CSL 201

DATA STRUCTURE LAB

2020-2021

Name :.....Fathimath Mufeedha K P.....

Roll Number : .....67.....

Class: .....S3CSE .....

College : .....GCEKannur.....

Experiment No.	Experiment Heading
1	Linear Search
2.	Binary Search and Selection Sort
3.	Insertion Sort
4.	Polynomial Addition using Array Representation
5.	Transpose of a sparse Matrix
6.	Addition of sparse matrices
7.	Array Insertion
8.	Array Deletion
9.	Stack using Arrays
10.	Infix to postfix conversion and evaluation of postfix expression
11.	Infix to prefix conversion and evaluation of prefix expression
12.	Queue using Arrays
13.	Circular Queue using Arrays
14.	Priority Queue using Arrays
15.	Deque using Arrays
16.	Single Linked list operations
17.	Doubly linked list operations
18.	Circular linked list operations
19.	Queue using linked list
20.	Polynomial addition using linked list
21.	Polynomial multiplication using linked list
22.	Binary Tree using Arrays
23.	Binary Tree using Linked list
24.	Binary Search Tree Using Linked list

Expt.No: 1

LINEAR SEARCH

Aim : - Write a program to search for an element in an array using Linear Search.

Algorithm AlgoLinearSearch()

Input Specification : An input array, A[SIZE] and a key, value to be searched
Output Specification: Position of the searched element, if found, else an appropriate message
Data Structures Used : Arrays

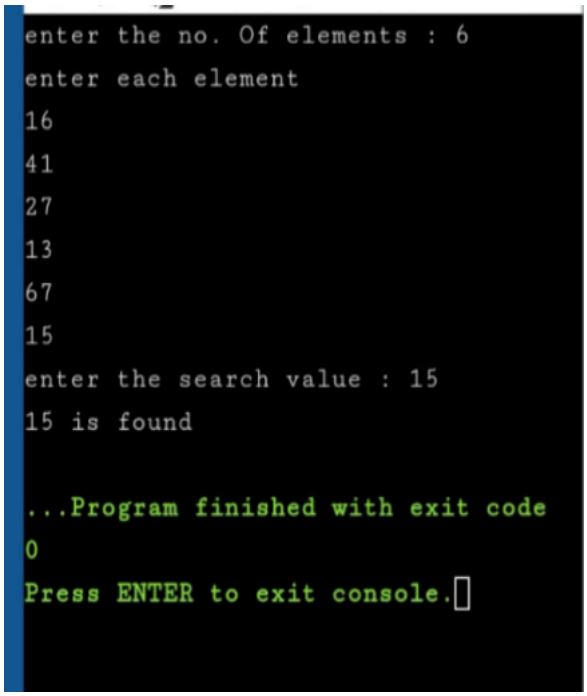
1. Start
2. Read A, n and key
3. i=1
4. while(i<=n)
5. if(A[i] == key)
6. break from the loop
7. end while
8. if i <=n then print 'key found'
9. else print 'key not found'
10. End if
11. Stop

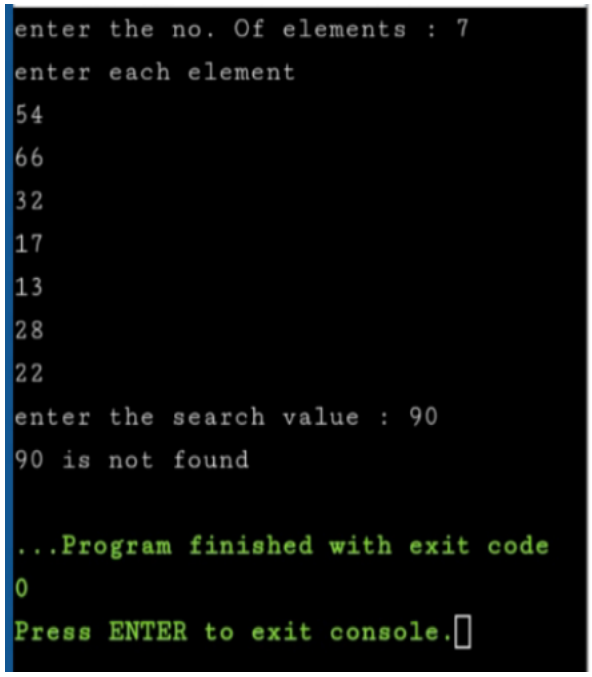
Program Code:

```
#include <stdio.h>
int main()
{
        int n,i,key;
        printf ("enter the no.Of elements:");
        scanf ("%d",&n);
        int a[n];
        printf ("enter eace element\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf ("enter the search value:");
        scanf ("%d",&key);
        i=0;
        //Linear search
        while(i<n)
        {
                if(a[i]==key)
```

```
            break;
            i++;
        }
        if (i<n)
            printf("%d is found at %d ",key,i);
        else
            printf("%d is not found",key);
        return 0;
}
```

Sample Output





Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 2

BINARY SEARCH AND SELECTION SORT

Aim : - Write a program to sort a set of numbers using selection sort and then search for a particular element in the array using Binary search.

Algorithm SelectionSort_BinarySearch(A,n,key)
Input Specification : An input array, A[SIZE] and a key, value to be searched
Output Specification: Array after Selection sort, Position of the searched element, if found, else an appropriate message
Data Structures Used : Arrays

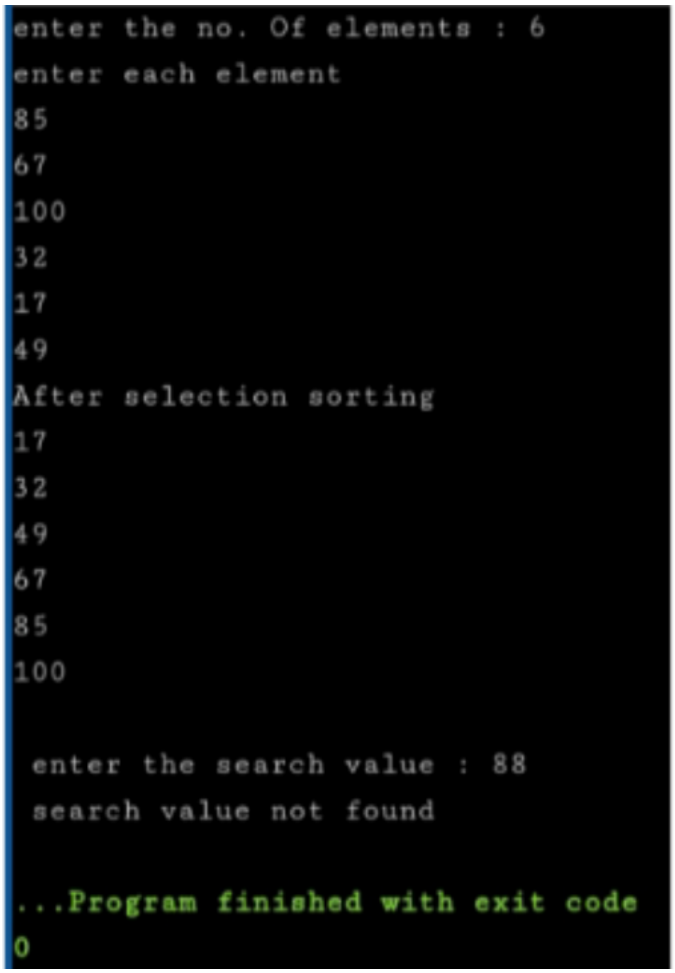
1. Start
2. for j ← 1 to n − 1
3. smallest ← j
4. for i ← j + 1 to n
5. if A[i] < A[smallest]
6. then smallest ← i
7. End if
8. End for
9. exchange A[j] ↔ A[smallest]
10. End for
11. Assign low =1, high = n,
12. while(low < = high)
13. mid = (low+ high)/2
14. If A[mid] < key then.
15. low = mid + 1
16. else if A[mid] > key then
17. high = mid -1
18. else print " Element found"
19. exit
20. end if
21. end while
22. Stop

