PROGRAMMING IN C

UNIT II ARRAYS AND STRINGS

Introduction to Arrays: Declaration, Initialization – One dimensional array – Example Program: Computing Mean, Median and Mode - Two dimensional arrays – Example Program: Matrix Operations (Addition, Scaling, Determinant and Transpose) - String operations: length, compare, concatenate, copy – Selection sort, linear and binary search

APEC

ARRAYS:

In C language, arrays are referred as structured data types. An array is defined as finite ordered collection of homogenous data, stored in contiguous memory locations.

- Ordered means data must be stored in continuous memory addresses.
- Homogenous means data must be of similar data type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

Array Declaration:

General syntax to create an array:

Syntax: <data-type> <array_name> [size];

Example: int a[10]

Arrays are referenced with the help of the index values. [] notation to represent the variable as an array. Assume that the array contains "n" integers, then the first element are indexed with the "0" value and the last element are indexed with "n-1" value.

Syntax for creating an array with size and initial values:

Syntax: <data-type> <array_name> [size] = [(Value1, value2…};

Example: int a[3] = {1,2,3}

Syntax for creating an array without specifying size:

Syntax: <data-type> <array_name> [] = [(Value1, value2…};

Example: int a[] = {1,2,3}

One Dimensional Array

Example(Array Initialization):

int a[5] = {100,200,300,400,500}

a→ array name

5 → Size of the array. It means, it can hold values from 0 to 4

Each individual element in the array is accessed as :

a[0] = 100
a[1] = 200
a[2] = 300
a[3] = 400
a[4] = 500

The above declaration allocates, 5 continuous memory locations of 2 bytes each with the name "a"

Index →	a[0]	a[1]	a[2]	a[3]	a[4]
Values →	100	200	300	400	500

Figure 3.1 Representing the Array Structure

Types of Arrays:

APE

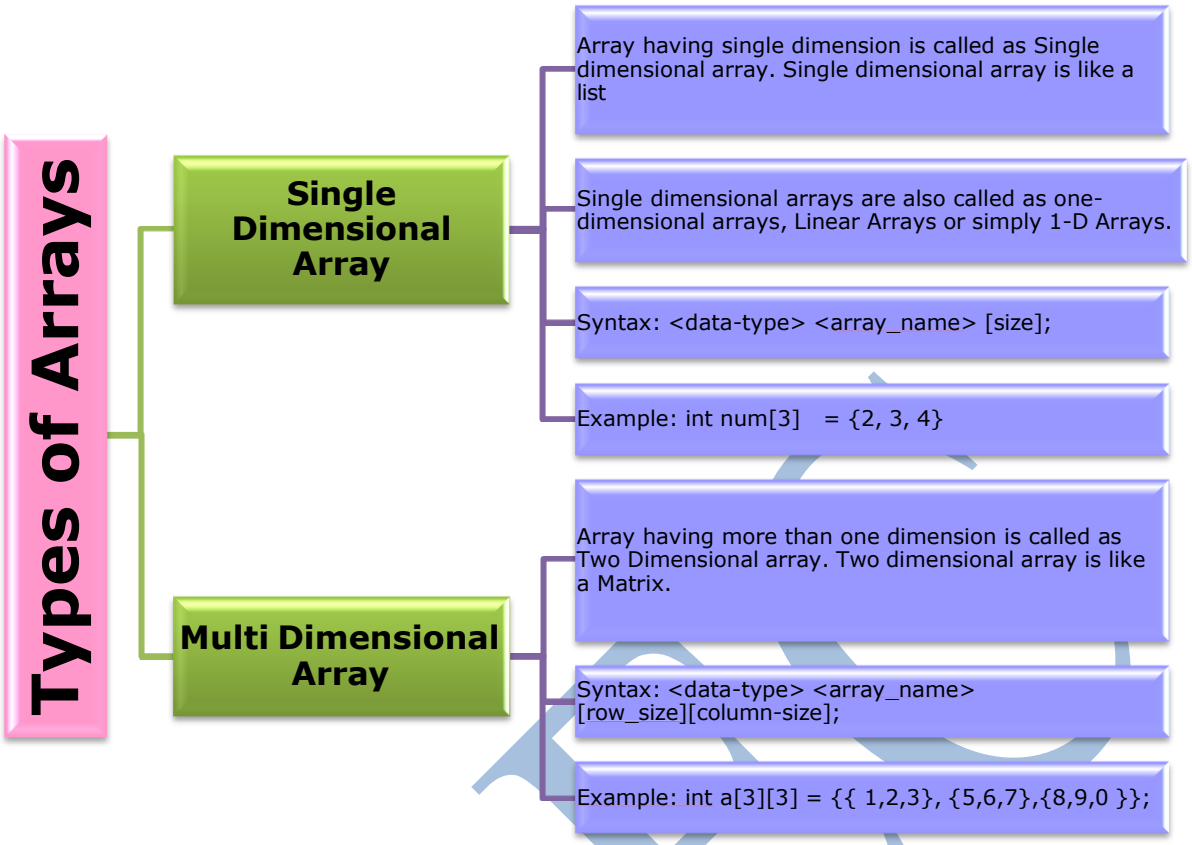


Figure 3.1 Types of Arrays

Examples of C Array Declaration

- *Declaring 1D array*
  - Example: int num[6]
  - In the example, the integer array of size 6 is declared and the array name is "num". It is one dimensional since it"s having only 1 subscript/index.
  - If the 1D array is initialized with integer values, it will look like the following.

    int num[6] = {10,43,25,65,34,49}

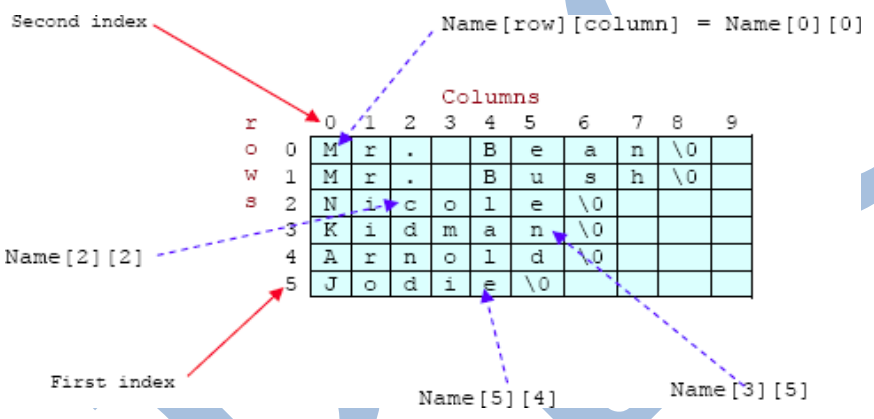
0	10
1	43
2	25
3	65
4	34
5	49

  - *Some more examples:*

    int marks[4]={ 67, 87, 56, 77 }; //integer array initialization

    float area[5]={ 23.4, 6.8, 5.5 }; //float array initialization

- *Declaring 2D array*
    - Example: char name[6][10]
    - Here we have two indexes/subscripts. These two indexes are referred as rows and columns. [6] refers to rows and [10] refers to columns.
        - [6] refers to “How many names?” and [10] refers to “How many characters in each name?”.
    - If the 2D array is initialized with string values, it will look like the following. char Name[6][10] = {"Mr. Bean", "Mr. Bush", "Nicole", "Kidman", "Arnold", "Jodie"};



C Array Declaration tells to Compiler about

- Type of the Array
- Name of the Array
- Size of the Array

Type of Array	Array Type	Array Name	Size of the Array
One Dimensional Array (1D Array)	integer	num	[6]
Two Dimensional Array (2D Array)	character	name	[6][10]

/* Simple C Program with Compile time Array Input */

```
#include<stdio.h>
void main()
{
 int i;
 int a[3]={2,3,4};     //Compile time array initialization
 for(i=0 ; i<3 ; i++)
    printf("%d\t",a[i]);
}
```

Execution:

Output

```
2   3   4
```

/* Simple C Program with Run time Array Input */

```
#include<stdio.h>
void main()
{
 int a[3],i;
 printf("Enter array elements : ");
 for(i=0;i<3;i++)
   scanf("%d\t",a[i]);     //Run time array input

 printf("Displaying array elements : ");
 for(i=0;i<3;i++)
   printf("%d\t",a[i]);
}
```

Execution:

Input:

```
Enter array elements : 4    5    6
```

Output

```
Displaying array elements : 4    5    6
```

APEC

/* Simple C Program by getting the size of the array and run time Array Input */

```
#include<stdio.h>
void main()
{
 int a[10],i,n;
 printf("Enter the size of the array :");
 scanf("%d", n);
 printf("Enter array elements : ");
 for(i=0;i<n;i++)
   scanf("%d\t",a[i]);     //Run time array input

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

Execution:

Input:

