# MODULE 3

# Module-3

Topics

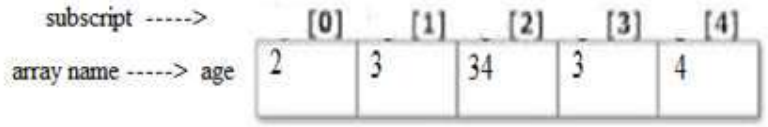
1. ARRAYS
2. STRINGS
3. SEARCHING ALGORITHMS
4. SORTING ALGORITHMS

# Chapter 1

# ARRAYS

## 1.1 Introduction

- Array is a collection of elements of same data type.
- The elements are stored sequentially one after the other in continuous memory location.
- Any element can be accessed by using
    - name of the array
    - position of element in the array
- Each value in an array is reference by a single name, which is the name of the array, and a subscript or index, which indicates the position of the value in the array.
- The subscript is a positive integer number, which is enclosed in a pair of square brackets.
- For ex: int age[5];




Array Elements

- Following are some rules that need to be kept in mind while using arrays:
    - The array should be declared with some data type.
    - The size of the array should be specified at the time of its declaration.
    - In an array, elements are stored sequentially, that is, in contiguous memory locations.
- Arrays are classified into 2 types:
    - **Single dimensional array**
    - **Multi-dimensional array**

## 1.2. SINGLE DIMENSIONAL ARRAY

- A single(one) dimensional array is a linear list consisting of related elements of same type.

- In memory, all the elements are stored in continuous memory-location one after the other.

**Declaration of one Dimensional Array:**

- A one dimensional array can be declared using the following syntax

  **data_type array_name[array_size]**

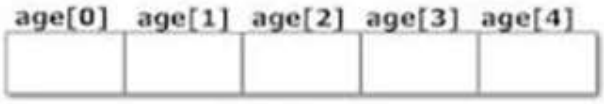
  where **data_type** can be int, float or char

  **array_name** is name of the array

  **array_size** indicates number of elements in the array

- For ex: int age[5];

  The above code can be pictorially represented as shown below:




Array Elements

- Here, the size of array "age" is 5 times the size of int because there are 5 elements.

  **Total amount of memory = size * [sizeof(data_type)]**

- Using this formula, for int age[5], The total amount of memory =20 bytes. (5*4).

## Initialization of One-Dimensional Array:

- The values can be stored in array using following methods:
  1. Initializing all specified memory locations (compile time initialization)
  2. Initializing individual elements (compile time initialization)
  3. Partial array initialization.
  4. Initialization during program execution. ( run time initialization)

**1. Initializing all specified memory locations**

- The syntax to initialize all the elements of a one-dimensional array at a same time is as Follows:

**data_type array_name[array_size] = { v1 , v2 , v3 };**

where **data_type** can be int, float or char

**array_name** is name of the array

**array_size** indicates number of elements in the array

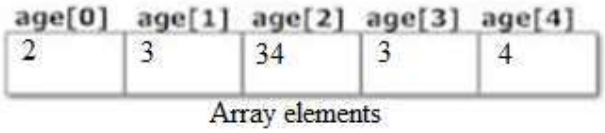
**v1, v2, v3** are values

- The elements values in the right hand side of the preceding syntax must be enclosed in braces and must be separated by comma.
- These values are assigned to the array elements in the same order as the order of these values.

- For ex:

int age[5]={2,4,34,3,4}

The above code can be pictorially represented as shown below:



Array elements

**Example: Program to illustrate the initialization of one-dimensional array**.

```
#include<stdio.h>
main( )
{
  int arr[5] = {10, 20, 30, 40, 50};       //array declaration and initialization
  int i;
  printf("Elements of the array are:\n");
  for(i=0; i<5; i++)
  {
  printf("%d", arr[i]);
  }
}
```

*Output:*
Elements of the array are:
10 20 30 40 50

**2. Initializing individual elements**

- The syntax for initializing an individual element of a one-dimensional array is as follows:

  **array_name[index] = value;**

- In the preceding syntax,
  **array_name** represents the name of the array,
  **index** represents the position of the array element in the array,
  **value** represents the value being assigned to the array element.

- For ex:

```
int age[5];

age[0]=2;        //value 2 stored in array  "age" at position 0
age[1]=4;        //value 4 stored in array  "age"at position 1
age[2]=34;       //value 34 stored in array  "age" at position 2
age[3]=3;        //value 3 stored in array  " age"at position 3
age[4]=4;        //value 4 stored in array  "age" at position 4
```

**Example: Program to illustrate assigning values to one-dimensional array.**

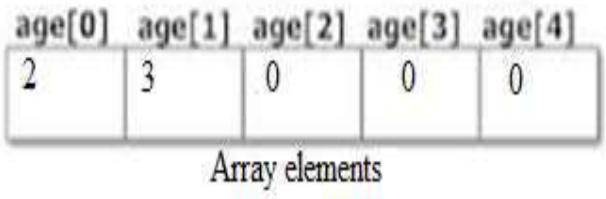
```
#include<stdio.h>
main()
{

        int age[5];
        age[0]=2;
        age[1]=4;
        age[2]=34;
        age[3]=3;
        age[4]=4;
        printf("Value in array age[0] : %d \n", age[0]);
        printf("Value in array age[1] : %d \n", age[1]);
        printf("Value in array age[2] : %d \n", age[2]);
        printf("Value in array age[3] : %d \n", age[3]);
        printf("Value in array  age[4] : %d ",age[4]);

}
```

Output:

```
Value in array age[0] : 2
Value in array age[1] : 4
Value in array age[2] : 34
Value in array age[3] : 3
Value in array age[4] : 4
```

**3. Partial array initialization**

- It is allowed in C language. If the number of values to be initialized is less than size of array, then the remaining locations will be initialized to zero automatically by compiler
- For ex: int age[5]={2,3};



