UNIT-3

Concepts:
Arrays and Strings: Arrays: One-Dimensional Arrays, Declaration, Array Initialization, Input and Output of Array Values, Two- Dimensional Arrays.
Strings: String Fundamentals, String Input and Output, String manipulation functions.

What is an array?

- An *Array* is defined as the collection of similar type of data items stored at contiguous memory locations.
- *Arrays* are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.
- The *Array* is the simplest data structure where each data element can be randomly accessed by using its index number.
- Important terms to understand the concept of array:
  - Element
  - Index

What are the advantages of an array?

- Arrays represent multiple data items of the same type using a single name.
- In arrays, the elements can be accessed randomly by using the index number.
- Arrays allocate memory in contiguous memory locations for all its elements. Hence there is no chance of extra memory being allocated in case of arrays. This avoids memory overflow or shortage of memory in arrays.
- Using arrays, other data structures like linked lists, stacks, queues, trees, graphs etc can be implemented.
- Two-dimensional arrays are used to represent matrices.

What are the disadvantages of an array?

- The number of elements to be stored in an array should be known in advance.
- An array is a static structure (which means the array is of fixed size). Once declared the size of the array cannot be modified. The memory which is allocated to it cannot be increased or decreased.
- Insertion and deletion are quite difficult in an array as the elements are stored in consecutive memory locations and the shifting operation is costly.
- Allocating more memory than the requirement leads to wastage of memory space and less allocation of memory also leads to a problem.

What are the properties of an array?

- An array is a derived data type, which is defined using basic data types like int, char, float and even structures.
- Array name represents its base address. The base address is the address of the first element of the array.
- Array's index starts with 0 and ends with N-1. Here, N stands for the number of elements. For Example, there is an integer array of 5 elements, then it's indexing will be 0 to 4.

What are the Applications of Arrays?

- Array stores data elements of the same data type.
- Arrays are used to Perform Matrix Operations. We use two dimensional arrays to create matrix and perform various operations on matrices using two dimensional arrays.
- Maintains multiple variable names using a single name. Arrays help to maintain large data under a single variable name. This avoids the confusion of using multiple variables.
- Arrays can be used for sorting data elements. Different sorting techniques like
  Bubblesort,
  Insertion sort,
  Selection sort etc. use arrays to store and sort elements easily.
- Arrays can be used for performing matrix operations. Many databases, small and large, consist of one-dimensional and two-dimensional arrays whose elements are records.
- Arrays can be used for CPU scheduling.
- Lastly, arrays are also used to implement other data structures like Stacks, Queues, Heaps,Hash tables etc.
- Arrays are used to implement Search Algorithms. We use single dimensional arrays to implement search algorihtms like ...
  Linear Search
  Binary Search

What are the different types of an array?

1. One dimensional array (1D) or single dimensional array
2. Two dimensional array (2D) or Multi-dimensional array

1. One dimensional (1D) array or single dimensional array:

In c programming language, single dimensional arrays are used to store list of values of same data type. In other words, single dimensional arrays are used to store a row of values. In single dimensional array, data is stored in linear form. Single dimensional arrays are also called as one- dimensional arrays, Linear Arrays or simply 1-D Arrays.

How to declare 1D-array?

- To declare an array in C, we need to specify the type of the elements and the number of elements required by an array i.e., size of the elements.
- The following is the syntax for declaring 1-D as follows:

data_type array_name[Size];

- We can create an array with size and also initialize values to it.

datatype arrayName [size] = {value1, value2, ...} ;

- Syntax for creating an array without size and with initial values

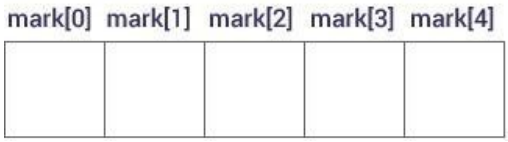
datatype arrayName [ ] = {value1, value2, ...} ;

- In the above syntax, the **data type** specifies the type of values we store in that array and **size** specifies the maximum number of values that can be stored in that array.