```
Enter the size of the array : 5
Enter array elements : 43   21   55   76   60
```

Output

```
Displaying array elements : 43   21   55   76   60
```

APEC

```
/* Simple C Program to access the Array elements using index values */

#include<stdio.h>
void main()
{
  int i,n;

  int mark[5] = {19, 10, 8, 17, 9};
  printf("Displaying array elements first time : ");
  for(i=0;i<n;i++)
    printf("%d\t",a[i]);

  // insert different value to third element
  mark[3] = 99;
  printf("Displaying array elements second time : ");
  for(i=0;i<n;i++)
    printf("%d\t",a[i]);

  // take input from the user and insert in third element
  printf("Get the input for mark[2] :")
  scanf("%d", &mark[2]);
  printf("Displaying array elements third time : ");
  for(i=0;i<n;i++)
    printf("%d\t",a[i]);

  // print only first element of an array
  printf("First element in the array : %d", mark[0]);
}
```

Execution:

```
Displaying array elements first time : 19, 10, 8, 17, 9
Displaying array elements second time : 19, 10, 8, 99, 9
Get the input for mark[2] :100
Displaying array elements third time : 19, 10, 100, 99, 9
First element in the array : 19
```

APEC

/* Simple C Program for Two Dimensional Array */

```
#include<stdio.h>
void main()
{
   int i,j;
   // declaring and Initializing array
   int arr[2][2] = {10,20,30,40};
   /* Above array can be initialized as below also
      arr[0][0] = 10; // Initializing array
      arr[0][1] = 20;
      arr[1][0] = 30;
      arr[1][1] = 40; */
   for (i=0;i<2;i++)
      for (j=0;j<2;j++)
         // Accessing variables
         printf("value of arr[%d][%d] : %d\n",i,j,arr[i][j]);
  }
```

Execution:

Output:

```
value of arr[0][0] is 10
value of arr[0][1] is 20
value of arr[1][0] is 30
value of arr[1][1] is 40
```

APEC

/* Simple C Program to find out the sum and average of n integers */

```
#include <stdio.h>
void main()
{
    int avg = 0,sum = 0;
    int a[10];

    printf("Enter the size of the array :");
    scanf("%d", n);
    printf("Enter array elements : ");
    for(i=0;i<n;i++)
       scanf("%d\t",a[i]);     //Run time array input

    for (i=0; i<4;i++)
        sum = sum + a[i];

    avg = sum/n;
    printf("Sum of n numbers : %d", sum);
    printf("Average of n numbers : %d", avg);
}
```

Execution:

Input:

```
Enter the size of the array : 5
Enter array elements : 10    80    20    40    300
```

Output:

```
Sum of n numbers : 450
Average of n numbers : 90
```

APEC

/* C Program to determine the mean of n numbers */

Formula and Example:

$$\text{Mean} = \frac{\text{Sum of all data values}}{\text{Number of data values}}$$
$$= \frac{15+13+18+16+14+17+12}{7}$$
$$= \frac{105}{7}$$
$$= 15$$

```
#include<stdio.h>
void main( )
{
  int n, a[20],i,sum=0;
  float avg;

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

  printf("Enter values \n");
  for(i=1;i<=n;i++)
  {
     scanf("%d",&a[i]);
     sum = sum + a[i];
     }
  avg = sum/n;
  printf("\n\nArithmetic Mean = %f",avg);
  }
```

Execution:

Input:

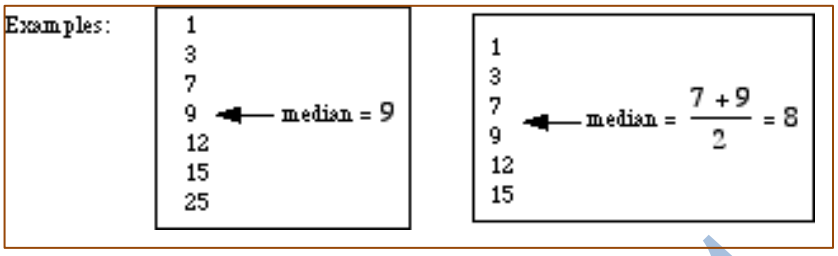
```
Enter number of terms: 6
Enter values:
12   23  56  78   2  56
```

Output

```
Arithmetic Mean = 37.83
```

APEC

/* C Program to sort the list of numbers and determine its median */




```
#include <stdio.h>

void main()
{
    int i,j,n, a[20],t;
    float median;

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

    /* Reading items into array a  */
    printf("Input %d values \n",n);
    for (i = 1; i <= n ; i++)
       scanf("%f", &a[i]);

    /* Sorting begins */
    for (i = 1 ; i <= n-1 ; i++)
       for (j = 1 ; j <= n-i ; j++)
              if (a[j] <= a[j+1])
              { /* Swapping values */

                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
              }

    /* calculation of median  */
     if ( n % 2 == 0)
          median = (a[n/2] + a[n/2+1])/2.0 ;
```

APEC

```
     else
          median = a[n/2 + 1];

    /*  Printing */
    printf("Sorted elements are :\n")
    for (i = 1 ; i <= n ; i++)
           printf("%f  ", a[i]);

   printf("\n\nMedian is %f\n", median);

   }
```

Execution:

Input:

```
Enter the number of items
   6
   Input 6 values
   3  5  8  9  4  6
```

Output:

```
   Sorted Elements are :
   9     8     6     5     4     3
  Median is 5.500000
```

APEC

/* C Program to determine the mode of n numbers */

Calculation of mode:

Example Values

12, 7, 17, 9, 16, 29, 8, 4, 15, 7, 29, 6, 17, 2, 5, 12, 8, 17, 6, 6, 17, 17

Step 1:

Sort all the values from low to high.

2, 4, 5, 6, 6, 6, 7, 7, 8, 9, 12, 12, 15, 16, 17, 17, 17, 17, 17, 29

Step 3:

Find the value that occurs the most times. This is the **mode**.

2,	4,	5,	6, 6, 6,	7, 7,	8,	9,	12, 12,	15,	16,	17, 17, 17, 17, 17,	29
1	1	1	3	2	1	1	2	1	1	5	1

Mode = 17

```
#include <stdio.h>
void main()
{
    int i, j, a[20], maxValue, maxCount = 0, count = 0;

    printf("\nEnter the limit\n");
    scanf("%d", &n);

    printf("Enter the set of numbers\n");
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    for (i = 0; i < n; ++i)
    {
      count = 0;
      for (j = 0; j < n; ++j)
      {
         if (a[j] == a[i])
         ++count;
      }

      if (count > maxCount)
      {
         maxCount = count;
         maxValue = a[i];
      }
```

APEC

```
    }
   printf("Mode = %d ", maxValue);
  }
```

Execution:

Input:

```
Enter the limit
10
Enter the set of numbers
1 5 2 3 4 5 5 6 7 8
```

Output:

```
Mode    =  5
```

APEC

/* C Program to perform Matrix Addition */

Formula:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} \end{bmatrix}$$

Example:

$$\begin{bmatrix} 3 & 6 \\ 4 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 7 \\ 1 & 9 \end{bmatrix} = \begin{bmatrix} 5 & 13 \\ 5 & 15 \end{bmatrix}$$

```
#include<stdio.h>

void main()
{
  int a[3][3],b[3][3],c[3][3],i,j;

  printf("Enter the First matrix->");
  for(i=0;i<3;i++)
      for(j=0;j<3;j++)
          scanf("%d",&a[i][j]);

  printf("\nEnter the Second matrix->");
  for(i=0;i<3;i++)
      for(j=0;j<3;j++)
          scanf("%d",&b[i][j]);

   for(i=0;i<3;i++)
       for(j=0;j<3;j++)
           c[i][j]=a[i][j]+b[i][j];

   printf("\nThe Addition of two matrix is\n");
   for(i=0;i<3;i++)
   {
       printf("\n");
```

APEC

```
        for(j=0;j<3;j++)
             printf("%d\t",c[i][j]);
      }
}
```

Execution:

Input

```
Enter the First matrix->
1 2 3
4 5 6
7 8 9
Enter the Second matrix->
9 8 7
6 5 4
3 2 1
```

Output

```
10  10  10
10  10  10
10  10  10
```

APEC

```
/* C Program to perform Matrix Subtraction     */
```

Formula:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}-b_{11} & a_{12}-b_{12} & a_{13}-b_{13} \\ a_{21}-b_{21} & a_{22}-b_{22} & a_{23}-b_{23} \\ a_{31}-b_{31} & a_{32}-b_{32} & a_{33}-b_{33} \end{bmatrix}$$

Example:

$$\begin{bmatrix} 3 & 6 \\ 4 & 6 \end{bmatrix} - \begin{bmatrix} 2 & 7 \\ 1 & 9 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 3 & -3 \end{bmatrix}$$

