|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **INDEX** | | | | | |
| **Sr.no.** | **Date** | **Exp.no.** | **Topic** | Page no. | |
| 1. | 14/3/22 | 01 | 1. Students Age Display | | 2-4 |
| 1. Smallest Number | 5-7 | |
| 1. Even and Odd Number | 8-11 | |
| 1. Sum Of Elements | 12-14 | |
| 1. Average Of Numbers | 15-17 | |
| 2. | 14/3/22 | 02 | 1. Sum Of Matrices | 18-22 | |
| 1. Temperature Conversion | 23-27 | |
| 1. Transpose Of Matrix | 27-32 | |
| 3. | 14/3/22 | 03 | 1. Linear Search | 33-36 | |
| 1. Binary Search | 37-40 | |
| 4. | 5/4/22 | 04 | 1. Bubble Sort | 41-44 | |
| 1. Insertion Sort | 45-48 | |
| 1. Selection Sort | 49-53 | |
| 1. Merge Sort | 54-63 | |
| 5. | 13/5/22 | 05 | 1. Stack | 64-69 | |
| 1. Queue | 70-76 | |
| 6. | 20/5/22 | 06 | 1. Singly Linked List | 77-100 | |

* Problem A

Students Age Display

**AIM :-** To write a ‘c’ Program to take input of Age of Students and display the Age using **Array.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare integer variable i and age for array.

Step 3:- Check FOR condition i<?, if it is true go to step 4 otherwise go to step 6.

Step 4:- Read variable age by For loop.

Step 5:- Increment i+1 and go to step 3.

Step 6:- Check FOR condition i<?, if it is true go to step 7 otherwise go to step 9.

Step 7:- Print age by for loop.

Step 8:- Increment i+1 and go to step 6.

Step 9:- Stop.

**Program :-**

#include<stdio.h>

int main()

{

int age[50];

int i;

printf("Enter the age of Students\n");

for(i=0;i<10;i++)

{

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

}

printf("The Age of Students is ......\n");

for(i=0;i<10;i++)

{

printf("Student No.%d = %d\n",i,age[i]);

}

return 0;

}

**Output :-** Enter the age of Students

12,13,10,14,15,18,17,16,9,7

The Age of Students is ......

Student No.0 = 12

Student No.1 = 13

Student No.2 = 10

Student No.3 = 14

Student No.4 = 15

Student No.5 = 18

Student No.6 = 17

Student No.7 = 16

Student No.8 = 9

Student No.9 = 7

**Observation :-** After performing the experiment we observed that Values can be stored in a single variable using Array and we were able to take input of many values using looping statements. Thus, reduced the time to write the scanf statement and print the values in the same manner using loop. It took 0.14 sec Compilation Time.

* Problem B

**Smallest Number**

**AIM :-** To write a ‘c’ Program to take input of Numbers and find out the Smallest Number in the Array**.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start

Step 2:- Declare ‘a’ variable for array and i,j,b

Step 3:- Check For condition i<?, if it is true go to step 4 otherwise go to step 6.

Step 4:- Read variable ‘a’ by loop.

Step 5:- Increment i+1 and go to step 3.

Step 6:- Check For condition i<?, if it is true go to step 7 otherwise go to step 8.

Step 7:- if(a[i]>a[j]) Swap the values

b=a[i];

a[i]=a[j];

a[j]=b;.

Step 8:- Increment i+1 and go to step 6.

Step 8:- Print the smallest number in Array.

Step 8:- Stop.

**Program :-**

#include<stdio.h>

int main()

{

int a[15];

int i,j,b;

printf("Enter the Elements in the Array ::\n");

for(i=0;i<7;i++)

{

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

}

for(i=0;i<7;i++)

{

for(j=i+1;j<7;j++)

{

if(a[i]>a[j])

{

b=a[i];

a[i]=a[j];

a[j]=b;

}

}

}

printf("\t%d is the Smallest in the Array\n",a[0]);

return 0;

}

**Output :- Enter the Elements in the Array ::**

**56,76,34,23,13,32,15**

**13 is the Smallest in the Array**

**Observation :-** After performing the experiment we observed that Values can be stored in a One dimensional Array and we were able to find out the smallest Number in the Array by swapping the values. The smallest number is located at the 0th location and after that print the 0th location. It took 0.16 sec Compilation Time.

* Problem C

**Even and Odd Numbers**

**AIM :-** To write a ‘c’ Program to take input of Numbers and find out the Even and odd Number in the Array**.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start

Step 2:- Declare ‘a[15]’ variable for array and i.

Step 3:- Check For condition i<?, if it is true go to step 4 otherwise go to step 6.

Step 4:- Read variable ‘a’ by loop.

Step 5:- Increment to i+1and go to step 3.

Step 6:- Check For condition i<?, if it is true go to step 7 otherwise go to step 9.

Step 7:- if(a[i]%2==0), print Even Numbers.

Step 8:- Increment to i+1 and go to step 6.

Step 9:- Check For condition i<?, if it is true go to step 10 otherwise go to step 12.

Step 10:- if(a[i]%2!=0), print Odd Numbers.

Step 11:- Increment to i+1 and go to step 9.

Step 12:- Stop.

**Program :-**

**#include<stdio.h>**

**int main()**

**{**

**int a[15];**

**int i;**

**printf("Enter the Elements in the Array ::\n");**

**for(i=0;i<7;i++)**

**{**

**scanf("%d",&a[i]);**

**}**

**printf("\n\nEven Numbers\n");**

**for(i=0;i<7;i++)**

**{**

**if(a[i]%2==0)**

**{**

**printf("%d\n",a[i]);**

**}**

**}**

**printf("\n\nOdd Numbers\n");**

**for(i=0;i<7;i++)**

**{**

**if(a[i]%2!=0)**

**{**

**printf("%d\n",a[i]);**

**}**

**}**

**return 0;**

**}**

**Output :- Enter the Elements in the Array ::**

**4**

**3**

**76**

**98**

**35**

**32**

**45**

**Even Numbers**

**4**

**76**

**98**

**32**

**Odd Numbers**

**3**

**35**

**45**

**Observation :-** After performing the experiment we observed that Values can be stored in a One dimensional Array and we were able to find out the Even and Odd Number in the Array by Dividing the number by 2 and then checking the remainder of the Division. If it is 0 ,the number is Even and if it is 1 ,the number is odd. It took 0.30 sec Compilation Time.

