waiting to get back to conditional state.

When break is encountered inside any loop, control automatically passes to the first statement after the loop. This break statement is usually associated with **if** statement.

Example :

```
void main()
{
int          j=0;
for(;j<6;j++)
if(j==4) break;
}
```

Output:

0 1 2 3

Continue statement (key word continue)

Continue statement is used for continuing next iteration of loop after skipping some statement of loop. When it encountered control automatically passes through the beginning of the loop. It is usually associated with the if statement. It is useful when we want to continue the program without executing any part of the program.

The difference between break and continue is, when the break encountered loop is terminated and it transfer to the next statement and when continue is encounter control come back to the beginning position.

In while and do while loop after continue statement control transfer to the test condition and then loop continue where as in, for loop after continue control transferred to the updating expression and condition is tested.

Example:-

```
void main()
{
int n;
for(n=2; n<=9; n++)
{
if(n==4) continue; printf("%d", n);
}

}
Printf("out of loop");
}
```

Output: 2 3 5 6 7 8 9 out of loop

ARRAY

Array is the collection of similar data types or collection of similar entity stored in contiguous memory location. Array of character is a string. Each data item of an array is called an element. And each element is unique and located in separated memory location. Each of elements of an array share a variable but each element having different index no. known as subscript.

An array can be a single dimensional or multi-dimensional and number of subscripts determines its dimension. And number of subscript is always starts with zero. One dimensional array is known as vector and two dimensional arrays are known as matrix.

ADVANTAGES: array variable can store more than one value at a time where other variable can store one value at a time.

Example:

```
int  arr[100];
```

int mark[100];

DECLARATION OF AN ARRAY :

Its syntax is :

Data type array name [size];

int arr[100];

int mark[100];

int a[5]={10,20,30,100,5}

The declaration of an array tells the compiler that, the data type, name of the array, size of the array and for each element it occupies memory space. Like for int data type, it occupies 2 bytes for each element and for float it occupies 4 byte for each element etc. The size of the array operates the number of elements that can be stored in an array and it may be a int constant or constant int expression.

We can represent individual array as :

int ar[5];

ar[0], ar[1], ar[2], ar[3], ar[4];

Symbolic constant can also be used to specify the size of the array as: #define SIZE 10;

INITIALIZATION OF AN ARRAY:

After declaration element of local array has garbage value. If it is global or static array then it will be automatically initialize with zero. An explicitly it can be initialize that

Data type array name [size] = {value1, value2, value3…}

Example:

int ar[5]={20,60,90, 100,120}

Array subscript always start from zero which is known as lower bound and upper value is known as upper bound and the last subscript value is one less than the size of array. Subscript can be an expression i.e. integer value. It can be any integer, integer constant, integer variable, integer expression or return value from functional call that yield integer value.

So if i & j are not variable then the valid subscript are ar

[i*7],ar[i*i],ar[i++],ar[3];

The array elements are standing in continuous memory locations and the amount of storage required for hold the element depend in its size & type.

Total size in byte for 1D array is:

Total bytes=size of (data type) * size of array.

Example : if an array declared is: int

[20];

Total byte= 2 * 20 =40 byte.

ACCESSING OF ARRAY ELEMENT:

/*Write a program to input values into an array and display them*/

```
#include<stdio.h>
int main()
{
int arr[5],i;
 for(i=0;i<5;i++)
{
printf("enter a value for arr[%d] \n",i);
scanf("%d",&arr[i]);
}
printf("the array elements are: \n"); for
(i=0;i<5;i++)
{
printf("%d\t",arr[i]);
}
return 0;
}
```

OUTPUT:

Enter a value for arr[0] = 12

Enter a value for arr[1] =45

Enter a value for arr[2] =59

Enter a value for arr[3] =98

Enter a value for arr[4] =21

The array elements are 12 45 59 98 21

single dimensional array

while initializing a single dimensional array, it is optional to specify the size of array. If the size is omitted during initialization then the compiler assumes the size of array equal to the number of initializers.

For example:-

```
int marks[]={99,78,50,45,67,89};
```

If during the initialization of the number the initializers is less then size of array, then all the remaining elements of array are assigned value zero .

For example:-

```
int marks[5]={99,78};
```

Here the size of the array is 5 while there are only two initializers so After this initialization, the value of the rest elements are automatically occupied by zeros such as

Marks[0]=99 , Marks[1]=78 , Marks[2]=0, Marks[3]=0, Marks[4]=0 Again if we initialize an array like

```
int array[100]={0};
```

Then the all the element of the array will be initialized to zero. If the number of initializers is more than the size given in brackets then the compiler will show an error.