```
#include<stdio.h>
void main()
{
  int a[3][3],b[3][3],c[3][3],i,j;

  printf("Enter the First matrix->");
  for(i=0;i<3;i++)
      for(j=0;j<3;j++)
           scanf("%d",&a[i][j]);

  printf("\nEnter the Second matrix->");
  for(i=0;i<3;i++)
      for(j=0;j<3;j++)
           scanf("%d",&b[i][j]);

   for(i=0;i<3;i++)
       for(j=0;j<3;j++)
            c[i][j]=a[i][j]-b[i][j];

   printf("\nThe Subtraction of two matrix is\n");
   for(i=0;i<3;i++)
   {
       printf("\n");
       for(j=0;j<3;j++)
            printf("%d\t",c[i][j]);
    }
}
```

Execution:

Input

Enter the First matrix->

1 2 3

4 5 6

7 8 9

Enter the Second matrix->

9 8 7

6 5 4

3 2 1

Output

The Addition of two matrix is

-8 -6 -4

APEC

```
-2  0  2
4  6  8
```

```
/* C Program to perform Matrix Scaling     */
```

Formula:

$$x\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} ax & bx \\ cx & dx \end{bmatrix}$$

Example:

$$\mathbf{a} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \mathbf{5a} = \begin{pmatrix} 5 & 10 & 15 \\ 20 & 25 & 30 \\ 35 & 40 & 45 \end{pmatrix}$$

```
#include<stdio.h>
void main()
{
  int a[3][3],num,i,j;

  printf("Enter the matrix->");
  for(i=0;i<3;i++)
      for(j=0;j<3;j++)
           scanf("%d",&a[i][j]);

  printf("Enter any number to multiply with matrix : ");
  scanf("%d", &num);

   for(i=0;i<3;i++)
       for(j=0;j<3;j++)
            a[i][j]=num * a[i][j];
   printf("\n Resultant matrix is\n");
   for(i=0;i<3;i++)
   {
       printf("\n");
```

APC

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

Execution:

Input

```
Enter the matrix->
1 2 3
4 5 6
7 8 9
Enter any number to multiply with matrix : 2
```

Output

```
Resultant matrix is

2    4    6
8    10   12
14   16   18
```

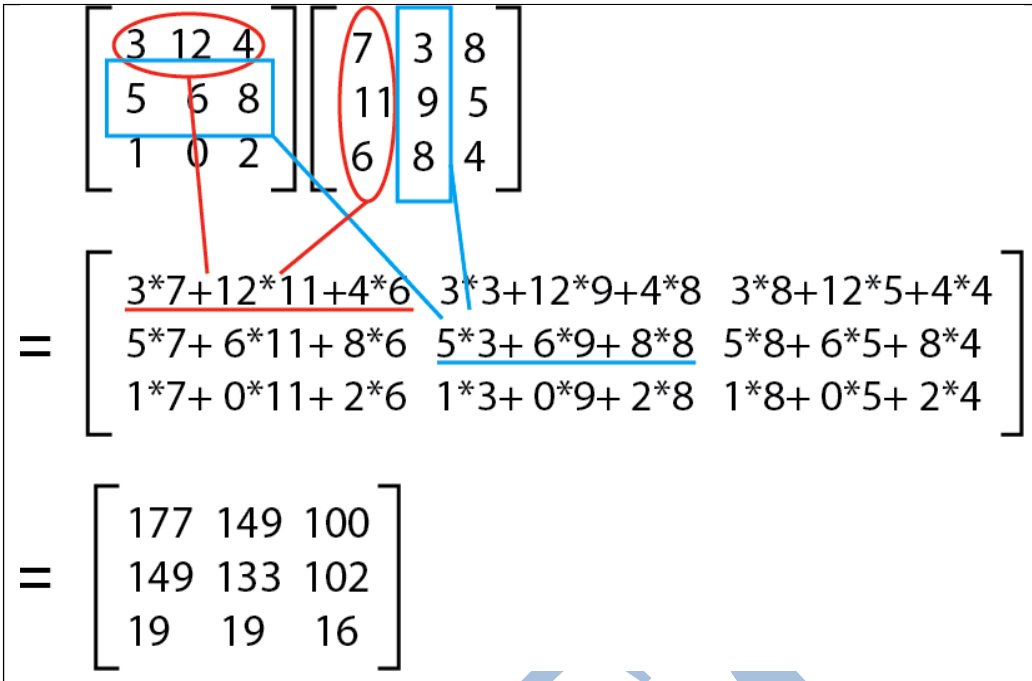
```
/* C Program to perform Matrix Multiplication     */
```

Formula

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}xb_{11}+a_{12}xb_{21}+a_{13}xb_{31} & a_{11}xb_{12}+a_{12}xb_{22}+a_{13}xb_{32} & a_{11}xb_{13}+a_{12}xb_{23}+a_{13}xb_{33} \\ a_{21}xb_{11}+a_{22}xb_{21}+a_{23}xb_{31} & a_{21}xb_{12}+a_{22}xb_{22}+a_{23}xb_{32} & a_{21}xb_{13}+a_{22}xb_{23}+a_{23}xb_{33} \\ a_{31}xb_{11}+a_{32}xb_{21}+a_{33}xb_{31} & a_{31}xb_{12}+a_{32}xb_{22}+a_{33}xb_{32} & a_{31}xb_{13}+a_{32}xb_{23}+a_{33}xb_{33} \end{bmatrix}$$

Example

APEC




```
#include<stdio.h>

void main()
{
   int a[10][10], b[10][10], c[10][10], i, j, k;
   int sum = 0;

   printf("\nEnter First Matrix : n");
   for (i = 0; i < 3; i++)
      for (j = 0; j < 3; j++)
         scanf("%d", &a[i][j]);

   printf("\nEnter Second Matrix:n");
   for (i = 0; i < 3; i++)
      for (j = 0; j < 3; j++)
         scanf("%d", &b[i][j]);

   //Multiplication Logic
   for (i = 0; i < 3; i++)
   {
      for (j = 0; j < 3; j++)
        {
          sum = 0;
```

APEC

```
            for (k = 0; k < 3; k++)
               sum = sum + a[i][k] * b[k][j];
            c[i][j] = sum;
         }
   }

   printf("\nMultiplication Of Two Matrices : \n");
   for (i = 0; i < 3; i++)
   {
      for (j = 0; j < 3; j++)
      {
         printf(" %d ", c[i][j]);
         }
       printf("\n");
    }
 }
```

Execution:

Input

```
Enter First Matrix :
1 2 0
0 1 1
2 0 1
Enter Second Matrix :
1 1 2
2 1 1
1 2 1
```

Output

```
Multiplication Of Two Matrices
5     3     4
3     3     2
3     4     5
```

APEC

/* C Program to find determinant of a Matrix */

Formula

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$
$$= a(ei - fh) - b(di - fg) + c(dh - eg)$$
$$= aei + bfg + cdh - ceg - bdi - afh.$$

Example

$$\det\begin{bmatrix} -5 & 0 & -1 \\ 1 & 2 & -1 \\ -3 & 4 & 1 \end{bmatrix} = -5 \cdot \det\begin{bmatrix} 2 & -1 \\ 4 & 1 \end{bmatrix} - (0) \cdot \det\begin{bmatrix} 1 & -1 \\ -3 & 1 \end{bmatrix} + (-1) \cdot \det\begin{bmatrix} 1 & 2 \\ -3 & 4 \end{bmatrix}$$
$$= -5[2-(-4)] - 0[1-(3)] - 1[4-(-6)]$$
$$= -5(2+4) - 0(1-3) - 1(4+6)$$
$$= -5(6) - 0(-2) - 1(10)$$
$$= -30 - 0 - 10$$
$$= -40$$

```
#include<stdio.h>
void main()
{
  int a[3][3],i,j;
  int determinant=0;

  printf("Enter the matrix: ");
  for(i=0;i<3;i++)
      for(j=0;j<3;j++)
           scanf("%d",&a[i][j]);

  for(i=0;i<3;i++)
      determinant = determinant +
                    (a[0][i]*(a[1][(i+1)%3]*a[2][(i+2)%3] -
```

```
                    a[1][(i+2)%3]*a[2][(i+1)%3]));

  printf("\nDeterminant of matrix is: %d",determinant);
}
```

Execution:

Input

```
Enter the Matrix:
1       2       3
4       5       6
7       8       9
```

Output

```
Determinant of matrix: 0
```

/* C Program to find Transpose of a Matrix */

Formula

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}^T = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}$$

Example

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
#include<stdio.h>
void main()
{
  int a[10][10],transpose[10][10],i,j;
  printf("\nEnter the matrix:");
```