For example:

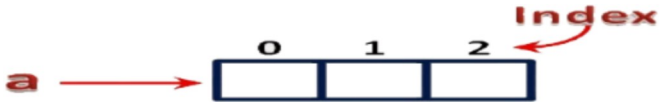
float mark[5];

Here, we declared an array called **mark**, of floating-point type and its size is 5. Meaning, it can hold 5 floating-point values.



int a[3];

In the above memory allocation, all the three memory locations have a common name 'a'. So accessing individual memory location is not possible directly. Hence compiler not only allocates the memory but also assigns a numerical reference value to every individual memory location of an array. This reference number is called "Index" or "subscript" or "indices". Index values for the above example are as follows...

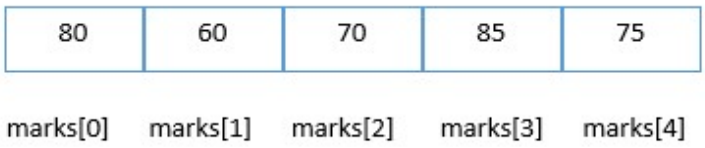


How to initialize one-dimensional array:

- The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index.

Example:

```
int marks[5];
marks[0]=80;  //initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```



- It is not necessary to define the size of array during initialization. We can also initialize the size of an array like this:

  int marks[]={80,60,70,85,75};
- The above example tells that memory is allocated at run time based on the number of elements given in an array. This is another approach to initialize an array.
- The number of elements in the initialization list should be less than the size of array because index is starts with 0.
- If the array size is greater than the list of elements in the array, then it first allocate memory for the elements are remaining space is filled with zero's. for example:

  int marks[5]={10,20,30};

marks[0]	marks[1]	marks[2]	marks[3]	marks[4]
10	**20**	**30**	**0**	**0**

- If the array elements are greater than the array size, then it shows compilation error.
  int age[4]={1,2,3,4,5} // ERROR

How to Access 1D-Array Elements:

In c programming language, to access the elements of single dimensional array we use array name followed by index value of the element that to be accessed. Here the index value must be enclosed in square braces. Index value of an element in an array is the reference number given to each element at the time of memory allocation. The index value of single dimensional array starts with zero (0) for first element and incremented by one for each element. The index value in an array is also called as subscript or indices.

Key points:

- Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
- If the size of an array is 'n', to access the last element, the 'n-1' index is used. In this example, mark[4].

Example:

- The elements can be accessed from an array with help of index values.
- Suppose declare an array **mark** with int data type followed by some size i.e. **int mark[5];**
- The first element is accessed with **mark[0],** the second element is **mark[1]** and so on.

marks[0]	marks[1]	marks[2]	marks[3]	marks[4]
10	**20**	**30**	**40**	**50**

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int mark[5]={10,20,30,40,50};
 printf("%d", mark[ [0]);          //print's the 0th element i.e. 10
 printf("%d", mark[ [1]);          //print's the 1st element i.e. 20
 printf("%d", mark[ [2]);          //print's the 2nd element i.e. 30
 printf("%d", mark[ [3]);          //print's the 3rd element i.e. 40
 printf("%d", mark[ [4]);          //print's the 4th element i.e. 50
 getch( );
}
```

How to Read 1D-array values:

```
int a[5];
for (i=0;i<5;i++)
 {
            scanf("%d",&a[i]);
 }
```

Reading values into array a []

How to Writing 1D-array values:

```
int a[5];
for (i=0;i<5;i++)
   {
                                                    Printing values into array a[]
       printf("%d", a[i]);
    }
```

Example Program: Write a C Program to print 1-D array element

```
#include<stdio.h>void main()
{
        int a[10], n, i;
        printf("Enter the number of elements=");
        scanf("%d", &n);
        printf("Read the elements =");
        for(i=0;i<n;i++)
        {
           scanf("%d", &a[i]);
        }
        printf("The elements are=");
        for(i=0;i<n;i++)
        {
          printf("%d\t", a[i]);
        }
}
```

2. Two dimensional Array (2D) or Multidimensional array:

An array of arrays is called as 2D or multi dimensional array. In simple words, an array created with more than one dimension (size) is called as multi dimensional array. Multi dimensional array can be of **two dimensional array** or **three dimensional array** or **four dimensional array** or more. Most popular and commonly used multi dimensional array is **two dimensional array**. The 2-D arrays are used to store data in the form of table. We also use 2-D arrays to create mathematical **matrices**.