For example:-

```
int arr[5]={1,2,3,4,5,6,7,8};//error
```

we cannot copy all the elements of an array to another array by simply assigning it to the other array like, by initializing or declaring as

```
int a[5] ={1,2,3,4,5};
int b[5];
b=a;//not valid
```

(**note**:-here we will have to copy all the elements of array one by one, using for loop.)

Single dimensional arrays and functions

```
/*program to pass array elements to a function*/
#include<stdio.h>
void main()
{
int arr[10],i;
printf("enter the array elements\n"); for(i=0;i<10;i++)
{
scanf("%d",&arr[i]);
check(arr[i]);
}
}
void check(int num)
{
if(num%2=0)
{
printf("%d is even \n",num);
}
else
{
printf("%d is odd \n",num);
}
}
```

Two dimensional arrays

Two dimensional array is known as matrix. The array declaration in both the array i.e.in single dimensional array single subscript is used and in two dimensional array two subscripts are is used.

Its syntax is

Data-type array name[row][column];

Or we can say 2-d array is a collection of 1-D array placed one below the other.

Total no. of elements in 2-D array is calculated as **row*column**

Example:-

int a[2][3];

Total no of elements=row*column is 2*3 =6

It means the matrix consist of 2 rows and 3 columns For example:-

20 2 7

8 3 15

String

Array of character is called a string. It is always terminated by the NULL character. String is a one dimensional array of character.

We can initialize the string as char name[]={'j','o','h','n','\o'};

String library function

There are several string library functions used to manipulate string and the prototypes for these functions are in header file "string.h". Several string functions are

strlen()

This function return the length of the string. i.e. the number of characters in the string excluding the terminating NULL character.

It accepts a single argument which is pointer to the first character of the string.

For example-

strlen("suresh");

It return the value 6.

In array version to calculate legnth:-

```
int str(char str[])
{
Int i=0;
while(str[i]!='\o')
{

i++;
```

```
}

return i;
}
```

Example:-

```
#include<stdio.h>
#include<string.h>   void
main()
{
char str[50]; print(”Enter a
string:”);
```

```
gets(str);
printf("Length of the string is %d\n",strlen(str));
}
```

Output:

Enter a string: C in Depth

Length of the string is 8

strcmp()

This function is used to compare two strings. If the two string match, strcmp() return a value 0 otherwise it return a non-zero value. It compare the strings character by character and the comparison stops when the end of the string is reached or the corresponding characters in the two string are not same.

```
strcmp(s1,s2)
return a value:
        <0 when s1<s2
        =0 when s1=s2
        >0 when s1>s2
```

The exact value returned in case of dissimilar strings is not defined. We only know that if s1<s2 then a negative value will be returned and if s1>s2 then a positive value will be returned.

For example:

```
/*String comparison……*/ #include<stdio.h>
#include<string.h>
void main()
{

    {
    }
    else
    {

    }
    }
```

```
char str1[10],str2[10];
printf("Enter two strings:");
gets(str1);
gets(str2);
if(strcmp(str1,str2)==0)
printf("String are same\n");

printf("String are not same\n");
```

strcpy()

This function is used to copying one string to another string. The function strcpy(str1,str2) copies str2 to str1 including the NULL character. Here str2 is the source string and str1 is the destination string.

The old content of the destination string str1 are lost. The function returns a pointer to destination string str1.

Example:-

```
#include<stdio.h>
#include<string.h>
void main()
{
char str1[10],str2[10];
printf("Enter a string:");
scanf("%s",str2);
strcpy(str1,str2);
printf("First string:%s\t\tSecond string:%s\n",str1,str2);
strcpy(str,"Delhi");
strcpy(str2,"Bangalore");
printf("First string :%s\t\tSecond string:%s",str1,str2);
}
```

strcat()

This function is used to append a copy of a string at the end of the other string. If the first string is ""Purva" and second string is "Belmont" then after using this function the string becomes "PusvaBelmont". The NULL character from str1 is moved and str2 is added at the end of str1. The 2nd string str2 remains unaffected. A pointer to the first string str1 is returned by the function.