APEC

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

  //Trnaspose of the Matrix
  for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       transpose[i][j]=a[j][i];

  printf("\nThe Transpose of matrix is:\n");
  for(i=0;i<3;i++)
    {
     for(j=0;j<3;j++)
       printf ("%d", transpose[i][j]);
    printf("\n";
   }
}
```

Execution:

Input

```
Enter the matrix:
3 7 9
2 7 5
6 3 4
```

Output

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

APEC

Strings in C:

In C programming, array of characters is called a string. A string is terminated by a null character \0. For example, consider: "C Programming". Here, “C Programming" is a string. When, compiler encounters strings, it automatically appends a null character \0 at the end of string.

C		P	r	o	g	r	a	m	m	i	n	g	\0
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]	c[10]	c[11]	c[12]	c[13]

```
char c[] = "C Programming";  /*One extra for string terminator*/
char c[15] = "C Programming";
char c[] = {„C“,“ „„“P“,“r“,“o“,“g“,“r“,“a“,“m“,“m“,“i“,“n“,“g“,“\0“};
char c[15] = {„C“,“ „„“P“,“r“,“o“,“g“,“r“,“a“,“m“,“m“,“i“,“n“,“g“,“\0“};
```

Reading Strings from user- String I/O

- scanf() function is used to read a string like any other data types. However, the scanf() function only takes the first entered word. The function terminates when it encounters a white space (or just space).

  Example:

  ```
  char c[20];
  scanf("%s", c);
  ```

- gets() and puts() - C standard library provides the gets function which allows us to read a line of characters (including spaces and tabs) until the newline character is entered, i. e., the enter key is pressed.

  ```
  char str[81];
  ```

  A line of text contains 80 characters. Thus, the array str has been declared to store 81 characters with a provision to store the null terminator.

Reading characters from user

- Single Character functions are used for reading and displaying a single character respectively.
  - getchar() – reads a character from the keyboard.
  - putchar() – prints a character to the monitor screen.

APEC

ch = getchar();

It reads a character and stores it in variable ch.

putchar(ch);

The above statement prints the given character onto the monitor screen.

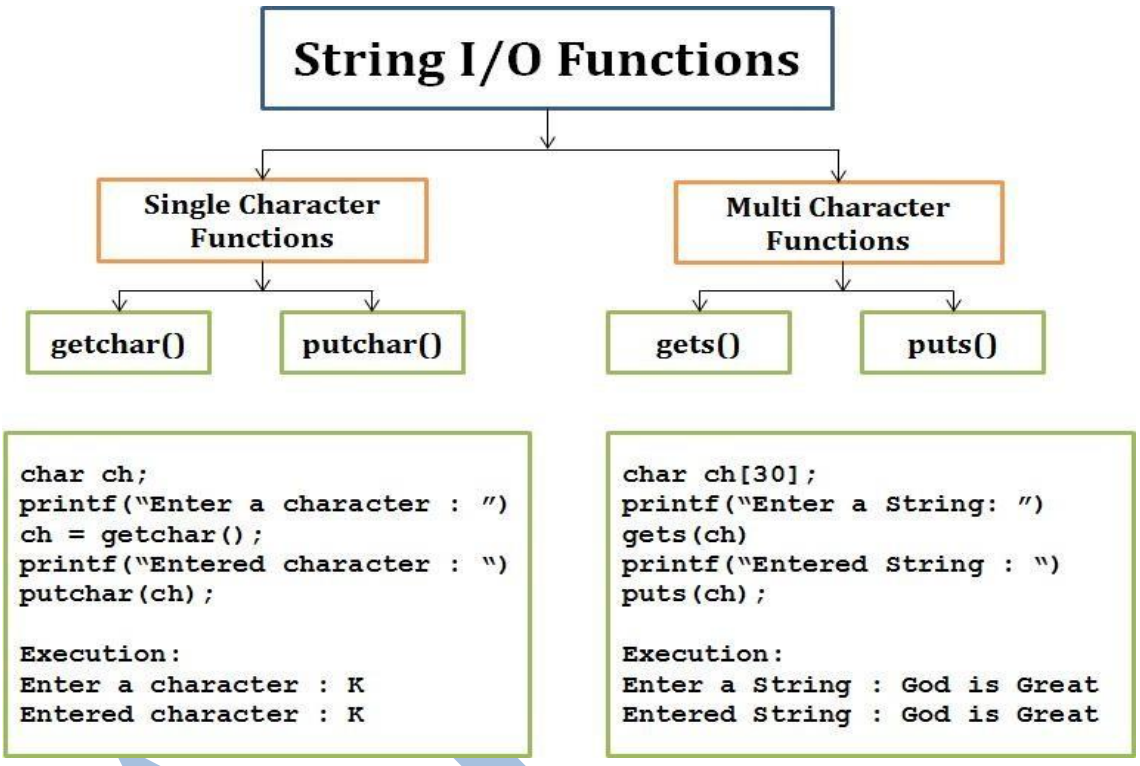


Figure 3.1 Character and String Input functions

/*C program to read a string */

Example #1: Initialize and display a string

```
#include<stdio.h>
void main()
{
    // declare and initialize string
    char str[] = "Hello";
    // print string
    printf("Given string : %s",str);
}
```

Execution:

Output:

```
Given string : Hello
```

Example #2: Using scanf() to read a string

```
#include <stdio.h>
void main()
{
    char name[20];
    printf("Enter name: ");
    scanf("%s", name);
    printf("Your name is %s.", name);
}
```

Execution:

Input:

```
Enter name: Mahatma Gandhi
```

Output:

```
Your name is Mahatma.
```

Example #3: Using getchar() to read a line, character by character.

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

APEC

```
void main()
{
    char name[30], ch;
    int i = 0;
    printf("Enter the string: ");
    while(ch != '\n')     // terminates if user hit enter
    {
        ch = getchar();
        name[i] = ch;
        i++;
    }
    name[i] = '\0';       // inserting null character at end
    printf("Name: %s", name);
}
```

Execution:

Input:

```
Enter the string: Mahatma Gandhi
```

Output:

```
Name: Mahatma Gandhi
```

Example #4: Using standard library function to read a line of text

```
#include <stdio.h>
void main()
{
    char name[30];
    printf("Enter name: ");
    gets(name);      //Function to read string from user.
    printf("Name : ");
    puts(name);     //Function to display string.
}
```

Execution:

Input:

```
Enter name: Mahatma Gandhi
```

Output:

```
Name : Mahatma Gandhi
```

Example #5: String formatting with printf to format a string

APEC

together with other strings and integer

```
#include <stdio.h>
void main()
{
  char name[] = "Mahatma Gandhi";
  int year = 1869;

 /* prints 'Mahatma Gandhi born in the year 1869 at Porbandar.' */
  printf("%s born in the year %d at Porbandar.\n", name, year);
  }
```

Execution:

Output:

```
Mahatma Gandhi born in the year 1869 at Porbandar.
```

```
/* Simple C Program to access the character array  */

#include<stdio.h>
void main()
{
  char name[10] = {„h‟, „e‟, „l‟, „l‟, „o‟};
  for (int i = 0; i < 5; i++ )
     printf("\nCharacter at %d is %c " , i, name[i]);
 }
```

Execution:

```
Character at 0 is h
Character at 1 is e
Character at 2 is l
Character at 3 is l
Character at 4 is o
```

APEC

/* Simple C Program to Calculate Length of String */

Example :

String	G	o	o	d		M	o	r	n	i	n	g	Total
Count	1	2	3	4	5	6	7	8	9	10	11	12	12

This program computes the length of a string manually without using strlen() funtion. However, scanf() consider only the first word.