Program Code:

```
#include <stdio.h>
int main()
{
        int n,i,small,t,j,key,low=0,mid;
        printf ("enter the no. Of elements:");
        scanf ("%d",&n);
        int a[n],high=n;
        printf ("enter each element\n");
        for (i=0;i<n;i++)
        {
```

```
        scanf("%d",&a[i]);
    }
    for (i=0;i<n-1;i++)          //Selection Sorting
    {
        small=i;
        for (j=i+1;j<n;j++)
        {
            if (a[j]<a[small])
            small=j;
        }
        t=a[i];
        a[i]=a[small];
        a[small]=t;
    }
    printf ("After selection sorting\n");
    for(i=0;i<n;i++)
    {
       printf("%d\n",a[i]);
    }

    printf ("\n enter the search value: ");
    scanf ("%d",&key);

    while (low<high)     //Binary Search
    {
        mid=(low+high)/2;
        if (a[mid]<key)
            low= mid+1;
        else if (a[mid]>key)
            high=mid-1;
        else
        {
            printf ("search value found at %d", mid);
            break;
        }
    }
    if (low>=high)
        printf ("search value not found");
    return 0;
}
```

Sample output:



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 3

INSERTION SORT

Aim : - Write a program to sort a set of numbers using Insertion sort.

Algorithm InsertionSort()

Input Specification : An input array, A[SIZE].
Output Specification: Sorted array
Data Structures Used : Arrays

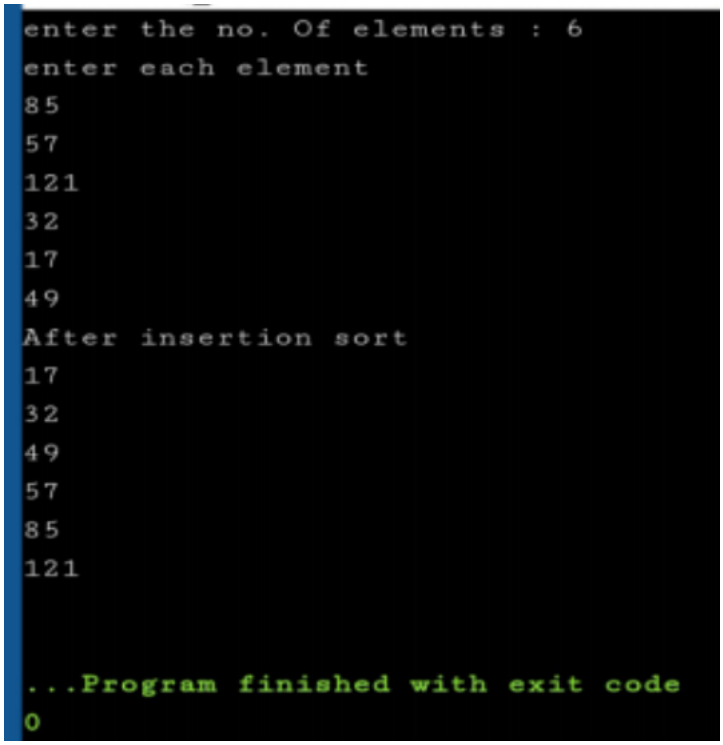
1. Start
2. For j = 2 to n
3. key = A[j]
4. i= j-1
5. While (i > 0 and A[i]> key) do
6. A[i+1] = A[i]
7. i = i - 1
8. End while
9. A[i + 1] = key
10. End for
11. Stop

Program Code:

```
#include <stdio.h>
int main()
{
        int n,i,j,key;
        printf ("enter the no. Of elements:");
        scanf ("%d",&n);
        int a[n];
        printf ("enter each element\n");
        for (i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        //Insertion_sort
        i=1;
        for (i=1;i<n;i++)
        {
            key=a[i];
            j= i-1;
            while (j>=0 && a[j]>key)
            {
            a[j+1]= a[j];
```

```
            j=j-1;
            }
            a[j+1]=key;
        }
        printf("After insertion sort\n");
        for (i=0;i<n;i++)
        {
             printf ("%d\n",a[i]);
        }
        return 0;
}
```

Sample Output:



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 4

POLYNOMIAL ADDITION USING ARRAY

Aim : - Write a program to read two polynomials and store them in an array. Find the sum of the two polynomials and display the input polynomials and the result and sum polynomial.

Algorithm Poly_Add(StartA, FinishA, StartB, FinishB, *StartD, *FinishD)
Input Specification : Input arrays, A[SIZE].
Output Specification: Input polynomials, Sum polynomial
Data Structures Used : Arrays, structure

1. Start
2. *StartD = avail
3. while(StartA <= FinishA AND StartB <=FinishB)
4. if (terms[StartA].expon < terms[StartB].expon)
5. attach(terms[StartB].coef, terms[StartB].expon)
6. StartB++
7. else if (terms[StartA].expon > terms[StartB].expon)
8. attach(terms[StartA].coef, terms[StartA].expon)
9. StartA++
10. else
11. coefficient = terms[StartA].coef + terms[StartB].coef
12. if(coefficient)
13. attach(coefficient, terms[StartA].expon)
14. end if
15. StartA++ , StartB++
16. end if
17. end while
18. while (startA<=finishA)
19. attach(terms[StartA].coef, terms[StartA].expon)
20. StartA++
21. end while
22. while(StartB <=FinishB)
23. attach(terms[StartB].coef, terms[StartB].expon)
24. StartB++
25. end while
26. *finishD= avail -1
27. Stop