Declaration of Two Dimensional Array:

We use the following general syntax for declaring a two dimensional array...

datatype arrayName [rowSize] [columnSize] ;

Example:

int array1[2][3] ;

The above declaration of two dimensional array reserves 6 continuous memory locations of 2 bytes each in the form of **2 rows** and **3 columns**.

Initialization of Two Dimensional Array:

We use the following general syntax for declaring and initializing a two dimensional array with specific number of rows and columns with initial values.

datatype arrayName [rows][columns] = {{val1, val2, ...},{val3, val4...}...} ;

Example:

```
int array1[2][3] = { {1, 2, 3},{4, 5, 6} } ;
```

The above declaration of two-dimensional array reserves 6 contiguous memory locations of 2 bytes each in the form of 2 rows and 3 columns. And the first row is initialized with values 1, 2 & 3 and second row is initialized with values 4, 5 & 6.

We can also initialize as follows...

Example:

```
int array1[2][3] = {
                     {1, 2, 3},
                     {4, 5, 6}
                   };
```

(OR)

Another way to initialize value into an array:

```
int a[2][2];
a[0][0]=10;
a[0][1]=20;
a[1][0]=30;
a[1][1]=40;
```

Example :

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        int a[2][2],i;
        a[0][0]=10;
        a[0][1]=20;
        a[1][0]=30;
        a[1][1]=40;
        for(i=0;i<2;i++)
        {
                printf("\n");
                for(j=0;j<2;j++)
                {
                        Printf("%d\t", a[i][j]);
                }
        }
 getch();
}
```

Accessing Individual Elements of Two Dimensional Array:

In a c programming language, to access elements of a two-dimensional array we use array name

followed by row index value and column index value of the element that to be accessed. Here the row and column index values must be enclosed in separate square braces. In case of the two- dimensional array the compiler assigns separate index values for rows and columns.
We use the following general syntax to access the individual elements of a two-dimensional array...

arrayName [rowIndex] [columnIndex];

Example:

array1[0][1] ; **//if we consider the above example then it will give the value '2'.**

Example :

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        int a[2][3]={ {1,2,3},{4,5,6}};
        int i,j;
        for(i=0;i<2;i++)                    //for rows
        {
                for(j=0;j<3;j++)            //for columns
                {
                        Printf("%d\t", a[i][j]);
                }
        }
 getch();
}
```

STRINGS

Strings Introduction:

- Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.
- The termination character ('\0') is important in a string since it is the only way to identify where the string ends.
- When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

Declaration of a String:

- Declaring a string is as simple as declaring a one-dimensional array.
- Syntax for declaring a string:

  char str_name[size];
- In the above syntax str_name is any name given to the string variable and size is used to define the length of the string.
- Example: **char str[10];**

Initialization of a string:

- A string can be initialized in different ways:
  1. char str[] = "welcome";

Here, size is not mentioned so that it will automatically allocate memory at run time and by default '\0' is appended at the end of string.

2. char str[20] = "welcome";
3. char str[] = {'w', 'e', 'l', 'c', 'o', 'm', 'e','\0'};
4. char str[8] = {'w', 'e', 'l', 'c', 'o', 'm', 'e','\0'};

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char str1[] = "welcome";
        char str2[20] = "welcome";
        char str3[] = {'w', 'e', 'l', 'c', 'o', 'm', 'e','\0'};
        char str4[8] = {'w', 'e', 'l', 'c', 'o', 'm', 'e','\0'};
        printf("%s", str1);
        printf("%s", str2);
        printf("%s", str3);
        printf("%s", str4);
        getch();
}
```

Output:

```
welcome
welcome
welcome
welcome
```

String Input and Output Functions:

Input Functions:

- scanf function
- gets function:
- getchar function

Output functions:

- printf function
- puts function
- putchar function

(1) **getchar() and putchar() Functions:**

The **getchar()** 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 **putchar()** 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 −

Example

```
#include<stdio.>
void main( )
 {
  char c;
  printf( "Enter a value :");
  c = getchar( );
  printf( "\nYou entered: ");
   putchar( c );
 }
```

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 −

Output:

```
Enter a value: welcome
You entered: w
```

(2) The gets() and puts() Functions::

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

Example:

```
#include <stdio.h>
void main( )
{
  char str[100];
  printf( "Enter a string :");
  gets( str );
  printf( "\n The string is: %s", str);
}
```

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:

Output:

```
Enter a string : welcome to c
The string is: welcome to c
```

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function.

Example:

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

```
#include <string.h>
void main()
{
char name[50];
printf("Enter your name: ");
gets(name);          //reads string from user
printf("Your name is: ");
puts(name);          //displays string
getch();
}
```

Output:

```
Enter your name: welcome to c program
Your name is: welcome to c program
```

(3) The scanf() and printf() Functions:

The **scanf()** function reads the input from the standard input stream **stdin** and scans that input according to the **format** provided. In scanf() function no need to '&' symbol because characters are entered into an array according to base address.

The **printf()** 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>
void main( )
 {
   char
   str[100];
   printf( "Enter a string :");
   scanf("%s", str);
   printf( "\nYou entered: %s", str);
}
```

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

Output 1:

```
Enter a string : HelloWorld
You entered: HelloWorld
```

Output 2:

```
Enter a string: Hello World
You entered: Hello
```

Here, while reading a string, scanf() stops reading as soon as it encounters a space, so "Hello World" are two strings for scanf().

String manipulation functions/library functions/pre defined functions/built-in functions:

1. **String length::strlen()**
2. **String copy::strcpy()**
3. **String concatenation::strcat();**
4. **String reverse::strrev()**
5. **String comparison::strcmp()**
6. **String upper :: strupr()**
7. **String lower::strlwr()**

String length: strlen( )

- String length function in C gives the length of the string.
- Syntax for this function is given below:

  strlen(string_name);
- strlen( ) function counts the number of characters in a given string and returns the integer value.
- It stops counting the character when null character is found. Because, null character indicates the end of the string in C.

Example:

```
#include <stdio.h>
#include<conio.h>
#include <string.h>
void main( )
{
        char name[20] ;
        int len;
        printf("Enter string");
        gets(name);
        len = strlen(name) ;
        printf ( "\string length = %d \n" , len ) ;
        getch();
}
```

Output:
Enter string hello world
String length= 11

String copy: strcpy( )

- String copy function copies contents of one string into another string.
- Syntax for strcpy( ) function is given below:
  strcpy (destination, source);
- Example:
  - strcpy (str1,str2) – It copies contents of str2 into str1.
  - strcpy (str2, str1) – It copies contents of str1 into str2.
- If destination string length is less than source string, entire source string value won't be copied into destination string.

Example:

```
#include<stdio.h>
#include<string.h>
void main()
{
   char str1[20];
   char str2[20];
   printf("Enter a string1");
   gets(str1);
   strcpy(str2,str1);
   printf("The copied string is=%s",str2);
}
```

Output:

```
Enter a string1  vignan
The copied string is=vignan
```

String concatenation: strcat( )

- strcat( ) function in C language concatenates two given strings. It concatenates source string at the end of destination string.
  Syntax:
  strcat (destination, source);
- As you know, each string in C is ended up with null character ('\0').
- In strcat( ) operation, null character of destination string is overwritten by source string's first character and null character is added at the end of new destination string which is created after strcat( ) operation

Example:

```
#include <stdio.h>
#include <string.h>
void main( )
{
char string1[ 20] ;
char string2[20];
printf ( "\n enter string1 :" ) ;
gets(string1);
printf ( "enter string2:") ;
gets(string2);
strcat( string1,string2);
printf ( "The string after concatenation= %s", string1);
}
```

Output:

```
Enter string1: vignan
Enter string2: institute
The string after concatenation=vignaninstitute
```