```
#include <stdio.h>
void main()
{
    char s[30], i;
    printf("Enter a string : ");
    gets(s);
    for(i = 0; s[i] != '\0'; ++i);
    printf("Length of string : %d", i);
}
```

Execution:

Input:

```
Enter a string : Good Morning
```

Output:

```
Length of string : 12
```

```
#include <stdio.h>
void main()
{
    char s[30], i;
    printf("Enter a string : ");
    scanf("%s",s);
    for(i = 0; s[i] != '\0'; ++i);
    printf("Length of string : %d", i);
}
```

Execution:

Input:

```
Enter a string : Good Morning
```

Output:

```
Length of string : 4
```

APEC

/* Simple C Program to Calculate the occurrence of the given character in the String */

Example :

Find the occurrence of the character „o" in the given string

String	G	o	o	d		M	o	r	n	i	n	g	Total
Count		1	2				3						3

```
#include <stdio.h>
void main()
{
   char str[25], ch;
   int i, frequency = 0;

   printf("Enter a string: ");
   gets(str);
   printf("Enter a character : ");
   getchar(ch);

   for(i = 0; str[i] != '\0'; ++i)
   {
        if(ch == str[i])
            ++frequency;
      }
   printf("Occurrence of %c = %d", ch, frequency);
  }
```

Execution:

Input:

Enter a string : Good Morning

Enter a character : o

Output:

Occurrence of o = 3

APEC

/* Simple C Program to count the numbers of characters and spaces in the given string */

Example :

String	G	o	o	d		M	o	r	n	i	n	g	Total
Characters Count	1	2	3	4		5	6	7	8	9	10	11	**11**
Spaces Count					1								**1**

```
#include<stdio.h>
void main()
{
  char str[81];
  int spaces = 0, chars = 0 ;

  puts("Enter a string :");
  gets(str);
  for(int i=0; str[i]!='\0'; i++)
     if(str[i] == ' ')
         spaces++;
     else
         chars++;
   printf(" Number of spaces :", spaces);
   printf(" Number of characters :", chars);
  }
```

Execution:

Input:

```
Enter a string : God is Great
```

Output:

```
Number of spaces : 2
Number of characters : 10
```

APEC

/* Simple C Program to Count number of Uppercase and Lowercase characters in the given string */

Example :

String	G	o	o	d		M	o	r	n	i	n	g	Total
Uppercase Count	1					2							2
Lowercase Count		1	2	3			4	5	6	7	8	9	9
Other Characters					1								1

```
#include<stdio.h>
void main()
{
   int upper = 0, lower = 0, others = 0;
   char ch[80];
   int i;

   printf("\nEnter The String : ");
   gets(ch);

   i = 0;
   while (ch[i] != '\0')
   {
      if (ch[i] >= 'A' && ch[i] <= 'Z')
         upper++;
      else if (ch[i] >= 'a' && ch[i] <= 'z')
         lower++;
      else
         others++;
      i++;
   }

   printf("\nUppercase Letters : %d", upper);
   printf("\nLowercase Letters : %d", lower);
   printf("\nOther characters  : %d", others);
}
```

APEC

Execution:

Input:

```
Enter The String : Good Morning
```

Output:

```
Uppercase Letters : 2
Lowercase Letters : 9
Other characters  : 1
```

APEC

/* Simple C Program to check whether the given character is alphabet or not */

```
#include <stdio.h>
void main()
{
    char c;
    printf("Enter a character: ");
    scanf("%c",&c);

    if( (c>='a' && c<='z') || (c>='A' && c<='Z'))
        printf("%c is an alphabet.",c);
    else
        printf("%c is not an alphabet.",c);
}
```

Execution:

Input:

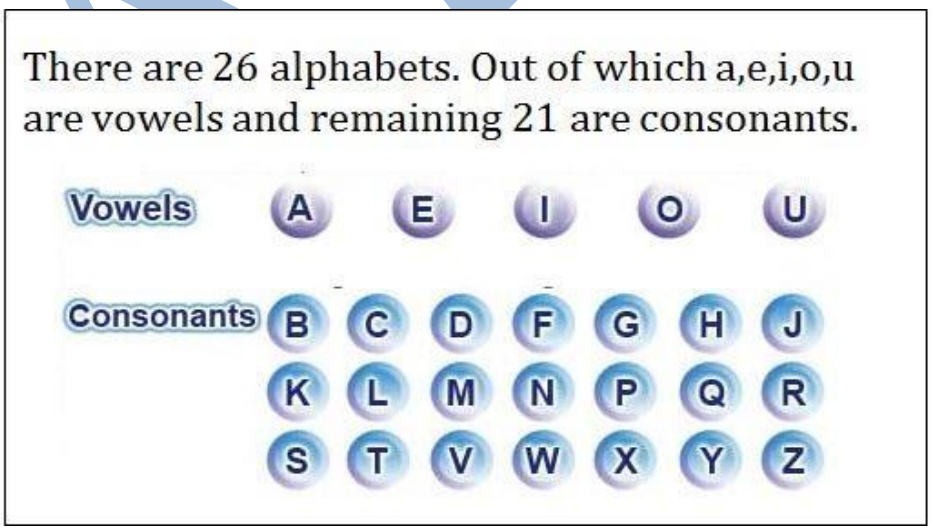
```
Enter a character: 9
```

Output:

```
9 is not an alphabet.
```

/* Simple C Program to check whether the given character is vowel or consonant */

Example :




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

```
void main()
{
    char c;
    int low, up;

    printf("Enter an alphabet: ");
    scanf("%c",&c);

    // evaluates to 1 (true) if c is a lowercase vowel
    low = (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');

    // evaluates to 1 (true) if c is an uppercase vowel
    up = (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U');

    if (low || up)
        printf("%c is a vowel.", c);
    else
        printf("%c is a consonant.", c);
  }
```

Execution:

Input:

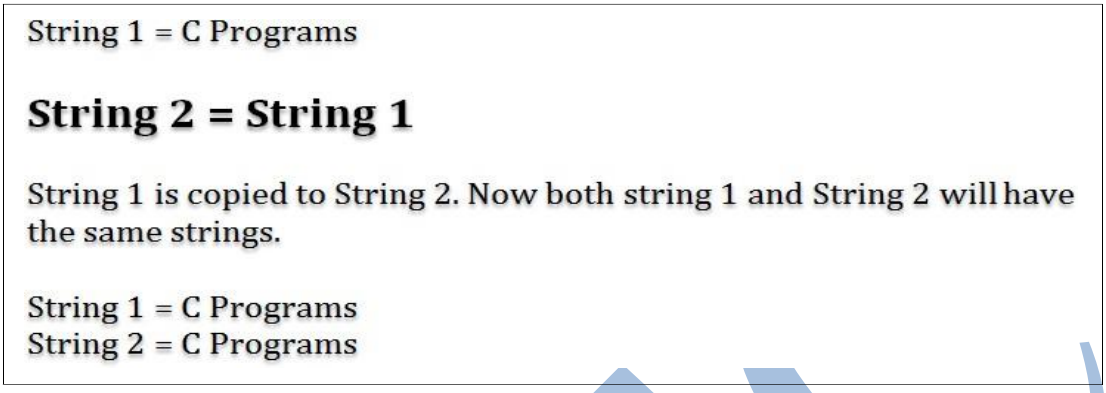
```
Enter an alphabet: i
```

Output:

```
i is a vowel.
```

APEC

/* Simple C Program to Copy one string to another without using strcpy() */




```
#include <stdio.h>
void main()
{
    char s1[100], s2[100], i;
    printf("Enter string s1: ");
    gets(s1);

    for(i=0;s1[i]!='\0';++i)
      s2[i]=s1[i];

    printf("\nCopied string in s2 is : ");
    puts(s2);
    printf("\nOriginal string s1 is : ");
    puts(s2);
  }
```

Execution:

Input:

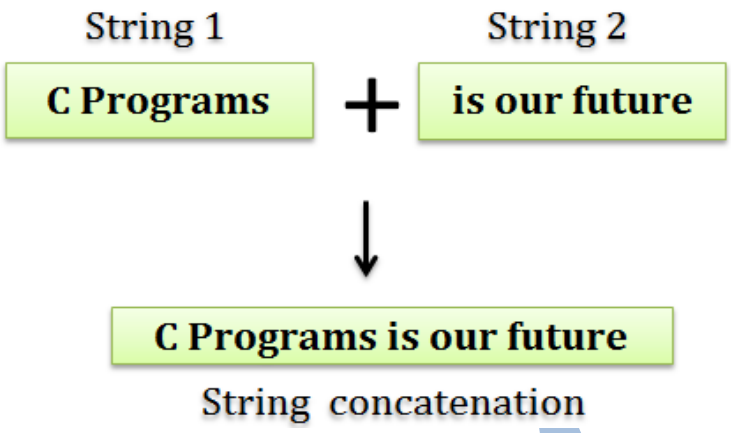
```
Enter string s1: Mother Terasa
```

Output:

```
Copied string in s2 is : Mother Terasa
Original string s1 is : Mother Terasa
```

APEC

/* Simple C Program to Concatenate two strings without using strcat()
***/**




```
#include <stdio.h>
void main()
{
   char s1[30],s2[30],s3[60];
   int i,j;

   printf("Enter first string:");
   gets(s1);
   printf("Enter second string:");
   gets(s2);

   for(i=0;s1[i]!='\0';++i)
      s3[i]=s1[i];

   for(j=0;s2[j]!='\0';++j)
      s3[i+j]=s2[j];

   s3[i+j]='\0';

   printf("nThe concatenated string is: %s",s3);
 }
```

Execution:

Input:

```
Enter first string: Hard work
Enter second string: never fails
```

Output:

```
The concatenated string is: Hard work never fails
```

APEC

/* Simple C Program to Compare two strings without using strcmp () */

Example :