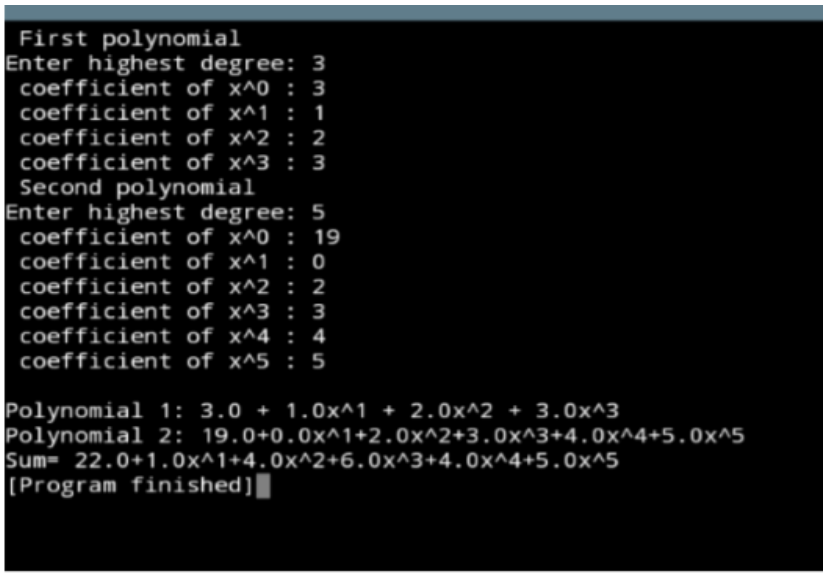
Program code:

```
#include <stdio.h>

struct polynomial {
      int expo;
      float coef;
} A[20], B[20], C[40];
```

```
int degreeA, degreeB, i, x = 0, y = 0, z = 0;

void readPoly()        //To Read Two Input Polynomials A and B
{
      printf (" First polynomial\nEnter highest degree: ");
      scanf ("%d", &degreeA);
      for (i = 0; i <= degreeA; i++)
      {
            printf (" coefficient of x^%d : ", i);
            scanf("%f", &A[i].coef);
            A[x++].expo = i;
      }
      printf(" Second polynomial\nEnter highest degree: ");
      scanf("%d", &degreeB);
      for (i = 0; i <= degreeB; i++)
      {
            printf(" coefficient of x^%d : ", i);
            scanf ("%f", &B[i].coef);
            B[y++].expo = i;
      }
}

void printPoly()       //To Print Polynomials
{
    printf("\nPolynomial 1: %.1f", A[0].coef);
    for (i = 1; i <= degreeA; i++)          //Printing First Polynomial A
          printf(" + %.1fx^%d", A[i].coef, A[i].expo);
    printf("\nPolynomial 2: %.1f", B[0].coef);
    for (i = 1; i <= degreeB; i++)          //Printing Second Polynomial B
          printf("+%.1fx^%d", B[i].coef, B[i].expo);
    printf("\nSum= %.1f", C[0].coef);
    for (i = 1; i < z; i++)                    //Printing Sum Polynomial C
          printf("+%.1fx^%d", C[i].coef, C[i].expo);
}

void addPoly()      //Polynomials Addtion
{
      if (degreeA > degreeB)
      {
            for (i = 0; i <= degreeB; i++)
            {
                C[z].coef = A[i].coef + B[i].coef;
                C[z].expo = A[i].expo;
                z++;
            }
             for (i = degreeB + 1; i <= degreeA; i++)
            {
                C[z].coef = A[i].coef;
```

```
                C[z].expo = A[i].expo;
                z++;
            }
        }
        else
        {
            for (i = 0; i <= degreeA; i++)
            {
                C[z].coef = A[i].coef + B[i].coef;
                C[z].expo = A[i].expo;
                z++;
            }
            int t = degreeA + 1;
            for (i = t; i <= degreeB; i++)
            {
                C[z].coef = B[i].coef;
                C[z].expo = B[i].expo;
                z++;
            }
        }
}
void main()        //Main Function
{
        readPoly();
        addPoly();
        printPoly();
}
```

Sample output:



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 5

TRANSPOSE OF SPARSE MATRIX

Aim : - Write program to read a matrix in normal form, converts them into tuple form and display it. And also find the transpose of the matrix represented in tuple form and display it.

Algorithm Transpose()
Input Specification : Input array- M[SIZE][SIZE], Structure - A[SIZE],B[SIZE][SIZE], row, column, matrix values
Output Specification: Input matrix and its transpose
Data Structures Used : Arrays, Structure

1. Start
2. t = A[0].value
3. B[0].row = A[0].col, B[0].col = A[0].row, B[0].value = t
4. if t > 0 then
5. currentB = 1
6. For i = 0 to A[0].col -1 do
7. For j = 1 to t do
8. if A[j].col == i then
9. B[currentB].row = A[j].col,
10. B[currentB].col = A[j].row,
11. B[currentB].value = A[j].value,
12. currentB++
13. end if
14. end For
15. end For
16. end if
17. Stop

Program code:

```
#include <stdio.h>
#include <stdlib.h>
#define Max_Term 100
int MyMatrix[50][50];

struct matrix
{
        int row, col, value;
} A[Max_Term], B[Max_Term];


void sparse(int,int);       //Function declaration
void read_print()           //To read and print Mymatrix
{
        int row, col,i, j;
        printf("Enter the number of rows :");
```

```
        scanf("%d", &row);
        printf("Enter the number of columns :");
        scanf("%d", &col);
        printf("Enter the matrix elements:\n");
        for(i=0; i<row;i++)
        {
            for(j=0; j<col; j++)
            {
                scanf("%d", &MyMatrix[i][j]);
            }
        }
        printf("\nYour matrix is :\n");
        for(i=0;i<row;i++)
        {
             for(j=0;j<col;j++)
             {
                  printf("%d\t", MyMatrix[i][j]);
             }
             printf("\n");
        }
        sparse(row, col);
}
void sparse( int r, int c)          //Creating sparse matrix
{
        int i, j, k=1;
        A[0].row= r;
        A[0].col= c;
        for(i=0;i<r;i++)
        {
             for(j=0;j<c;j++)
             {
                 if(MyMatrix[i][j]!= 0)
                 {
                     A[k].row=i;
                     A[k].col=j;
                     A[k].value = MyMatrix[i][j];
                     k++;
                 }
             }
        }
        A[0].value = k-1;     //No. Of Non-Zero Values
}
void printSparse ()        //To print in tuple form
{
        int i,t;
```

```
        t = A[0].value;
        printf ("\nSparse Matrix\n");
        for(i = 0; i <= t; i++)
        printf("%d\t%d\t%d\n", A[i].row, A[i].col, A[i].value);
}
void Transpose()     //To find transpose
{
        int t, i, j, currentB;
        t=A[0].value;
        B[0].row=A[0].col;
        B[0].col=A[0].row;
        B[0].value=t;
        if (t>0)
        {
            currentB=1;
            for(i=0; i<A[0].col; i++)
            {
                for(j=1; j<=t; j++)
                {
                    if(A[j].col == i)
                    {
                        B[currentB].row = A[j].col;
                        B[currentB].col = A[j].row;
                        B[currentB].value = A[j].value;
                        currentB++;
                    }
                }
            }
        }
}
void printTranspose()    //To print transpose
{
      int i;
      printf("\nTranspose of the Sparse Matrix\n");
      for(i=0; i<=B[0].value; i++)
            printf("%d\t%d\t%d\n", B[i].row, B[i].col, B[i].value);
}
void main()
{
       read_print();
       printSparse();
       Transpose();
       printTranspose();
}
```

Sample Output:

```
Enter the number of rows :6
Enter the number of columns :6
Enter the matrix elements:
0 0 0 0 9 0
0 8 0 0 0 0
4 0 0 2 0 0
0 0 0 0 0 5
0 0 2 0 0 0
0 0 0 0 0 6

Your matrix is :
0       0       0       0       9       0
0       8       0       0       0       0
4       0       0       2       0       0
0       0       0       0       0       5
0       0       2       0       0       0
0       0       0       0       0       6

Sparse Matrix
6       6       7
0       4       9
1       1       8
2       0       4
2       3       2
3       5       5
4       2       2
5       5       6

Transpose of the Sparse Matrix
6       6       7
0       2       4
1       1       8
2       4       2
3       2       2
4       0       9
5       3       5
5       5       6

[Program finished]
```

Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 6

ADDITION OF SPARSE MATRICES

Aim : - Write program to read a matrix in normal form, converts them into tuple form and display it. And also find the sum of the matrices represented in tuple form and display it.

