# MODULE 3

# Arrays

## INTRODUCTION

**Arrays**: Array is a sequential collection of similar data items.

Pictorial representation of an array of 5 integers

| 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|
| A[0] | A[1] | A[2] | A[3] | A[4] |

- ➢ An array is a collection of similar data items.
- ➢ All the elements of the array share a common name .
- ➢ Each element in the array can be accessed by the subscript(or index) and array name.
- ➢ The arrays are classified as:
  1. **Single dimensional array**
  2. **Multidimensional array.**

## <u>Single Dimensional Array.</u>

- ➢ A single dimensional array is a linear list of related data items of same data type.
- ➢ In memory, all the data items are stored in contiguous memory locations.

**<u>Declaration of one-dimensional array(Single dimensional array)</u>**

**<u>Syntax:</u>**

```
datatype  array_name[size];
```

- ➢ **datatype** can be int,float,char,double.
- ➢ **array_name** is the name of the array and it should be an valid identifier.
- ➢ **Size** is the total number of elements in array.

**For example**:

int  a[5];

The above statement allocates 5*2=10 Bytes of memory for the array **a.**

| | | | | |
|---|---|---|---|---|
| | | | | |
| a[0] | a[1] | a[2] | a[3] | a[4] |

float b[5];

The above statement allocatests 5*4=20 Bytes of memory for the array **b.**

- ➢ Each element in the array is identified using integer number called as i**ndex.**
- ➢ If n is the size of array, the array index starts from **0** and ends at **n-1.**

**<u>Storing Values in Arrays</u>**

- ➢ Declaration of arrays only allocates memory space for array. But array elements are not initialized and hence values has to be stored.
- ➢ Therefore to store the values in array, there are 3 methods
  1. Initialization
  2. Assigning Values
  3. Input values from keyboard through **scanf()**

**<u>Initialization of one-dimensional array</u>**

- ➢ **Assigning the required values to an array elements before processing is called initialization.**

```
data type array_name[expression]={v1,v2,v3…,vn};
```

Where

- ✓ datatype can be char,int,float,double
- ✓ array name is the valid identifier
- ✓ size is the number of elements in array
- ✓ v1,v2,v3…......vn are values to be assigned.

- ➢ Arrays can be initialized at declaration time.

  Example:

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

| | | | | |
|---|---|---|---|---|
| 2 | 4 | 34 | 3 | 4 |
| a[0] | a[1] | a[2] | a[3] | a[4] |

- ➢ The various ways of initializing arrays are as follows:
  1. **Initializing all elements of array(Complete array initialization)**
  2. **Partial array initialization**

3. **Initialization without size**
4. **String initialization**

1. **Initializing all elements of array:**

- Arrays can be initialized at the time of declaration when their initial values are known in advance.
- In this type of array initialization, initialize all the elements of specified memory size.
- Example:

**int a[5]={10,20,30,40,50};**

| | | | | |
|---|---|---|---|---|
| **10** | **20** | **30** | **40** | **50** |

2. **Partial array initialization**
   - If the number of values to be initialized is less than the size of array then it is called as partial array initialization**.**
   - In such a case elements are initialized in the order from 0th element.
   - The remaining elements will be initialized to **zero automatically by the compiler.**
   - Example:
     **int a[5]={10,20};**

| | | | | |
|---|---|---|---|---|
| **10** | **20** | **0** | **0** | **0** |

3. **Initialization without size**

- In the declaration the array size will be set to the total number of initial values specified.
- The compiler will set the size based on the number of initial values.
- Example:

**int a[ ]={10,20,30,40,50};**

- **In the above example the size of an array is set to 5**

4. **String Initialization**

- Sequence of characters enclosed within double quotes is called as string.
- The string always ends with NULL character**(\0)**

```
char s[5]=”SVIT”;
```

We can observe that string length is 4,but size is 5 because to store NULL character we need one more location.

So pictorial representation of an array **s** is as follows:

| S | V | I | T | \0 |
|---|---|---|---|---|
| S[0] | S[1] | S[2] | S[3] | S[4] |

**<u>3.1.2 Assigning values to arrays</u>**

Using assignment operators, we can assign values to individual elements of arrays.
For example:

```
        int  a[3];
                a[0]=10;
                a[1]=20;
                a[2]=30;
```

| 10 | 20 | 30 |
|---|---|---|
| a[0] | a[1] | a[2] |

**<u>Reading and writing single dimensional arrays.</u>**

To read array elements from keyboard we can use **scanf()** function as follows:

To read **0th** element: scanf("%d",&a[0]);
To read **1st** element: scanf("%d",&a[1]);
To read **2nd** element: scanf("%d",&a[2]);
……
…….
To read **n th** element : scanf("%d",&a[n-1]);
**In general**
To read **i th** element:
**scanf("%d",&a[i]);** where i=0; i<n; i++

To print array elements we can use **printf()** function as follows:

To print **0th** element: printf("%d",a[0]);
To print **1st** element: printf("%d",a[1]);
To print **2nd** element :printf("%d",a[2]);
……..
……..

