# C PROGRAMMING FOR PROBLEM SOLVING (18CPS23)

## Module 3
## Arrays and Strings

**Prof. Prabhakara B. K.**
Associate Professor
Department of Computer Science and Engineering
Vivekananda College of Engineering and Technology, Puttur.

# C PROGRAMMING FOR PROBLEM SOLVING (18CPS23)

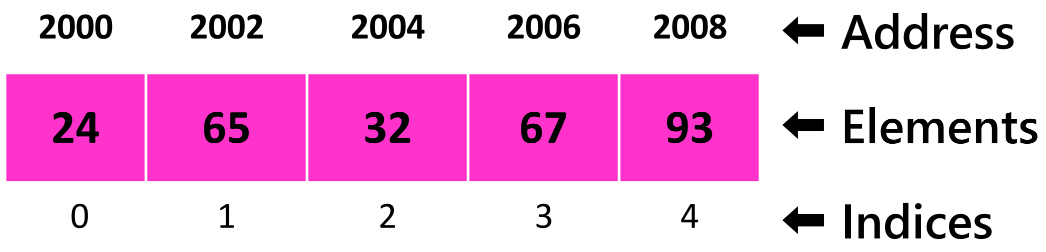
## Module 3
## Arrays and Strings

Arrays: One dimension arrays (1-D), Two dimension arrays (2-D), Character arrays and Strings Basic Algorithms: Searching and Sorting Algorithms (Linear search, Binary search, Bubble sort and Selection sort)

# ARRAY(Derived Data Type)

- An array is a collection of similar data items.
- Data items are stored at contiguous memory locations.
- Data items are also known as elements of array.
- Elements of an array is accessed randomly using indices / subscripts
- These indices / subscripts are always integer.
- They are used to store collection one particular primitive data types

  such as int, float, double, char, etc.



| 2000 | 2002 | 2004 | 2006 | 2008 | ⬅ Address |
|---|---|---|---|---|---|
| **24** | **65** | **32** | **67** | **93** | ⬅ Elements |
| 0 | 1 | 2 | 3 | 4 | ⬅ Indices |

**Definition**

**An array is fixed size sequenced collection of elements of same data type.**

OR

List of elements of same data type within single variable.

# TYPES OF ARRAY

We can implement three types of array in C.

1. One dimension array
2. Two dimension array
3. Multi dimension array

## One dimension Array ( 1D array)

Fixed size sequenced collection of elements of same data type are having one subscript (index) is called one dimension array.

Example int a[5];

## Two dimension Array ( 2D array)

Fixed size sequenced collection of elements of same data type are having two subscripts (indices) is called two dimension array.

Example int a[5][4];

## Multi dimension Array

Fixed size sequenced collection of elements of same data type are having more than one subscripts (indices) is called multi dimension array.

Example int a[5][4][3];

# ONE DIMENSION ARRAY ( 1D ARRAY)

Fixed size sequenced collection of elements of same data type are in one row or one column having one subscript (index) is called one dimension array.

**Declaration**

Syntax

```
datatype  arrayname[size];
```

datatype - may be int, float, char, double etc.

size - number of elements store in an array.

arrayname – is a valid names like identifiers.

Example 1 int a[5];

Example 2 float avg[3];

Example 3 char name[10];

# INITIALIZATION OF ONE DIMENSION ARRAY

Initialization of an array means storing the elements in an array.

Array can be initialized in two different fashions.

1. Compilation time initialization
2. Run time initialization

## 1. Compile time initialization

Storing values in an array when they are declared.

Syntax

```
datatype  arrayname[size]={ V1, V2, V3,… Vn};
```

Values are separated by commas.

Example 1 int a[3]={10,20,30};

Example 2 float b[2]={1.1,2.2};

# INITIALIZATION OF ONE DIMENSION ARRAY

## a) Basic Initialization

Specifying the size of an array and its values during declaration.

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

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

## b) Without size Initialization

If the array is initialized during its declaration, mentioning its size (number of elements) is optional.

Example1 int a[ ] = {10,20,30,40,50} ;

Example 2 float avg[ ] = {1.1,2.2} ;

Example 3 char name[ ]={‘I’,’N’,’D’,’I’,’A’};

# INITIALIZATION OF ONE DIMENSION ARRAY

## c) Partial Initialization

Storing less number of elements in an array than its size.

Example 1 int a[5]= {10,20,30};

| 10 | 20 | 30 | 0 | 0 |
|---|---|---|---|---|
| a[0] | a[1] | a[2] | a[3] | a[4] |

Example 2 char name[5] = {‘a',‘b','c'};

Example 3 char place[10] = “Puttur”;

## d) Null Initialization

Storing same values in all locations of an array.

Example int a[5]={0};

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

# INITIALIZATION OF ONE DIMENSION ARRAY

## 2. Run time initialization

Initialization of an array when program is under execution.

Example 1

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

Using for loop to initialize elements of an array

(**READING ELEMENTS INTO AN ARRAY**)

Example 2

```
int i, a[3];
…………
…………
for(i=0; i<3; i++)
{
        scanf("%d", &a[i]);
}
```

**TRACE**

i=0
0< 3
a[0]= ?
i=0+1=1
-------------------
1< 3
a[1]= ?
i=1+1=2
------------------
2< 3
a[2]= ?
i=2+1=3
------------------
3< 3

# PRINTING ONE DIMENSION ARRAY

Example

```
int i, a[3];
………….
………….
for(i=0; i<3; i++)
{
        printf("%d\t", a[i]);
}
```

| 24 | 65 | 32 |
|---|---|---|
| a[0] | a[1] | a[2] |

**TRACE**

i=0<br>
0< 3<br>
a[0]= 24<br>
i=0+1=1

------------------

1< 3<br>
a[1]= 65<br>
i=1+1=2

-----------------

2< 3<br>
a[2]= 32<br>
i=2+1=3

-----------------

3< 3

Output

```
24      65      32
```

# Write a C program to read and print an array elements

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

Output

```
Enter number of elements
3
Enter elements of array
10
20
30
Array elements are
10      20      30
```

**TRACE**

```
n=3
i=0
0< 3
a[0]= ?   10
i=0+1=1
------------------
1< 3
a[1]= ?   20
i=1+1=2
-----------------
2< 3
a[2]= ?   30
i=2+1=3
-----------------
3< 3
=============
i=0
0< 3
a[0]= 10
i=0+1=1
------------------
1< 3
a[1]= 20
i=1+1=2
-----------------
2< 3
a[2]= 30
i=2+1=3
-----------------
3< 3
```

# Write a C program to find the largest among n integers.

```
#include <stdio.h>
void main()
{
        int i, n, a[100], big;
        printf("Enter the no. of elements\n");
        scanf("%d", &n);
        printf("Enter elements\n");
        for(i=0; i<n; i++)
        {
            scanf("%d", &a[i]);
        }
        big= a[0];
        for(i=1; i<n; i++)
        {
                if(a[i]>big)
                {
                     big = a[i];
                }
        }
        printf("Biggest element is %d", big);
}
```

| 25 | 93 | 42 |
|---|---|---|
| a[0] | a[1] | a[2] |

**TRACE**

n=3, big=25

i=1

1< 3

a[1]>25

93>25

big=93

i=1+1=2

------------------

2< 3

a[2]>93

42>93

i=2+1=3

-----------------

3< 3

# SEARCHING

The searching is an operation or a technique to locate a given element in any data structure where it is stored.