* Problem D

**Sum of Elements**

**AIM :-** To write a ‘c’ Program to take input of Numbers and find out the Sum of Number in the Array**.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start

Step 2:- Declare ‘a[15]’ variable for array and i, b=0.

Step 3:- Check For condition i<?, if it is true go to step 4 otherwise go to step 6.

Step 4:- Read variable ‘a’ by loop.

Step 5:- Increment to i+1and go to step 3.

Step 6:- Check For condition i<?, if it is true go to step 7 otherwise go to step 9.

Step 7:- calculate sum by b+=a[i];.

Step 8:- Increment to i+1 and go to step 6.

Step 9:- Print the value of ‘b’ variable.

Step 10:- Stop.

**Program :-**

**#include<stdio.h>**

**int main()**

**{**

**int a[15];**

**int i, b=0;**

**printf("Enter the Elements in the Array ::\n");**

**for(i=0;i<7;i++)**

**{**

**scanf("%d",&a[i]);**

**}**

**for(i=0;i<7;i++)**

**{**

**b+=a[i];**

**}**

**printf("The Sum of Values in the Array is ....%d\n",b);**

**return 0;**

**}**

**Output :- Enter the Elements in the Array ::**

**56**

**45**

**32**

**53**

**12**

**75**

**89**

**The Sum of Values in the Array is ....362**

**Observation :-** After performing the experiment we observed that Values can be stored in a One dimensional Array and we were able to find out the sum of Number in the Array by adding each element in the array and stored in b variable. It took 0.20 sec Compilation Time.

* Problem E

**Average of Numbers**

**AIM :-** To write a ‘c’ Program to take input of Numbers and find out the Average of Numbers in the Array**.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start

Step 2:- Declare ‘a[15]’ float variable for array, b=0 and integer variable i.

Step 3:- Check For condition i<?, if it is true go to step 4 otherwise go to step 6.

Step 4:- Read variable ‘a’ by loop 7 times.

Step 5:- Increment to i+1and go to step 3.

Step 6:- Check For condition i<?, if it is true go to step 7 otherwise go to step 9.

Step 7:- calculate sum by b+=a[i];.

Step 8:- Increment to i+1 and go to step 6.

Step 9:- Print the value of ‘b’ variable.

Step 10:-Calculate Average by b/=7;.

Step 11:- Print the value of ‘b’ variable.

Step 11:- Stop.

**Program :-**

**#include<stdio.h>**

**int main()**

**{**

**float a[15],b=0;**

**int i;**

**printf("Enter the Elements in the Array ::\n");**

**for(i=0;i<7;i++)**

**{**

**scanf("%f",&a[i]);**

**}**

**for(i=0;i<7;i++)**

**{**

**b+=a[i];**

**}**

**printf("Sum :: %f\n",b);**

**b/=7;**

**printf("Average :: %f",b);**

**return 0;**

**}**

**Output :-**

**Enter the Elements in the Array ::**

**1**

**23**

**45**

**6**

**54**

**34**

**12**

**Sum :: 175.000000**

**Average :: 25.000000**

* **Observation :-** After performing the experiment we observed that Values can be stored in a One dimensional Array and we were able to find out the Average of Numbers in the Array by adding each element in the array and stored in b variable. Then divide the sum with the number of elements present in the Array. It took 0.36 sec Compilation Time.
* Problem A

**Sum of 2 Matrices**

**AIM:-** To write a ‘c’ program to take input of 2 Matrices and display the sum of it using **Array.**

**THEORY:-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare a[2][2], b[2][2], c[2][2] for Array and i, j for loops.

Step 3:- Check FOR condition i<?, if it is true go to step 4 otherwise go to step 8.

Step 4:- Check FOR condition j<?, if it is true go to step 5 otherwise go to step 7.

Step 5:- Read variable ‘a’ by nested loop.

Step 6:- Increment j+1 and go to step 4.

Step 7:- Increment i+1 and go to step 3.

Step 8:- Check FOR condition i<?, if it is true go to step 9 otherwise go to step 13.

Step 9:- Check FOR condition j<?, if it is true go to step 10 otherwise go to step 12.

Step 10:- Read variable ‘b’ by nested loop.

Step 11:- Increment j+1 and go to step 9.

Step 12:- Increment i+1 and go to step 8.

Step 13:- Check FOR condition i<?, if it is true go to step 14 otherwise go to step 19.

Step 14:- Check FOR condition j<?, if it is true go to step 15 otherwise go to step 18.

Step 15:- Calculate Sum by c[i][j]=a[i][j]+b[i][j]; and values are stored in variable ‘c’.

Step 16:- Print Values of ‘c’ variable.

Step 17:- Increment j+1 and go to step 14.

Step 18:- Increment i+1 and go to step 13.

Step 19:- Stop.

**Program:-**

#include<stdio.h>

int main()

{

int a[2][2],b[2][2],c[2][2];

int i,j;

printf("Enter the Values in the 1st Matrix!!\n");

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

printf("Row=%d And Column=%d\n",i,j);

scanf("%d",&a[i][j]);

}

}

printf("\n\nEnter the Values in the 2nd Matrix!!\n");

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

printf("Row=%d And Column=%d\n",i,j);

scanf("%d",&b[i][j]);

}

}

printf("\n\nThe Sum of 2 Matrices !!\n");

for(i=0;i<2;i++)

{

for(j=0;j<2;j++)

{

c[i][j]=a[i][j]+b[i][j];

printf("%d",c[i][j]);

}

printf("\n");

}

return 0;

}

**Output:-**

Enter the Values in the 1st Matrix!!

Row=0 And Column=0

1

Row=0 And Column=1

2

Row=1 And Column=0

1

Row=1 And Column=1

2

Enter the Values in the 2nd Matrix!!

Row=0 And Column=0

1

Row=0 And Column=1

2

Row=1 And Column=0

1

Row=1 And Column=1

2

The Sum of 2 Matrices !!

24

24

**Observation:-** After performing the experiment we observed that Values can be stored in a Multi-dimensional Array and we were able to find out the Sum of 2 Matrices. The Sum was stored in ‘c’ variable. It took 0.31 sec Compilation Time.

* **Problem B**

**Temperature Conversion**

**AIM:-** To write a ‘c’ program to take input of Temperature in Celsius and display the Temperature in Fahrenheit in one Multi-dimensional **Array.**

**THEORY:-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare variables a[2][5] for Array and i, j for loops, ‘f’ for Fahrenheit.

Step 3:- Check FOR condition i<?, if it is true go to step 4 otherwise go to step 14.