To **n th** element : printf("%d",&a[n-1]);
**In general**
To read **i th** element:
**printf("%d",a[i]); where i=0; i<n; i++**

1. Write a C program to read N elements from keyboard and to print N elements on screen.

```
/* program to read N elements from keyboard and to print N elements on
screen */
#include<stdio.h>
void main()
{
        int i,n,a[10];
        printf("enter number of array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }

        Printf("array elements are\n"):
        for(i=0; i<n;i++)
        {
                printf("%d",a[i]);
        }
}
```

2. Write a C program to find sum of n array elements .

```
/* program to find the sum of n array elements.*/
#include<stdio.h>
void main()
{
        int i,n,a[10],sum=0;
        printf("enter number of array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n; i++)
        {
                scanf("%d",&a[i]);
        }

        for(i=0; i<n;i++)
        {
                sum=sum+ a[i];
        }
        printf("sum is %d\n",sum):

}
```

3. Write a c program to find largest of n elements stored in an array a.

```
#include<stdio.h>
void main()
{
        int i,n,a[10],big;
        printf("enter number of array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        big=a[0];
        for(i=0; i<n;i++)
        {
                if(a[i]>big)
                 big=a[i];
        }
        printf("the biggest element in an array is %d\n",big);
}
```

4. **Write a C program to generate Fibonacci numbers using arrays.**

```
#include<stdio.h>
void main()
{
        int i,n,a[10];
        a[0]=0;
        a[1]=1;
        printf("enter n\n");
        scanf("%d",&n);
        if(n==1)
        {
                printf("%d\t",a[0]);
        }
        if(n==2)
        {
                printf("%d\t %d\t",a[0],a[1]);
        }
        if(n>2)
```

| | |
|---|---|
| | ```<br>        {<br>                printf("%d \t %d\t",a[0],a[1]);<br>                for(i=2;i<n;i++)<br>                {<br>                        a[i]=a[i-1]+a[i-2];<br>                        printf("%d\t",a[i]);<br>                }<br>        }<br>}<br>``` |

## Two Dimensional arrays:

- In two dimensional arrays, elements will be arranged in rows and columns.
- To identify two dimensional arrays we will use two indices(say i and j) where I index indicates row number and j index indicates column number.

### Declaration of two dimensional array:

**data_type array_name[exp1][exp2];**
**Or**
**data_type**
**array_name[row_size][column_size];**

- **data_type** can be int,float,char,double.
- **array_name** is the name of the array.
- **exp1 and exp2** indicates number of rows and columns

**For example**:
**int a[2][3];**

✓ The above statements allocates memory for 3*4=12 elements i.e 12*2=24 bytes.

### Initialization of two dimensional array

Assigning or providing the required values to a variable before processing is called initialization.

```
Data_type array_name[exp1][exp2]={

                                        {a1,a2,......an}
                                        {b1,b2,.....bn}
                                        .............
                                        {z1,z2........zn}

                              }
```

- Data type can be int,float etc.
- exp1 and exp2 are enclosed within square brackets .
- both exp1 and exp2 can be integer constants or constant integer expressions(number of rows and number of columns).
- a1 to an are the values assigned to 1st row ,
- b1 to bn are the values assigned to 2nd row and so on.
- Example:

```
int a[3][3]={
              {10,20,30},
              {40,50,60},
              {70,80,90}
          };
```

| | | |
|---|---|---|
| **10** | **20** | **30** |
| **40** | **50** | **60** |
| **70** | **80** | **90** |

## Partial Array Initialization

- If the number of values to be initialized is less than the size of array, then the elements are initialized from left to right one after the other.
- The remaining locations initialized to zero automatically.
- Example:

```
int a[3][3]={
              {10,20},
              {40,50},
              {70,80}
          };
```

| | | |
|---|---|---|
| **10** | **20** | **0** |
| **40** | **50** | **0** |
| **70** | **80** | **0** |

1. **Write a c program to read & print 2d array as a Array.**

```
#include<stdio.h>
void main()
{
        int m,n,i,j,a[3][3];
        printf("enter number of rows and columns\n");
        scanf("%d %d",&m,&n);
        printf("eneter array elements\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("array elements are\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        printf("%d",a[i][j]);
                }
                printf("\n");
        }
}
```

2 **Write a c program to add two matrices.**

```
#include<stdio.h>
void main()
{
        int m,n,i,j,a[3][3],b[3][3] ,c[3][3];
        printf("enter number of rows and columns\n");
        scanf("%d %d",&m,&n);
        printf("enter array a elements\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
```

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

    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            c[i][j]=a[i][j]+b[i][j];
        }
    }
    printf(“resultant matrix c is \n”);

    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            printf(“%d\t”,c[i][j]);
        }
        printf(“\n”);
    }
}
```

**3** **Write a c program to copy one 2d array in to another 2d array**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int m,n,i,j,a[3][3],b[3][3];
    clrscr();
    printf(“enter number of rows and columns\n”);
    scanf(“%d %d”,&m,&n);
    printf(“enter array a elements\n”);
    for(i=0; i<m; i++)
    {
        for(j=0; j<n; j++)
        {
            scanf(“%d”, &a[i][j]);
        }
    }

    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
```

```
                {
                    b[i][j]=a[i][j];
                }
            }
            printf("matrix b is \n");

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

4 **Write a c program to find biggest element in a matrix or 2D array.**

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

            int m,n,i,j,a[3][3];
            clrscr();
            printf("enter number of rows and columns\n");
            scanf("%d %d",&m,&n);
            printf("enter array elements\n");
            for(i=0;i<m;i++)
            {
                for(j=0;j<n;j++)
                {
                    scanf("%d",&a[i][j]);
                }
            }
            big=a[0][0];
            for(i=0;i<m;i++)
            {
                for(j=0;j<n;j++)
                {
                    if(big>a[i][j])
                    big=a[i][j];
                }
            }
            printf("big is %",big);
}
```