## Types of Searching

### 1. Sequential Search

The list of elements are sequentially compared with the searching element. Example Linear Search

### 2. Interval Search

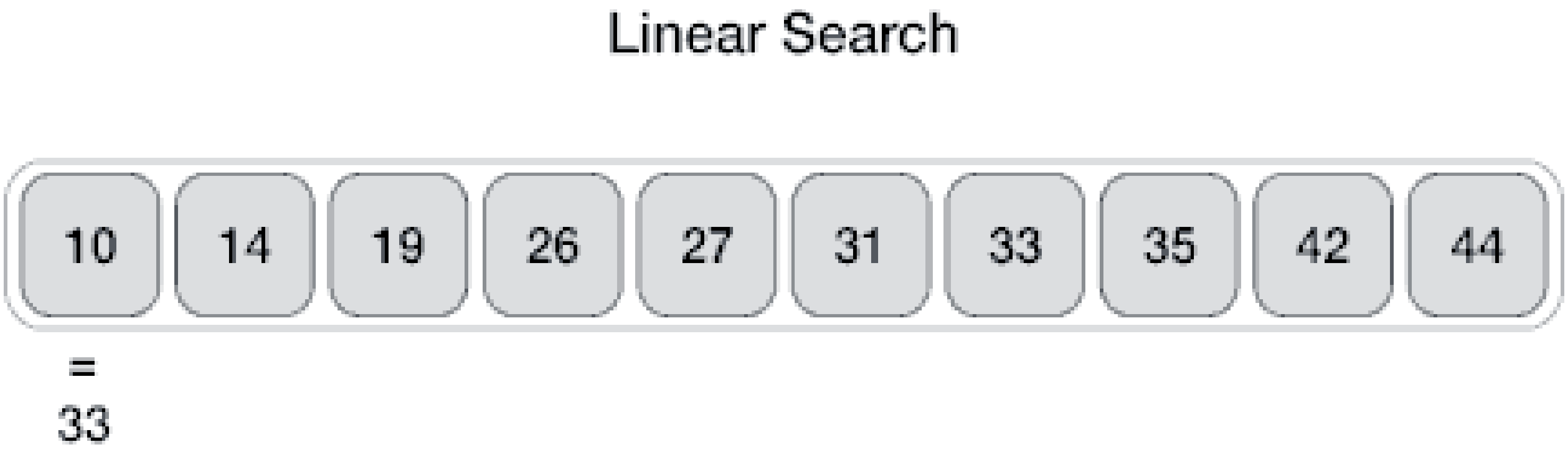
This search uses **sorted list** of elements.

This is an efficient searching technique compared with sequential search. Example Binary Search

# SEARCHING ALGORITHMS

- Linear Search
- Binary Search
- Jump Search
- Interpolation Search
- Exponential Search

# LINEAR SEARCH ALGORITHM



# LINEAR SEARCH PROGRAM

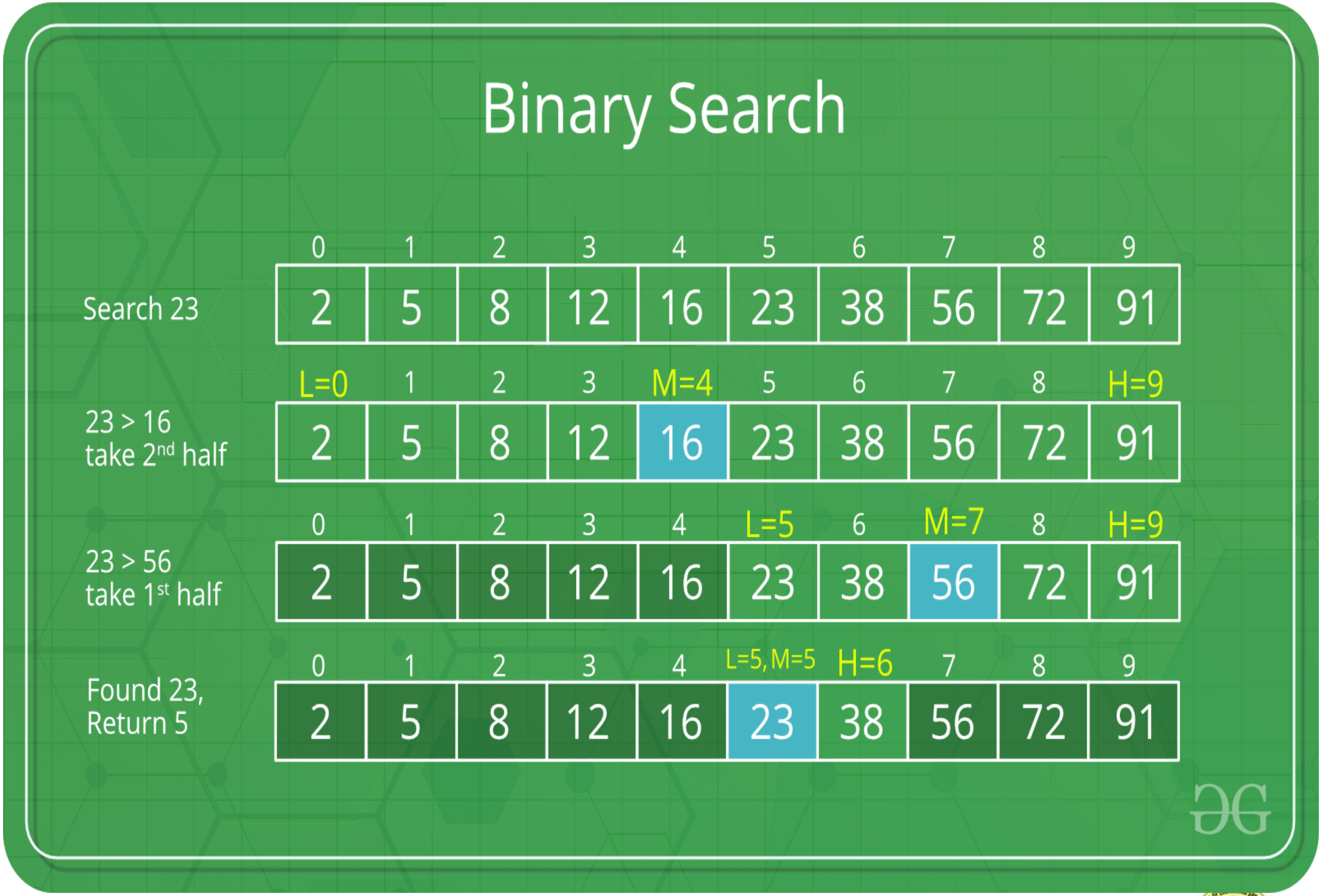
```
#include <stdio.h>
#include<stdlib.h>
void main()
{
    int a[10], i, n,key;
    printf("Enter no.of elements\n");
    scanf("%d",&n);
    printf("Enter the elements");
    for (i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
    printf("Enter item to be searched \n");
    scanf("%d", &key);
    for (i=0; i<n; i++)
    {
        if (key == a[i])
        {
            printf("Search successful\n");
            exit(0);
        }
    }
    printf("Search unsuccessful\n");
}
```

# BINARY SEARCH ALGORITHM

Search for 47

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 7 | 10 | 14 | 23 | 45 | 47 | 53 |

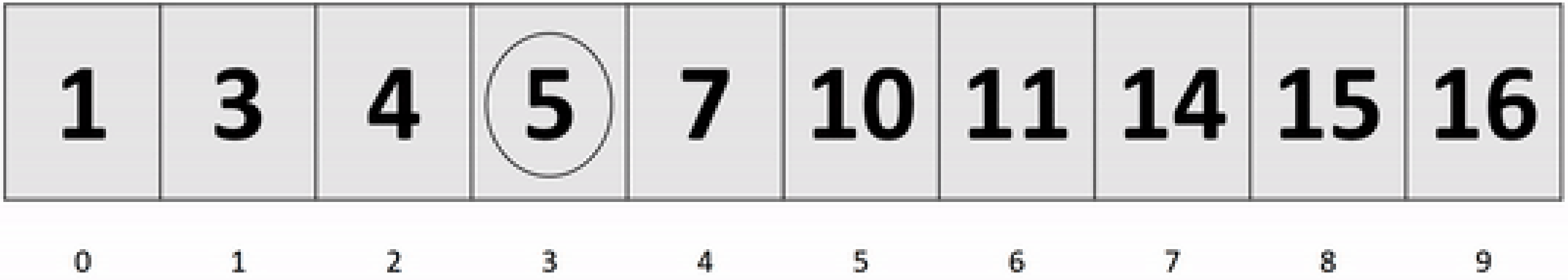
# BINARY SEARCH ALGORITHM



## Binary Search

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Search 23 | 2 | 5 | 8 | 12 | 16 | 23 | 38 | 56 | 72 | 91 |
| | L=0 | 1 | 2 | 3 | M=4 | 5 | 6 | 7 | 8 | H=9 |
| 23 > 16<br>take 2nd half | 2 | 5 | 8 | 12 | 16 | 23 | 38 | 56 | 72 | 91 |
| | 0 | 1 | 2 | 3 | 4 | L=5 | 6 | M=7 | 8 | H=9 |
| 23 > 56<br>take 1st half | 2 | 5 | 8 | 12 | 16 | 23 | 38 | 56 | 72 | 91 |
| | 0 | 1 | 2 | 3 | 4 | L=5, M=5 | H=6 | 7 | 8 | 9 |
| Found 23,<br>Return 5 | 2 | 5 | 8 | 12 | 16 | 23 | 38 | 56 | 72 | 91 |