Step 4:- Check Condition if (i==0), if it is true go to step 5 otherwise go to step 8.

Step 5:- Check FOR condition j<?, if it is true go to step 6 otherwise go to step 13.

Step 6:- Read variable ‘a’ by nested loop for 1st Row.

Step 7:- Increment j+1 and go to step 5.

Step 8:- else

Step 9:- Check FOR condition j<?, if it is true go to step 10 otherwise go to step 13.

Step 10:- Calculate Temperature by f=(a[0][j]\*9/5+32;.

Step 11:- Assign values to 2nd Row of ‘a’ variable by a[i][j]=f;.

Step 12:- Increment j+1 and go to step 9.

Step 13:- Increment i+1 and go to step 3.

Step 14:- Check FOR condition i<?, if it is true go to step 15 otherwise go to step 19.

Step 15:- Check FOR condition j<?, if it is true go to step 16 otherwise go to step 18.

Step 16:- Print variable ‘a[i][j]’.

Step 17:- Increment j+1 and go to step 15.

Step 18:- Increment i+1 and go to step 14.

Step 19:- Stop.

**Program:-**

#include<stdio.h>

int main()

{

int a[2][5];

int i,j,f;

printf("Enter the Temperature in Degrees!!\n");

for(i=0;i<2;i++)

{

if(i==0)

{

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

{

scanf("%d",&a[i][j]);

}

}

else

{

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

{

f=(a[0][j]\*9/5)+32;

a[i][j]=f;

}

}

}

printf("\n\nCelsius To Fahrenheit\n");

for(i=0;i<2;i++)

{

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

{

printf("%d\t",a[i][j]);

}

printf("\n");

}

return 0;

}

**Output:-**

Enter the Temperature in Degrees!!

28

89

-23

23

13

Celsius To Fahrenheit

28 89 -23 23 13

82 192 -9 73 55

**Observation :-** After performing the experiment we observed that Values can be stored in a Multi-dimensional Array and we were able to convert the temperature from Celsius to Fahrenheit in a given Matrix. The Result of Temperature Conversion was stored in ‘f’ variable and then passed to ‘a’ variable. It took 0.20 sec Compilation Time.

* **Problem C**

**Transpose of Matrix**

**AIM:-** To write a ‘c’ program to take input from the user of a Matrix and display its Transpose in Multi-dimensional **Array.**

**THEORY:-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare variables a[3][3], b[3][3] for Array and i, j for loops.

Step 3:- Check FOR condition i<?, if it is true go to step 4 otherwise go to step 8.

Step 4:- Check FOR condition j<?, if it is true go to step 5 otherwise go to step 13.

Step 5:- Read variable ‘a’ by nested loop.

Step 6:- Increment j+1 and go to step 4.

Step 7:- Increment i+1 and go to step 3.

Step 8:- Check FOR condition i<?, if it is true go to step 9 otherwise go to step13 .

Step 9:- Check FOR condition j<?, if it is true go to step 10 otherwise go to step 12.

Step 10:- Print the values of ‘a’ in matrix form.

Step 11:- Increment j+1 and go to step 9.

Step 12:- Increment i+1 and go to step 8.

Step 14:- Check FOR condition i<?, if it is true go to step 15 otherwise go to step 19.

Step 15:- Check FOR condition j<?, if it is true go to step 16 otherwise go to step 18.

Step 16:- Assign b[i][j]=a[j][i], i.e rows of ‘a’ will become ‘b’ variable’s column and columns of ‘a’ will become ‘b’ variable’s row .

Step 17:- Print the Transposed Matrix.

Step 18:- Increment j+1 and go to step 15.

Step 19:- Increment i+1 and go to step 14.

Step 20:- Stop.

**Program:-**

#include<stdio.h>

int main()

{

int a[3][3],b[3][3];

int i,j;

printf("Enter the Values in the Matrix!!\n");

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

scanf("%d",&a[i][j]);

}

}

printf("\n\nThe Matrix\n");

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

printf("%d\t",a[i][j]);

}

printf("\n");

}

printf("\n\nThe Transpose of the Matrix !!\n");

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

b[i][j]=a[j][i];

printf("%d\t",b[i][j]);

}

printf("\n");

}

return 0;

}

**Output:-**

Enter the Values in the Matrix!!

12

34

32

45

65

36

16

41

12

The Matrix

12 34 32

45 65 36

16 41 12

The Transpose of the Matrix !!

12 45 16

34 65 41

32 36 12

**Observation :-** After performing the experiment we observed that Values can be stored in a Multi-dimensional Array and we were able to find the Transpose of the Matrix. By assigning rows with columns and columns with rows we got the Transposed Matrix . It took 0.16 sec Compilation Time.

* Problem A

**Linear Search**

**AIM :-** To write a ‘c’ Program to take input of the Numbers and search the element in the Array and display the location of the element**.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare integer variable a for array, i, j for loops and ‘r’, pos=-1.

Step 3:- Check FOR condition i<?, if it is true go to step 4 otherwise go to step 6.

Step 4:- Read variable ‘a’ by For loop.

Step 5:- Increment i+1 and go to step 3.

Step 6:- Read variable ‘r’ to search the element.

Step 7:- Check FOR condition i<?, if it is true go to step 8 otherwise go to step 12.

Step 8:- check condition if(a[j]==r), if it is true go to step 9 otherwise go to step 11.

Step 9:- Assign “pos” by ‘j’, i.e pos=j;

Step 10:- Print the value of “pos” variable.

Step 11:- Increment i+1 and go to step 7.

Step 12:- check condition if(pos==-1), if it is true go to step 13 otherwise go to step 14.

Step 13:- Print Value not found.

Step 14:- Stop.

**Program :-**

#include<stdio.h>

int main()

{

int a[6];

int i,j,r,pos=-1;

printf("Enter the Values in the Array\n");

for(i=0;i<6;i++)

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

printf("Enter the value you want to Search??\n");

scanf("%d",&r);

for(j=0;j<6;j++)

{

if(a[j]==r)

{

pos=j;

printf("Position of the Value is %d\n",pos);

}

}

if(pos==-1)

printf("Value Not Found!!!!");

return 0;

}

**Output :-**

Enter the Values in the Array

1

6

7

3

5

9

Enter the value you want to Search??

7

Position of the Value is 2

**Observation :-** After performing the experiment we observed that Values can be stored in a single variable using Array and we were able to take input of many values using looping statements and also were able to search the element in a single dimension Array. It took 0.24 sec Compilation Time.