**4. Initialization during program execution**

- In this type the initialization is done during program execution.
- For ex:

```
for(i=0;i<100;i++)
{
 if(i<50)
   sum[i]=0;
else
   sum[i]=1;
}
```

**Reading and writing one dimensional array**

The following shows the reading and writing to the one dimensional array:

- For ex:

```
int age[5]
scanf("%d", &age[0]);      //read data from keyboard into array age at position 0.
scanf("%d", &age[1]);      //read data from keyboard into array age at position 1.
scanf("%d", &age[2]);      //read data from keyboard into array age at position 2.
scanf("%d", &age[3]);      //read data from keyboard into array age at position 3.
scanf("%d", &age[4]);      //read data from keyboard into array age at position 4.
```

- In general, to read 5 values, we can write as follows

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

- Similarly, to display 5 elements stored in the array, we can write

```
for(i=0;i<5;i++)
{
printf("%d", age[i]);
}
```

- **Example: Program to illustrate reading/writing to one-dimensional array.**

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

{
        int age[5], i;
        printf("Enter 5 numbers:\n”);
        for(i=0;i<5;i++)
        {

                scanf("%d", &age[i]);
        }

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

        {
                printf("Value in array age[%d] : %d \n ", i, age[i]);
        }

}
```

Output:

```
Enter 5 numbers: 2  4 34 3 4
Value in array age[0] :2
Value in array age[1] :4
Value in array age[2] :34
Value in array age[3] :3
Value in array age[4] :4
```

## 1.3. MULTI DIMENSIONAL ARRAYS

- Arrays with two or more dimensions are called multi-dimensional arrays.
- For ex:

  int matrix[2][3];

- The above code can be pictorially represented as shown below:

| | col 1 | col 2 | col 3 |
|---|---|---|---|
| row 1 | a[0][0] | a[0][1] | a[0][2] |
| row 2 | a[1][0] | a[1][1] | a[1][2] |

Multidimensional array

- A two dimensional array is used when elements are arranged in a tabular fashion.

- Here, to identify a particular element, we have to specify 2 indices:
  →First index identifies the row number of the element.
  →Second index identifies the column number of the element.

**Declaration of Two Dimensional Array**

- The syntax is shown below:

  **data_type array_name[size1][size2];**

  - **data_type** represents the data type of the array to be declared,
  - **array_name** represents the name of the array,
  - **size1** represents the number or rows,
  - **size2** represents the number of columns.

- For ex:

  int matrix[2][3];

  The above code can be pictorially represented as shown below

| | col 1 | col 2 | col 3 |
|---|---|---|---|
| row 1 | | | |
| row 2 | | | |

**Initialization of Two Dimensional Arrays**

- Similar to one-dimensional array, the elements of a two-dimensional array can be initialized Either one at a time or all at once.

The syntax is:

**data_type array_name[size1][size2] = {value1, value2, ……., value n}**

• For ex:

int matrix[2][3]= {1, 23, 11, 44, 5, 67};

Initialization of the array elements in the preceding line of code is equivalent to initializing each array element separately as follows:

matrix[0][0] = 1;
matrix [0][1] = 23;
matrix [0][2] = 11;
matrix [1][0] = 44;
matrix [1][1] = 5;
matrix [1][2] = 67;

The above code can be pictorially represented as shown below:

| | col 1 | col 2 | col 3 |
|---|---|---|---|
| row 1 | 1 | 23 | 11 |
| row 2 | 44 | 5 | 67 |

**Partial array initialization**

- If the number of values to be initialized are less than the size of the array, then the remaining locations will be initialized to 0 automatically.
-

For ex:
int matrix[2][3]= {1, 23,44, 5 };

| | col 1 | col 2 | col 3 |
|---|---|---|---|
| row 1 | 1 | 23 | 0 |
| row 2 | 44 | 5 | 0 |

**Reading and Writing Two Dimensional Arrays**

**For ex**:
int matrix[2][3]

scanf("%d", &matrix[0][0]); //read data from keyboard into matrix at row=0 col=0.
scanf("%d", &matrix[0][1]); //read data from keyboard into matrix at row=0 col=1.
scanf("%d", &matrix[0][2]); //read data from keyboard into matrix at row=0 col=2.
scanf("%d", &matrix[1][0]); //read data from keyboard into matrix at row=1 col=0.
scanf("%d", &matrix[1][1]); //read data from keyboard into matrix at row=1 col=1.
scanf("%d", &matrix[1][2]); //read data from keyboard into matrix at row=1 col=2.

- In general, to read 6 values, we can write:

```
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
  {
        scanf("%d", &matrix[i][j]);
  }
}
```

- Similarly to display 6 elements stored in the matrix, we can write

```
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
  {
        printf("%d ", matrix[i][j]);
  }
}
```

- **Example: Program to illustrate reading/writing to two-dimensional array.**

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

        int matrix[2][3], i, j;
        printf("Enter elements of 2*3 matrix: \n");
        for(i=0;i<2;i++)
        {
                for(j=0;j<3;j++)
                {
                        scanf("%d", &matrix[i][j]);
                }
        }
```

```
        printf("Elements of the matrix are: \n");
        for(i=0;i<2;i++)
        {
                for(j=0;j<3;j++)
                {
                        printf("%d  \t", matrix[i][j]);

                }

                printf("\n");
        }
}
```

*Output:*

```
Enter elements of 2*3 matrix:
1      23    11
44     5     67

Elements of the matrix are:
1              23        11
44             5         67
```

**Limitations of Arrays**

1. The size of an array that you specify at the time of creation cannot be altered later. In other words, you cannot add more elements to an array than specified in its size.
2. It is very difficult to insert an element between two existing elements. If you want to insert a new element between two existing elements, you first need to make space for the new element by moving all the existing elements up by one index.
3. Shortage of memory, if we don't know the size of memory in advance.
4. Wastage of memory, if array of large size is defined.