Algorithm AlgoMatrixAdd(Matrix, row,col)
Input Specification : Input arrays, rows, columns, matrix values
Output Specification: Input matrices and its sum
Data Structures Used : Arrays

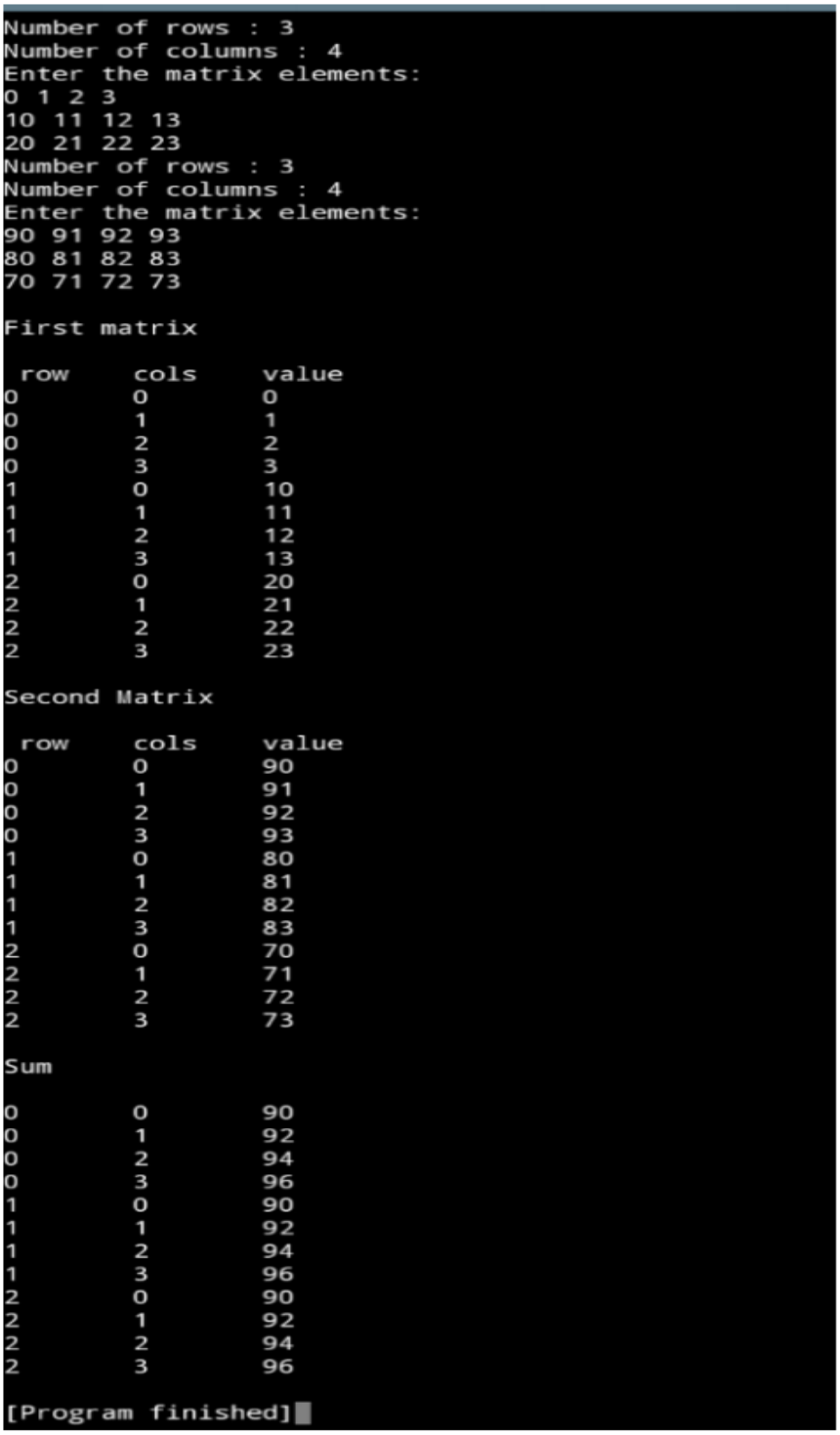
1. Start.
2. Read two matrices
3. If the matrices are compatible
4. For i=0 to row-1
5. For j=0 to column-1
6. Find sum
7. If sum>0 Print it in tuple form
8. End for
9. End for
10. End if
11. Else print 'matrices are not compatible'
12. Stop

Program code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 30

void print_tuple (int m[MAX][MAX],int r, int c)    //To print input matrix
{
    printf("\n row\tcols\tvalue\n");
    for(int i=0; i<r; i++)
    {
        for(int j=0; j<c; j++)
        printf("%d\t%d\t%d\n", i, j, m[i][j]);
    }
}

void add(int m1[MAX][MAX],int m2[MAX][MAX],int r,int c)
{
    printf("\nSum\n\n");
    for(int i=0; i<r;i++)        //printing sum
    {
        for(int j=0; j<c; j++)
            if (m1[i][j]+m2[i][j] >0)
```

```
                        printf("%d\t%d\t%d\n", i, j, (m1[i][j] + m2[i][j]));
    }
}

void main()
{
    int r1, c1,r2, c2, i, j, n1=0, n2=0;
    int a[MAX][MAX], b[MAX][MAX];

    printf("Number of rows : ");     //Read First Matrix
      scanf("%d", &r1);
    printf("Number of columns : ");
      scanf("%d", &c1);
    printf("Enter the matrix elements:\n");
    for(i=0; i<r1;i++)
    {
        for(j=0; j<c1; j++)
              scanf("%d", &a[i][j]);
    }

    printf("Number of rows : ");    //Read Second Matrix
      scanf("%d", &r2);
    printf("Number of columns : ");
      scanf("%d", &c2);
    printf("Enter the matrix elements:\n");
    for(i=0; i<r2; i++)
    {
        for(j=0; j<c2; j++)
              scanf("%d", &b[i][j]);
    }

    if(r1 == r2 && c1 == c2)
    {
        printf("\nFirst matrix\n");
        print_tuple(a,r1, c1);
        printf ("\nSecond Matrix\n");
        print_tuple(b, r2, c2);
        add(a, b,r1,c1);
      }
    else
    {
        printf(" Can not add!!\n");
        Exit();
    }
}
```

Sample output:



```
Number of rows : 3
Number of columns : 4
Enter the matrix elements:
0 1 2 3
10 11 12 13
20 21 22 23
Number of rows : 3
Number of columns : 4
Enter the matrix elements:
90 91 92 93
80 81 82 83
70 71 72 73

First matrix

 row    cols    value
0       0       0
0       1       1
0       2       2
0       3       3
1       0       10
1       1       11
1       2       12
1       3       13
2       0       20
2       1       21
2       2       22
2       3       23

Second Matrix

 row    cols    value
0       0       90
0       1       91
0       2       92
0       3       93
1       0       80
1       1       81
1       2       82
1       3       83
2       0       70
2       1       71
2       2       72
2       3       73

Sum

0       0       90
0       1       92
0       2       94
0       3       96
1       0       90
1       1       92
1       2       94
1       3       96
2       0       90
2       1       92
2       2       94
2       3       96