String reverse: strrev( )

- strrev( ) function reverses a given string in C language.
- Syntax for strrev( ) function is given below.

strrev(string);

Example:

```
#include<stdio.h>
#include<string.h>
void main()
{
  char name[30] = "Hello";
  printf("String before reverse : %s\n", name);
  strrev(name);
  printf("String after  reverse : %s", name);
}
```

Output:

```
String before  reverse: Hello
String after reverse: olleH
```

String Comparison: strcmp( ):

- strcmp( ) function in C compares two given strings and returns zero if they are same.
- If length of string1 < string2, it returns < 0 value.
- If length of string1 > string2, it returns > 0 value.
- Syntax for strcmp() function is given below.

strcmp (str1, str2);

- strcmp( ) function is case sensitive. i.e, “A” and “a” are treated as different characters.

Example:

```
#include <stdio.h>
#include <string.h>
void main( )
{
char str1[50] = "abcd" ;
char str2[50] = "abCd";
char str3[50] = "abcd";
int res;
res = strcmp ( str1, str2 ) ;
printf("%d \n", res);
 res = strcmp ( str1, str3 ) ;
printf ("%d ", res ) ;
getch();
}
```

Output:

32
0

String upper: strupr( ):

- strupr( ) function converts a given string into uppercase.
- Syntax for strupr( ) function is given below.

  strupr(string);
- strupr( ) function is non standard function which may not available in standard library in C.

Example:

```
#include<stdio.h>
#include<string.h>
void  main()
{
char str[ ] = "vignan";
strupr(str);
printf("%s", str);
getch();
}
```

Output:

VIGNAN

String Lower: strlwr( ):

- strlwr( ) function converts a given string into lowercase.
- Syntax for strlwr( ) function is given below.

  strlwr(string);
- strlwr( ) function is non standard function which may not available in standard library in C.

Example:

```
#include<stdio.h>
#include<string.h>
void main()
{
   char str[ ] = "VIGNAN";
   strlwr(str);
    printf("%s", str);
}
```

Output:

vignan

String manipulation functions without using built-in functions:

1. string length:

```
#include<stdio.h>
#include<conio.h>
void main( )
{
        char str[50];
        printf("Enter a string=");
        gets(str);
        int i, length=0;
        for(i=0;str[i]!='\0';i++)
        {
                length++;
        }
        printf("the length of the string is=%d",length);
        getch();
}
```

Output:

Enter a string= hello world
the length of the string is=11

2. string copy:

```
#include<stdio.h>
#include<conio.h>
void main( )
{
        char str[50],str1[50];
        printf("Enter a string=");
        gets(str);
        int i;
        for(i=0; str[i]!= '\0'; i++)
        {
                str1[i]=str[i];
        }
        str1[i]='\0';
        printf(" copied string is=%s",str1);
        getch();
}
```

Output:

Enter a string=vignan
copied string is=vignan

3. string comparison:

```
#include<stdio.h>
Void main()
{
      char str[50], str2[50];
      int i;
      printf(“enter string1 and string2:”);
      gets(str1);
      gets(str2);
      for(i=0; str1[i]!='\0'&&str2[i]!=’\0’; i++)
      {
            if(str[i]!=str2[i])
             break;
      }
      printf("Comparison result=%d",str1[i]-str2[i]);
     getch();
}
```

Output:

Enter string1 and string2 : ABCD
ABCD
Comparison result=0

Enter string1 and string2: ABCD
ABCd
Comparison result= -32

4. string concatenation:

```
#include <stdio.h>
void main()
{
   char str1[20], str2[20];
   int i, length=0;
   printf("Enter string 1 and string 2=”);
   gets(str1);
   gets(str2);
   for(i=0;str1[i] != '\0';i++)
   {
      length++;
   }
   for(i=0;str2[i] != '\0';i++)
   {
       str1[length+i]=str2[i];
   }
   str1[length+i] = '\0';
   printf("\nConcatenated string: %s", str1);
}
```

Output:
Enter string 1 and string 2: welcome
vignan
Concatenated string: welcomevignan