| 5 | **Write a C program to implement Matrix Multiplication** |
|---|---|
| | see code below |

```
#include<stdio.h>
void main()
{
        int m,n,i,j,sum,p,q,k,a[3][3],b[3][3],c[3][3];
        printf("enter number of rows and columns of matrix a \n");
        scanf("%d %d",&m,&n);
        printf("enter number of rows and columns of matrix b \n");
        scanf("%d %d",&p,&q);
        if(n!=p)
        {
                printf("multiplication not possible\n"):
                exit(0);
        }

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

        printf("enter array b elements\n");
        for(i=0;i<p;i++)
        {
                for(j=0;j<q;j++)
                {
                        scanf("%d",&b[i][j]);
                }
        }

        for(i=0;i<m;i++)
        {
                for(j=0;j<q;j++)
                {
                        {
                                c[i][j]=0;
                                for(k=0;k<n;k++)
                                {
                                        c[i][j]= c[i][j]+a[i][k]*b[k][j];
                                }

                }
        }
        printf("resultant matrix a is \n");

        for(i=0;i<m;i++)
```

```
    {
        for(j=0;j<n;j++)
        {
            printf("%d\t",a[i][j]);
        }
        printf("\n");
    }
    printf("resultant matrix a is \n");

    for(i=0;i<p;i++)
    {
        for(j=0;j<q;j++)
        {
            printf("%d\t",b[i][j]);
        }
        printf("\n");
    }
printf("resultant matrix a is \n");

    for(i=0;i<m;i++)
    {
        for(j=0;j<q;j++)
        {
            printf("%d\t",c[i][j]);
        }
        printf("\n");
    }

}
```

6 **Write a program to find sum of each row and sum of each column**

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

    int m,n,i,j,rsum,csum,a[3][3];
    printf("enter number of rows and columns\n");
    scanf("%d %d",&m,&n);
    printf("enter array elements\n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    for(i=0;i<m;i++)
    {
        rsum=0;
        for(j=0;j<n;j++)
        {
            rsum=rsum+a[i][j];
        }
        printf("sum is %d",rsum);
    }

    for(i=0;i<m;i++)
    {
        csum=0;
        for(j=0;j<n;j++)
        {
            csum=csum+a[i][j];
        }
        printf("sum is %d",csum);
    }

}
```

| | |
|---|---|
| **7** | **Write a C program to add all 2D array elements** |

```
#include<stdio.h>
void main()
{
    int m,n,i,j,sum=0,a[3][3];
    printf("enter number of rows and columns\n");
    scanf("%d %d",&m,&n);
    printf("enter array elements\n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    for(i=0;i<m;i++)
    {

        sum=sum+a[i][j];

    }
    printf("sum is %d",rsum);
}
```

## Searching

- The process of finding a particular item in the large amount of data is called searching.
- The element to be searched is called key element.

There are two methods of searching:

1] Linear search.

2] Binary search.

### 1] Linear Search:

- Linear search also called sequential search is a simple searching technique.
- In this technique we search for a given key item in linear order i.e,one after the other from first element to last element.
- The search may be successful or unsuccessful.
- If key item is present, the search is successful, otherwise unsuccessful search.

| 1. | **Program to implement linear search.** |
|---|---|

```
#include<stdio.h>
void main()
{
        int i,n,a[10],key;
        clrscr( );
        printf("enter array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("enter the key element\n");
        scanf("%d",,&key);

        for(i=0; i<n;i++)
        {if(key==a[i])
```

| | |
|---|---|
| | <pre>                    {<br>                              printf(“successful search\n”);<br>                              exit(0);<br>                    }<br>          }<br>          printf(“unsuccessful search\n”);<br>}</pre> |

**Advantages of linear search**

- Very simple Approach.
- Works well for small arrays.
- Used to search when elements are not sorted.

**Disadvantages of linear search**

- Less efficient if array size is large
- If the elements are already sorted, linear search is not efficient.

**2] <u>Binary Search:</u>**

- Binary search is a simple and very efficient searching technique which can be applied if the items are arranged in either ascending or descending order.
- In binary search first element is considered as low and last element is considered as high.
- Position of middle element is found by taking first and last element is as follows.

  mid=(low+high)/2

- Mid element is compared with key element, if they are same, the search is successful.
- Otherwise if key element is less than middle element then searching continues in left part of the array.
- If key element is greater than middle element then searching continues in right part of the array.
- The procedure is repeated till key item is found or key item is not found.

| | **write a C program to perform binary search on the array of integers** |
|---|---|
| | <pre>/* C program to search a name in a list of names using Binary<br>searching technique*/<br>          #include<stdio.h><br>          void main()<br>          {<br>                    int i, n, low, high, mid,a[50],key;<br>                    printf(“enter the number of elements\n”):<br>                    scanf(“%d”,&n);<br>                    printf(“enter the elements\n”);<br>                    for(i=0;i<n;i++)<br>                    {<br>                              Scanf(“%d”,&a[i]);<br>                    }<br><br>                    printf(“enter the key element to be searched\n”);<br>                    scanf(“%d”,&key);</pre> |

```
                  low=0;
                  high=n-1;
                  while(low<=high)
                  {
                          mid=(low+high)/2;
                          if(key==a[mid])
                          {
                                  printf("successful search\n");
                                  exit(0);
                          }
                          if(key<a[mid])
                          {
                                  high=mid-1;
                          }
                          else
                          {
                                  low=mid+1;
                          }
                  }
                  printf("unsuccesfull seasrch\n");
}
```

**Advantages of binary search**

1. Simple technique
2. Very efficient searching technique

Disadvantages

1. The elements should be sorted.
2. It is necessary to obtain the middle element, which are stored in array. If the elements are stored in linked list, this method cannot be used.

## Sorting

- The process of arranging elements in either ascending order or descending order is called Sorting.

### Bubble Sort

- This is the simplest and easiest sorting technique.
- In this technique two successive elements of an array such as a[j] and a[j+1] are compared.
- If a[j]>=a[j+1] the they are exchanged, this process repeats till all elements of an array are arranged in ascending order.
- After each pass the largest element in the array is sinks at the bottom and the smallest element in the array is bubble towards top. So this sorting technique is also called as sinking sort and bubble sort.

Write a C program to sort numbers in ascending order using Bubble Sort