[Program finished]
```

Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 7

ARRAY INSERTION

Aim : - Write a C program to insert an element into an array. You have to input the index position.

Algorithm ArrayInsertion(a, n, pos, value)
Input Specification : Input array, index, value to insert
Output Specification: Array after insertion
Data Structures Used : Arrays

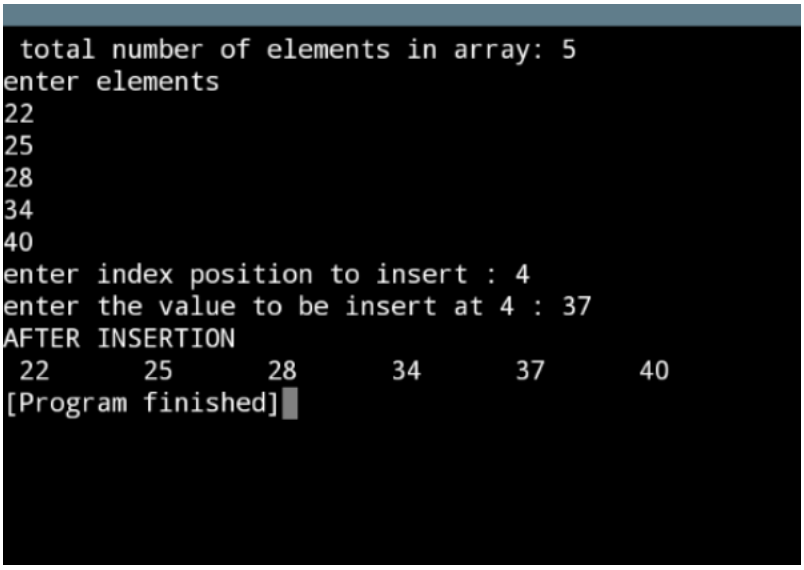
1. Start
2. Read an array a of size n
3. Read index position pos where you have to insert.
4. If the index is valid then,
5. for i=n to pos , i--
6. a[i]= a[i-1]
7. End for
8. a[pos]= value
9. Display the new array
10. Stop

Program Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 30
void insertion (int a[MAX], int n)
{
        int i, pos, value;
        printf("enter index position to insert : ");
        scanf("%d", &pos);
        if(pos>n || pos<0)
        {
            printf(" INVALID INDEX!! \n");
            exit(0);
        }
        printf("enter the value to be insert at %d : ", pos);
        scanf("%d", &value);
        for( i=n; i>=pos; i--)
        {
             a[i]= a[i-1];
        }
        a[pos]= value;
        printf("AFTER INSERTION\n");
        for (i=0; i<=n; i++)
        {
              printf(" %d\t", a[i]);
        }
}
```

```
int main()
{
        int n, i, a[MAX];
        printf(" total number of elements in array: ");
          scanf("%d", &n);
        printf("enter elements \n");
        for (i=0; i<n; i++)
        {
                scanf("%d", &a[i]);
        }
        insertion(a,n);
}
```

Sample output:



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 8

ARRAY DELETION

Aim : - Write a C program to delete an element from an array. You have to input the index position where the element to be deleted.

Algorithm ArrayDeletion(a, n, pos)
Input Specification : Input array, index
Output Specification: Array before and after deletion
Data Structures Used : Array

1. Start
2. Read an array a of size n
3. Read index pos , where you have to delete.
4. If the index is not exist then exit
5. Else
6. For i=pos to count-1
7. a[i]=a[i+1]
8. End for
9. Display array after and before deletion
10. Stop

Program code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 30

void deletion (int a[MAX], int n)
{
    int i, pos;
    printf("enter index position to delete : ");
    scanf("%d", &pos);
    if(pos>(n-1) || pos<0)
    {
        printf(" INVALID INDEX!! \n");
        exit(0);
    }
    printf("BEFORE DELETION\n");
    for (i=0; i<n; i++)
    {
        printf(" %d\t", a[i]);
    }
    for( i=pos; i<(n-1); i--)
    {
     a[i]= a[i+1];
    }
    printf("\nAFTER DELETION\n");
```

```
    for (i=0; i<(n-1); i++)
    {
        printf(" %d ", a[i]);
    }
}

int main()
{
    int n, i, a[MAX];
    printf(" total number of elements in array: ");
      scanf("%d", &n);
    printf("enter elements \n");
    for (i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
    deletion(a,n);
}
```

Sample output:

```
 total number of elements in array:
 5
enter elements
4
8
75
12

7
enter index position to delete : 4
BEFORE DELETION
 4        8        75       12       7
AFTER DELETION
 4  8  75  12
```

Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 9

STACK USING ARRAYS

Aim : - Write a C program to implement stack data structure using arrays.

Algorithm Stack(top, n)
Input Specification : Input array, push, pop, traverse, top
Output Specification: Stack after push and pop.
Data Structures Used : Array

1. Start
2. Read the size of stack, n
3. Read the choice of user, op
4. If it's push Then
5. Check whether the stack is not full, then
6. Push value into top of the stack.
7. Otherwise print 'Overflow'
8. If it's pop
9. Check whether the stack is not empty, then
10. Pop top element from the stack
11. Otherwise print 'Underflow'
12. If it's Print
13. Check whether the stack is not empty, then
14. Display the values
15. Otherwise print 'stack is empty'
16. Else exit
17. Stop

Program code:

```
#include <stdio.h>
#include <stdlib.h>
int top=-1, n;

void push(int stk[n])     //push operation
{
      int x;
      if (top==n)
      printf(" Overflow \n ");
      else
      {
            printf ("eneter element to be push: ");
            scanf("%d", &x);
            stk[++top]= x;
      }
}
```

```
void pop(int stk[n])       //pop operation
{
      int temp;
      if(top==-1)
            printf("Empty stack!!\n");
      else
      {
            temp=stk[top--];
            printf("Popped top element : %d  ", temp);
      }
}

void print(int stk[n])    //printing stack elements
{
      if(top==-1)
            printf("Your stack is empty.!! You have to push values \n");
      else
      {
            printf("  Stack elements \n");
            for(int i=0; i<=top; i++)
                  printf("  %d ", stk[i]);
      }
}

int main()
{
      int op=3;
      printf ("enter the size of stack : ");
        scanf ("%d", &n);
      int stk[n];
      n--;
      while (op!= 4)
      {
          switch(op)
          {
                  case 1: push(stk);
                  break;
                  case 2: pop(stk);
                  break;
                  case 3: print(stk);
                  break;
                  case 4: exit(0);
                  default: printf ("Invalid choice\n");
          }
          printf ("\n 1:Push  2:Pop  3:Display  4:Exit \n Enter Your Choice: ");
          scanf("%d", &op);
      }
}
```

Sample output:

```
enter the size stack : 4
Your stack is empty.!!
 You have to push values

  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 1
enetr element to push: 67

  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 1
enetr element to push: 32

  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 1
enetr element to push: 26

  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 3
  Stack elements
  67   32   26
  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 1
enetr element to push: 15

  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 1
  Overflow

  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 3
  Stack elements
  67   32   26   15
  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 2
  Popped top element : 15
  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 3
  Stack elements
  67   32   26
  1:Push  2:Pop  3:Display  4:Exit
 Enter Your Choice: 4