Example:-

#include<stdio.h> #include<string.h> void main()

```
{
char str1[20],str[20];
printf(“Enter two strings:”);
gets(str1);
gets(str2);
 strcat(str1,str2);
printf(“First string:%s\t second string:%s\n”,str1,str2);
strcat(str1,”-one”);
printf(“Now first string is %s\n”,str1);
}
```

Output

Enter two strings: data Base

First string: database second string: database

`Now first string is: database-one

SELECTION SORTING

```
#include <stdio.h>

int main()
{
  int array[100], n, c, d, position, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for ( c = 0 ; c < n ; c++ )
    scanf("%d", &array[c]);

  for ( c = 0 ; c < ( n - 1 ) ; c++ )
  {
    position = c;

    for ( d = c + 1 ; d < n ; d++ )
    {
      if ( array[position] > array[d] )
        position = d;
    }
    if ( position != c )
    {
      swap = array[c];
```

```
        array[c] = array[position];
        array[position] = swap;
      }
    }

    printf("Sorted list in ascending order:\n");

    for ( c = 0 ; c < n ; c++ )
      printf("%d\n", array[c]);

    return 0;
}
```

LINEAR SEARCH

```
#include <stdio.h>

int main()
{
  int array[100], search, c, n;

  printf("Enter the number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
```

```
   scanf("%d", &array[c]);

  printf("Enter a number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
  {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);

  return 0;
}
```

BINARY SEARCH

```
#include <stdio.h>

int main()
{
   int c, first, last, middle, n, search, array[100];

   printf("Enter number of elements\n");
   scanf("%d",&n);
```

```
  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d",&array[c]);

  printf("Enter value to find\n");
  scanf("%d", &search);

  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == search) {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d isn't present in the list.\n", search);

  return 0;
}
```

MATRIX ADDITION

```
#include <stdio.h>

int main()
{
  int m, n, c, d, first[10][10], second[10][10], sum[10][10];

  printf("Enter the number of rows and columns of matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");

  for (c = 0; c < m; c++)
    for (d = 0; d < n; d++)
      scanf("%d", &first[c][d]);

  printf("Enter the elements of second matrix\n");

  for (c = 0; c < m; c++)
    for (d = 0 ; d < n; d++)
      scanf("%d", &second[c][d]);

  printf("Sum of entered matrices:-\n");

  for (c = 0; c < m; c++) {
    for (d = 0 ; d < n; d++) {
      sum[c][d] = first[c][d] + second[c][d];
      printf("%d\t", sum[c][d]);
    }
    printf("\n");
```

```
  }

  return 0;
}
```

Operator

This is a symbol use to perform some operation on variables, operands or with the constant. Some operator required 2 operand to perform operation or Some required single operation.

Several operators are there those are, arithmetic operator, assignment, increment , decrement, logical, conditional, comma, size of , bitwise and others.

1. Arithmatic operator

This operator used for numeric calculation. These are of either Unary arithmetic operator, Binary arithmetic operator. Where Unary arithmetic operator required only one operand such as +,-, ++, --,!, tiled. And these operators are addition, subtraction, multiplication, division. Binary arithmetic operator on other hand required two operand and its operators are +(addition), -(subtraction) *(multiplication), /(division), %(modulus). But modulus cannot applied with floating point operand as well as there are no exponent operator in c.Unary (+) and Unary (-) is different from addition and subtraction.

When both the operand are integer then it is called integer arithmetic and the result is always integer. When both the operand are floating point then it is called floating arithmetic and when operand is of integer and floating point then it is called mix type or mixed mode arithmetic . And the result is in float type.

2. Assignment Operator

A value can be stored in a variable with the use of assignment operator. The assignment operator(=) is used in assignment statement and assignment expression. Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression. When variable on the left hand side is occur on the right hand side then we can avoid by writing the

compound statement. For example,

```
int x= y;
int  Sum=x+y+z;
```

3.Increment and Decrement

The Unary operator ++, --, is used as increment and decrement which acts upon single operand. Increment operator increases the value of variable by one .Similarly decrement operator decrease the value of the variable by one. And these operator can only used with the variable, but cann't use with expression and constant as ++6 or ++(x+y+z).

It again categories into prefix post fix . In the prefix the value of the variable is incremented 1^{st}, then the new value is used, where as in postfix the operator is written after the operand(such as m++,m--).

EXAMPLE

```
let y=12;
z= ++y;
y= y+1;
 z= y;
```

Similarly in the postfix increment and decrement operator is used in the operation . And then increment and decrement is perform.

EXAMPLE

```
let x= 5; y= x++; y=x;
x= x+1;
```

4.Relational Operator

It is use to compared value of two expressions depending on their relation. Expression that contain relational operator is called relational expression.

Here the value is assign according to true or false value. a.

```
(a>=b) || (b>20)
```

b.(b>a) && (e>b)

c. 0(b!=7)

5. Conditional Operator

It sometimes called as ternary operator. Since it required three expressions as operand and it is represented as (? , :).

SYNTAX

exp1 ? exp2 :exp3

Here exp1 is first evaluated. It is true then value return will be exp2 . If false then exp3.