# BINARY SEARCH ALGORITHM



| 1 | 3 | 4 | 5 | 7 | 10 | 11 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# BINARY SEARCH PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        int low, high, mid, key,  n,  i, a[100];
        printf("Enter the number of elements\n");
        scanf("%d",&n);
        printf("Enter the elements in ascending order\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("Enter the element to be searched\n");
        scanf("%d",&key);
        low=0;
        high=n-1;
```

# BINARY SEARCH PROGRAM

```
  while(low<=high)
  {
          mid=(low+high)/2;
           if(key == a[mid])
           {
               printf("Successful Search & element is found at=%d\n",mid+1);
               exit(0);
           }
           if(key>a[mid])
           {
               low=mid+1;
            }
           else
           {
               high=mid-1;
           }
   }
   printf("Unsuccessful Search\n");
}
```

# SORTING

The sorting is an operation or a technique to arranging elements of a given list either in an ascending order or in descending order according to a comparison operator.

## Sorting Algorithms

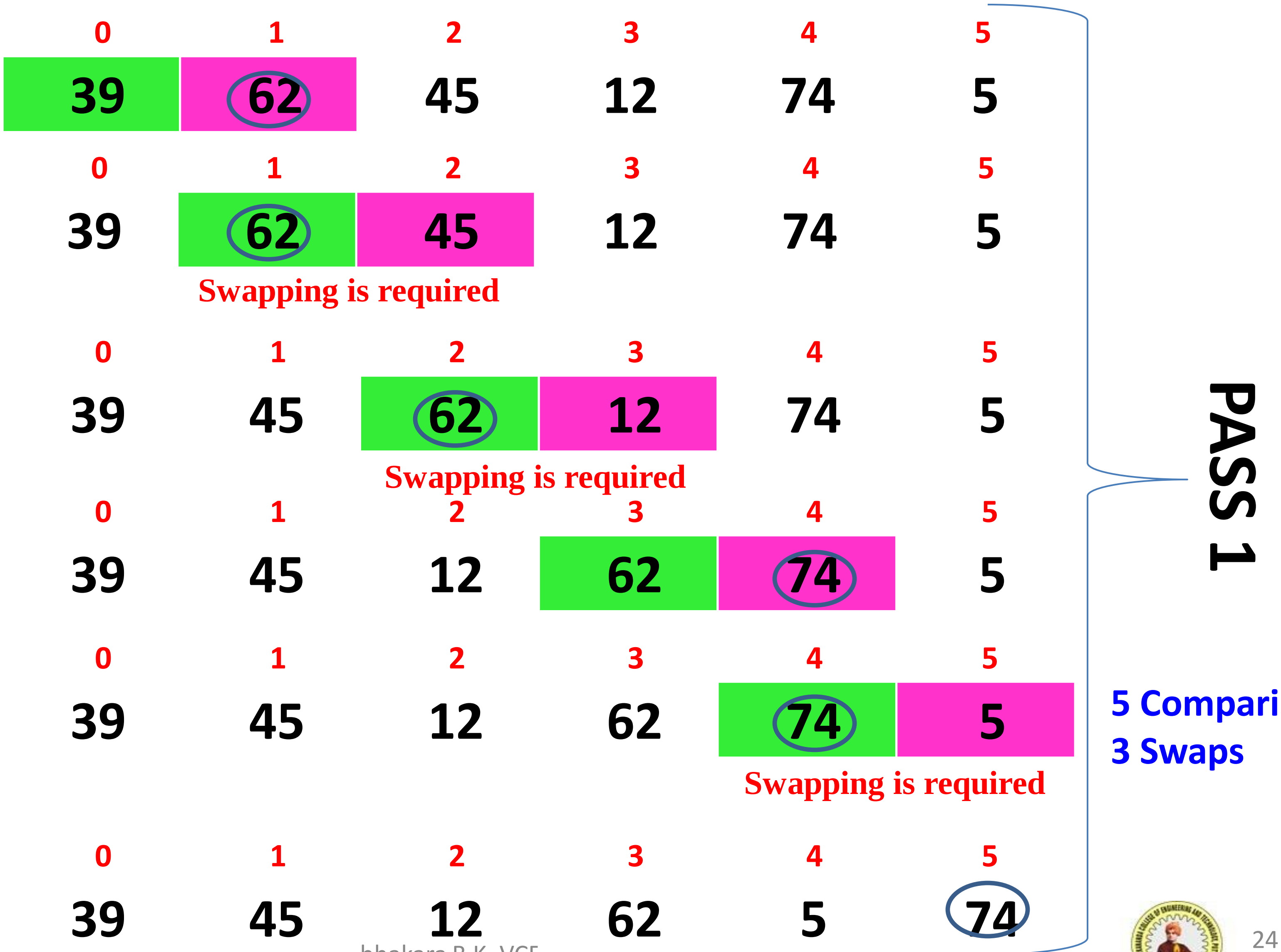
- Bubble Sort
- Selection Sort
- Quick Sort ← fastest sorting algorithm
- Insertion Sort
- Merge Sort
- Heap Sort
- Shell Sort
- Radix Sort
- Bucket Sort etc.

# BUBBLE SORT ALGORITHM

6 5 3 1 8 7 2 4

# BUBBLE SORT

## To arrange 6 elements of an array in Ascending order



| INDEX | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 39 | 62 | 45 | 12 | 74 | 5 |

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 62 | 45 | 12 | 74 | 5 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 45 | 62 | 12 | 74 | 5 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 45 | 12 | 62 | 74 | 5 |

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 45 | 12 | 62 | 74 | 5 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 45 | 12 | 62 | 5 | 74 |

5 Comparisons
3 Swaps

# BUBBLE SORT

## To arrange 6 elements of an array in Ascending order

| INDEX | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 39 | 45 | 12 | 62 | 5 | 74 |

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 45 | 12 | 62 | 5 | 74 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 12 | 45 | 62 | 5 | 74 |

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 12 | 45 | 62 | 5 | 74 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 39 | 12 | 45 | 5 | 62 | 74 |

**PASS 2**

4 Comparisons

2 Swaps

# BUBBLE SORT

## To arrange 6 elements of an array in Ascending order

| INDEX | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 39 | 12 | 45 | 5 | 62 | 74 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 12 | 39 | 45 | 5 | 62 | 74 |