[Program finished]
```

Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 10

Infix to postfix conversion and Evaluation

Aim : - Write a C program to convert an infix expression to postfix equivalent and evaluate it.

Algorithm InfixToPostfix(top, n)
Input Specification : Infix, MAX_SIZE,top
Output Specification: Infix expression, postfix expression and postfix evaluation.
Data Structures Used : Array

1. Start
2. Read infix expression
3. Push ')' to the stack
4. While (top>-1)
    5. Scan each *item* in the expression
    6. x=Pop()
    7. If the scanned item is operand then
        - push(x)
        - output(item)
    8. Else If the item is ')' then
        While (x!= '(')
        - output (x)
        - x= pop()

        End while
    9. Else if (ISP(x)>=ICP(item)) then
        - output(x)
        - x=pop() , until the same condition occurs
    10. else if (ISP(x)<ICP(item))
        - Push(x)
        - push(item)
    11. Else print INVALID EXPRESSION Then break
12. End while
13. For postfix evaluation
14. put '#' to end of postfix expression
15. item=ReadSymbol(post)
16. While (item!='#')
    17. If item is operand
        - push(item)
    18. Else
        - y=pop()
        - x=pop()
        - z=x op y
        - push(z)
    19. End if
    20. item= ReadSymbol(post)
21. Return z
22. Stop

Program Code

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX 20
char in[MAX],stk[MAX],post[MAX];
int top=-1,i=0,j=0;

int push(char x){     //PUSH OPERATION
  if(top>=MAX)
            printf("Maximum terms reached\n");
  else
            stk[++top]=x;
}
int pop(){        //POP OPERATION
  if(top==-1)
            printf("Underflow");
  else
            return stk[top--];
}
void output(int item){    //POSTFIX EXPRESSION
            post[j++]=item;
}
int operand(char op) {      //OPERAND or Not
          if(op=='('|| op==')'|| op=='*'|| op=='/'|| op=='+'|| op=='-'|| op=='^')
                    return 0;
          else
                    return 1;
}
int ISP(char op)         //ISP Value
{
            if(op=='+' || op=='-')
                      return 2;
            else if(op=='*' || op=='/')
                      return 4;
           else if(op=='^')
                      return 5;
            else if(op=='(')
                      return 0;
             else
                     return 8;
}
int ICP(char op)     //ICP Value
{
            if(op=='+' || op=='-')
                      return 1;
```

```
        if(op=='*' || op=='/')
                return 3;
        if(op=='^')
                return 6;
        if(op=='(')
                return 9;
        if(op==')')
                return 0;
        else
                return 7;
}
char ReadSymbol(char A[MAX])   //reading symbol
{
        if(A[i]=='\0')
                return ')';
        else
                return A[i++];
}

//POSTFIX CONVERSION
void ToPostfix(char a[MAX])
{
  char item,x;
  push('(');
  while(top>=0)
  {
        item=ReadSymbol(a);
        x=pop();
         if(operand(item)==1)
         {
                push(x);
                output(item);
        }
        else if(item==')')
        {
                while(x!='(')
                {
                        output(x);
                        x=pop();
                }
        }
        else if(ISP(x)>=ICP(item))
        {
                while(ISP(x)>=ICP(item))
                {
                        output(x);
                        x=pop();
```

```
                }
                push(x);
                push(item);
        }
        else if(ISP(x)<ICP(item))
        {
                push(x);
                push(item);
        }
        else
        {
                printf("INVALID EXPRESSION");
                break;
        }
   } //End while loop
}
//POSTFIX EVALUATION
int postEval(char post[MAX])
{
  char item,op;
  i=0;
  int x,y,t=0,k,res=0;
       post[j]='#';
  item =ReadSymbol(post);
     while(item!='#')
     {
                if(operand(item)==1)
                        push(item);
                else
                {
                        op=item;
                        y=pop()-48;     //ASCII to integer
                        x=pop()-48;
                        if(op=='+')
                                t=x+y;
                        else if(op=='-')
                                t=x-y;
                        else if(op=='*')
                                t=x*y;
                        else if(op=='/')
                                t=x/y;
                        else if(op=='^'){
                                t=1;
                                for(k=1;k<=y;k++)
                                        t=t*x;
                        }
                        else
                                printf("Invalid expression!!\n");
                        push(t+48);
```

```
                }
                item =ReadSymbol(post);
    }
  res=pop()-48;
  return res;
}
int main()
{
        printf("\n Enter Infix expression : ");
        scanf("%s",in);
        ToPostfix(in);
        printf("\n Postfix Expression : %s \n",post);
        int v=postEval(post);
        printf("\n Postfix evaluation : %d \n",v);
}
```

Sample Output:

Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 11

Infix to prefix conversion and Evaluation

Aim : - Write a C program to convert an infix expression to prefix equivalent and evaluate it.

Algorithm InfixToPrefix(top, n)
Input Specification : Infix, MAX_SIZE, top
Output Specification: Infix expression, prefix expression and prefix evaluation.
Data Structures Used : Arrays

1. Start
2. Read infix expression
3. Reverse infix expression
4. Push '(' to the stack
5. While (top>-1)
    6. Scan each *item* in the expression
    7. x=Pop()
    8. If (item is Operand)
        - push(x)
        - output(item)
    9. Else If (item is '**)**')
        - While (x!= '(')
        - output (x)
        - x= pop()
        - End while
    10. else if(item== '**(**')
        - push(x);
        - push(item);
    11. Else if (prcd(x)>=prcd(item))
        - output(x)
        - x=pop() , until the same condition occurs
    12. else if prcd(x)<prcd(item)
        - push(x)
        - push(item)
    13. Else print INVALID EXPRESSION Then break
14. End while
15. Reverse(pre)
16. For evaluation put '#' to end of pre[]
17. item=ReadSymbol(pre)
18. While (item!='#')
    19. If item is operand
        - push(item)
    20. Else
        - op=item
        - y=pop()
        - x=pop()
        - z=x op y
        - push(z)

21. End if
22. item= ReadSymbol(pre)

23. End while
24. Return (Stack[top])
25. Stop

Program code:

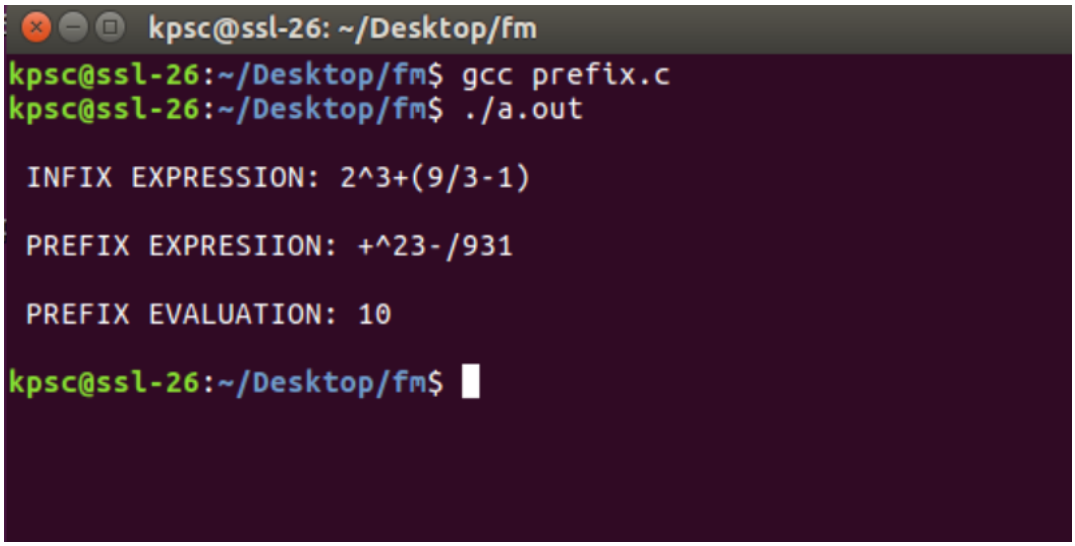
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 20
char in[MAX],stk[MAX],pre[MAX],r[MAX];
int top,i=0,j=0,len;