**Advantages of Array**

1. It is used to represent multiple data items of same type by using only single name.
2. It can be used to implement other data structures like linked lists, stacks, queues, trees, etc.
3. 2D arrays are used to represent matrices.

**Example Programs on Array**

**Program to find largest element of an array**

```
#include <stdio.h>
void main()
{
        int i,n;
        float arr[100];
        printf("Enter total number of elements(1 to 100): ");
        scanf("%d",&n);
        for(i=0;i<n;++i)  // Stores number entered by user.
        {

          printf("Enter Number %d: ",i+1);
          scanf("%f",&arr[i]);

        }
        for(i=1;i<n;i++)          // Loop to store largest number to arr[0]
        {

          if(arr[0]<arr[i])       // Change < to > if you want to find smallest element
             arr[0]=arr[i];
        }
        printf("Largest element = %.2f",arr[0]);

}
```

*Output:*

```
Enter total number of elements (1 to 100): 8
Enter Number 1: 23.4
Enter Number 2: -34.5
Enter Number 3: 50
Enter Number 4: 33.5
Enter Number 5: 55.5
Enter Number 6: 43.7
Enter Number 7: 5.7
Enter Number 8: -66.5
Largest element = 55.5
```

**Program to calculate sum & average of an array**

```
#include <stdio.h>
 void main()
 {
    int array[100];
    int i, num, total = 0,average;
```

```
    printf ("Enter the value of N \n");
    scanf("%d", &num);
    printf("Enter %d numbers \n", num);
    for (i = 0; i < num; i++)
    {
       scanf("%d", &array[i]);

    }
    for (i = 0; i < num; i++)
    {
      total =total + array[i];
    }
    average = total / num;
    printf("Sum of all numbers = %d \n", total);
    printf("Average of all input numbers = %d", average);
}
```

**Output**

Enter the value of N
5
Enter 5 numbers
1 2 3 4 5
Sum of all numbers=15
Average of all input numbers=3

**Program to generate fibonacci series using an array**

```
#include <stdio.h>
 main()
{
    int fib[100],i, n;
    fib[0] = 0;
    fib[1] = 1;
    printf("enter n value: “);
    scanf(“%d”, &n);
    for(i = 2; i < n; i++)
    {
        fib[i] = fib[i-1] + fib[i-2];
    }
```

```
        printf("fibo series is ….. \n");
         for(i = 0; i < n; i++)
         {
             printf("%d\t", fib[i]);
        }
}
```

**Output**:

```
enter n value:7
fibo series is ….. 0 1 1 2 3 5 8
```

### 1.4 SORTING

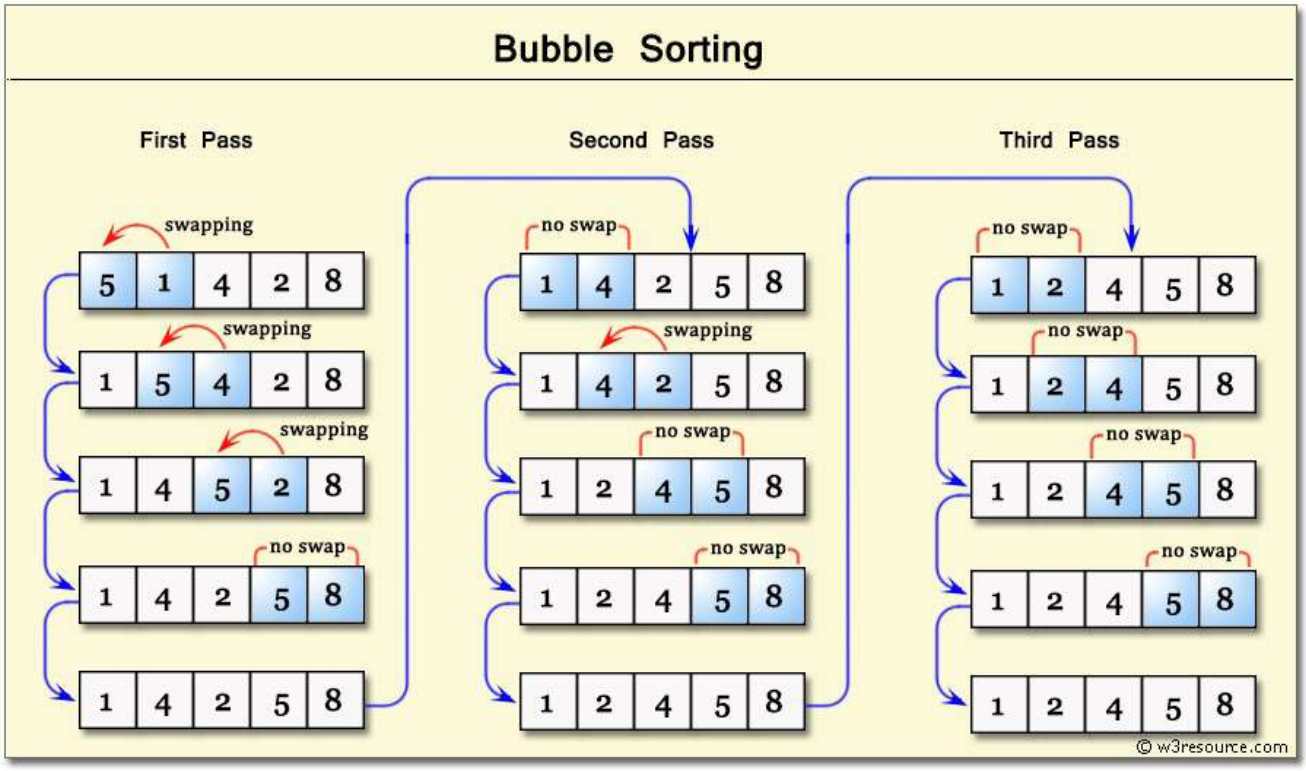
A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

**Bubble Sort**

- Bubble sort may be defined as the sorting algorithm that follows the approach of replacing the value in the first index with the smallest value in the array and keep it repeating until the list is sorted. It is a very simple way of performing sorting. In this way to sort the array, the value has to be assigned to the array in the beginning before starting the sorting.