```
#include<stdio.h>
void main()
{
   char str1[30], str2[30];
   int i;

   printf("\nEnter the two strings :");
   gets(str1);
   gets(str2);

   i = 0;
   while (str1[i] == str2[i])
      i++;
   if (str1[i] > str2[i])
      printf("str1 > str2");
   else if (str1[i] < str2[i])
      printf("str1 < str2");
   else
      printf("str1 = str2");
 }
```

Execution:

Input:

```
Enter the two strings:
program
```

APEC

```
program
```

Output:

```
str1 = str2
```

```
/* Simple C Program to access the string array  */

#include<stdio.h>
void main()
{
  string months[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "July",
                     "Aug", "Sep", "Oct", "Nov", "Dec"};
//use the length attribute to get the number of elements in an array
for (int i = 0; i < months.length; i++ )
  printf("\n Month: " + month[i]);
 }
```

Execution:

Output:

```
Month : Jan
Month : Feb
Month : Mar
Month : Apr
Month : May
Month : Jun
Month : July
Month : Aug
Month : Sep
Month : Oct
Month : Nov
Month : Dec
```

APEC

Sorting:

Sorting is the process of arranging a list of elements in a particular order (Ascending or Descending).

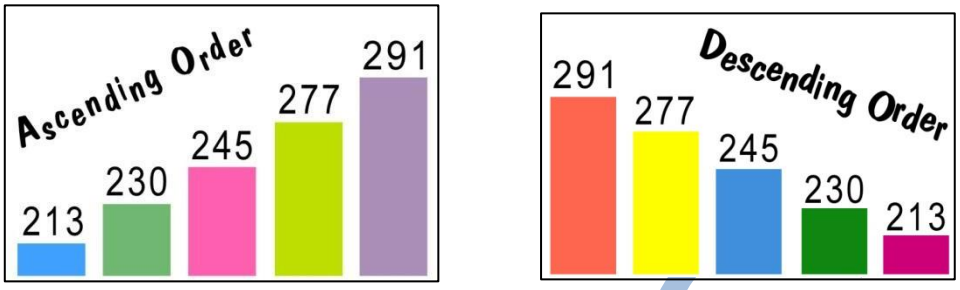


Figure 2. Sorting of Numbers

There are many types of Sorting techniques, differentiated by their efficiency and space requirements.

- Bubble Sort
- Insertion Sort
- Selection Sort
- Quick Sort
- Merge Sort
- Heap Sort

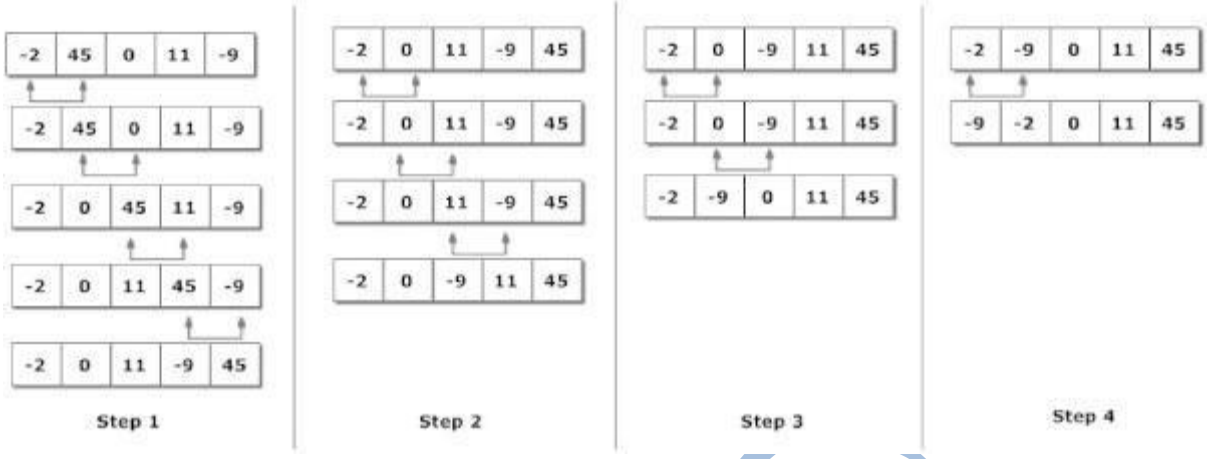
Bubble Sort:

Bubble sort, sometimes is a simple sorting algorithm that repeatedly compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the array is repeated until no swaps are needed, which indicates that the list is sorted.

- Number of passes = Array_size –1.
- In each iteration, the largest element in the list bubbles up towards the last place.

```
/*C Program to sort the given numbers in ascending order using bubble
sort.*/
Example
```

APEC




```
#include <stdio.h>
void main()
{
    int a[20],i,n,temp;
    printf("Enter the size of the Array: ");
    scanf("%d",&n);
    printf("\nEnter the array elements: ");
    for(i=0;i<n;++i)
        scanf("%d",&a[i]);

    for(i=1;i<n;++i)
     for(j=0;j<n-i;++j)
         if(a[j] > a[j+1])    //For descending order, change > to <
        {
            temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
        }
    }
    printf(" Sorted array: ");
    for(i=0;i<n;++i)
         printf("%d \t ",a[i]);
}
```

Execution:

Input

```
Enter the size of array: 4
Enter the array elements: 3 7 9 2
```

Output

APEC

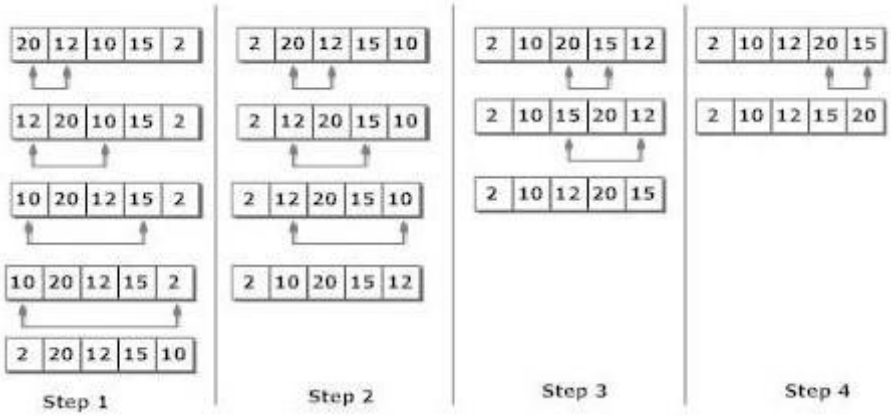
```
Sorted array: 2 3 7 9
```

Selection Sort:

Selection sorting is conceptually the simplest sorting algorithm. This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then finds the second smallest element and exchange it with the element in the second position, and continues in this way until the entire array is sorted.

/*C Program to sort the given numbers in ascending order using Selection sort.*/

Example




```
#include <stdio.h>
void main()
 {
    int a[100],i,n,j,temp;
    printf("Enter the size of the array: ");
    scanf("%d",&n);
    printf("Enter the array elements: ")
    for(i=0;i<n;++i)
       scanf("%d",&a[i]);
    for(i=0;i<n;++i)
      for(j=i+1;j<n;++j)
      {
         if(a[i]>a[j])
```

```
            /* To sort in descending order, change > to <. */
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
        printf("Sorted array: ");
        for(i=0;i<n;++i)
            printf("%d  ",a[i]);
}
```

Execution:

Input

```
Enter the size of array: 4
Enter the array elements: 3 7 9 2
```

Output

```
Sorted array: 2 3 7 9
```

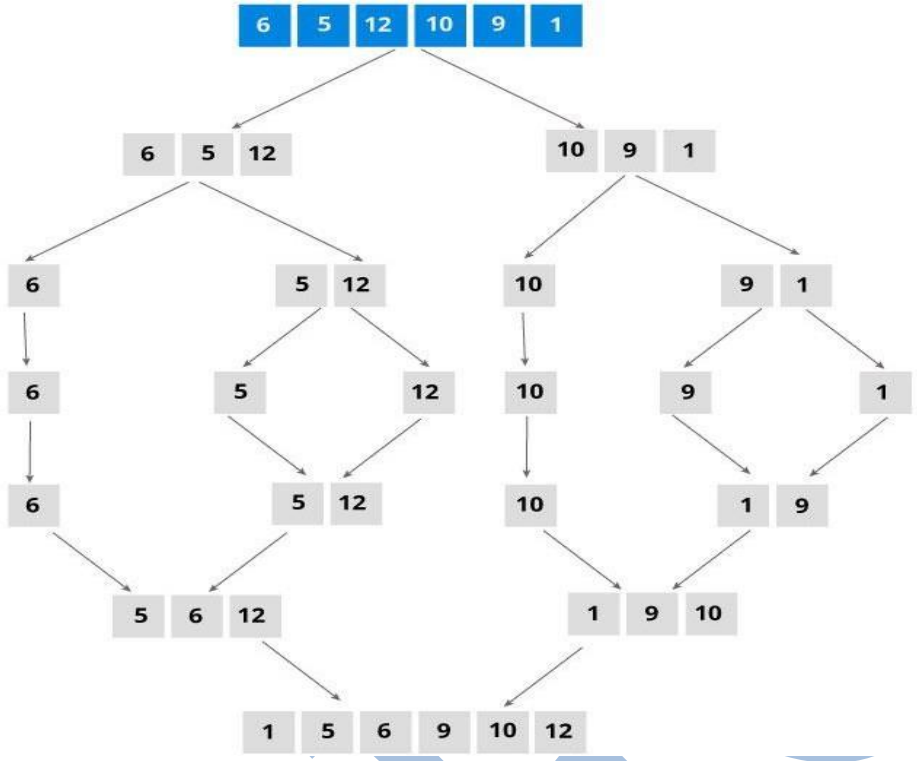
Merge Sort:

Merge Sort is a kind of Divide and Conquer algorithm in computer programming. It is one of the most popular sorting algorithms in recursive algorithms. Using the Divide and Conquer technique, we divide a problem into sub problems. When the solution to each sub problem is ready, we 'combine' the results from the sub problems to solve the main problem.