int push(char x)
{
	if(top>=MAX)
		printf(" Stack is full\n");
	else
		stk[++top]=x;
}
int pop()
{
	if(top==-1)
		printf(" Underflow\n");
	else
		return stk[top--];
}
void output(char item)
{
	pre[j++]=item;
}
int operand(char op)  //Operand or not
{
	if(op=='('|| op==')'|| op=='*'|| op=='/'|| op=='+'|| op=='-'|| op=='^')
		return 0;
	else
		return 1;
}
int prcd(char op)  //precedence
{
	if(op=='+' || op=='-')
		return 2;
	else if(op=='*' || op=='/')
		return 4;
	else if(op=='^')
		return 5;
	else
```

```
                return 0;
}
char ReadSymbol(char A[MAX])   //reading symbol
{
        if(A[i]=='\0')
                return ')';
        else
                return A[i++];
}
void reverse(char a[MAX])//REVERSE
{
        int z=0,k;
        len= strlen(a);
        char temp[MAX];
        for (k=len-1; k>=0;k--)
        {
                if(a[k]=='(')
                        a[k]=')';
                else if(a[k]==')')
                        a[k]='(';
                temp[z++]=a[k];
        }
        temp[z]='\0';
        strcpy(r,temp);
}
//INFIX TO PREFIX CONVERSION
void ToPrefix()
{
        char item,x;
        top=-1;
        reverse(in);
        push('(');
        while(top>=0)
        {
                item=ReadSymbol(r);
                x=pop();
                if(operand(item)==1){
                        push(x);
                        output(item);
                }
                else if(item==')') {
                        while (x!='(')
                        {
                                output(x);
                                x=pop();
                        }
                  }
                else if(item=='(')
                  {
```

```
                        push(x);
                        push(item);
                  }
                else if(prcd(x)>prcd(item)) {
                        while(prcd(x)>=prcd(item))
                        {
                                output(x);
                                x=pop();
                        }
                        push(x);
                        push(item);
                }
                else if(prcd(x)<=prcd(item)) {
                        push(x);
                        push(item);
                }
                else {
                        printf("INVALID EXPRESSION");
                        break;
                 }
        }
        reverse(pre);
}
//PREFIX EVALUATION
int preEval()
{
        char item,op;
        i=0;
        int x,y,t=0,k,res=0;
        pre[j]='#';
        item =ReadSymbol(pre);
        while(item!='#') {
                if(operand(item)==1)
                        push((item-'0'));  //Push Integer value (ASCII of '0' = 48)
                else {
                                op=item;
                                y=pop();
                                x=pop();
                                if(op=='+')
                                        t=x+y;
                                else if(op=='-')
                                        t=y-x;
                                else if(op=='*')
                                        t=x*y;
                                else if(op=='/')
                                        t=y/x;
                                else if(op=='^') {
                                        t=1;
                                        for(k=1;k<=x;k++)
```

```
                                        t=t*y;
                                }
                                else
                                        printf("Invalid expression!!\n");
                                push(t);
                        }
                        item =ReadSymbol(pre);
                }   //End while loop
                res=pop();
                return res;
        }
        int main()
        {
                printf("\n INFIX EXPRESSION: ");
                scanf("%s",in);
                ToPrefix();
                printf("\n PREFIX EXPRESIION: %s \n",r);
                int v=preEval();
                printf("\n PREFIX EVALUATION: %d \n\n",v);
        }
```

Sample output:



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 12

Queue using arrays

Aim : - Write a C program to implement queue using array with the operation insertion, Deletion and display the elements.

Algorithm Queue(q, front, rear)
Input Specification : q[MAX_SIZE], Front, rear
Output Specification: q[] after insertion, deletion.
Data Structures Used : Arrays

1. Start
2. Create q[MAX_SIZE]
3. front=rear=-1
4. Read the choice
5. If it's Insertion, then
   - check the q[] is full ,
   - Print q is full
   - Else if front=rear=-1 ,
   - increment both, q[front]= value
   - Else
   - q[++rear]=value
   - End if
6. if It's Deletion Then
   - If the q is empty,
   - print Empty queue
   - Else
   - temp=q[rear--]
   - End if
7. If it's Display then
   - If the q is empty,
   - Print Empty queue
   - Else, for(i=front to rear)
   - Print q[i]
   - End if
8. Else exit()
9. Stop

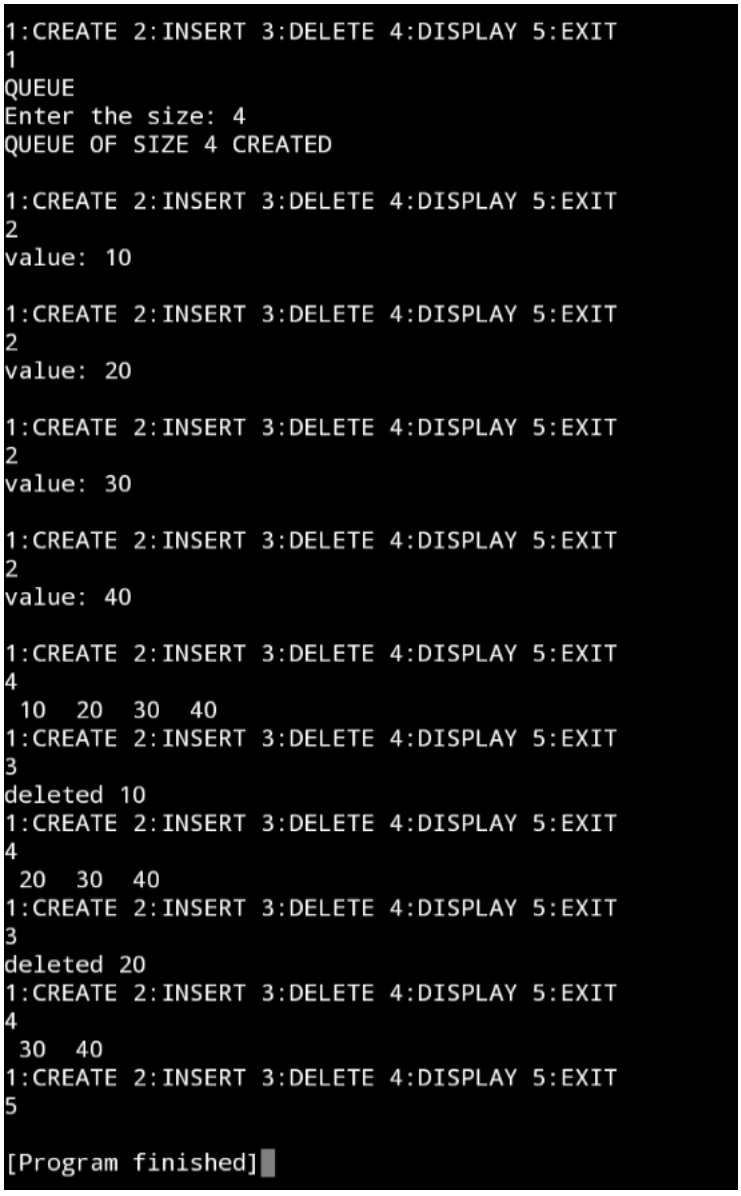
Program code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 30
int q[MAX],n,front=-1,rear=-1;

void create()     //CREATION
{
    printf("QUEUE\nEnter the size: ");
    scanf("%d",&n);
    printf("QUEUE OF SIZE %d CREATED\n",n);
}
```

```
void insert()      //INSERTION
{
    if(rear>=MAX)
            printf("QUEUE FULL!!\n");
    else{
            int v;
            printf("value: ");
            scanf("%d",&v);
            if(rear==-1 && front==-1)
             {
                    front=rear=0;
                    q[front]=v;
             }
            else
                    q[++rear]=v;
    }
}
void delete()      //DELETION
{
    if(rear==-1 && front==-1)
            printf("Queue is empty!!\n");
    else{
            int x=q[front++];
            printf("deleted %d ",x);
    }
}
void display()     //PRINTING ELEMENTS
{
    if(rear==-1 && front==-1)
            printf("Your Queue is empty!\nInsert values!");
    else
    {
            for(int i=front;i<=rear;i++)
                    printf(" %d ",q[i]);
    }
}
int main()
{
    int op;
    printf("\n1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT\n");
            scanf("%d",&op);
    while(op<=5)
    {
            switch (op)
            {
                    case 1: create();
                            break;
                    case 2: insert();
                            break;
                    case 3: delete();
```

```
                    break;
                case 4: display();
                    break;
                case 5: exit(0);
                    break;
                default: printf("INVALID CHOICE\n");
                    break;
            }
            printf("\n1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT\n");
            scanf("%d",&op);
        }
    }
```