```
#include<stdio.h>
void main()
{
        int i,j,n,a[50],temp;
        printf("enter n:\n");
        scanf("%d",&n);
        printf("enter array efficient \n");
        for(i=0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        for(i=0; i<n-1;i++)
        {
                for(j=0; j<n-1; j++)
                {
                        if (a[ j ]>a[ j+1] )
                        {
                                temp= a[ j ];
                                a[ j ]= a[ j+1 ];
                                a[ j+1 ]=temp;
                        }
                }
        }

printf("\n the sorted numbers are:\n");
for(i=0;i<n;i++)
{
        printf("%d\t",a[i]);
}

getch();
}
```

for descending order sorting give a[j]<a[j+1]

## Selection Sort

In Selection sort, the smallest element is exchanged with the first element of the unsorted list of elements (the exchanged element takes the place where smallest element is initially placed). Then the second smallest element is exchanged with the second element of the unsorted list of elements and so on until all the elements are sorted. In the following C program we have implemented the same logic.

Before going through the program, lets see the steps of selection sort with the help of an example:
Entered elements: 22 0 -90 89 17
Step 1: -90 0 22 89 17 (22 and -90 exchanged position)
Step 2: -90 0 22 89 17 (0 is at right place, no exchange needed)
Step 3: -90 0 17 89 22 (22 and 17 exchanged position)
Step 4: -90 0 17 22 89 (89 and 22 exchanged position)

### Selection sort Program

```
  #include<stdio.h>
      void main()
 {
         int i,j,n,a[20],temp,min;
         printf("enter n:\n");
         scanf("%d",&n);
         printf("enter array efficient \n");
         for(i=0; i<n;i++)
         {
                 scanf("%d",&a[i]);
         }

         for(i=0; i<n-1;i++)
         {
                min=i;
         //finding min value starting from index i+1
                 for(j=i+1; j<n; j++)
                {
                        if (a[ j ]>a[ min] )
                        {
                                min=j;
                        }
                 }
        //starting 1st element of unsorted part with min value
                temp= a[ j ];
                a[ j ]= a[ j+1 ];
                a[ j+1 ]=temp;

        }
        printf("\n the sorted numbers are:\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t",a[i]);
        }

        getch();
}
```

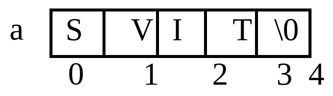
Write a C program to evaluate the polynomial using Horners method.

```
#include<stdio.h>
void main()
{
        int i,x,n,a[10],sum;
        printf("enter n:\n");
        scanf("%d",&n);
        printf("enter n+1 co efficient\n");
        for(i=0; i<=n;i++)
        {
                scanf("%d",&a[i]);
        }
        sum=a[n]* x;
        for(i=n-1; i>0; i--)
        {
                sum=(sum+a[i]) *x;
        }
        sum=sum+a[0];
        printf("sum of polynomial equation is %d",sum);
}
```

# Strings

## Definition:

- A string is a sequence of characters within double quotes. A string constant is always terminated y null character.
- A string is pictorially represented as follows:



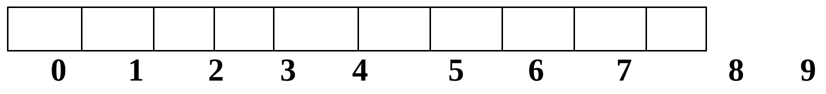
## String Declaration:

- Like all other variable a string variable a string variable also has to be declared before it is used.

**Syntax:** **char string_name[size];**

**Example: char s[10];**

The above declaration statement allocates 10 bytes of memory to string s as follows:



## String Initialization:

Initialization is a process of assigning values to a string, before doing manipulation.

Strings are initialized in 4 ways:

1. Initializing character by character
2. Partial Array Initialization
3. Initialization without size
4. Initialization of Array with string

**1.Initializing character by character**

✓ Consider following declaration and initialization

**char b[5]={ 'S','V','I','T'};**

✓ The complier allocates 5 memory locations and these locations are initialized with the character in the order specified.

| b | S | V | I | T | \0 |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |

**2. Partial Array Initialization**

✓ If the number of characters to be initialized is less than the size of array then the remaining locations will be initialized to NULL as follows:

**Char b[5]={ 'H','I'};**

| b | H | I | \0 | \0 | \0 |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |

**3.Initialization without Size**

✓ If a string is declared without size then compiler will set the array size to the total number of initialized values.

**char b[]={ 'S','V','I','T'};**

| b | H | I | \0 | \0 | \0 |
|---|---|---|---|---|---|
| | 0 | 12 | | 3 | 4 |

Size of b is 4

**4.Initialization of array with string**

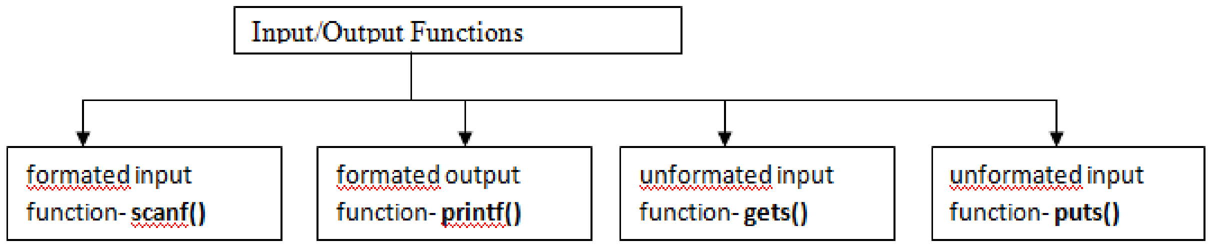
**char b[]="SVIT";**

In the above initialization the string length is 4 bytes but size is 5 bytes.

| b | S | V | I | T | \0 |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |

## String Input and Output functions:

➢ The strings can be read from the keyboard and can be displayed onto the monitor using various functions:


Input/Output Functions
formated input function- scanf()
formated output function- printf()
unformated input function- gets()
unformated input function- puts()

**Formatted Input Function: scanf()**

- The formatted input function is scanf().
- It reads a string from the keyboard
- The format specifier is **%s**
- **The string is terminated by NULL character(\0)**

**Syntax: scanf("%s",str);**

- Example:

char str[5]="SVIT";
scanf("%s",str);

| b | S | V | I | T | \0 |
|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 |

**NOTE:** scanf() cannot read spaces and any special symbols i.e conversion code cannot read spaces, it will terminated as soon as space appear.

**Formatted Output Function: printf()**

- The formatted output function is printff().
- It prints/displays a string which is stored on memory locations on monitor

**Syntax: printf("%s",str);**

- Example:

char str[5]="SVIT";
printf("%s",str);

**Write a program to read and print an string using scanf() and printf()**