5. string upper:

```
#include <stdio.h>
void main()
{
   char str1[20];
   int i=0;
   printf("Enter string=”);
   gets(str1);
   while(str1[i] != '\0')
  {
      if(str1[i]>=’a’ && str1[i]<=’z’)
      {
          Str1[i]=str1[i]-32;
       }
     i++;
   }
  printf(“Upper case letter is=%s”,str1);
  getch();
}
```

Output:
Enter string: hello world
Upper case letter is: HELLO WORLD

6. string lower:

```
#include <stdio.h>
void main()
{
   char str1[20];
   int i=0;
   printf("Enter string=”);
   gets(str1);
   while(str1[i] != '\0')
  {
      if(str1[i]>=’A’ && str1[i]<=’Z’)
      {
          Str1[i]=str1[i]+32;
       }
     i++;
   }
  printf(“Lower case letter is=%s”,str1);
  getch();
}
```

Output:
Enter string: HELLO WORLD
Lowercase letter is: hello world

Programs on Arrays

1. Write a C program to read and print array elements

Program:

```
#include<stdio.h>
 void main()
 {
          int a[10], n, i;
          printf("Enter the number of elements=");
          scanf("%d", &n);
          printf("Read the elements =");
          for(i=0;i<n;i++)
          {
             scanf("%d", &a[i]);
          }
          printf("The elements are=");
          for(i=0;i<n;i++)
          {
             printf("%d\t", a[i]);
          }
 }
```

Output:

```
Enter the number of elements= 5
Read the elements:1
2
3
4
5
The elements are:
1 2 3 4 5
```

2. Write a C program to access index 3 from an array

Source code:

```
#include<stdio.h>
void main()
{
        int a[10],i,n;
        prntf("enter no.of elements:");
        scanf("%d'",&n);
        printf("enter elements into array:");
        for(i=0;i<n;i++)
```

```
        {
            scanf(%d",&a[i]);
        }
        printf("element at index 3 is %d",a[3]);
    }
```

Output:

```
enter no.of elements:5
enter elemnts into array: 43 54 21 76 23
element at index 3 is 76
```

3. Write a c program to print array elements in reverse order

Source code:

```
#include<stdio.h>
void main()
{
 int a[10], n, i;
 printf("Enter the number of elements=");
 scanf("%d", &n);
 printf("Read the elements =");
 for(i=0;i<n;i++)
 {
    scanf("%d", &a[i]);
 }
 printf("The reverse of an array is=");
 for(i=n-1;i>=0;i--)
 {
     printf("%d\t", a[i]);
  }
}
```

Output:

```
Enter the number of elements= 5
Read the elements:1
2
3
4
5
The reverse of an array is= 5 4 3 2 1
```

4. write a c program to print the sum of array elements

Source code:

```
#include<stdio.h>
void main()
{
  int a[10], n, i,sum=0;
  printf("Enter the number of elements=");
  scanf("%d", &n);
  printf("Read the elements =");
  for(i=0;i<n;i++)
  {
     scanf("%d", &a[i]);
```

```
  }
  for(i=0;i<n;i++)
  {
      sum=sum+a[i];
   }
  printf("The sum of an array is=%d\t", sum);
  }
```

Output:

Enter the number of element: 4
Read the elements=10
20
30
40
The sum of an array is=100

5. Write a c program to find largest and smallest element in an array

Source code:

```
#include<stdio.h>
void main()
{
int a[10], n, i,min,max;
printf("Enter the number of elements=");
scanf("%d", &n);
printf("Enter the elements =");
for(i=0;i<n;i++)
{
   scanf("%d", &a[i]);
}
min=a[0],max=a[0];
for(i=0;i<n;i++)
{
    if(min>a[i])
      min=a[i];
   if(max<a[i])
      max=a[i];
 }
printf("The Minimum of array is=%d\t",min);
printf("The Maximum of array is=%d\t",max);
getch();
}
```

Output:
Enter the number of elements: 5
Enter the elements: 80
25
45
77
9
The Minimum of array is=9
The Maximum of array is =80