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 12 | 39 | 45 | 5 | 62 | 74 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 12 | 39 | 5 | 45 | 62 | 74 |

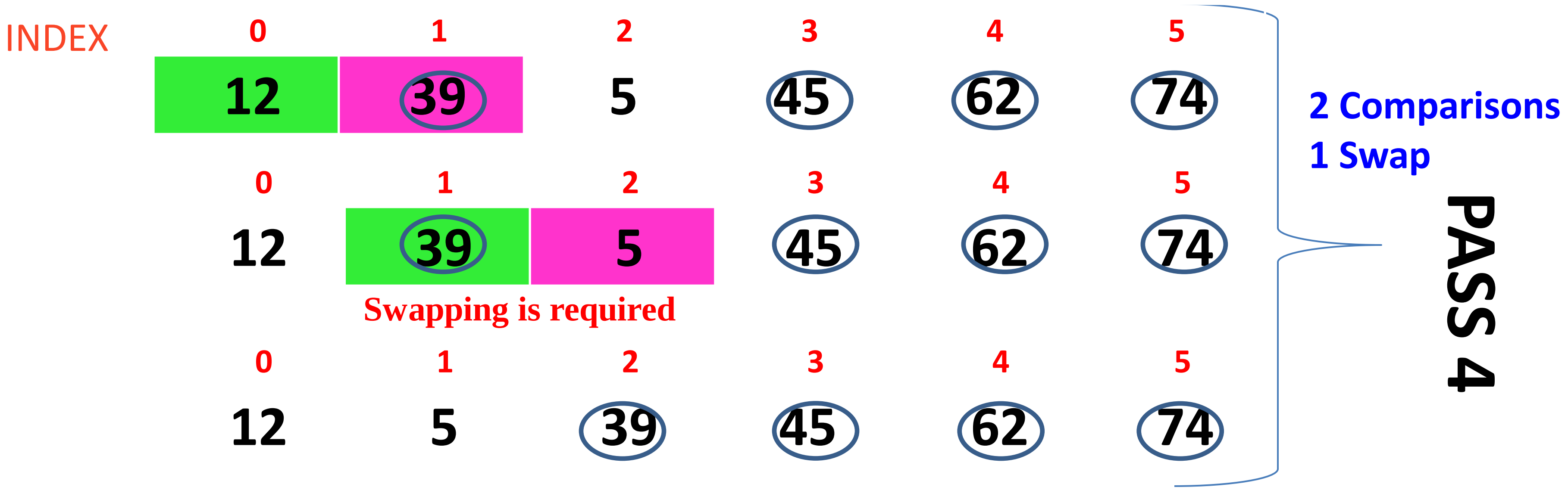
PASS 3

3 Comparisons

2 Swaps

# BUBBLE SORT

## To arrange 6 elements of an array in Ascending order



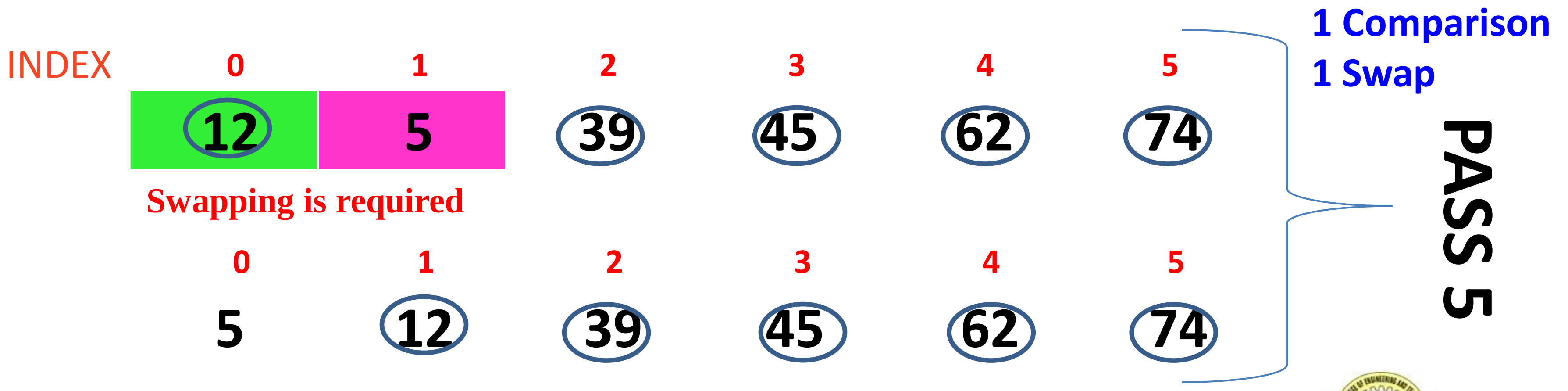
| INDEX | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 12 | 39 | 5 | 45 | 62 | 74 |

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 12 | 39 | 5 | 45 | 62 | 74 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 12 | 5 | 39 | 45 | 62 | 74 |

2 Comparisons
1 Swap



| INDEX | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| | 12 | 5 | 39 | 45 | 62 | 74 |

Swapping is required

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 5 | 12 | 39 | 45 | 62 | 74 |

1 Comparison
1 Swap

# BUBBLE SORT ALGORITHM

| | 0 | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|
| 1,0 | 39 | 62 | 45 | 12 | 74 | 5 | PASS 1 |
| 1,1 | 39 | 62 | 45 | 12 | 74 | 5 | |
| 1,2 | 39 | 45 | 62 | 12 | 74 | 5 | |
| 1,3 | 39 | 45 | 12 | 62 | 74 | 5 | |
| 1,4 | 39 | 45 | 12 | 62 | 74 | 5 | |
| | 39 | 45 | 12 | 62 | 5 | 74 | |
| 2,0 | 39 | 45 | 12 | 62 | 5 | 74 | PASS 2 |
| 2,1 | 39 | 45 | 12 | 62 | 5 | 74 | |
| 2,,2 | 39 | 12 | 45 | 62 | 5 | 74 | |
| 2,3 | 39 | 12 | 45 | 62 | 5 | 74 | |
| | 39 | 12 | 45 | 5 | 62 | 74 | |
| 3,0 | 39 | 12 | 45 | 5 | 62 | 74 | PASS 3 |
| 3,1 | 12 | 39 | 45 | 5 | 62 | 74 | |
| 3,,2 | 12 | 39 | 45 | 5 | 62 | 74 | |
| | 12 | 39 | 5 | 45 | 62 | 74 | |
| 4,0 | 12 | 39 | 5 | 45 | 62 | 74 | PASS 4 |
| 4,1 | 12 | 39 | 5 | 45 | 62 | 74 | |
| | 12 | 5 | 39 | 45 | 62 | 74 | |
| 5,0 | 12 | 5 | 39 | 45 | 62 | 74 | PASS 5 |
| | 5 | 12 | 39 | 45 | 62 | 74 | |

## TO ARRANGE 6 ELEMENTS OF AN ARRAY IN ASCENDING ORDER

- Requires exactly 5 Passes.
- Requires exactly 15 Comparisons.
- Requires 9 Swaps.

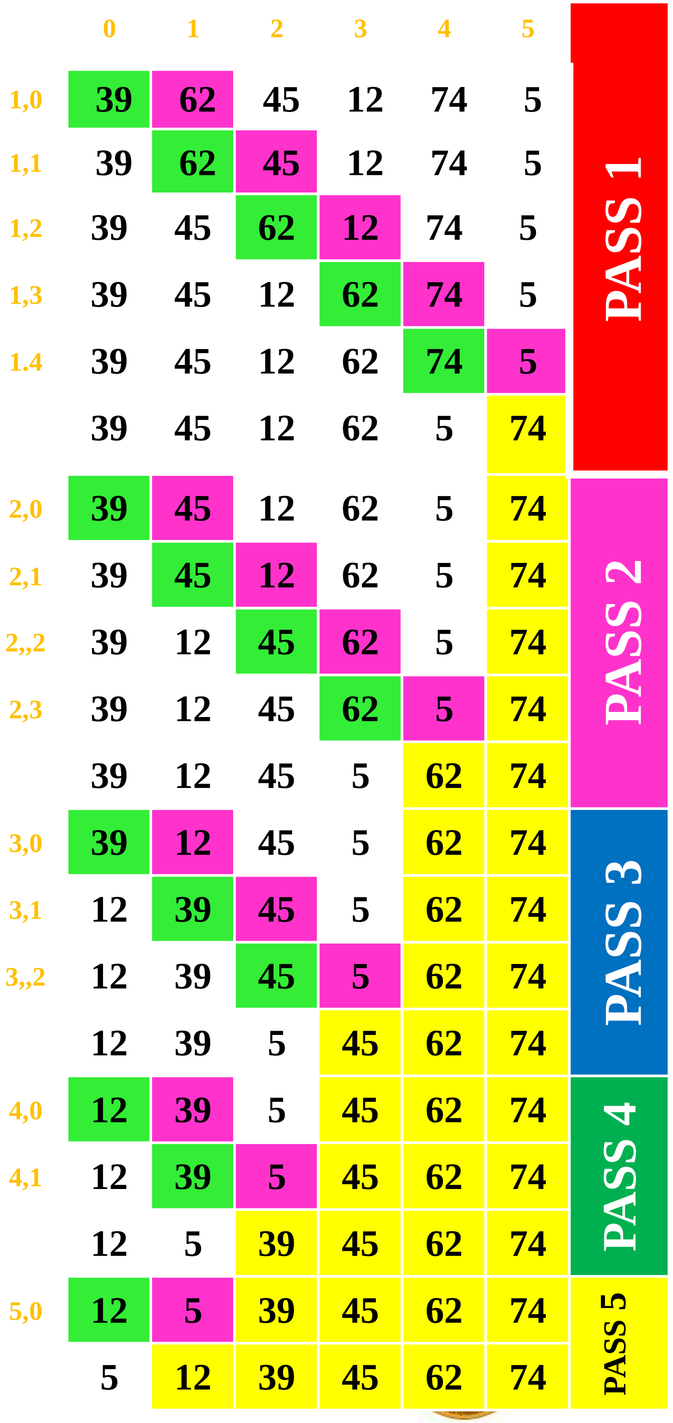
| Pass | Comparisons | Swaps |
|---|---|---|
| Pass 1 | 5 | 3 |
| Pass 2 | 4 | 2 |
| Pass3 | 3 | 2 |
| Pass 4 | 2 | 1 |
| Pass 5 | 1 | 1 |
| Total | 15 | 9 |

# BUBBLE SORT PROGRAM

```
#include<stdio.h>
 void main()
{
            int n, i, j, temp, a[100];
            printf("Enter the value for n ");
            scanf("%d",&n);
           printf("Enter %d elements into array\n",n);
            for(i=0;i<n;i++)
            {
                        scanf("%d",&a[i]);
            }
            printf("The unsorted array is\n");
            for(i=0;i<n;i++)
            {
                        printf("%d\t",a[i]);
           }
```

# BUBBLE SORT PROGRAM