- Below is the program to sort the array using bubble sort where the values have been taken from the user. Once the program is compiled and run, it will ask the user for the number of elements that they want to sort. Once the number is provided, the program will ask the user to provide values equivalent to the count that they have provided. The values will be stored in the array and will be processed further using nested for loop together with decision making using "if" in order to sort the array.

- The first smallest value found in the array has been moved to the first index of the array and then the search begins again to find the other smallest number. Once the next smallest number is found, it replaces the value in the second index and the process keeps on repeating until the array consists of a sorted list of values.




**Write a program to sort the element in ascending order using Bubble Sort**

```
#include<stdio.h>
void main()
{
int a[100], n, i, j, temp;
printf("Enter number of elements to be sorted\n");
scanf("%d", &n);
printf("Enter  the %d integer elements \n", n);
for (i = 0; i < n; i++)
{
scanf("%d", &a[i]);
}
//sorting
for (i = 0 ; i < n - 1; i++)
{
for (j = 0 ; j < n - i - 1; j++)
{
if (a[j] > a[j+1])
{
temp    = a[j];
a[j]      = a[j+1];
a[j+1] = temp;
}
```

```
}
}
printf("\n Sorted list in ascending order:\n");
for ( i = 0 ; i < n ; i++ )
{
printf("%d\t", a[i]);
}
}
```

**Output**

```
Enter number of elements to be sorted
4
Enter the 4 integer elements
4
3
2
1
Sorted list in ascending order
1       2       3       4
```

**Write a program to sort the element in Ascending and Descending order.**

```
#include<stdio.h>
void main()
{
int A[100], n, i, j, temp;
printf("Enter the number of elements to be sorted\n");
scanf("%d", &n);
printf("Enter %d integers\n", n);
for (i = 0; i < n; i++)
{
scanf("%d", &A[i]);
}
for (i = 0 ; i < n - 1; i++)
{
for (j = 0 ; j < n - i - 1; j++)
{
if (A[j] > A[j+1])
{
temp   = A[j];
A[j]   = A[j+1];
A[j+1] = temp;
}
}
```

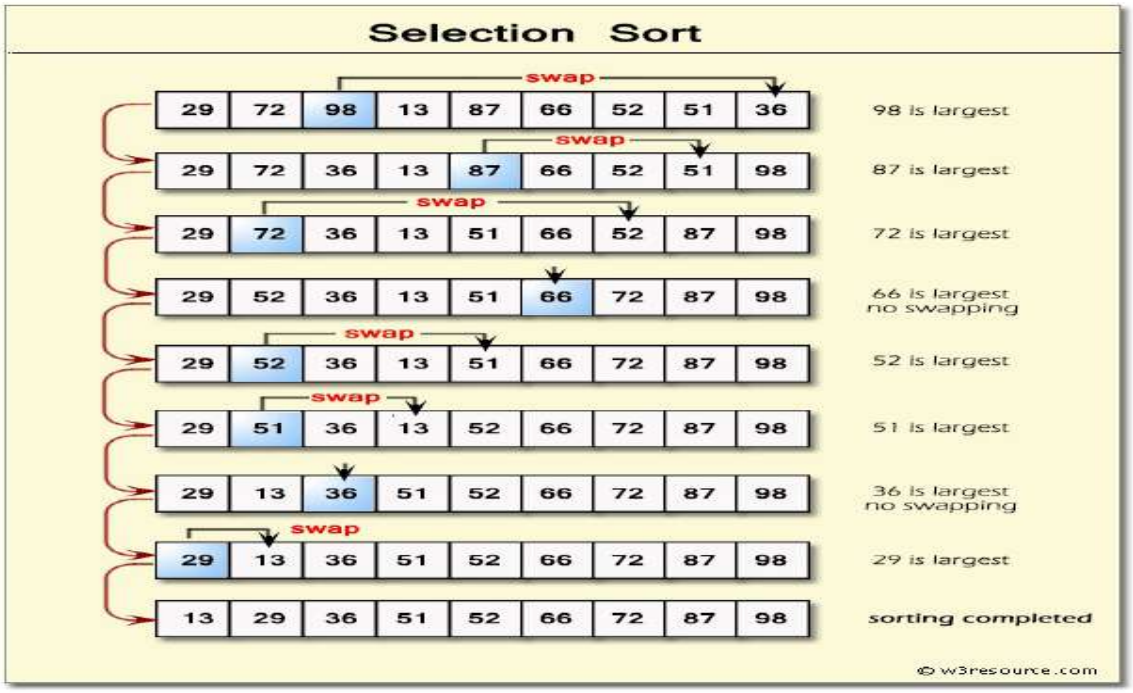
```
}
printf("\n Sorted list in ascending order:\n");
for ( i = 0 ; i < n ; i++ )
{
printf("%d\t", A[i]);
}
printf("\n Sorted list in descending order:\n");
for ( i = n-1 ; i >= 0 ; i-- )
{
printf("%d\t", A[i]);
}
}
```

**Output**

```
Enter number of elements to be sorted
5
Enter the 4 integer elements
4
3
5
2
1
Sorted list in ascending order
1       2       3       4       5

Sorted list in descending order
5       4       3       2       1
```

**Selection Sort**

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.