* Problem B

**Binary Search**

**AIM :-** To write a ‘c’ Program to search the element in the Array, when an **Array** is **Sorted** and display the location of the element**.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. For example it is written as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare integer variable ‘a’ for array, ‘I’ for loops and ‘f’, ‘low’, ‘high’, ‘mid’.

Step 3:- Read variable ‘f’ to find the element.

Step 4:- Initialize low=0 and high=9.

Step 5:- Check **While** condition low<=high, if it is true go to step 6 otherwise go to step 14.

Step 6:- Assign ‘mid’, mid=(low+high)/2.

Step 7:- Check **if** condition a[mid]==f, if it is true go to step 8 otherwise go to step 10.

Step 8:- Print the element’s Location.

Step 9:- Break the loop by ‘break’ statement and go to step 14.

Step 10:- check **else if** condition a[mid]>f, if it is true go to step 11 otherwise go to step 12.

Step 11:- Update ‘high’ with high=mid +1 and go to step 5.

Step 12:- **Else**, go to step 13.

Step 13:- Update ‘low’ with low=mid -1 and go to step 5.

Step 14:- Stop.

**Program :-**

#include<stdio.h>

int main()

{

int a[10]={2,4,7,9,10,13,16,20,25,30};

int i,f,low,high,mid;

printf("Enter the Element to Search\n");

scanf("%d",&f);

low=0;

high=9;

while(low<=high)

{

mid=(low+high)/2;

if(a[mid]==f)

{

printf("The Element is present in %d location\n",mid);

break;

}

else if(a[mid]>f)

{

high=mid-1;

}

else

{

low=mid+1;

}

}

return 0;

}

**Output :-**

Enter the Element to Search

7

The Element is present in 2 location

**Observation :-** After performing the experiment we observed that Values can be stored in a single variable using Array and we were able to search the element from the **Sorted** Array in a single dimension Array. It divides the Array into 2 parts until the element is found. It took 0.31 sec Compilation Time.

* Problem **A**

**Bubble Sort**

**AIM :-** To write a ‘c’ Program to take input of Unsorted numbers and display the sorted list of numbers using **Array.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. Swapping means exchanging the values between 2 variables. For example Array is initialized as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare integer variable i, j for loop and ‘a’ for array and ‘b’ as temporary.

Step 3:- Check **FOR** condition i<5, if it is true go to step 4 otherwise go to step 6.

Step 4:- Read variable ‘a’ by **For** loop.

Step 5:- Increment i+1 and go to step 3.

Step 6:- Check **FOR** condition i<=4-1, if it is true go to step 7 otherwise go to step 12.

Step 7:- Check **FOR** condition j<=4-1, if it is true go to step 8 otherwise go to step 11.

Step 8:- Check **if** condition a[j]>a[j+1], if it is true go to step 9 otherwise go to step 10.

Step 9:- Swap the values by b=a[j] then a[j]=a[j+1] and then a[j+1]=b.

Step 10:- Increment j+1 and go to step 7.

Step 11:- Increment i+1 and go to step 6.

Step 12:- Check **FOR** condition i<5, if it is true go to step 13 otherwise go to step 15.

Step 13:- Print variable ‘a’.

Step 14:- Increment i+1 and go to step 12.

Step 15:- Stop.

**Program :-**

#include<stdio.h>

int main()

{

int a[10],b,i,j;

printf("Enter the elements in the Array...\n");

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

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

for(i=0;i<=4-1;i++)

{

for(j=0;j<=4-1;j++)

{

if(a[j]>a[j+1])

{

b=a[j];

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

a[j+1]=b;

}

}

}

printf("\nAfter Sorting:-\n");

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

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

return 0;

}

**Output :-**

Enter the elements in the Array...

69

45

12

43

76

After Sorting:-

12 43 45 69 76

**Observation :-** After performing the experiment we observed that Values can be stored in a single variable using Array and we were able to take input of some values in an Array using looping statements. The largest value is placed at the End of an Array. Thus, numbers are sorted from Ascending to Descending order. Printing the sorted elements in the same manner using loop. It took 0.16 sec Compilation Time.

* Problem **B**

**Insertion Sort**

**AIM :-** To write a ‘c’ Program to take input of Unsorted numbers and display the sorted list of numbers using **Array.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. Swapping means exchanging the values between 2 variables. For example Array is initialized as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare integer variable i, j for loop and ‘a’ for array and ‘temp’ as temporary.

Step 3:- Check **FOR** condition i<5, if it is true go to step 4 otherwise go to step 6.

Step 4:- Read variable ‘a’ by **For** loop.

Step 5:- Increment i+1 and go to step 3.

Step 6:- Check **FOR** condition i<=5-1, if it is true go to step 7 otherwise go to step 13.

Step 7:- Assign temp=a[i] and j=i-1.

Step 8:- Check **while** condition (j>=0 && a[j]>temp) , if it is true go to step 9 otherwise go to step 11.

Step 9:- Swap the values by a[j+1]=a[j].

Step 10:- Decrement j-1 and go to step 8.

Step 11:- Assign a[j+1]=temp.

Step 12:- Increment i+1 and go to step 6.

Step 13:- Check **FOR** condition i<5, if it is true go to step 14 otherwise go to step 16.

Step 14:- Print variable ‘a’.

Step 15:- Increment i+1 and go to step 12.

Step 16:- Stop.

**Program :-**

#include<stdio.h>

int main()

{

int a[10];

int i,temp,j;

printf("Enter the elements in the Array...\n");

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

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

for(i=1;i<=5-1;i++)

{

temp=a[i];

j=i-1;

while(j>=0 && a[j]>temp)

{

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

j--;

}

a[j+1]=temp;

}

printf("\nFinal Sort\n");

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

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

return 0;

}

**Output :-**

Enter the elements in the Array...

45

69

23

12

32

Final Sort

12 23 32 45 69

**Observation :-** After performing the experiment we observed that Values can be stored in a single variable using Array and we were able to take input of some values in an Array using looping statements. The Smallest value is placed at the Front of an Array. Thus, numbers are sorted from Ascending to Descending order. Printing the sorted elements in the same manner using loop. It took 0.22 sec Compilation Time.

* Problem **C**

**Selection Sort**

**AIM :-** To write a ‘c’ Program to take input of Unsorted numbers and display the sorted list of numbers using **Array.**

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. Swapping means exchanging the values between 2 variables. For example Array is initialized as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** Step 1:- Start.