```
        for(i=1;i<n;i++)
        {
            for(j=0;j<(n-i);j++)
             {
                        if(a[j]>a[j+1])
                        {
                              temp=a[j];
                              a[j]=a[j+1];
                              a[j+1]=temp;
                        }
             }
       }
      printf("\nThe sorted array is\n");
      for(i=0;i<n;i++)
      {
             printf("%d\t",a[i]);
       }
}
```

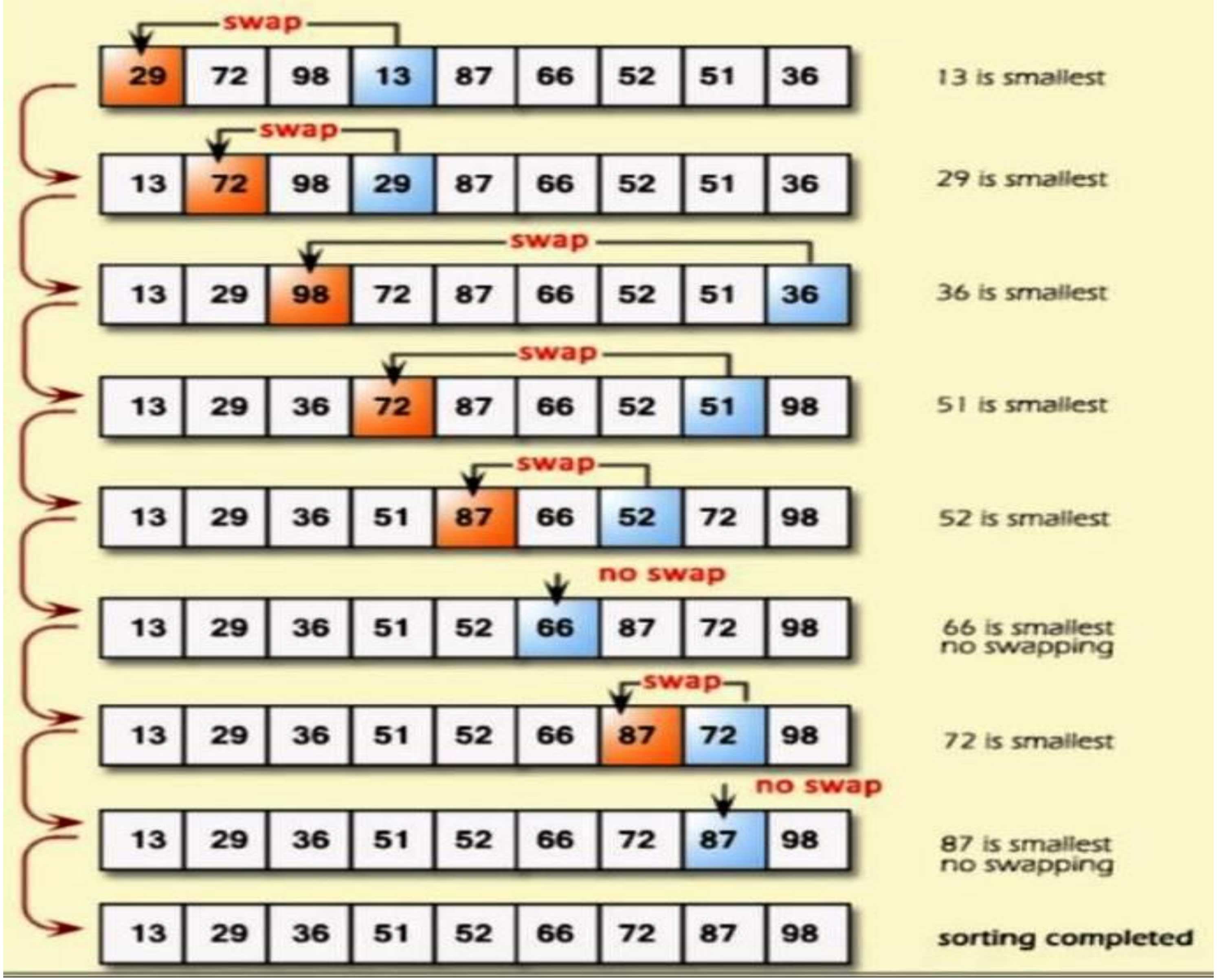


| | 0 | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|
| 1,0 | 39 | 62 | 45 | 12 | 74 | 5 | PASS 1 |
| 1,1 | 39 | 62 | 45 | 12 | 74 | 5 | |
| 1,2 | 39 | 45 | 62 | 12 | 74 | 5 | |
| 1,3 | 39 | 45 | 12 | 62 | 74 | 5 | |
| 1.4 | 39 | 45 | 12 | 62 | 74 | 5 | |
| | 39 | 45 | 12 | 62 | 5 | 74 | |
| 2,0 | 39 | 45 | 12 | 62 | 5 | 74 | PASS 2 |
| 2,1 | 39 | 45 | 12 | 62 | 5 | 74 | |
| 2,,2 | 39 | 12 | 45 | 62 | 5 | 74 | |
| 2,3 | 39 | 12 | 45 | 62 | 5 | 74 | |
| | 39 | 12 | 45 | 5 | 62 | 74 | |
| 3,0 | 39 | 12 | 45 | 5 | 62 | 74 | PASS 3 |
| 3,1 | 12 | 39 | 45 | 5 | 62 | 74 | |
| 3,,2 | 12 | 39 | 45 | 5 | 62 | 74 | |
| | 12 | 39 | 5 | 45 | 62 | 74 | |
| 4,0 | 12 | 39 | 5 | 45 | 62 | 74 | PASS 4 |
| 4,1 | 12 | 39 | 5 | 45 | 62 | 74 | |
| | 12 | 5 | 39 | 45 | 62 | 74 | |
| 5,0 | 12 | 5 | 39 | 45 | 62 | 74 | PASS 5 |
| | 5 | 12 | 39 | 45 | 62 | 74 | |

# SELECTION SORT ALGORITHM

This image cannot currently be displayed.

# SELECTION SORT ALGORITHM



29 72 98 13 87 66 52 51 36 — 13 is smallest

13 72 98 29 87 66 52 51 36 — 29 is smallest

13 29 98 72 87 66 52 51 36 — 36 is smallest

13 29 36 72 87 66 52 51 98 — 51 is smallest

13 29 36 51 87 66 52 72 98 — 52 is smallest

13 29 36 51 52 66 87 72 98 — 66 is smallest no swapping

13 29 36 51 52 66 87 72 98 — 72 is smallest

13 29 36 51 52 66 72 87 98 — 87 is smallest no swapping

13 29 36 51 52 66 72 87 98 — sorting completed

# SELECTION SORT PROGRAM

```
#include <stdio.h>
void main()
{
        int i, j, n, pos, temp, a[100];
        printf("Enter number of elements");
        scanf("%d",&n);
        printf("Enter elements\n");
        for(i=0;i<n;i++)
        {
          scanf("%d",&a[i]);
        }
```

# SELECTION SORT PROGRAM

```
    for (i=0;i< n-1;i++)
     {
        pos=i;    // Position for fixing smallest element of an array
        for(j=i+1;j<n;j++)
        {
            if(a[pos]>a[j]) //Searching for smallest element from rest of array
            pos=j;
        }
        if(pos!=i)   //Exchange only when smallest element's position is different
        {
            temp=a[i];
            a[i]=a[pos];
            a[pos]=temp;
        }
     }
     printf("Enter elements\n");
     for(i=0;i<n;i++)
         printf("%d\t",a[i]);
}
```

# TWO DIMENSION ARRAY ( 2D ARRAY)

Fixed size sequenced collection of elements of same data types having two subscripts (indices) is called two dimension array.

- First subscript (index) for number of rows.
- Second subscript (index) for number of columns

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

## Declaration

Syntax `datatype  arrayname[Number of rows][Number of columns];`

datatype - may be int, float, char, double etc.

Number of rows/columns – rows x columns= elements in an array.

arrayname – is a valid names like identifiers.

Example 1    int a[3][2];

Example 2    float avg[2][2];

Example 3    char name[5][30];  - can store 5 names with 30 characters each

# INITIALIZATION OF TWO DIMENSION ARRAY

Initialization of an array means storing the elements in an array.

Array can be initialized in two different fashions.

1. Compilation time initialization
2. Run time initialization

## 1. Compile time initialization

Storing values in an array when they are declared.

Syntax