**//Program to Implement Selection Sort**

```
#include<stdio.h>
void main()
{
   int a[100],n,i,j,min,temp;
   printf("\n Enter the Number of Elements: ");
   scanf("%d",&n);
   printf("\n Enter %d Elements: ",n);
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
  for(i=0;i<n-1;i++)
   {
      min=i;
      for(j=i+1;j<n;j++)
```

```
        {
            if(a[min]>a[j])
            min=j;
        }
        if(min!=i)
        {
            temp=a[i];
            a[i]=a[min];
            a[min]=temp;
        }
    }
    printf("\n The Sorted array in ascending order: ");
    for(i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
}
```

**Output**

```
Enter number of elements to be sorted
4
Enter the 4 elements
4
3
2
1
Sorted array in ascending order
1       2       3       4
```

## 1.5. SEARCHING

### Linear Search

Linear search is to find whether a number is present in an array. If it's present, then at what location it occurs. It is also known as a sequential search. It is straightforward and works as follows: we compare each element with the element to search until we find it or the list ends.

**Program to implement linear search**

```
#include <stdio.h>
void main()
{
 int array[10];
 int i, num, key,found = 0;
 printf("Enter the value of num \n");
 scanf("%d", &num);
 printf("Enter the elements one by one \n");
 for (i = 0; i < num; i++)
 {
    scanf("%d", &array[i]);
 }
 printf("Enter the element to be searched \n");
 scanf("%d", &key);       // Linear search begins
 for (i = 0; i < num ; i++)
 {
    if (key == array[i] )
    {
       found = 1;
       break;
    }
 }
 if (found == 1)
    printf("Element is present in the array at position %d\n",i+1);
 else
    printf("Element is not present in the array\n");
}
```

**Output**

Enter the value of num

5

Enter the elements one by one

23

90

56

15

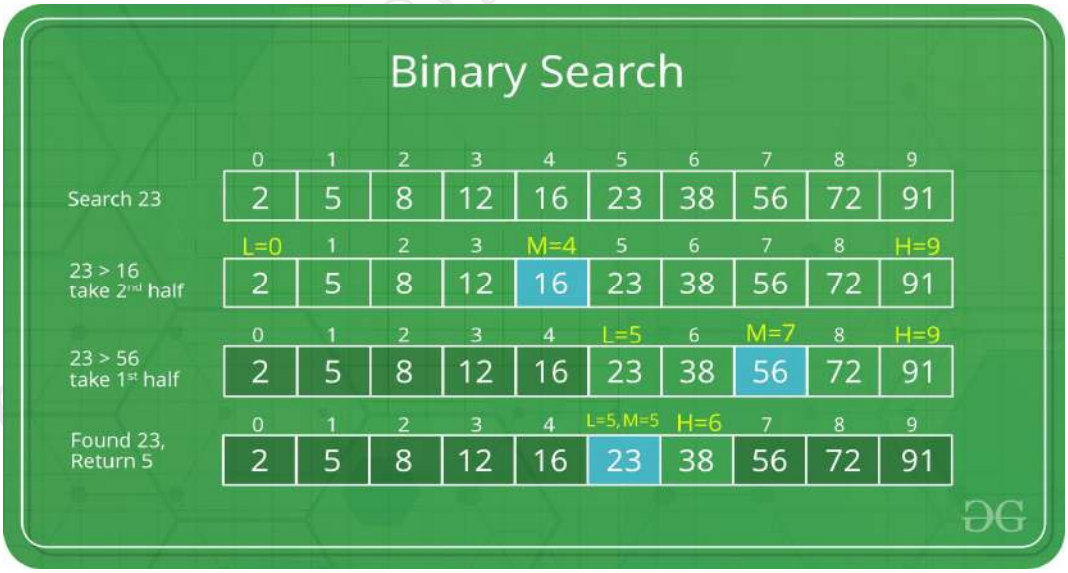
58

Enter the element to be searched 56

Element is present in the array at position 3

**Binary search**

- Binary search in C language to find an element in a sorted array. If the array isn't sorted, you must sort it using a sorting technique such as **bubble sort (refer lab program)**.
- If the element to search is present in the list, then we print its location. The program assumes that the input numbers are in *ascending* order.
- Search a sorted array by repeatedly dividing the search interval in half.
- Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.



**//Program to implement binary search**

```
#include <stdio.h>
 void main()
{
      int array[10];
      int i, j, num, temp, key; int low, mid, high;
      printf("Enter the value of num:");
      scanf("%d", &num);
      printf("Enter the elements in ascending order \n");
      for (i = 0; i < num; i++)
      scanf("%d", &array[i]);
      printf("Enter the element to be searched:");
      scanf("%d", &key);
      low = 1
      do
      {
          mid = (low + high) / 2;
           if (key< array[mid])
               high = mid - 1;
          else if (key > array[mid])
               low = mid + 1;
      } while (key!= array[mid] && low <= high);

      if (key== array[mid])
      printf("SEARCH SUCCESSFUL ");
      else
       printf("SEARCH FAILED ");
}
```

*Output:*

```
Enter the value of num: 5
Enter the elements in ascending order
 23 90 56 15 58
Enter the element to be searched: 58
SEARCH SUCCESSFUL
```

**Program to find the transpose of a given matrix**

```
#include <stdio.h>
 void main()
{
      int array[10][10]; int i, j, m, n;
      printf("Enter the order of the matrix \n");
      scanf("%d %d", &m, &n);
      printf("Enter the coefficients of the matrix\n");
```

```
        for (i = 0; i < m; ++i)
        {
            for (j = 0; j < n; ++j)
            {
                scanf("%d", &array[i][j]);
            }
            printf("\n");
        }

        printf("Transpose of matrix is \n");
        for (j = 0; j < n; ++j)
        {
            for (i = 0; i < m; ++i)
            {
                printf(" %d", array[i][j]);
            }
            printf("\n");
        }
    }
```

*Output:*

```
Enter the order of the matrix 3
3
Enter the coefficients of the matrix
 3 7 9
 2 7 5
 6 3 4

Transpose of matrix is
 3 2 6
 7 7 3
 9 5 4
```