Step 2:- Declare variable i, j for loop and ‘a’ for array and ‘temp’ , ‘mini’ & ‘n’.

Step 3:- Read variable ‘n’.

Step 4:- Check **FOR** condition i<n, if it is true go to step 5 otherwise go to step 7.

Step 5:- Read variable ‘a’ by **For** loop.

Step 6:- Increment i+1 and go to step 4.

Step 7:- Check **FOR** condition i<n, if it is true go to step 8 otherwise go to step 15.

Step 8:- Assign mini=i.

Step 9:- Check **FOR** condition j<n , if it is true go to step 10 otherwise go to step 13.

Step 10:- Check **if** condition a[j]<a[mini], if it is true go to step 11 otherwise go to step 12.

Step 11:- Assign mini=j.

Step 12:- Increment j+1 and go to step 9.

Step 13:- Check **if** condition mini!=j, if it is true go to step 14 otherwise go to step 15.

Step 14:- Swap the values by temp=a[j] then a[j]=a[mini] and then a[mini]=temp.

Step 15:- Increment i+1 and go to step 7.

Step 16:- Check **FOR** condition i<n, if it is true go to step 17 otherwise go to step 19.

Step 17:- Print variable ‘a’.

Step 18:- Increment i+1 and go to step 14.

Step 19:- Stop.

**Program :-**

#include<stdio.h>

int main()

{

int a[10];

int i,j,n,mini,temp,k;

printf("How many values you want to Enter???\n");

scanf("%d",&n);

printf("Enter The values in the Array!!\n");

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

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

for(i=0;i<n-1;i++)

{

mini=i;

for(j=i+1;j<n;j++)

{

if(a[j]<a[mini])

{

mini=j;

}

}

if(mini!=i)

{

temp=a[i];

a[i]=a[mini];

a[mini]=temp;

}

}

printf("\n\nSorted Array is.......\n");

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

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

return 0;

}**Output :-**

How many values you want to Enter???

5

Enter The values in the Array!!

54

69

96

87

34

Sorted Array is.......

34 54 69 87 96

**Observation :-** After performing the experiment we observed that Values can be stored in a single variable using Array and we were able to take input of some values in an Array using looping statements. The Smallest value is placed at the Front of an Array. Thus, numbers are sorted from Ascending to Descending order. Printing the sorted elements in the same manner using loop. It took 0.14 sec Compilation Time.

* Problem **D**

**Merge Sort**

**AIM :-** To write a ‘c’ Program to take input of Unsorted numbers and display the sorted list of numbers using **Array** and **Merge sort** Technique.

**Theory :-** The collection of multiple Data in a single variable is known as **Array**. Swapping means exchanging the values between 2 variables. For example Array is initialized as a[100];, a[5]={1,2,3,4,5};, a[]={1,2,3,4,5}; .

**Algorithms :-** UDF(User Defined Function) mergesort ( )

Step 1:- Start.

Step 2:- Receive Values for integer variable ‘A’ for array, ‘low’ and ‘high’ from main function().

Step 3:- Check **if** condition low<high, if it is true go to step 4 otherwise go to step 8.

Step 4:- Declare and Assign integer variable ‘mid’ and mid=(low+high)/2.

Step 5:- Call user defined function “mergesort” **Recursively** by passing ‘A’, ‘low’, ‘mid’ variable’s values as arguments.

Step 6:- Call user defined function “mergesort” **Recursively** by passing ‘A’, ‘mid+1’, ‘high’ variable’s values as arguments.

Step 7:- Call user defined function “merge” by passing ‘A’, ‘low’, ‘mid’, ‘high’ variable’s values as arguments.

Step 8:- Stop.

UDF merge ()

Step 1:- Start.

Step 2:- Receive Values for integer variable ‘A’ for array, ‘low’, ‘mid’ and ‘high’ from mergesort function().

Step 3:- Declare integer variable ‘b’ for array, ‘i’, ‘j’ and ‘k’.

Step 4:- Assign i=low, j=mid+1, k=low.

Step 5:- Check **while** condition **j<=mid && j<=high**, if it is true go to step 6 otherwise go to step 13.

Step 6:- check **if** condition A[i]<=A[j], if it is true go to step 7 otherwise step 9.

Step 7:- Assign b[k]=A[i].

Step 8:- Increment i+1 and go to step 12.

Step 9:- **else,** go to step 10 otherwise step .

Step 10:- Assign b[k]=A[j].

Step 11:- Increment j+1 and go to step 12.

Step 12:- Increment k+1 and go to step 5.

Step 13:- Check **if** condition i>mid, if it is true go to step 14 otherwise go to step 17.

Step 14:- Check **while** condition **j<=high**, if it is true go to step 15 otherwise go to step 17.

Step 15:- Assign b[k]=A[j].

Step 16:- Increment j+1 & k+1 and go to step 14.

Step 17 :- **else** go to step 18 otherwise step 21.

Step 18 :- Check **while** condition **i<=mid**, if it is true go to step 19 otherwise go to step 21.

Step 19 :- Assign b[k]=A[i].

Step 20 :- Increment i+1 & k+1 and go to step 18.

Step 21 :- initialize k=low; Check **For** condition **k<=high**, if it is true go to step 22 otherwise go to step 24.

Step 22:- Assign A[k]=b[k].

Step 23:- Increment k+1 and go to step 21.

Step 24:- Stop.

UDF printarray()

Step 1:- Start.

Step 2:- Receive Values for integer variable ‘A’ for array, ‘size’ from main function().

Step 3:- Declare ‘i’ variable for loop.

Step 4:- initialize i=0; Check **For** condition **i<size**, if it is true go to step 5 otherwise go to step 6.

Step 5:- Print values of variable ‘A’.

Step 6:- Stop.

Main function ()

Step 1:- Start.

Step 2:- Declare integer variable ‘i’ and ‘n’.

Step 3:- Read value for ‘n’ variable.

Step 4:- Declare ‘a’ for Array.

Step 5:- initialize i=0; Check **For** condition **i<n**, if it is true go to step 6 otherwise go to step 7.

Step 6:- Read value for ‘a’ by loop.

Step 7:- Call the printArray function and pass the arguments ‘a’ and ‘n’.

Step 8:- Call the mergesort function and pass arguments ‘a’, ‘0’ and ‘n-1’.

Step 9:- Call again the printArray function and the arguments ‘a’ and ‘n’.

Step 10:- Stop.