6. Write a C program to perform Linear Search:

Program :

```
#include<stdio.h>
void main()
{
   int a[10],i,n,count=0,key;
   printf("enter number of elements=");
   scanf("%d",&n);
   printf("enter elements=");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   printf("enter key value");
   scanf("%d",&key);
   for(i=0;i<n;i++)
   {
      if(a[i]==key)
      {
         count++;
      }
   }
   if(count==1)
   {
      printf("elements is found");
   }
   else
   {
      printf("elements not found");
   }
getch();
}
```

Output:

```
enter number of elements=5
enter elements=55
44
8
52
74
enter key value 52
elements found
```

7. Write a C program to perform transpose of a matrix.

Program:

```
#include<stdio.h>
void main()
{
   int a[10][10],trans[10][10],i,j,r,c;
   printf("enter number of row and columns=");
   scanf("%d%d",&r,&c);
   printf("enter elements=");
```

```
    for(i=0;i<r;i++)
    {
       for(j=0;j<c;j++)
       {
          scanf("%d",&a[i][j]);
       }
    }
    printf("the matrix is:");
    for(i=0;i<r;i++)
    {
       printf("\n");
       for(j=0;j<c;j++)
       {
          printf("%d\t",a[i][j]);
       }
    }
    for(i=0;i<r;i++)
    {
       for(j=0;j<c;j++)
       {
          trans[ j ][ i ] =a[ i ][ j ];
       }
    }
    printf("\n the transpose of a matrix");
    for(i=0;i<r;i++)
    {
       printf("\n");
       for(j=0;j<c;j++)
       {
          printf("%d\t",trans[i][j]);
       }
    }
getch();
}
```

Output:

```
enter number of row and columns=2 2
enter elements=10
20
30
40
the matrix is:
10      20
30      40
 the transpose of a matrix
10      30
20      40
```

8. Write a C program to read and print a matrix

Source code:

```
#include<stdio.h>
 void main()
{
        int a[20][20],m,n,i,j;
        printf("enter the size of row and column:");
        scanf("%d%d",&m,&n);
        printf("enter elements into matrix:");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        Printf("The elements are:\n");
        for(i=0;i<m;i++)
        {
                 printf("\n");
                for(j=0;j<n;j++)
                {
                                printf("%d\t",a[i][j]);
                }

        }
```

Output:

```
        Enter the size of row and column:2 2
        Enter elements into array:1 2 3 4
        The elements are:
                        1       2
                        3       4
```

9. Write a C program to add two matrixes

Source Code:

```
        #include<stdio.h>
        void main()
        {
        int c[20][20], a[20][20],b[20][20],r,c,i,j;
        printf("enter the size of row and column:");
        scanf("%d%d",&r,&c);
        printf("enter elements into a matrix:");
        for(i=0;i<r;i++)
        {
                for(j=0;j<c;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        Printf("enter elements into matrix b:");
```

```
        for(i=0;i<r;i++)
        {
                for(j=0;j<c;j++)
                {
                        scanf("%d",&b[i][j]);
                }
        }
        for(i=0;i<r;i++)
        {
                for(j=0;j<c;j++)
                {
                        c[i][j]=a[i][j]+b[i][j];
                }
        }
        printf("Addition of two matrix is:\n");
        for(i=0;i<c;i++)
        {
                for(j=0;j<c;j++)
                {
                        printf("%d\t",c[i][j]);
                }
                printf("\n");
        }
        }
```

Output:

```
Enter size of row and column: 2 2
Enter elements into a matrix: 1
 2
 3
 4
Enter elements into b matrix : 5
 6
 7
 8
 Addition of two matrix is:
6       8

10      12
```