```
datatype  arrayname[Number of rows][Number of columns] ={V1, V2,  V3…Vn};
```

Values are separated by commas.

Example 1 `int a[2][3]={{10,20,30},{40,50,60}};`

Example 2 `int a[2][3]={10,20,30,40,50,60};`

# INITIALIZATION OF TWO DIMENSION ARRAY

## a) Basic Initialization

Specifying the number of rows and number of columns of an array and its values during declaration.

Example

```
int a[2][3] = {10,20,30,40,50,60};
int a[2][3]={{10,20,30},{40,50,60}};
```

| a[0][0] | a[0][1] | a[0][2] |
|---|---|---|
| **10** | **20** | **30** |
| **40** | **50** | **60** |
| a[1][0] | a[1][1] | a[1][2] |

## b) Without size Initialization

While initializing the two dimension arrays during their declaration, the row number is optional. But column number is mandatory.

Example

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

| a[0][0] | a[0][1] | a[0][2] |
|---|---|---|
| **10** | **20** | **30** |
| **40** | **50** | **60** |
| a[1][0] | a[1][1] | a[1][2] |

# INITIALIZATION OF TWO DIMENSION ARRAY

## c) Partial Initialization

Storing less number of elements in an array than its rows x columns value.

Example

```
int a[2][3] = {10,20,30,40};
            or
int a[2][3]={{10,20,30},{40}};
            or
int a[ ][3] = {10,20,30,40};
```

| a[0][0] | a[0][1] | a[0][2] |
|---|---|---|
| **10** | **20** | **30** |
| **40** | **0** | **0** |
| a[1][0] | a[1][1] | a[1][2] |

## d) Null Initialization

Storing same values in all locations of an array.

Example `int a[2][3]={0};`

| a[0][0] | a[0][1] | a[0][2] |
|---|---|---|
| **0** | **0** | **0** |
| **0** | **0** | **0** |
| a[1][0] | a[1][1] | a[1][2] |

# INITIALIZATION OF TWO DIMENSION ARRAY

**2. Run time initialization**

Initialization of an array when program is under execution.

Example 1 int a[2][2];

a[0][0] =10; a[0][1] =20; a[1][0] =30; a[1][1]=40;

Using for loop to initialize elements of an array

(**READING ELEMENTS INTO AN 2D ARRAY**)

Example 2

```
int i, j, a[10][10];
………….
………….
for(i=0; i<2; i++)
{
        for(j=0; j<2; j++)
         {
                scanf("%d", &a[i][j]);
         }
}
```

**TRACE**

i=0<br>
0<2<br>
------------------<br>
j=0<br>
0<2<br>
a[0][0] = ?<br>
j=0+1=1<br>
------------------<br>
1<2<br>
a[0][1] = ?<br>
j=1+1=2<br>
------------------<br>
2<2<br>
i=0+1=1<br>
------------------

1<2<br>
-----------------<br>
j=0<br>
0<2<br>
a[1][0] = ?<br>
j=0+1=1<br>
------------------<br>
1<2<br>
a[1][1] = ?<br>
j=1+1=2<br>
------------------<br>
2<2<br>
i=1+1=2<br>
------------------<br>
2<2

# PRINTING TWO DIMENSION ARRAY

Example

```
int i, j, a[10][10];
………….
………….
for(i=0; i<2; i++)
{
        for(j=0; j<2; j++)
        {
                printf(“%d\t”, a[i][j]);
        }
        printf(“\n”);
}
```

| a[0][0] | a[0][1] |
|---|---|
| **10** | **20** |
| **30** | **40** |
| a[1][0] | a[1][1] |

**TRACE**

i=0
0<2
------------------
j=0
0<2
a[0][0] = 10
j=0+1=1
------------------
1<2
a[0][1] = 20
j=1+1=2
------------------
2<2
i=0+1=1
------------------

1<2
------------------
j=0
0<2
a[1][0] = 30
j=0+1=1
------------------
1<2
a[1][1] = 40
j=1+1=2
------------------
2<2
i=1+1=2
------------------
2<2

Output

```
10        20
30        40
```

# Write a C program to find sum of two matrices

```
#include<stdio.h>
void main()
{
    int m, n, i, j, a[10][10], b[10][10], c[10][10];
    printf("Enter the order of matrices\n");
    scanf("%d%d",&m, &n);
    printf("Enter elements of matrix A\n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("Enter elements of matrix B\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("Sum of two matrices \n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d\t",c[i][j]);
        }
        printf("\n");
    }
}
```

# Write a C program to find sum of two matrices

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

m x n = 2 x 2

| a[0][0] | a[0][1] |
|---|---|
| **10** | **20** |
| **30** | **40** |
| a[1][0] | a[1][1] |

m x n =2 x 2

| b[0][0] | b[0][1] |
|---|---|
| **50** | **60** |
| **70** | **80** |
| b[1][0] | b[1][1] |

| c[0][0] | c[0][1] |
|---|---|
| **60** | **80** |
| **100** | **120** |
| c[1][0] | c[1][1] |

**TRACE**

i=0

0<2

-------------------------

j=0

0<2

c[0][0] = 10+50=60

j=0+1=1

-------------------------

1<2

c[0][1] =20+60=80

j=1+1=2

-------------------------

2<2

-------------------------

i=0+1=1

1<2

-----------------------------

j=0

0<2

c[1][0] = 30+70=100

j=0+1=1

-----------------------------

1<2

c[1][1] = 40+80=120

j=1+1=2

-----------------------------

2<2

-----------------------------

i=1+1=2

2<2

# Write a C program to find transpose of given matrix.

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

# TRACING TRANSPOSE OF MATRIX.

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

m x n = 2 x 2

a[0][0] a[0][1]

| | |
|---|---|
| 10 | 20 |
| 30 | 40 |

a[1][0] a[1][1]

b[0][0] b[0][1]

| | |
|---|---|
| 10 | 30 |
| 20 | 40 |

b[1][0] b[1][1]

**TRACE**

i=0
0<2
-----------------
j=0
0<2
b[0][0] =10
j=0+1=1
-----------------
1<2
b[1][0] =20
j=1+1=2
-----------------
2<2
-----------------

i=0+1=1
1<2
-----------------
j=0
0<2
b[0][1] =30
j=0+1=1
-----------------
1<2
b[1][1] =40
j=1+1=2
-----------------
2<2
-----------------
i=1+1=2
2<2

# CHARACTER ARRAYS AND STRINGS

String is zero or more characters enclosed within " " (quotation marks) and terminated by NULL character \0

Example 1 "VCET"

| | | | | |
|---|---|---|---|---|
| V | C | E | T | \0 |

Delimiter

Example 2 "123"

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | \0 |

Example 3 "1+2-3*5"

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | + | 2 | - | 3 | * | 5 | \0 |

Example 4 ""

| |
|---|
| \0 |

**Length of a string**

Number of characters till the delimiter \0. Exclude \0 (NULL character).

| | | |
|---|---|---|
| Example 1 | "VCET“ | Where length is 4 |
| Example 2 | "123“ | Where length is 3 |
| Example 3 | "1+2-3*5“ | Where length is 7 |
| Example 4 | "“ | Where length is 0 |

# CHARACTER ARRAYS AND STRINGS

## Declaring String Variable

Syntax

```
char stringname[size];
```

- Size of the string is always number of characters plus 1.
- Because string ends with delimiter \0.
- **Array of character not ends with \0 (NULL character).**

Example char str[5];

Where str can store up to 5 characters.

Syntax

```
char stringname[number of strings][number of characters];
```

Example char str[5][30];

Where str can store up to 5 strings and each string can have characters up to 30.

# INITIALIZATION OF STRINGS

Initializing string means storing characters / strings in an character array.

Strings can be initialized in two different fashions.

1. Compilation time initialization
2. Run time initialization

**1. Compile time initialization**

Storing characters in character array when they are declared.

Syntax

```
char stringname[size]={List of characters};
```

Example 1 `char str[6]={‘I’,’N’,’D’,’I’,’A’,’\0’};`

Example 2 `char str[ ]={‘I’,’N’,’D’,’I’,’A’,’\0’};`

This type of initialization treats the string as an array of character and assigns value to individual position. So we must add NULL character at the end.

Example 3 `char str[6]=“INDIA”;`

Example 4 `char str[ ]=“INDIA”;`

In above example 3 & 4, the NULL character is automatically added by the compiler.

# INITIALIZATION OF STRINGS

Storing strings in two dimensions of character array when they are declared.

Syntax