**Program :-**

#include <stdio.h> //Merge sort

void mergeSort(int A[], int low, int high)

{

if(low<high)

{

int mid=(low+high)/2;

mergeSort(A,low,mid);

mergeSort(A,mid+1,high);

merge(A,low,mid,high);

}

}

void merge(int A[], int low, int mid, int high)

{

int b[high];

int i, j, k;

i=low;

j=mid+1;

k=low;

while(i<=mid && j<=high)

{

if(A[i]<=A[j])

{

b[k]=A[i];

i++;

}

else

{

b[k]=A[j];

j++;

}

k++;

}

if(i>mid)

{

while(j<=high)

{

b[k]=A[j];

j++;

k++;

}

}

else

{

while(i<=mid)

{

b[k]=A[i];

i++;

k++;

}

}

for(k=low;k<=high;k++)

A[k]=b[k];

}

void printArray(int A[], int size)

{

int i;

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

printf("%d ",A[i]);

printf("\n");

}

int main()

{

int i,n;

printf("How Many values you want to Enter???\n");

scanf("%d",&n);

int a[n];

printf("Enter the Values in the Array???\n");

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

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

printf("\n\nGiven array is \n");

printArray(a, n);

mergeSort(a, 0, n-1);

printf("\nSorted array is \n");

printArray(a, n);

return 0;

}

**Output :-**

How Many values you want to Enter???

5

Enter the Values in the Array???

9

8

7

6

5

Given array is

9 8 7 6 5

Sorted array is

5 6 7 8 9

**Observation :-** After performing the experiment we observed that Values can be stored in a single variable using Array and we were able to take input of some values in an Array using looping statements. The Smallest value is placed at the Front of an Array. Thus, numbers are sorted from Ascending to Descending order. Printing the sorted elements in the same manner using loop. It took 0.31 sec Compilation Time.

* Problem **A**

**Stack**

**AIM :-** To write a ‘c’ Program to perform operations on **Stack** of Push and Pop**.**

**Theory :-** A **Stack** is also a **Linear Data Structure** which Stores the data. The Insertion and Deletion Happen from only one end. It uses the concept of **FILO** or **LIFO.**

**Algorithms :-** Step 1:- Start.

Step 2:- Declare integer Global Variables stack[10] and top=-1.

Step 3:- Stop.

UDF Push ()

Step 1:- Start.

Step 2:- Declare integer variable x and n=sizeof(stack)/sizeof(stack[0]).

Step 3:- Read variable ‘x’.

Step 4:- Check **if** condition top==n-1, if it is true go to step 5 otherwise go to step 6.

Step 5:- Print Stack is Full.

Step 6:- Increment top++ and stack[top]=x.

Step 7:- Stop.

UDF Pop ()

Step 1:- Start.

Step 2:- Check **IF** condition top==-1, if it is true go to step 3 otherwise go to step 4.

Step 3:- Print Stack is Empty.

Step 4:- Declare “loc” variable and assign loc=stack[top] and decrement top--.

Step 5:- Print Popped element and display “loc”.

Step 6:- Stop.

Main()

Step 1:- Start.

Step 2:- Declare ‘i’ variable for loop.

Step 3:- Call Push operation 6 times.

Step 4:- Call Pop operation 4 times.

Step 5:- Print stack by loop.

Step 6:- Stop.

**Program :-**

#include<stdio.h>

int stack[10]; //Global Variables

int top=-1;

void push()

{

int x, n=sizeof(stack)/sizeof(stack[0]);

printf("Enter the Element in the Stack???\n");

scanf("%d",&x);

if(top==n-1)

printf("Stack is full!!!\n");

else

{

top++;

stack[top]=x;

}

}

void pop()

{

if(top==-1)

printf("Stack is Empty\n");

else

{

int loc=stack[top];

top--;

printf("Popped element is %d\n",loc);

}

}

int main()

{

int i;

push();

push();

push();

push();

push();

push();

pop();

pop();

pop();

pop();

printf("\n\nElements of Stack are....\n");

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

printf("%d\n",stack[i]);

return 0;

}

**Output :-**

Enter the Element in the Stack???

1

Enter the Element in the Stack???

2

Enter the Element in the Stack???

3

Enter the Element in the Stack???

4

Enter the Element in the Stack???

5

Enter the Element in the Stack???

6

Popped element is 6

Popped element is 5

Popped element is 4

Popped element is 3

Elements of Stack are....

1

2

**Observation :-** After performing the experiment we observed that Values can be stored in a Stack and we were able to use UDF Push and Pop operations. Insertion and Deletion of Values happened from one end only. It took 0.31 sec Compilation Time.

* Problem **B**

**Queue**

**AIM :-** To write a ‘c’ Program to perform operations on **Queue** of **Enqueue** and **Dequeue.**

**Theory :-** A **Queue** is also a **Linear Data Structure** which Stores the data. The Insertion and Deletion Happen from only Two end. The Insertion happens from last and Deletion from First. It uses the concept of **FIFO.**

**Algorithms :-** Step 1:- Start.

Step 2:- Declare integer Global Variables queue[10] and f=-1, r=-1.

Step 3:- Stop.

UDF Enqueue ()

Step 1:- Start.

Step 2:- Declare integer variable “data” and n=sizeof(queue)/sizeof(queue[0]).

Step 3:- Read variable “data”.

Step 4:- Check **if** condition r==n-1, if it is true go to step 5 otherwise go to step 6.

Step 5:- Print Queue is Full.

Step 6:- Check **if** condition f==-1, if it is true go to step 7 otherwise go to step 8.

Step 7:- Increment f++.

Step 8:- Increment r++ and queue[r]=data

Step 9:- Stop.

UDF Dequeue ()

Step 1:- Start.

Step 2:- Check **IF** condition f==r, if it is true go to step 3 otherwise go to step 4.

Step 3:- Print Queue is Empty.

Step 4:- Declare “loc” variable and assign loc=queue[f] and Increment f++.

Step 5:- Print “loc” variable element has left the Queue.

Step 6:- Stop.

Main()

Step 1:- Start.

Step 2:- Declare ‘i’ variable for loop.

Step 3:- Call Enqueue operation 6 times.

Step 4:- Call Dequeue operation 2 times.

Step 5:- Print “queue” by loop.

Step 6:- Stop.