```
/*C Program to sort the given numbers in ascending order using Merge
sort.*/
```

Example

APEC




```
#include<stdio.h>

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

void main()
{
    int a[30],n,i;
    printf("Enter the size of array :");
    scanf("%d",&n);
    printf("Enter the array elements:");

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

    mergesort(a,0,n-1);

    printf("\nSorted array :");
    for(i=0;i<n;i++)
```

```
        printf("%d ",a[i]);

}

void mergesort(int a[],int i,int j)
{
    int mid;

    if(i<j)
    {
        mid=(i+j)/2;
        mergesort(a,i,mid);         //left recursion
        mergesort(a,mid+1,j);     //right recursion
        merge(a,i,mid,mid+1,j);     //merging two sorted sub-arrays
    }
}

void merge(int a[],int i1,int j1,int i2,int j2)
{
    int temp[50];     //array used for merging
    int i,j,k;
    i=i1;     //beginning of the first list
    j=i2;     //beginning of the second list
    k=0;

    while(i<=j1 && j<=j2)     //while elements in both lists
    {
        if(a[i]<a[j])
            temp[k++]=a[i++];
        else
            temp[k++]=a[j++];
    }

    while(i<=j1)     //copy remaining elements of the first list
        temp[k++]=a[i++];

    while(j<=j2)     //copy remaining elements of the second list
```

APEC

```
        temp[k++]=a[j++];

    //Transfer elements from temp[] back to a[]
    for(i=i1,j=0;i<=j2;i++,j++)
        a[i]=temp[j];
}
```

Execution:

Input

```
Enter the size of array: 6
Enter the array elements: 6 5 12 10 9 1
```

Output

```
Sorted array: 1 5 6 9 10 12
```

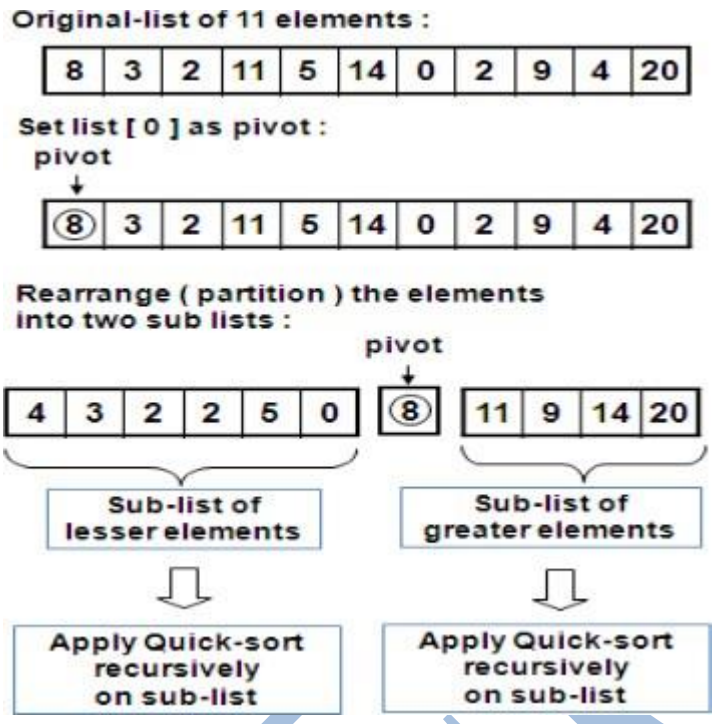
Quick Sort:

The Quick Sort is based on partition. It is also known as Partition Exchange sorting. The basic concept of quick sort process is pick one element from an array and rearranges the remaining elements around it. This element divides the main list into two sub lists. This chosen element is called pivot. Once pivot is chosen, then it shifts all the element less than pivot to left of value pivot and all the elements greater than pivot are shifted to the right side. This procedure of choosing pivot and partition the list is applied recursively until sub-lists consisting of only one element.

```
/*C Program to sort the given numbers in ascending order using Quick
sort.*/
```

Example

APEC




```
#include<stdio.h>
void quicksort(int a[25],int first,int last)
{
   int i, j, pivot, temp;
   if(first<last)
  {
      pivot=first;
      i=first;
      j=last;

      while(i<j)
     {
         while(a[i]<=a[pivot]&&i<last)
            i++;
         while(a[j]>a[pivot])
            j--;
         if(i<j)
        {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
         }
```

APEC

```
        }

        temp=a[pivot];
        a[pivot]=a[j];
        a[j]=temp;
        quicksort(a,first,j-1);
        quicksort(a,j+1,last);
    }
}

void main()
 {
    int i, count, a[25];

    printf("Enter the size of array : ");
    scanf("%d",&count);

    printf("Enter the array elements :");
    for(i=0;i<count;i++)
        scanf("%d",&a[i]);

    quicksort(a,0,count-1);

    printf("Sorted array: ");
    for(i=0;i<count;i++)
        printf(" %d",a[i]);
}
```

Execution:

Input

```
Enter the size of array: 6
Enter the array elements: 6 5 12 10 9 1
```

Output

```
Sorted array: 1 5 6 9 10 12
```

APEC

Sorting algorithms comparison:

Sorting algorithms are usually judged by their efficiency. In this case, efficiency refers to the algorithmic efficiency is generally based on the number of elements to sort. Each algorithm has particular strengths and weaknesses. The choice of sorting algorithm depends on the application.

Table 2.x Sorting Algorithms Average time Complexity

Sorting Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n\ log\ n)$	$O(n\ log\ n)$	$O(n\ log\ n)$
Quick Sort	$O(n\ log\ n)$	$O(n\ log\ n)$	$O(n^2)$

APEC

Searching

Searching is a technique that helps to find whether the given element is present in the set of elements. Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not. Some of the standards searching techniques are:

- Linear Search or Sequential Search
- Binary Search

Linear Search:

Linear search is implemented using following steps...

Step 1: Read the search element from the user

Step 2: Compare, the search element with the first element in the array.

Step 3: If both are matched, then display "Given element found!!!" and terminate the program

Step 4: If both are not matched, then compare search element with the next element in the array.

Step 5: Repeat steps 3 and 4 until the search element is compared with the last element in the array.

Step 6: If the last element in the array is also not matched, then display "Element not found!!!" and terminate the function.

```
/*C Program to search the given element in the list of items using
Linear Search */
```

Example

Given a array, Search a given element in array.

Case 1:

Input: Search 20.

12	5	10	15	31	20	25	2	40
0	1	2	3	4	5	6	7	8

Output: True (20 is present in array)

Case 2:

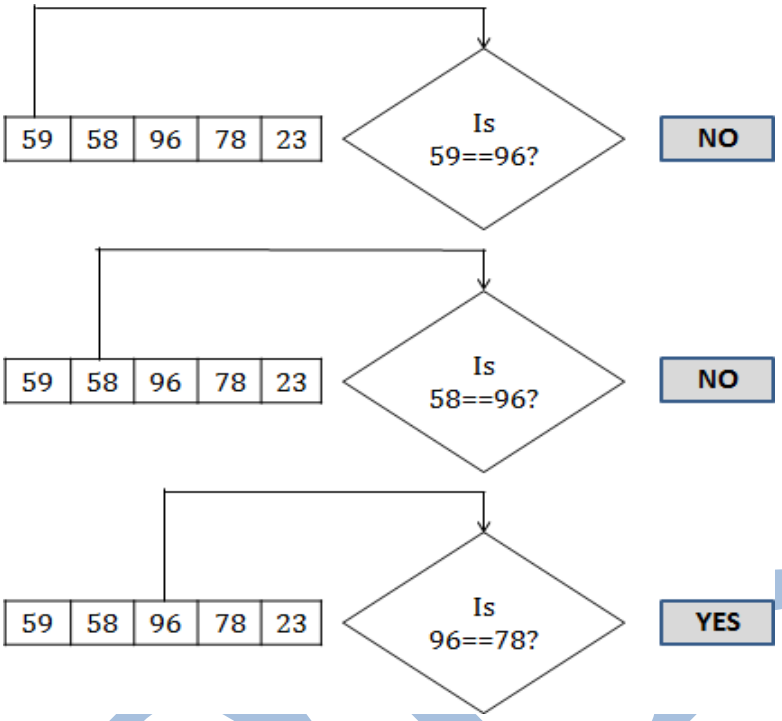
Input: Search 26.

12	5	10	15	31	20	25	2	40
0	1	2	3	4	5	6	7	8

Output: False (26 is not present in array)

APEC

Given the array of elements: 59,58,96,78,23 and the element to be searched is 96, the working of linear search is as follows:



The Element is FOUND. Hence stop the searching process.