EXAMPLE

```
void main()
{
int a=10, b=2
int s= (a>b) ? a:b;
 printf("value is:%d");
}
```

Output:

Value is:10

6. Comma Operator

Comma operator is use to permit different expression to be appear in a situation where only one expression would be used. All the expression are separator by comma and are evaluated from left to right.

EXAMPLE

```
int i, j, k, l;
for(i=1,j=2;i<=5;j<=10;i++;j++)
```

7. Sizeof Operator

Size of operator is a Unary operator, which gives size of operand in terms of byte that occupied in the memory. An operand may be variable, constant or data type qualifier.

Generally it is used make portable program(program that can be run on different

machine) . It determines the length of entities, arrays and structures when their size are not known to the programmer. It is also use to allocate size of memory dynamically during execution of the program.

EXAMPLE

```
main( )
{
int sum; float f;
printf( "%d%d" ,size of(f), size of (sum) );
printf("%d%d", size of(235 L), size of(A));
}
```

8. Bitwise Operator

Bitwise operator permit programmer to access and manipulate of data at bit level.

Various bitwise operator enlisted are

one's complement	(~)
bitwise AND	(&)
bitwise OR	(|)
bitwise XOR	(^)
left shift	(<<)
right shift	(>>)

These operator can operate on integer and character value but not on float and double. In bitwise operator the function showbits( ) function is used to display the binary representation of any integer or character value.

In one's complement all 0 changes to 1 and all 1 changes to 0. In the bitwise OR its value would obtaining by 0 to 2 bits.

As the bitwise OR operator is used to set on a particular bit in a number. Bitwise AND the logical AND.

It operate on 2operands and operands are compared on bit by bit basic. And hence both the operands are of same type.

Logical or Boolean Operator

Operator used with one or more operand and return either value zero (for false) or one (for true). The operand may be constant, variables or expressions. And the

expression that combines two or more expressions is termed as logical expression. C has three logical operators :

Operator Meaning

&&	AND
||	OR
!	NOT

Where logical NOT is a unary operator and other two are binary operator. Logical AND gives result true if both the conditions are true, otherwise result is false. And logial OR gives result false if both the condition false, otherwise result is true.

Managing I/O statement

In C Language input and output function are available as C compiler function or C library provided with each C compiler implementation. These all functions are collectively known as **Standard I/O Library function.**

Unformatted I/O functions

Formatted I/O functions

Unformatted I/O functions

There are mainly six unformatted I/O functions discussed as follows:

a) getchar()

b) putchar()

c) gets()

d) puts()

e) getch()

f) getche()

a) getchar()

This function is an Input function. It is used for reading a single character from the keyborad.

The general syntax is as:

v = getchar();

```
/*To read a single character from the keyboard using the getchar() function*/
#include
main()
{
charn;
n=getchar();
}
```

b) putchar()

This function is an output function. It is used to display a single character on the screen. The general syntax is as:

putchar(v);

```
/*Program illustrate the use of getchar() and putchar() functions*/
#include
main()
{
char n;
n = getchar();
putchar(n);
}
```

c) gets()

This function is an input function. It is used to read a string from the keyboar. It is also buffered function.

The general syntax is as:

gets(v);

```
/*Program to explain the use of gets() function*/
#include
main()
{
char n[20];
gets(n);
}
```

d) puts()

This is an output function. It is used to display a string inputted by gets() function.

The general syntax is as:

puts(v);

```
/*Program to illustrate the concept of puts() with gets() functions*/
#include
main()
{
char name[20];
puts("Enter the Name");
gets(name);
puts("Name is :");
puts(name);
}
```

Formatted I/O functions

Formatted I/O functions which refers to an Input or Ouput data that has been arranged in a particular format. There are mainly two formatted I/O functions discussed as follows:

a) scanf()

b) printf()

a) scanf()

The scanf() function is an input function. It used to read the mixed type of data from keyboard. You can read integer, float and character data by using its control codes or format codes. The general syntax is as:

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

```
/*Program to illustrate the use of formatted code by using the formatted scanf() function */
#include
main()
{
char n,name[20];
int abc;
float xyz;
printf("Enter the single character, name, integer data and real value");
scanf("\n%c%s%d%f", &n,name,&abc,&xyz);
getch();
}
```

b) printf()

This ia an output function. It is used to display a text message and to display the mixed type (int, float, char) of data on screen. The general syntax is as:

printf("control strings",&v1,&v2,&v3,.....&vn);

```
/*Below the program which show the use of printf() function*/ #include
main()
{
int a;
float b;
char c;
printf("Enter the mixed type of data");
scanf("%d",%f,%c",&a,&b,&c);
```

```
getch();
}
```