10. Write a c program to multiply two matrices

Source code:

```
#include<stdio.h>
void main()
{
   int a[10][10],b[10][10],c[10][10],i,j,k,r1,c1,r2,c2;
   printf("enter number of row and columns into matrix a");
   scanf("%d%d",&r1,&c1);
   printf("enter number of row and columns into matrix b");
   scanf("%d%d",&r2,&c2);
```

```
if(c1==r2)
 {
   printf("enter elements into matrix a");
   for(i=0;i<r1;i++)
   {
      for(j=0;j<c1;j++)
      {
         scanf("%d",&a[i][j]);
      }
   }
   printf("enter elements into matrix b");
   for(i=0;i<r2;i++)
   {
      for(j=0;j<c2;j++)
      {
         scanf("%d",&b[i][j]);
      }
   }
   for(i=0;i<r1;i++)
   {
      for(j=0;j<c1;j++)
      {
         c1[i][j]=0;
         for(k=0;k<c1;k++)
         {
           c1[i][j]=c1[i][j]+(a[i][k]*b[k][j]);
         }
      }
   }
   printf("\n the multiplication of a matrix");
   for(i=0;i<r1;i++)
   {
      printf("\n");
      for(j=0;j<c1;j++)
      {
         printf("%d\t",c1[i][j]);
      }
   }
 else
 {
  printf("Matrix multiplication not possible");
 }
getch();
}
```

Output:

```
enter number of row and columns=2
2
enter a[i][j] elements=1
2
3
4
enter b[i][j] elements=5
6
7
```

8

the multiplication of a matrix
19 22
43 50

11. Write a C program to perform Binary Search

Source code:

```
#include<stdio.h>
void main()
{
   int a[10],i,n,count=0,key,low,high,mid;
   printf("enter number of elements=");
   scanf("%d",&n);
   printf("enter elements=");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   printf("enter key value");
   scanf("%d",&key);
   low=0,
   high=n-1;
   while(low<=high)
   {
       mid=(low+high)/2;
       if(key<a[mid])
       {
          high=mid-1;
       }
        if(key>a[mid])
       {
          low=mid+1;
       }
     if(key==a[mid])
       {
          count++;
          break;
       }
   }
if(count==1)
   printf("Element is found");
else
    Printf("Element is not found");
}
```

Output:
Enter the number of elements:5
Enter the elements: 10
20
30
40

50
Enter key value: 40
Element is found

12. Write a c program to print the list of elements in a sorted order (bubble sort)

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[10],i,j,n,temp;
   printf("enter number of elements=");
   scanf("%d",&n);
   printf("enter elements=");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
  for(i=0;i<n;i++)
  {
     for(j=0;j<n;j++)
    {
       if(a[j]>a[j+1])
      {
            temp=a[j];
           a[j]=a[j+1];
           a[j+1]=temp;
      }
    }
  }
printf("after sorting")
for(i=0;i<n;i++)
{
    printf("%d\t",&a[i])
 }
getch();
}
```

Output:
Enter no of elements: 5
Enter elements: 10
50
30
40
20
After sorting: 10 20 30 40 50

Programs on Strings

1. Write a C program to find reverse of a string.

Source code:

```
#include<stdio.h>
#include<string.h>
void main() {
  char str[100];
```

```
    int i, j, temp, len=0;
    printf("\nEnter the string :");
    gets(str);
    for(i=0;str[i]!='\0';i++)
    {
      len++;
    }
    i = 0;
    j = len-1 ;
    while (i<j)
    {
          temp=str[i];
          str[i]=str[j];
          str[j]=temp;
          i++;
          j--;
    }
    printf("\nReverse string is :%s", str);
    getch();
}
```

Output:

Enter the string: vignan

Reverse string is : nangiv

2. Write a C program to check whether a string is palindrome or not.

Source code:

```
#include<stdio.h>
#include<string.h>
void main() {
  char str[100];
   int i,j,len=0;
  printf("\nEnter the string :");
  gets(str);
  for(i=0;str[i]!='\0';i++)
  {
    len++;
  }
  i = 0;
  j = len-1;
  while (i<j)
  {
        if(str[i]! = str[j])
        {
          break;
        }
```

```
            i++;
            j--;
      }
     if(i>=j)
     {
       printf("string is a palindrome");
     }
     else
     {
        printf("string is not a palindrome");
     }
   getch();
   }
```

Output:

Enter the string: madam

madam is a palindrom

Output 2:

Enter the string: hai

Given string is not a palindrom