**Program to calculate the addition of 2 matrices**

```
#include <stdio.h>
void main()
{
    int array1[10][10], array2[10][10], arraysum[10][10];
    int i, j, m, n, option;
    printf("Enter the order of the matrix array1 and array2 \n");
    scanf("%d %d", &m, &n);
    printf("Enter the elements of matrix array1 \n");
     for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &array1[i][j]);
        }
    }
```

```
        printf("Enter the elements of matrix array2 \n");

        for (i = 0; i < m; i++)
        {
            for (j = 0; j < n; j++)
            {
                scanf("%d", &array2[i][j]);
            }
        }

            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                {
                    arraysum[i][j] = array1[i][j] + array2[i][j];
                }
            }
            printf("Sum matrix is \n");
            for (i = 0; i < m; i++)
            {
                for (j = 0; j < n; j++)
                {
                    printf("%3d", arraysum[i][j]) ;
                }
                printf("\n");
            }

}
```

**Write a program to copy one array element to another <u>(parallel array)</u>**

```
#include<stdio.h>
main()
{
float a[5] = { 5.90, 6.45, 2.56,-9.86, 7.32};
float b[5];
int i;
printf(" elements of array a:\n");
for(i=0;i<5;i++)
printf("\t %.3f", a[i]);
for(i=0; i<5; i++)
b[i] = a[i];                    /*parallel array */
printf(" elements of array b:\n");
for(i=0;i<5;i++)
printf("\t%.3f", b[i]);
}
```

**Write C program to find the trace and normal of a matrix**

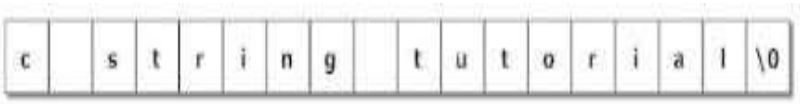
Trace is defined as the sum of main diagonal elements and Normal is defined as square root of the sum of all the elements

```
#include <stdio.h>
#include <math.h>
void main ()
{
    int A[10][10];
    int i, j, n, sum = 0, trace=0;
    float normal;
    printf("Enter the order of the matrix\n"); scanf("%d",&n);
    printf("Enter the n coefficients of the matrix \n"); for (i = 0; i < n; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            scanf("%d", &A[i][j]);
            sum = sum + A[i][j] * A[i][j];
        }
    }
    normal = sqrt(sum);
    printf("The normal of the given matrix is = %.2f\n", normal);
    for (i = 0; i < n; ++i)
    {
        trace = trace + A[i][i];
    }
    printf("Trace of the matrix is = %d\n", trace);
}
```

# Chapter 2

## STRINGS

- Array of character are called strings.
- A string is terminated by '/0'(NULL Character).
- For ex:
  "c string tutorial"
- The above string can be pictorially represented as shown below:




**STRING VARIABLE**

- There is no separate data type for strings in C.
- They are treated as arrays of type "char".
- So, a variable which is used to store an array of characters is called a string variable.

**Declaration of String**

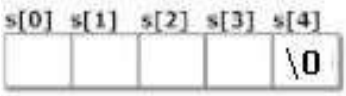
- Strings are declared in C in similar manner as arrays. Only difference is that, strings are of "char" type. a character array or a string is declared as follows:

  **char variable_name[array_length];**

  For ex:

  char s[5]; //string variable name can hold maximum of 5 characters including NULL character

  The above code can be pictorially represented as shown below:




**Initialization of String**

- When you initialize a string variable by assigning character constants individually, use single quotes ' ',
- For ex:

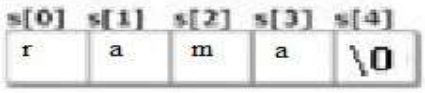
  char s[5] = {'r', 'a', 'm', 'a' };

- When we initialize a string variable by assigning all the characters collectively, we use double quotes **" "**,.

  For ex:

  char s[9] = {"rama"};

- The above code can be pictorially represented as shown below:




- Example: Program to illustrate the initialization of string.

```
#include<stdio.h>
void main()
{
        char s[5]={'r','a','m','a'};    //or char s[5]="rama";
        printf("Value in array s[0] : %c \n", s[0]);
        printf("Value in array s[1] : %c \n", s[1]);
        printf("Value in array s[2] : %c \n", s[2]);
        printf("Value in array s[3] : %c ", s[3]);
}
```

*Output:*

```
Value in array s[0] : r
Value in array s[1] : a
Value in array s[2] : m
Value in array s[3] : a
```

## Reading & Printing Strings

### 1) Using Formatted input output function: (scanf and printf):

- The strings can be read from the keyboard and can be displayed onto the monitor using following 2 formatted functions:

  - Formatted input function: scanf()
  - Formatted output function: printf()

- The scanf( ) function reads strings from the keyboard and stores them in variables by using %s format specifier.
- scanf( ) function read strings which do not have any white spaces, that is, a white space or a blank space in the string will terminate the reading of a string. To read a string with white space **gets( )** function is used.

- **scanf: (scanformat)**
  → **%s** is the format string/ control string for string.
  → %s removes all white spaces present before the string is removed.

- The printf( ) function along with the %s format specifier prints a string stored in character array to the console.
- We can use the printf( ) function to display the string data with the help of two ways. Either we can pass the string data directly within the printf( ) function or we can store the string data in a character array.

Example: **printf**("Test string");
Or
Example: char str[ ] = {"Test string"};
**printf**(%s", str);

**Example: Program to illustrate the use of scanf() and printf().**

```
#include<stdio.h>
void main()
{
        char name[10];
        printf("enter your name: \n");
        scanf("%s", name);
        printf("welcome: ");
        printf("%s", name);
}
```

*Output:*

```
enter your name:
canara
welcome: canara
```

## 2) Using unformatted INPUT/OUTPUT FUNCTIONS: gets, puts

• The strings can be read from the keyboard and can be displayed onto the monitor using following 2 unformatted functions:

- Unformatted input function: gets( )
- Unformatted output function: puts( )

• The gets( ) function reads a string from keyboard. This function stops reading the string only when we press the Enter key from the keyboard.