```
#include<stdio.h>
void main()
{
char str[20];
printf("enter the string\n");
scanf("%s",str);
printf("The entered string is \n");
printf("%s",str);
}
```

**<u>UnFormatted Input Function: scanf()</u>**

- The Unformatted input function is gets().
- It reads a sequence of characters(line) from the keyboard with spaces in between and store them in memory locations.

  **Syntax: gets(str);**
- Example:

```
char str[20];
printf("enter the string\n");
gets(str);
```

**<u>UnFormatted output Function: puts()</u>**

- The Unformatted output function is puts().
- This function displays all the character(line) stored in variable **str** on the monitor till it encounters **\0(Null Character)**

  **Syntax: gets(str);**
- Example:

```
char str[20]="HELLO";
printf("the string is \n");
puts(str);
```

**<u>Write a program to read and print an string using gets() and puts()</u>**

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[20];
printf("enter the string\n");

gets(str);
printf("The entered string is \n");
puts(str);
}
```

OutPut:

```
Enter the string
HELLO HOW R U
The entered string is
HELLO HOW R U
```

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | E | L | L | O | | H | O | W | | R | | U | \0 | | | | |

**Based on the kind of data processed, the I/O function are classified into**

1. **Token Oriented I/O functions:**
2. **Line Oriented I/O functions**
3. **Character Oriented I/O functions**

1. **<u>Token Oriented I/O functions:</u>**

- The I/O functions processes individual units such as characters,integers,double values,float values and are separated by whitespaces characters. Since these individual units are called tokens,the functions that perform these kind of operations are called ***Token Oriented I/O*** **functions.**
- The functions **scanf() and printf()** are ***Token Oriented I/O*** **functions.**

2. **<u>Line Oriented I/O functions</u>**

- The I/O functions that process entire line are called ***line oriented I/O functions.***
- The functions **gets() and puts()** are ***Line Oriented I/O*** **functions.**

3. **<u>Character Oriented I/O functions</u>**

a) **<u>getchar() and putchar()</u>**
   - To read a character from the keyboard and store this character into memory location, ***getchar()*** function is used.
   - We have to press the ENTER KEY after typing character.
     **Syntax: ch=getchar();**
   - To display a character stored in the memory on the screen ***,putchar()*** function is used.
     **Syntax:putchar(ch);**

**<u>Write a C program to Find the section of student.</u>**

```
#include<stdio.h>
#include<string.h>
void main()
{
        char sec;
   printf("Enter your section\n");
   scanf("%d",&sec);
```

**<u>OUTPUT</u>**

```
Enter Your Section
B
```

```
  sec=getchar();                                    Your section is
 printf("Your section is \n");                      B
 putchar(sec);
}
```

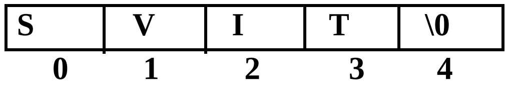
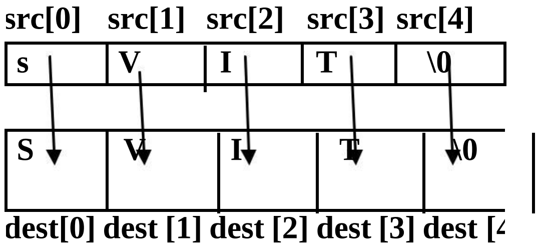
**b) <u>getch(),getche(),putch()</u>**