**Program** :-

#include<stdio.h>

int queue[10]; //Global Variables

int f=-1, r=-1;

void enqueue()

{

int data, n=sizeof(queue)/sizeof(queue[0]);

printf("Enter the Element in the Queue???\n");

scanf("%d",&data);

if(r==n-1)

{

printf("The Queue is Full\n");

}

else

{

if(f==-1)

f++;

r++;

queue[r]=data;

}

}

void dequeue()

{

int loc;

if(f==r)

printf("The Queue is Empty\n");

else

{

loc=queue[f];

f++;

printf("%d Element has left the Queue\n",loc);

}

}

int main()

{

int i;

enqueue();

enqueue();

enqueue();

enqueue();

enqueue();

enqueue();

dequeue();

dequeue();

printf("\n\nElements of Queue are....\n");

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

if(i>=f)

printf("%d\n",queue[i]);

return 0;

}

**Output** :-

Enter the Element in the Queue???

1

Enter the Element in the Queue???

2

Enter the Element in the Queue???

3

Enter the Element in the Queue???

4

Enter the Element in the Queue???

5

Enter the Element in the Queue???

6

1 Element has left the Queue

2 Element has left the Queue

Elements of Queue are....

3

4

5

6

**Observation :-** After performing the experiment we observed that Values can be stored in a Queue and we were able to use UDF Enqueue and Dequeue operations. Insertion and Deletion of Values happened from two end only. The Insertion happens from last of the Queue and Deletion happens from the front of the Queue. It took 0.20 sec Compilation Time.

* Problem **A**

**Singly Linked List**

**AIM :-** To write a ‘c’ Program to perform operations on **Singly Linked List** of Insert, Delete and Display**.**

**Theory :-** A **Singly Linked List** is also a **Linear Data Structure** which Stores the data. The **Data** and **Address** of the next list is stored in the block of memory called **NODE** which is **Dynamically Allocated**. The **NODES** are created anywhere in the memory block which are linked with their respective **Addresses.**

**Algorithms :-**

Declare a Structure node and create there members of int type ‘data’ and of struct type ‘next’ pointer.

Declare struct type Global Pointer Variables struct node **\*head** and struct node **\*tail** pointing to **NULL**.

**UDF(User Defined Function) start\_insert()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘new\_node’ of struct node datatype and dynamically allocate it **new\_node=(struct node\*)malloc(sizeof(struct node))**.

Step 3:- Read value for ‘new\_node->data’.

Step 4:- Assign **new\_node->next=head** and **head=new\_node**.

Step 5:- Stop.

**UDF last\_insert ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘new\_node’ and ‘temp’ of struct node datatype and dynamically allocate it **new\_node=(struct node\*)malloc(sizeof(struct node))**.

Step 3:- Read value for ‘new\_node->data’.

Step 4:- Check **IF** Condition **head==tail**?, if true go to step 5 otherwise step 6.

Step 5:- Assign **new\_node->next=tail** and **head=new\_node**.

Step 6:- Else.

Step 7:- Assign **temp=head**.

Step 8:- Check **while** condition temp->next!=tail, if true go to step 9 otherwise step 10.

Step 9:- Assign **temp->next=new\_node** and **new\_node->next=tail**.

Step 10:- Stop.

**UDF random\_insert ()**

Step 1:- Start.

Step 2:- Declare integer variables ‘i’, ‘pos’.

Step 3:- Declare a pointer variable ‘new\_node’ and ‘temp’ of struct node datatype and dynamically allocate it **new\_node=(struct node\*)malloc(sizeof(struct node))**.

Step 4:- Check **IF** condition new\_node== NULL?, if true go to step 5 otherwise step 6.

Step 5:- Print OVERFLOW.

Step 6:- Else.

Step 7:- Read value in ‘new\_node->data’.

Step 8:- Read value for ‘Pos’.

Step 9:- Assign temp=head.

Step 10:- Check **for** loop condition, if i<pos go to step 11 otherwise step .

Step 11:- Assign **temp=temp->next**.

Step 12:- Check **if condition**, if temp=tail go to step 13 otherwise step 14.

Step 13:- Print Can’t insert.

Step 14:- Assign **new\_node->next=temp->next** and **temp->next=new\_node**.

Step 15:- Stop.

**UDF start\_delete ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘new\_node’ of struct node datatype.

Step 3:- Check **if** condition, if **head==tail** go to step 4 otherwise go to step 5.

Step 4:- Print “List is Empty”.

Step 5:- Else.

Step 6:- Assign **new\_node=head** and **head=new\_node->next**.

Step 7:- Delete the node using free function **free(new\_node)**.

Step 8:- Stop.

**UDF last\_delete ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘new\_node’ and ‘temp’ of struct node datatype.

Step 3:- Assign **new\_node=head**.

Step 4:- Check **if** condition, if **head==tail** is true go to step 5 otherwise step 6.

Step 5:- Print “List is Empty”.

Step 6:- Check **else if** condition, if **new\_node->next==tail** go to step 7 otherwise step 9.

Step 7:- Assign **head=tail**.

Step 8:- Delete the node using free function **free(new\_node)**.

Step 9:- Else.

Step 10:- Check **while** condition, if **new\_node->next!=tail** go to step 11 otherwise step 12.

Step 11:- Assign **temp=new\_node** and **new\_node=new\_node->next**.

Step 12:- Assign **temp->next=tail**.

Step 13:- Delete the node using free function **free(new\_node)**.

Step 14:- Stop.

**UDF random\_delete ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘**new\_node**’ and ‘**temp**’ of struct node datatype.

Step 3:- Declare integer variables ‘**pos**’ and ‘**i**’.

Step 4:- Read value for ‘**pos**’.

Step 5:- Assign **new\_node=head**.

Step 6:- Check **for loop** condition, if **i<pos** go to step 7 otherwise step 10.

Step 7:- Assign **temp=new\_node** and **new\_node=new\_node->next**.

Step 8:- Check **if** condition, if **new\_node ==NULL** go to step 9 otherwise step 10.

Step 9:- Print “Can’t Delete”.

//End of **for loop**

Step 10:-Assign **temp->next=new\_node->next**

**Step 11:-** Delete the node using free function **free(new\_node)**.

Step 12:- Stop.

**UDF display ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘**temp**’ of struct node datatype.

Step 3:- Assign **temp=head**.

Step 4:- Check **if** condition, if **temp==tail** go to step 5 otherwise step 6.

Step 5:- Print “Nothing to Print”.

Step 6:- Else.

Step 7:- Print values using **While loop**.

Step 8:- Check **while loop** condition, if **temp!=tail** go to step 9 otherwise step 11.

Step 9:- Print the values of **‘temp->data’**.