• The gets( ) function is similar to the scanf( ) function. The difference between the gets( ) and scanf( ) functions is that the gets( ) function can read the whole sentence, which is a combination of multiple words; while the scanf( ) function can read the characters until a space or a newline character is encountered.

• The puts( ) function, on the other hand, prints a string or a value stored in a variable to the console in the next line.

**Example: Program to illustrate the use of gets() and puts().**

```
#include<stdio.h>
void main()
{
        char name[10];
        printf("enter your name: \n");
        gets(name);                    //same as scanf("%s", name);
        printf("welcome: ");
        puts(name);                    //same as printf("%s", name);
}
```

*Output:*

```
enter your name:
canara
welcome: canara
```

## 3) Using the getchar( ) and putchar( ) Functions

- The getchar( ) function, is used to read a single character from the standard input device. getchar( ) returns the character it reads, or, the special value EOF (end of file), if there are no more characters available.
- The putchar( ) function writes a character to the standard output device.

Program using getchar( ) and putchar( ) Function.

```
#include<stdio.h>
main( )
{
char ch;
printf("Enter a word");
ch=getchar( );
putchar(ch);
}
```

**Output:**

```
Enter a word
String
S
```

## Creating an Array of Strings

- An array of strings that is known as two-dimensional character array, which consists of strings as its individual elements.
- The order of the subscripts in the array declaration is important. The first subscript gives the number of strings in the array, while the second subscript gives the length of each string in the array.

Syntax:

**char arr_name[row][col];**

→ col indicates maximum length of each string.
→ row indicates number of string

**Example**:

char alpha[3][5] = {“pawan”, “pooja”, “punya”} ;

- The names would be stored in the memory as shown in the below Figure. Note that each string ends with a “\0”. The arrangement is similar to that of a two-dimensional numeric array.

char alpha[3][5] = { "PAWAN", "POOJA" , "PUNYA" };

| | alpha[ ][0] | alpha[ ][1] | alpha[ ][2] | alpha[ ][3 ] | alpha[ ][4] |
|---|---|---|---|---|---|
| alpha[0] | P | A | W | A | N |
| alpha[ 1] | P | O | O | J | A |
| alpha[ 2] | P | U | N | Y | A |

**Program: Creating and Displaying an Array of String**

```
#include<stdio.h>
main()
{
        int i;
        char alpha[3][5] =  {“Pawan”, “Pooja”, “Punya”} ;
        for(i=0;i<3;i++)
        {
        printf("%s \t",alpha[i]);
        }
}
```

**Output:** Pawan Pooja Punya

## STRING MANIPULATION FUNCTIONS FROM THE STANDARD LIBRARY

- Strings are often needed to be manipulated by programmer according to the need of a problem.
- All string manipulation can be done manually by the programmer but, this makes programming complex and large.
- To solve this, the C supports a large number of string handling functions.
- There are numerous functions defined in <string.h> header file.

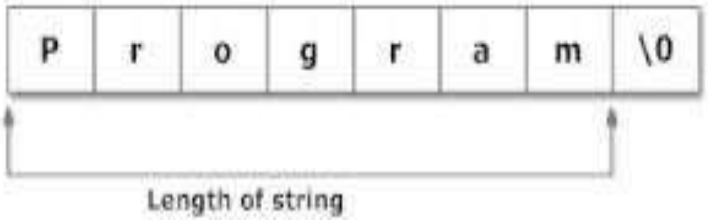
| S.N. | Function & Purpose |
|---|---|
| 1 | **strcpy(s1, s2);**<br>Copies string s2 into string s1. |
| 2 | **strcat(s1, s2);**<br>Concatenates string s2 onto the end of string s1. |
| 3 | **strlen(s1);**<br>Returns the length of string s1. |
| 4 | **strcmp(s1, s2);**<br>Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5 | **strchr(s1, ch);**<br>Returns a pointer to the first occurrence of character ch in string s1. |
| 6 | **strstr(s1, s2);**<br>Returns a pointer to the first occurrence of string s2 in string s1. |

- **strlen()**
  - This function calculates the length of string. It takes only one argument, i.e., string-name.
  - The syntax is shown below:

**temp_variable = strlen(string_name);**

```
char c[]={'P', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};
temp=strlen(c);
```

Then, temp will be equal to 7 because, null character '\0' is not counted.



• Example: Program to illustrate the use of strlen().

```
#include<string.h>
#include<stdio.h>
void main()
{
   char c[20];
   int len;
   printf("Enter string whose length is to be found:");
   gets(c);
   len=strlen(c);
   printf("\n Length of the string is %d ", len);
}
```

*Output:*

```
Enter string whose length is to be found: Program
Length of the string is 7
```

- **strcpy()**
  - This function copies the content of one string to the content of another string. It takes 2 arguments.
  - The syntax is shown below:

    **strcpy(destination,source);**

    where source and destination are the name of the strings.

  - Example: Program to illustrate the use of strcpy().

```
#include<string.h>
#include<stdio.h>
void main()
{
   char      src[20],dest[20];
   printf("Enter   string:   ");
   gets(src);
   strcpy(dest, src);    //Content of string src is copied to string dest
   printf("Copied string: ");
   puts(dest);
}
```

*Output:*

```
Enter string: canaracollege
Copied string: canaracollege
```

- **strcat()**

  - This function joins 2 strings. It takes two arguments, i.e., 2 strings and resultant string is stored in the first string specified in the argument.