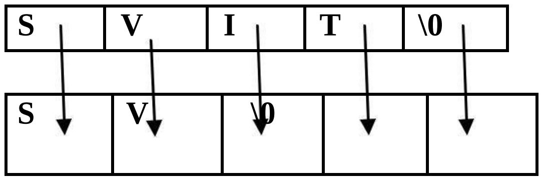
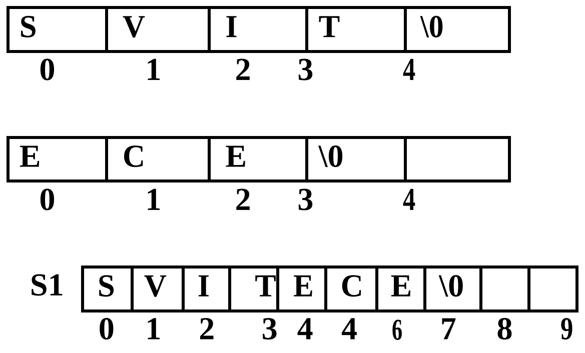
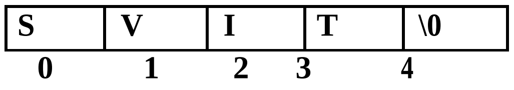
- The **getch()** function reads a character from the keyboard and copies it into specified memory location identified by **ch.**
- **Syntax: ch=getch()**
- The typed character will not be echoed(displayed) on the screen if we use **getch()** function.
- No arguments are required for this function

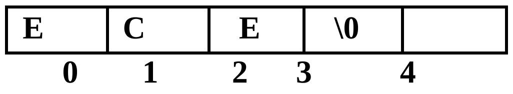
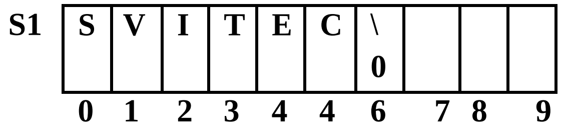
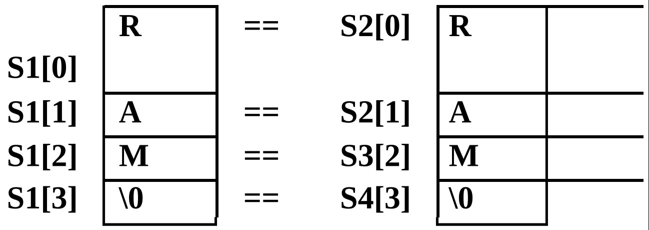
- The **getche()** function reads a character from the keyboard and copies it into specified memory location identified by **ch.**
- **Syntax: ch=getche()**
- The typed character will be echoed(displayed) on the screen if we use **getche()** function. No arguments are required for this function

- The **putchar()** displays a character stored in memory location identified by variable **ch** on the screen.
- **Syntax: putch(ch)**

## *String handling Functions*

| SL. No | Name | Syntax | Example | Explanation |
|---|---|---|---|---|
| **1** | **strlen** | **int strlen (char str[ ]);** | **char str[15]="SVIT";**<br>**int count;**<br>**count=strlen(str);**<br> <br>**The example str variable contains 4 characters S,V,I,T , hence count is 4** | **-This function returns the length of the string str.**<br>**-It counts all the characters until null character is encountered.** |
| **2** | **strcpy** | **strcpy(char dest[ ] , char src[ ]);** | **char src[5] ="SVIT";**<br>**char dest[5];**<br>**strcpy(dest ,src);**<br> src[0] src[1] src[2] src[3] src[4] s V I T \0 S V I T \0 dest[0] dest [1] dest [2] dest [3] dest [4 | ✓ **This function copies content from source string to destination string including \0.**<br>✓ **Size of dest string should be greater or equal to the size of source string src to store the entire source string.** |

| | | | | |
|---|---|---|---|---|
| 3. | **strncpy** | **strcpy(char dest[ ] , char src[ ],int n);** | **char src[5] =”SVIT”;**<br>**char dest[5];**<br>**strcpy(dest ,src,2);**<br>**src[0] src[1] src[2] src[3] src[4]**<br> <br>**dest[0] dest [1] dest [2] dest [3] dest [4]** | ✓ **This function copies n characters from source string to destination string .**<br>✓ **In thisexample only 2 characters are copied from src to dest.** |
| 4 | **strcat** | **strcat(char s1[ ] , char s2[ ]);** | **char s1[5]=”SVIT”;**<br>**char s2[5]=”ECE”;**<br>**strcat(s1,s2);**<br> S<br>V<br>I<br>T<br>\0<br>0 1 2 3 4<br>E<br>C<br>E<br>\0<br>0 1 2 3 4<br>S1<br>S V I T E C E \0<br>0 1 2 3 4 4 6 7 8 9 | ✓ **This function copies the all characters of s2 string to the end of s1 string.**<br>✓ **The delimiter of s1 is replaced by first character of s2.**<br>✓ **Size of s1 string should be greater or store the contents of both the string** |
| 5 | **strncat** | **strncat(char s1[ ] , char s2[ ],n);** | **char s1[5]=”SVIT”;**<br>**char s2[5]=”ECE”;**<br>**strncat(s1,s2,2);**<br> | ✓ **This function copies the n characters of s2 string to the end of s1 string.**<br>✓ **The delimiter of s1 is replaced by first character of s2.** |