```
char stringname[number of strings][number of characters]={List of strings};
```

Example 1

```
char str[2][4]={{‘A’,’B’,’C’,’\0’}, {‘X’,’Y’,’\0’}};
```

or

```
char str[2][4]={“ABC”,”XY”};
```

Example 2

```
char str[ ][4]={{‘A’,’B’,’C’,’\0’}, {‘X’,’Y’,’\0’}};
```

or

```
char str[ ][4]={“ABC”,”XY”};
```

# INITIALIZATION OF STRINGS

## 2. Run time initialization (READING A STRING)

Initialization of character array when program is under execution.

### a) scanf( )

Syntax

```
scanf("Control String",argument1, arguement2,…);
```

- The control string %s is used without prefix & in arguments.
- While reading a string, special character space is not allowed.

i.e, user can read a word from keyboard. A sentence is not possible to read.

Example 1

```
char str[100];
………
scanf("%s", str);
```

Example 2

```
char str[5][20];
int i;
………
for(i=0; i<5; i++)
{
        scanf("%s", str[i]);
}
```

# INITIALIZATION OF STRINGS

## Scan set conversion code

Syntax

```
scanf("%width[allowed characters]", argument);
```

Below example is used to read a sentence (line).

Example 1

```
char str[100];
………
scanf("%[^\n]", str);
```

The below example is used to read a string containing characters mentioned inside the bracket and their combinations. The strings containing any other characters than mentioned inside the bracket are rejected.

Example 2

```
char str[100];
………
scanf(“%[126*-ab]”, str);
```

| Valid | Invalid |
|---|---|
| 12666 | 110 |
| 2 | a+b |
| ***** | vcet |
| ---*** | ab_ab |
| 126*-ab | 1234 |
| abba | a/c |

# INITIALIZATION OF STRINGS

## 2. Run time initialization (READING A STRING)

### b) gets( )

Syntax

```
gets(stringname);
```

- This is unformatted input statement to read a string.
- s in gets() stands for string.
- control string is not used here.
- While reading a string, special character space is allowed.
- User can read a word or a sentence from keyboard.

Example 1

```
char str[100];
………
gets(str);
```

# PRINTING A STRING

The formatted output statement **printf( )** is used to display a string .

Which we have already discussed in MODULE 2 .

## puts( )

Syntax

```
puts(stringname);
```

- This is unformatted output statement to display a string.
- s in gets() stands for string.
- control string is not used here.

Example 1

```
char str[100];

………

puts(str);
```

```
#include <stdio.h>
void main()
{
   char str[100];
   puts("Enter a string");
   gets(str);
   puts("Entered string is ");
   puts(str);
}
```

# READING A CHARACTER

## a) getch( )

Syntax

```
charactername=getch( );
```

- This is unformatted input statement to read a character.
- ch in getch( ) stands for character.
- control string is not used here.
- Reads character without echo.

(The read character is not going to display in the screen)

- After reading character enter key press is not required.

Example

```
char ch;

………

ch=getch();
```

# READING A CHARACTER

## b) getchar( )

Syntax

```
charactername=getchar( );
```

- This is unformatted input statement to read a character.
- char in getchar( ) stands for character.
- control string is not used here.
- Reads character with echo.

  (The read character will display in the screen)
- After reading character enter key press is required.

Example

```
char ch;

 ………

 ch=getchar();
```

# PRINTING A CHARACTER

## a) putch( )

Syntax

```
putch(charactername);
```

- This is unformatted output statement to display a character.
- ch in getch( ) stands for character.
- control string is not used here.

Example

```
char ch;

 ………

 putch(ch);
```

# PRINTING A CHARACTER

## b) putchar( )

Syntax

```
putchar(charactername);
```

- This is unformatted output statement to display a character.
- char in getch( ) stands for character.
- control string is not used here.

Example

```
char ch;

 .........

 putchar(ch);
```

# STRING OPERATIONS

1. **Reading and writing strings**

   scanf( )   printf( )  gets( )  puts( )

2. **Finding length of a string**

   strlen( )

3. **Converting a string**

   strlwr( )      strupr( )

4. **Reverse a string**

   strrev( )

5. **Copy a string**

   strcpy( )   strncpy( )

6. **Concatenate (Combining/Joining) strings**

   strcat( )    strncat( )

7. **Compare strings**

   strcmp( )    strncmp( )

8. **Extracting substring**

   strstr( )

# STRING MANIPULATIONS FUNCTIONS

- Below listed functions are string manipulation functions.
- Because they perform some operations on given strings.
- They are library (built-in/predefined) functions.
- The header file included is string.h

1. **Finding length of a string**

   strlen( )

2. **Converting a string**

   strlwr( )     strupr( )

3. **Reverse a string**

   strrev( )

4. **Copy a string**

   strcpy( )   strncpy( )

5. **Concatenate (Combining/Joining) strings**

   strcat( )   strncat( )

6. **Compare strings**

   strcmp( )   strncmp( )

7. **Extracting substring**

   strstr( )

# 1. FINDING LENGTH OF A STRING strlen( )

- This library function returns length (number of characters) of given string.
- Only one parameter is used in this library function.
- The header file to be included is string.h

Syntax

```
strlen(stringname);
```

Example

strlen(“India”);

Above statement returns 5

```
#include <stdio.h>
#include<string.h>
void main()
{
    char str[ ]= “Programming”;
    printf(“The length of string is %d\n", strlen(str));
}
```

Output

```
The length of string is 11
```

# 2. CONVERTING A STRING

There two string converting function strlwr( ) and strupr( )

## strlwr( )

- This library function returns given string in lowercase.
- Only one parameter is used in this library function.
- The header file to be included is string.h

Syntax

```
strlwr(stringname);
```

Example

strlwr(“INDIA”);

Above statement returns india

```
#include <stdio.h>
#include<string.h>
void main()
{
   char str[ ]= “GOOD”;
   printf(“The converted string is %s\n",strlwr(str));
}
```

Output

```
The converted string is good
```

# 2. CONVERTING A STRING

## strupr()

- This library function returns given string in uppercase.
- Only one parameter is used in this library function.
- The header file to be included is string.h

Syntax

```
strupr(stringname);
```

Example

strupr(“india”);

Above statement returns INDIA

```
#include <stdio.h>
#include<string.h>
void main()
{
   char str[ ]= “good”;
   printf(“The converted string is %s\n",strupr(str));
}
```

Output

```
The converted string is GOOD
```

# 3. REVERSE A STRING strrev( )

- This library function returns given string in reverse.
- Only one parameter is used in this library function.
- The header file to be included is string.h

Syntax

```
strrev(stringname);
```

Example

strrev(“india”);

Above statement returns aidni

```
#include <stdio.h>
#include<string.h>
void main()
{
    char str[ ]= “good”;
    printf(“Reversed string is %s\n",strrev(str));
}
```

Output

```
Revered string is doog
```

# Write a C program to find length of given string without using built-in function strlen( )

```
#include <stdio.h>
void main()
{
        char str[100];
        int i;
        printf("Enter a string\n");
        scanf("%s",str);
        i=0;
        while(str[i]!='\0')
        {
             i++;
        }
        printf("Length of the string is %d\n",i);
}
```

| C | P | S | \0 |
|---|---|---|---|
| str[0] | str[1] | str[2] | str[3] |

```
for(i=0; str[i] !=‘\0’; i++);
```

**TRACE**

i=0

str[0]≠ \0

C ≠ \0

i=0+1=1

------------

str[1]≠ \0

P ≠ \0

i=1+1=2

------------

str[2]≠ \0

S ≠ \0

i=2+1=3

------------

str[3]≠ \0

\0 ≠ \0

# Write a C program to reverse a given string without using built-in function strrev( )

```
 #include <stdio.h>
 #include <string.h>
  void main()
 {
      char s[100], r[100];
      int i, j=0, len;
      printf("Enter any String \n");
      gets(s);
      len = strlen(s);
      for (i = len - 1; i >= 0; i--)
      {
          r[j] = s[i];
          j++;
      }
      r[j] = '\0';
      printf("\n String after Reversing = %s", r);
 }
```

| | | | |
|---|---|---|---|
| P | A | Y | \0 |
| s[0] | s[1] | s[2] | s[3] |

| | | | |
|---|---|---|---|
| Y | | | |
| r[0] | r[1] | r[2] | r[3] |

| | | | |
|---|---|---|---|
| Y | A | | |
| r[0] | r[1] | r[2] | r[3] |

| | | | |
|---|---|---|---|
| Y | A | P | |
| r[0] | r[1] | r[2] | r[3] |

| | | | |
|---|---|---|---|
| P | A | Y | \0 |
| r[0] | r[1] | r[2] | r[3] |

**TRACE**

j=0, len=3
i=2, 2>=0
r[0]=s[2]
r[0]=Y
j=0+1=1
i=2-1=1
------------
1>=0
r[1]=s[1]
r[1]=A
j=1+1=2
i=1-1=0
------------
0>=0
r[2]=s[0]
r[2]=P
j=2+1=3
i=0-1=-1
------------
-1>=0

# Reverse a given string without using  built-in function strrev( )