  - The syntax is shown below:

    **strcat(first_string,second_string);**

  - Example: Program to illustrate the use of strcat().

```
#include <stdio.h>
#include <string.h>
void main()
{
  char str1[10], str2[10};
  printf("Enter First String:");
  gets(str1);
  printf("\n Enter Second String:");
  gets(str2);
  strcat(str1,str2);        //concatenates str1 and str2
  printf("\n Concatenated String is ");
  puts(str1);               //resultant string is stored in str1
}
```

*Output:*

```
Enter First String: Abdul
Enter Second String: Khalam
Concatenated String is  AbdulKhalam
```

- **strcmp()**

  - This function compares 2 string and returns value 0, if the 2 strings are equal. It takes 2 arguments, i.e., name of two string to compare.

  - The syntax is shown below:

    **temp_variable=strcmp(string1,string2);**

Example: Program to illustrate the use of strcmp().

```
#include <string.h>
#include<stdio.h>
void main()
{
 char str1[30],str2[30];
 printf("Enter first string: ");
 gets(str1);
 printf("Enter second string: ");
 gets(str2);
```

```
  if(strcmp(str1,str2)==0)
      printf("Both strings are equal");
  else
      printf("Strings are unequal");
}
```

*Output:*

```
Enter first string: Asection
Enter second string: Asection
Both strings are equal
```

**SAMPLE PROGRAMS WITHOUT USING BUILT IN FUNCTIONS**

**Program to find length of a string without using strlen( ).**

```
#include<stdio.h>
main()
{
   char str1[25];
   int len=0;
   printf("Enter string whose length is to be found:");
   gets(str1);
   while(str1[len]!='\0')
       len++;                    //here the length of string is calculated.
   printf("Length of the string is %d", len);
}
```

*Output:*

```
Enter string whose length is to be found: Fsection
Length of the string is 8
```

**Program to concatenate two strings without using strcat( ).**

```
 #include<stdio.h>
  main()
{
   char str1[25],str2[25];
   int i=0,j=0;
   printf(" Enter First String:");
   gets(str1);
   printf("\n Enter Second String:");
   gets(str2);
   while(str1[i]!='\0')
       i++;
   while(str2[j]!='\0')
   {
      str1[i]=str2[j];
      j++;
      i++;
   }
```

```
   str1[i]='\0';
   printf("\n Concatenated String is ");
   puts(str1);
}
```

*Output:*

```
Enter First String: Canara
Enter Second String: College
Concatenated String is CanaraCollege
```

**Program to copy one string into other without using strcpy( ).**

```
#include<stdio.h>
void main()
{
   char src[100], dest[100];
   int i;
   printf("Enter string:");
   gets(src);
   i = 0;
   while (src[i] != '\0')
  {
      dest[i] = src[i];
      i++;
   }
   dest[i] = '\0';
   printf("Copied String ; %s ", dest);
}
```

*Output:*

```
Enter string : Fsection
Copied String : Fsection
```

**Program to compare 2 strings without using strcmp( ).**

```
#include<stdio.h>
void main()
{
    char str1[100],str2[100];
    int i=0,flag=0;
    printf("Enter first string: ");
    scanf("%s",str1);
    printf("Enter second string: ");
    scanf("%s",str2);
    while(str1[i]!='\0' && str2[i]!='\0')
    {
        if(str1[i]!=str2[i])
       {
            flag=1;
            break;
```

```
            }
            i++;
        }
        if (flag==0 && str1[i]=='\0' && str2[i]=='\0')
            printf("Both strings are equal");
        else
            printf("Both strings are not equal");

    }
```

*Output:*

```
Enter first string: Rama
Enter second string: Rama
Both strings are equal
```

- **strncpy( )**
  - This function allows us to copy a block of "N" character from one string to another.

    Syntax: **strncpy(target, source, num_of_char);**
  - Example program

```
#include<stdio.h>
#include<string.h>
void main()
{
 char str2[25] = "Hello CEC";
char str1[25];
strncpy(str1,str2,5);
printf("str1= %s\n", str1);
printf("str2= %s\n", str2);
}
```

Output:

```
Str1= Hello
Str2= Hello CEC
```

- **strncmp( )**
  - This function allows us to compare a block of „n" characters from one string with those in another.

    Syntax: **strncmp(string1, string2, num_of_char);**
  - The function returns zero if first n chars of strings are equal. Otherwise, it returns a value less than zero or greater than zero if string1 is less than or greater than string2, respectively.

- Example program

```
#include<stdio.h>
#include<string.h>
void main()
{
        char str1[10] = "hello";
        char str2[10] = "hey";
        if(strncmp(str1,str2,2) == 0)
        printf("First 2 characters are identical");
        else
         printf("First 2 characters are not identical");
}
```

## Additional String Manipulation/Handling Functions:

| Function | Purpose | Example | Result |
|---|---|---|---|
| strupr( ) | To convert all alphabets in a string to upper case letters. | Strupr("Delhi") | "DELHI" |
| strlwr( ) | To convert all alphabets in a string to lower case letters. | strlwr("CITY") | "city" |
| strrev( ) | To reverse a string | strrev("SACHIN") | "NIHCAS" |
| strncmp( ) | To compare the first n characters of two strings. | m=strncmp("DELHI", "DIDAR",2) | m = -4 |
| strcmpi( ) | To compare two strings with case insensitive(neglecting upper/lower case) | m = strcmpi("DELHI", "Delhi"); | m = 0 |
| strncat( ) | To join specific number of letters to another string. | char s1[10] = "New"; char s2[10] = "Delhi"; strncat(s1,s2,3); | S1 will be "Newdel" |

**Question Bank**

1. Define array. Write syntax for declaring and initialization of one and two dimensional arrays with example.
2. Define string. List and explain all string manipulation functions.
3. Write a c program to search a key integer element in the given array of N elements using binary search technique. Print the output with suitable headings.
4. Write a C program to find the transpose of a given matrix.
5. Define searching. Explain linear search algorithm and write a program to implement the same.
6. Write a C program to sort the elements of a given array using bubble sort.
7. Write a c program to accept a sentence from keyboard and count the no of vowels and consonants.