| | | | | |
|---|---|---|---|---|
| | | |  <br> S1<br>S<br>V<br>I<br>T<br>E<br>C<br>\0<br>0 1 2 3 4 4 6 7 8 9<br>**In the above example only 2 characters(EC) from string s2 copies to string s1.** | |
| 6 | **strcmp** | **int strcmp( char s1[ ] , char s2[ ]);** | **1) Strings are equal**<br>**S1[4]=”RAM”;**<br>**S2[4]=”RAM”;**<br>**Strcmp(S1,S2);**<br> S1[0]<br>R<br>==<br>S2[0]<br>R<br>S1[1]<br>A<br>==<br>S2[1]<br>A<br>S1[2]<br>M<br>==<br>S3[2]<br>M<br>S1[3]<br>\0<br>==<br>S4[3]<br>\0<br>**S1[0]==S2[0]**<br>**R==R(ASCII value of R is compared) similarly for other characters.**<br>**S1[3]==S2[3]**<br>**\0==\0(ASCII value of \0 is compared and it is 0.)**<br>**2)String S1 is Lesser than String S2** | **where:**<br>**s1 is first string**<br>**s2 is second string**<br>✓ **This function used to compare two strings.**<br>✓ **The comparison starts with first character of each string.**<br>✓ **This comparison continues till the corresponding character differ or until the end of the character is reached.**<br>✓ **The strcmp Returns 3values Possibly:**<br>**returns 0 if both strings are equal.**<br>**returns positive value ,if s1>s2**<br>**returns negative value if s1<s2** |

**S1[4]="ABC";**
**S2[4]="BAC";**
**Strcmp(S1,S2);**

| | | | | |
|---|---|---|---|---|
| **S1[0]** | **A** | == | **S2[0]** | **B** |
| **S1[1]** | **B** | == | **S2[1]** | **A** |
| **S1[2]** | **C** | == | **S3[2]** | **C** |
| **S1[3]** | **\0** | == | **S4[3]** | **\0** |

**S1[0]==S2[0]**
**A==B(ASCII value of A is compared with ASCII value of B)**
**i.e 65==66 returns S1<S2**

**3)String S1 is Greater than String S2**
**S1[4]="BBC";**
**S2[4]="ABC";**
**Strcmp(S1,S2);**

| | | | | |
|---|---|---|---|---|
| **S1[0]** | **B** | == | **S2[0]** | **A** |
| **S1[1]** | **B** | == | **S2[1]** | **B** |
| **S1[2]** | **C** | == | **S3[2]** | **C** |
| **S1[3]** | **\0** | == | **S4[3]** | **\0** |

**S1[0]==S2[0]**
**A==B(ASCII value of A is compared with ASCII value of B)**
**i.e 66==65 returns S1>S2**

| | | | | |
|---|---|---|---|---|
| 7 | **strncmp** | **int strcmp( char s1[ ] , char s2[ ], n);** | **1) Strings are equal**<br>**S1[4]=”RAM”;**<br>**S2[4]=”RAM”;**<br>**strcmp(S1,S2,2);**<br>**S1[0] R == S2[0] R**<br>**S1[1] A == S2[1] A**<br>**S1[2] M == S3[2] M**<br>**S1[3] \0 == S4[3] \0**<br>**Only 2 characters from each S1 and S2 is compared.**<br>**Other function is similar to strcmp().** | **where:**<br>**s1 is first string**<br>**s2 is second string**<br>✓ **This function used to comparen number of characters two strings.**<br>✓ **The comparison starts with first character of each string.**<br>✓ **This comparison continues till the corresponding character differ or until the end of the character is reached or specified number of characters have been tested..**<br>✓ **The strcmp Returns 3values Possibly:**<br>**returns 0 if both strings are equal.**<br>**returns positive value ,if s1>s2**<br>**returns negative value if s1<s2** |
| 8 | **strrev( )** | **void strrev(char str[ ]);** | **Given string**<br>**S1 \| S \| V \| I \| T \| E \| C \| E \| \0 \| \| \|**<br>**0 1 2 3 4 5 6 7 8 9**<br>**strrev(s1**<br>**Reverse String**<br>**S1 \| E \| C \| E \| T \| I \| V \| S \| \0 \| \| \|**<br>**0 1 2 3 4 5 6 7 8 9** | ✓ **This function reverse all characters in the S1 except Null character.**<br>✓ **The original string is lost.** |

| | | | | |
|---|---|---|---|---|
| | | | **S1[6]==S1[0]**<br>**S1[5]==S1[1]**<br>**S1[4]==S1[2]**<br>**S1[3]==S1[3]**<br>**S1[2]==S1[4]**<br>**S1[1]==S1[5]**<br>**S1[0]==S1[6]** | |

## Example Programs for string handling functions

**strlen()**

```
#include<stdio.h>
#include<string.h>
void main()
{ char name[15];
    int len;
   printf("Enter the string\n");
   gets(name);
    len=strlen(name);
    printf("\n The string length is %d",len);
}
```

**OUTPUT**

```
Enter the string
COMPUTER
The string length is 8
```

**strrev()**

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

**OUTPUT**

```
String=AIDNI
```

**strcpy()**

```
#include<stdio.h>
#include<string.h>
void main()
{
      char src[15],char dest[15];
      printf("Enter the source string\n");
     gets(src);
      strcpy(dest,src);
      printf("\n The copied string is \n");
     puts(dest);
}
```

**OUTPUT**

```
Enter the source string
COMPUTER
The copied string is
 COMPUTER
```

**strncpy()**

```
#include<stdio.h>
#include<string.h>
void main()
{
      char src[15],char dest[15];
     int n;
     printf("Enter the source string\n");
     gets(src);
     printf("Enter n");
     scanf("%d",&n);
      strncpy(dest,src);
      printf("\n The copied string is \n");
     puts(dest);
}
```

**OUTPUT**

```
Enter the source string
COMPUTER
Enter n
3
The copied string is
 COMPUTER
```

**strcat()**

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

**strncat()**

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

```
                {
                     char S1[15],char S2[15];
                    printf("Enter the string 1\n");
                    gets(S1);
                    printf("Enter the string 2\n");
                    gets(S2);
                     strcat(S1,S2);
                     printf("\n The Concatenated string is \n");
                    puts(S1);
                }
```