Step 10:- Assign **temp=temp->next.**

Step 11:- Stop.

**Main()**

Step 1:- Start.

Step 2:- Declare ‘**choice**’ integer variable.

Step 3:- Check **while loop** condition, if choice!=8 go to step 4 otherwise step .

Step 4:- Read Value for ‘choice’ variable from 1 to 8 for different operations.

Step 5:- Check **if** condition, if choice==1 go to step 6 otherwise step 7.

Step 6:- Call “**start\_insert()**” function.

Step 7:- Check **else if** condition, if choice==2 go to step 8 otherwise step 9.

Step 8:- Call “**last\_insert()**” function.

Step 9:- Check **else if** condition, if choice==3 go to step 10 otherwise step 11.

Step 10:- Call “**random\_insert()**” function.

Step 11:- Check **else if** condition, if choice==4 go to step 12 otherwise step 13.

Step 12:- Call “**start\_delete()**” function.

Step 13:- Check **else if** condition, if choice==5 go to step 14 otherwise step 15.

Step 14:- Call “**last\_delete()**” function.

Step 15:- Check **else if** condition, if choice==6 go to step 16 otherwise step 17.

Step 16:- Call “**random\_delete()**” function.

Step 17:- Check **else if** condition, if choice==7 go to step 18 otherwise step 19.

Step 18:- Call “**display()**” function.

Step 19:- Check **else if** condition, if choice==8 go to step 20 otherwise step 21.

Step 20:- Terminate the loop using ‘**break’ statement**.

Step 21:- **Else**.

Step 22:- Print “Invalid Value!!!”.

//End loop

Step 6:- Stop.

**Program :-**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

struct node \*tail;

void start\_insert()

{

struct node \*new\_node;

new\_node=(struct node\*)malloc(sizeof(struct node));

printf("\n\nEnter the value in the node??\n");

scanf("%d",&new\_node->data);

new\_node->next=head;

head=new\_node;

printf("Node Inserted\n");

}

void last\_insert()

{

struct node \*new\_node, \*temp;

new\_node=(struct node\*)malloc(sizeof(struct node));

printf("\n\nEnter the value in the node??\n");

scanf("%d",&new\_node->data);

if(head==tail)

{

new\_node->next=tail;

head=new\_node;

printf("\nNode inserted");

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

temp->next=new\_node;

new\_node->next=tail;

printf("Node Inserted\n");

}

}

void random\_insert()

{

int i,pos,value;

struct node \*ptr, \*temp;

ptr=(struct node\*)malloc(sizeof(struct node));

if(ptr==NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter Data value\n");

scanf("%d",&value);

ptr->data=value;

printf("\nEnter the location after which you want to insert???\n");

scanf("%d",&pos);

temp=head;

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

{

temp=temp->next;

if(temp==NULL)

{

printf("\ncan't insert\n");

return;

}

}

ptr->next=temp->next;

temp->next=ptr;

printf("\nNode inserted");

}

}

void start\_delete()

{

struct node \*ptr;

if(head==tail)

printf("\n\nList is Empty\n");

else

{

ptr=head;

head=ptr->next;

printf("\n\n%d Data \nNode Deleted\n",ptr->data);

free(ptr);

}

}

void last\_delete()

{

struct node \*ptr,\*temp;

ptr=head;

if(head==tail)

{

printf("\nlist is empty");

}

else if(ptr->next==tail)

{

head=tail;

free(ptr);

printf("\nOnly node of the list deleted ...\n");

}

else

{

while(ptr->next!=tail)

{

temp=ptr;

ptr=ptr->next;

}

temp->next=tail;

free(ptr);

printf("\nDeleted Node from the last ...\n");

}

}

void random\_delete()

{

struct node \*ptr,\*temp;

int pos,i;

printf("\nEnter the location of the node after which you want to perform deletion \n");

scanf("%d",&pos);

ptr=head;

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

{

temp=ptr;

ptr=ptr->next;

if(ptr==NULL)

{

printf("\nCan't delete");

return;

}

}

temp->next=ptr->next;

free(ptr);

printf("\nDeleted node from %d location",pos+1);

}

void display()

{

struct node \*temp;

temp=head;

if(temp==tail)

printf("\n\nNothing to print");

else

{

printf("\n\nprinting values . . . . .\n");

while(temp!=tail)

{

printf("%d\t",temp->data);

temp=temp->next;

}

}

}

int main()

{

int choice;

while(choice!=6)

{

printf("\n\nChoose one option from the following list ...\n");

printf("\n1.To Insert at Start\n2.To Insert at Last\n3.To Insert at Random Location\n4.To Delete a Node from Start\n5.To Delete a Node from Last\n6.To Delete a Node from Random Location\n7.To Display and Traverse\n8.To Exit\n\n");

scanf("%d",&choice);

if(choice==1)

start\_insert();

else if(choice==2)

last\_insert();

else if(choice==3)

random\_insert();

else if(choice==4)

start\_delete();

else if(choice==5)

last\_delete();

else if(choice==6)

random\_delete();

else if(choice==7)

display();

else if(choice==8)

break;

else

printf("\n\nInvalid value!!!!\n");

}

return 0;

}

**Output :-**

Choose one option from the following list ...

1.To Insert at Start

2.To Insert at Last

3.To Insert at Random Location

4.To Delete a Node from Start

5.To Delete a Node from Last

6.To Delete a Node from Random Location

7.To Display and Traverse

8.To Exit

2

Enter the value in the node??

1

Node inserted

Tell Your choice from above

2

Enter the value in the node??

2

Node Inserted

Tell Your choice from above

2

Enter the value in the node??

3

Node Inserted

Tell Your choice from above

2

Enter the value in the node??

4

Node Inserted

Tell Your choice from above

2

Enter the value in the node??

5

Node Inserted

Tell Your choice from above

2

Enter the value in the node??

6

Node Inserted

Tell Your choice from above

5

Deleted Node from the last ...

Tell Your choice from above

4

1 Data

Node Deleted

Tell Your choice from above

7

printing values . . . . .

2 3 4 5

Tell Your choice from above

6

Enter the location of the node on which you want to perform deletion

2

Deleted node from 2 location

Tell Your choice from above

7

printing values . . . . .

2 3 5

Tell Your choice from above

8

**Observation :-** After performing the experiment we observed that Values can be stored in a memory in any location and we were able to link them, so they can be accessed easily one after the other. Insertion and Deletion of Values can happen from anywhere like from starting or Ending or Randomly. It took 0.16sec Compilation Time.