```
for (i = len - 1; i >= 0; i--)
{
        r[j++] = s[i];
}
```

```
for (i = len - 1; i >= 0; i--,j++)
{
        r[j] = s[i];
}
```

```
for (i = len - 1; i >= 0; i--)
{
        r[j] = s[i];
        j++;
}
```

# 4. COPY A STRING strcpy( )

- This library function returns the destination string after copying the source string to it.
- Two parameters are used in this library function.
- The header file to be included is string.h

Syntax

```
strcpy(destinationstring, sourcestring);
```

Example

char s1[ ] = “Morning”

char s2[ ]= “Good”

strcpy(s1,s2)

Above statement returns

s1=Good

```
#include <stdio.h>
#include<string.h>
void main()
{
    char s1[10],s2[10];
    printf("Enter first string\n");
    gets(s1);
    printf("Enter second string\n");
    gets(s2);
    strcpy(s1,s2);
    printf("Copied Sting is = %s",s1);
}
```

Output

```
Enter first string
Programming
Enter second string
Problem
Copied Sting is = Problem
```

# 4. COPY A STRING strncpy( )

- This library function returns the destination string after copying the specified number of characters from source string to it.

Syntax

```
strncpy(destinationstring, sourcestring, no.of characters);
```

- Three parameters are used in this library function.
- The header file to be included is string.h

Example

char s1[ ] = “Morning”

char s2[ ]= “Good”

strncpy(s1,s2,2)

Above statement returns

s1=Go

```
#include <stdio.h>
#include<string.h>
void main()
{
    char s1[10],s2[10];
    printf("Enter first string\n");
    gets(s1);
    printf("Enter second string\n");
    gets(s2);
    strncpy(s1,s2,4);
    printf("Copied Sting is = %s",s1);
}
```

Output

```
Enter first string
Programming
Enter second string
Problem
Copied Sting is = Prob
```

# 5. CONCATENATE STRINGS strcat( )

- This library function returns the destination string after joining it with source string.
- Two parameters are used in this library function.
- The header file to be included is string.h

Syntax

```
strcat(destinationstring, sourcestring);
```

Example

```
char s1[ ] = “Morning”
char s2[ ]= “Good”
strcat(s1,s2)
```

Above statement returns

s1=MorningGood

```
#include <stdio.h>
#include<string.h>
void main()
{
    char s1[10],s2[10];
    printf("Enter first string\n");
    gets(s1);
    printf("Enter second string\n");
    gets(s2);
    strcat(s1,s2);
    printf("Combined Sting is= %s",s1);
}
```

Output

```
Enter first string
Programming
Enter second string
Problem
Combined Sting is=ProgrammingProblem
```

# 5. CONCATENATE STRINGS strncat( )

- This library function returns the destination string after joining it with specified number of characters from source string to it.

Syntax

```
strncat(destinationstring, sourcestring, no.of characters);
```

- Three parameters are used in this library function.
- The header file to be included is string.h

Example

char s1[ ] = “Morning”

char s2[ ]= “Good”

strncat(s1,s2,2)

Above statement returns

s1=MorningGo

```
#include <stdio.h>
#include<string.h>
void main()
{
    char s1[10],s2[10];
    printf("Enter first string\n");
    gets(s1);
    printf("Enter second string\n");
    gets(s2);
    strcat(s1,s2,4);
    printf("Combined Sting is = %s",s1);
}
```

Output

```
Enter first string
Programming
Enter second string
Problem
Combined Sting is = ProgrammingProb
```

# 6. COMPARE STRINGS strcmp( )

- This library function compares destination string with source string.
- It returns an integer value by subtracting ASCII values of source string from destination string.
- This library function returns one among the following three values.
  - An positive integer – When source string ASCII value is smaller than the ASCII value of destination string.
  - An negative integer – When source string ASCII value is greater than the ASCII value of destination string.
  - Zero – When source string ASCII value is equal to the ASCII value of destination string.
- There are TWO parameters in this library function.
- The header file to be included is string.h

**Syntax**

```
strcmp(destinationstring, sourcestring);
```

# 6. COMPARE STRINGS strcmp( )

Syntax

```
strcmp(destinationstring, sourcestring);
```

Example 1

char s1[ ] = “AB”

char s2[ ]= “AE”

strcmp(s1,s2)

Above statement returns -3

i.e., s1<s2

Example 2

char s1[ ] = “ZA”

char s2[ ]= “AF”

strcmp(s1,s2)

Above statement returns 25

i.e., s1>s2

Example 3

char s1[ ] = “AA”

char s2[ ]= “AA”

strcmp(s1,s2)

Above statement returns 0

i.e., s1=s2

# Write a C program to compare given two strings

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[10],s2[10];
    printf("Enter first string\n");
    gets(s1);
    printf("Enter second string\n");
    gets(s2);
```

```
    if(strcmp(s1,s2)==0)
    {
       printf(“String are equal”);
    }
    else if(strcmp(s1,s2)>0)
         {
           printf(“%s is biggest”,s1);
         }
         else
         {
           printf(“%s is  biggest”,s2);
         }
}
```

Output

```
Enter first string
Age
Enter second string
Eat
Eat is biggest
```

# 6. COMPARE STRINGS strncmp( )

- This library function compares destination string with source string by specified number of characters from it.
- It returns an integer value by subtracting ASCII values of source string from destination string.
- This library function returns one among the following three values.
  - An positive integer – When source string ASCII value is smaller than the ASCII value of destination string.
  - An negative integer – When source string ASCII value is greater than the ASCII value of destination string.
  - Zero – When source string ASCII value is equal to the ASCII value of destination string.
- There are THREE parameters in this library function.
- The header file to be included is string.h

**Syntax**

```
strncmp(destinationstring, sourcestring,no.of characters);
```

# 6. COMPARE STRINGS strncmp( )

**Syntax** strcmp(destinationstring, sourcestring,no.of characters);

Example 1

char s1[ ] = “ABC”

char s2[ ]= “AEF”

strncmp(s1,s2,2)

Above statement returns -3

i.e., s1<s2

Example 2

char s1[ ] = “ZA”

char s2[ ]= “AF”

strncmp(s1,s2,1)

Above statement returns 25

i.e., s1>s2

Example 3

char s1[ ] = “AACD”

char s2[ ]= “AA”

strncmp(s1,s2,2)

Above statement returns 0

i.e., s1=s2

# Write a C program that read a sentence and print frequency of vowels and total count of consonants.

```
#include<stdio.h>
#include<ctype.h>
void main()
{
    char line[100];
    int i, cv=0,cc=0;
    printf("Enter a sentence\n");
    scanf("%[^\n]",line);
```

```
for(i=0;line[i]!='\0';i++)
{
    if(isalpha(line[i]))
    {
        switch(tolower(line[i]))
       {
               case 'a':
               case 'e':
               case 'i':
               case 'o':
               case 'u': cv++;
                             break;
              default : cc++;
         }
     }
 }
  printf(“No. of Vowels=%d\n",cv);
  printf(“No. of Consonants=%d\n",cc);
}
```

Output

```
Enter a sentence
I LOVE INDIA
No. of Vowels=6
No. of Consonants=4
```

# MODULE 3 REVIEW QUESTIONS

1. Define array. How to declare and initialize one dimensional and two dimensional array.
2. Explain string manipulation functions.
3. Without using built in functions write a program to find
   i) String length ii) Copy string iii) Compare strings iv) Concatenate strings
4. Write a C program to find the largest among n numbers using arrays.
5. Write a C program to find sum of two one dimensional arrays and store the result in another array.
6. Write a C program to find sum and average of given integers.
7. Write a C program to find transpose of given matrix.
8. Write a C program to find trace of given matrix.
9. Write a C program to print Fibonacci numbers up to given limit using array.
10. Write a C program to evaluate the given polynomial $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ for given value of x and the coefficients using Horner's method.
11. Write a C program to find  sum of each row and sum of each columns of given matrix.