**OUTPUT**

```
Enter the string1
HELLO
Enter the string 2
ALL
The Concatenated string is
 HELLOALL
```

```
                {
                     char S1[15],char S2[15];
                    int n;
                    printf("Enter the string 1\n");
                    gets(S1);
                    printf("Enter the string 2\n");
                    gets(S2);
                    printf("Enter n");
                    scanf("%d",&n);
                     strncat(S1,S2,n);
                     printf("\n The Concatenated string is \n");
                    puts(S1);
                }
```

**OUTPUT**

```
Enter the string1
HELLO
Enter the string 2
SVIT
Enter n
2
The Concatenated string is
 HELLOSV
```

**strcmp()**

```
                #include<stdio.h>
                #include<string.h>
                void main()
                {
                     char S1[15],char S2[15];
```

**strcnmp()**

```
                #include<stdio.h>
                #include<string.h>
                void main()
                {
                     char S1[15],char S2[15];
```

```
                         int res;
                         printf("Enter the string 1\n");
                         gets(S1);
                         printf("Enter the string 2\n");
                         gets(S2);
                          res=strcmp(S1,S2);
                          if(res==0)
                               printf("Strings are same\n");
                          else if(res>0)
                                printf("String1 is greater than string2\n");
                          else
                               printf("String1 is lesser than string2\n");
                      }
```

**OUTPUT**

```
Enter the string1
HELLO
Enter the string 2
HELLO
The Strings are equal
```

```
                         int res,n;
                         printf("Enter the string 1\n");
                         gets(S1);
                         printf("Enter the string 2\n");
                         gets(S2);
                         printf("Enter n");
                         scanf("%d",&n);
                          res=strcmp(S1,S2,n);
                          if(res==0)
                               printf("Strings are same\n");
                          else if(res>0)
                                printf("String1 is greater than string2\n");
                          else
                               printf("String1 is lesser than string2\n");
                      }
```

**OUTPUT**

```
Enter the string1
SVIT
Enter the string 2
SVCE
Enter n
2
String1 is greater than String2
```

## <u>Edit Set Conversion Code</u> (%[…])

- Using edit set conversion code ,it is possible to enter a line of text with white spaces such as blank character,**\t etc…**
- **Syntax:scanf(%[....... ] ,str)**

$\underbrace{\qquad\qquad}$

edit characters

- edit characters represent the valid characters that are allowed in the string and should be enclosed within **‘ [ ‘** and **‘]’**

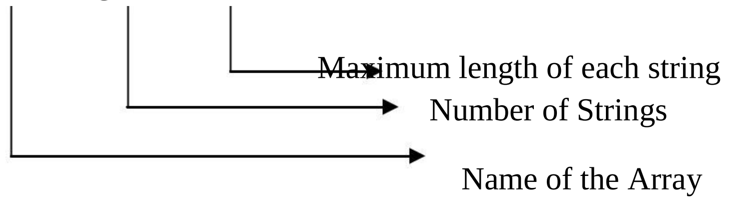
**Working:**

- Each character read by ***scanf*** is compared with ***edit character set***.
- If the character read is in **edit character set,** it is copied into **str**
- The above procedure is repeated as long as the character read is in **edit character set.**
- If the character read does not match with **edit character set**, the reading process is stopped and remaining characters will still be available in keyboard buffer.
- If the first character read is not in **edit characters set**, the **scanf** is terminated and a **‘\0’** is copied into **str** indicating **null string.**
- The **edit characters set** is indicated by the symbol **caret**(^)

```
scanf(“%[^\n]”,str);          // read all characters except \n
```

## Array of Strings

- Group of Strings is known as array of strings.
- To represent array of strings two dimensional array is required.
  Declaration Syntax: **char string_name[row][column];**



Example: char a[5][20];

Where a is the string name

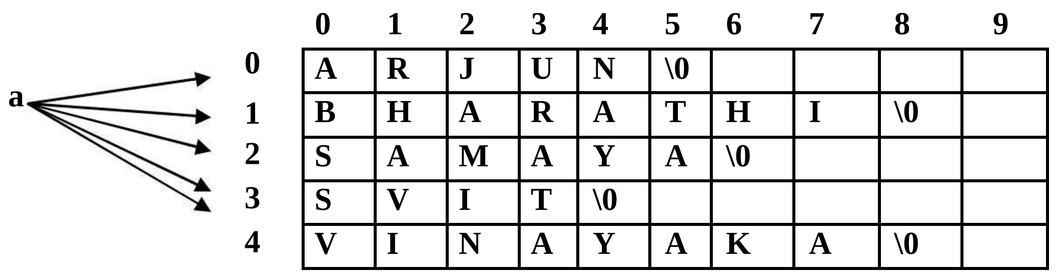
5 indicates at most 5 names can be stored

20 indicate that each name can have at most 20 characters

**Initialization**

```
char a[5][10]={
                    "ARJUN",
                     "BHARATHI",
                     "SAMAYA",
                      "SVIT",
                      "VINAYAKA"
                  };
```

- Memory representation of the above initialization is as follows..



|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | A | R | J | U | N | \0 |  |  |  |  |
| 1 | B | H | A | R | A | T | H | I | \0 |  |
| 2 | S | A | M | A | Y | A | \0 |  |  |  |
| 3 | S | V | I | T | \0 |  |  |  |  |  |
| 4 | V | I | N | A | Y | A | K | A | \0 |  |

**Note: Example program is binary search**