```
#include<stdio.h>
void main()
 {
  int a[20],size,i,sElement;

  printf("Enter size of the array: ");
  scanf("%d",&size);

  printf("Enter the array elements: ");
  for(i = 0; i < size; i++)
    scanf("%d",& a[i]);

  printf("Enter the element to Search : ");
  scanf("%d",&sElement);

  // Linear Search Logic
  for(i = 0; i < size; i++)
```

```
      if(sElement == a[i])
      {
         printf("Element is found at index %d ", i);
         break;
      }

  if(i == size)
     printf("Given element is not found ");
}
```

Execution:

Input

```
Enter size of the array: 5
Enter the array elements:  6 10 9 15 3
Enter the element to Search : 15
```

Output

```
Element is found at index 3
```

Binary search

Binary search is implemented using following steps:

Step 1: Read the search element from the user

Step 2: Find the middle element in the sorted array

Step 3: Compare, the search element with the middle element in the sorted array.

Step 4: If both are matched, then display "Given element found!!!" and terminate the function

Step 5: If both are not matched, then check whether the search element is smaller or larger than middle element.

Step 6: If the search element is smaller than middle element, then repeat steps 2, 3, 4 and 5 for the left sub array of the middle element.

Step 7: If the search element is larger than middle element, then repeat steps 2, 3, 4 and 5 for the right sub array of the middle element.

Step 8: Repeat the same process until we find the search element in the array or until the sub array contains only one element.

Step 9: If that element also doesn't match with the search element, then display "Element not found in the array!!!" and terminate the function.

APEC

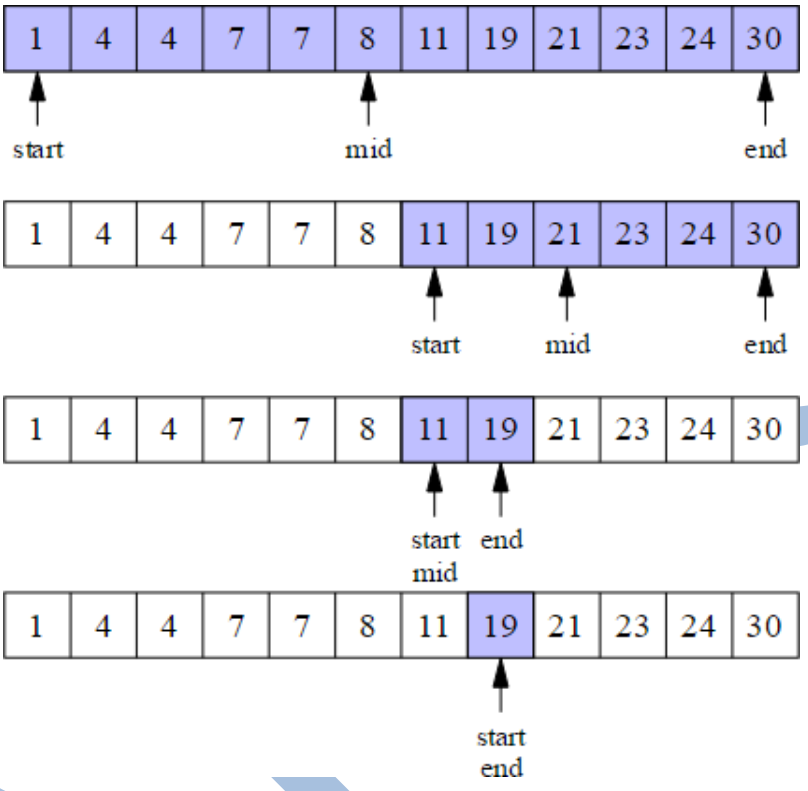
/*C Program to search the given element in the list of items using Binary Search */

Example 1:

Given an array of elements: 1,4,4,7,7,8,11,19,21,23,24,30

The element to be searched: 19

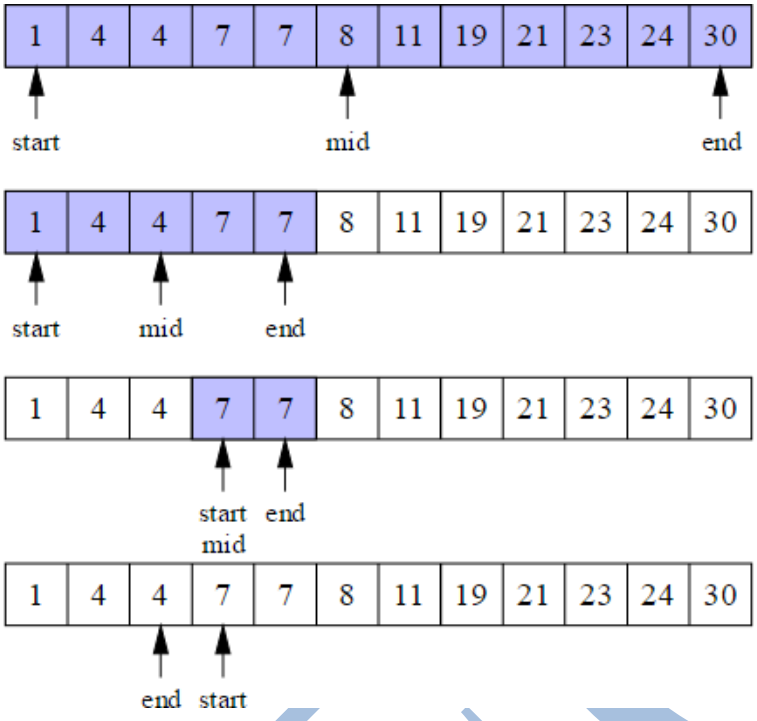
Formula for calculating middle is, $Mid = \frac{start + end}{2}$



The Element is FOUND. Hence stop the searching process.

Example 2:

Assume the element to be searched: 6



The Element is NOT FOUND.

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

void main()
{
   int start, end, mid, size, i, sElement, a[100];
   clrscr();

   printf("Enter the size of the array: ");
   scanf("%d",&size);

   printf("Enter array elements in Ascending order: ");

   for (i = 0; i < size; i++)
      scanf("%d",&a[i]);

   printf("Enter value to be search: ");
   scanf("%d", &sElement);

   start = 0;
   end = size - 1;
   mid = (start+end)/2;
```

APEC

```
   while (start <= end)
   {
      if (a[mid] < sElement)
         start = mid + 1;
      else if (a[mid] == sElement)
        {
         printf("Element found at index %d.\n",mid);
         break;
        }
      else
         end = mid - 1;

    mid = (start + end)/2;
   }

   if (start > end)
      printf("Element Not found in the array");

   getch();
 }
```

Execution:

Input

```
Enter the size of the array: 5
Enter array elements in Ascending order: 6 31 69 75 93
Enter the element to Search : 69
```

Output

```
Element is found at index 2
```

APEC

Merits of using Arrays:

In C programming language, arrays are used in wide range of applications.

- Arrays are used to Store List of values
    - Single dimensional arrays are used to store list of values of same data type.

- Arrays are used to perform Matrix Operations
    - Two dimensional arrays to create matrix and perform various matrix operations
- Arrays are used to implement Search Algorithms
    - Linear Search
    - Binary Search
- Arrays are used to implement Sorting Algorithms
    - Insertion Sort
    - Bubble Sort
    - Selection Sort
    - Quick Sort
    - Merge Sort, etc.,
- Arrays are used to implement Data structures
    - Stack Using Arrays
    - Queue Using Arrays
- Arrays are also used to implement CPU Scheduling Algorithms

Demerits of using Arrays:

- The elements of array are stored in consecutive memory locations. So insertions and deletions are very difficult and time consuming.
- The number of elements stored in array should be known in advance.

APEC