Sample Output:



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 13

Circular Queue using Arrays

Aim : - Write a C program to implement Circular queue using array with the operation insertion, Deletion and display the elements.

Algorithm CircularQueue(q, front, rear)
Input Specification : q[MAX_SIZE], Front, rear
Output Specification: q[] after insertion, deletion.
Data Structures Used : Arrays

1.Start
2.Create q[MAX_SIZE]
3.front=rear=-1
4.Case Insertion:
5. If front==(rear+1)%size, Then
6. Print q is full
7. Else if front=rear=-1 ,
8. front =0
9. End if
10. rear=(rear+1)%size
11. q[rear]=value
12. Case Deletion:
13. If front= -1,
14. print Empty queue
15. Else If (front= rear)
16. front=0
17. Else
18. front=(front+1)%size
19. End if
20. Case Display :
21. If front== -1,
22. Print Empty queue
23. Else,
24. i=front
25. While(i<= rear)
26. Print q[i]
27. i=(i+1)%size
28. End while
29. End if
30. Default: exit()
30. Stop

Program code:

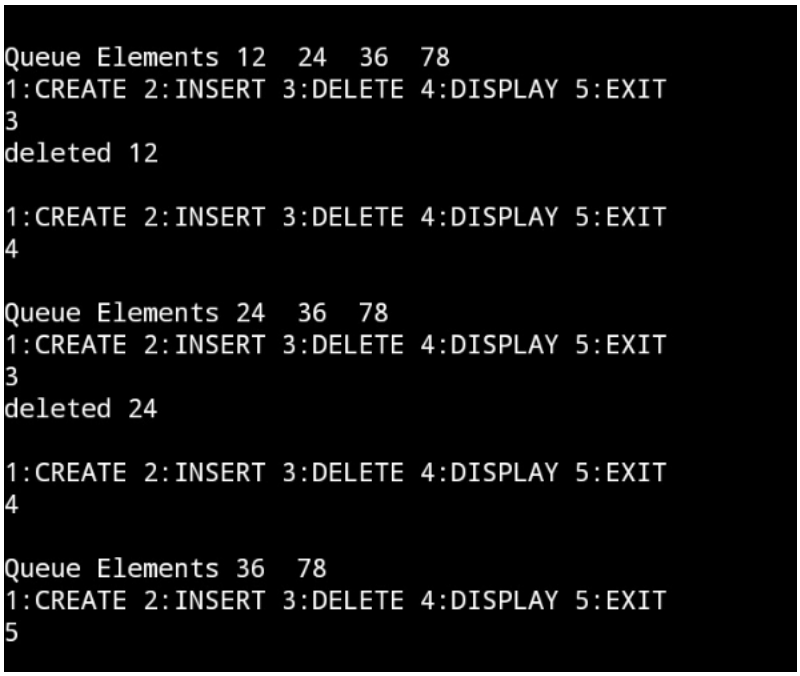
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 30

int cq[MAX],size,front=-1,rear=-1;
```

```
void create() //CREATION
{
	printf("CIRCULAR QUEUE\nEnter the size: ");
	scanf("%d",&size);
	printf("CIRCULAR QUEUE OF SIZE %d CREATED\n",size);
}
void insert()  //INSERTION
{
	if(front==(rear+1)%size)
		printf("FULL!!\n");
	else
	{
		int v;
		printf("value: ");
		scanf("%d",&v);

		if(front==-1)
			front=0;
		rear=(rear+1)%size;
		cq[rear]=v;
	}
}
void delete()  //DELETION
{
	if(front==-1)
		printf("EMPTY!!\n");
	else {
		int x= cq[front];
		printf("deleted %d \n",x);
		if(front==rear)
			front=rear=-1;
		else
			front=(front+1)%size;
	}
}
void display()  //PRINTING ELEMENTS
{
	if(front==-1)
		printf("Your Queue is empty!\nInsert values!");
	else
	{
		printf("\nQueue Elements");
		int i=front;
		while(i<rear){
			printf(" %d ",cq[i]);
			i=i+1%size;
		}
		printf(" %d ",cq[i]);
	}
}
```

```
int main()
{
	int op;
	printf("\n1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT\n");
	scanf("%d",&op);
	while(op<=5)
	{
		switch(op)
		{
			case 1: create();
				break;
			case 2: insert();
				break;
			case 3: delete();
				break;
			case 4: display();
				break;
			case 5: exit(0);
				break;
			default: printf("INVALID CHOICE\n");
				break;
		}
		printf("\n1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT\n");
		scanf("%d",&op);
	}
}
```

Sample output:

```
1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT
1
CIRCULAR QUEUE
Enter the size: 4
CIRCULAR QUEUE OF SIZE 4 CREATED

1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT
2
value: 12

1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT
2
value: 24

1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT
2
value: 36

1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT
2
value: 78

1:CREATE 2:INSERT 3:DELETE 4:DISPLAY 5:EXIT
4
```



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 14

Priority Queue using Array

Aim : - Write a C program to implement priority queue using array with the operation insertion, Deletion and display the elements.

Algorithm PriorityQueue(q, front, rear,priority)
Input Specification : q[MAX_SIZE], Front, rear,priority,value
Output Specification: q[] after insertion, deletion.
Data Structures Used : Arrays, structure

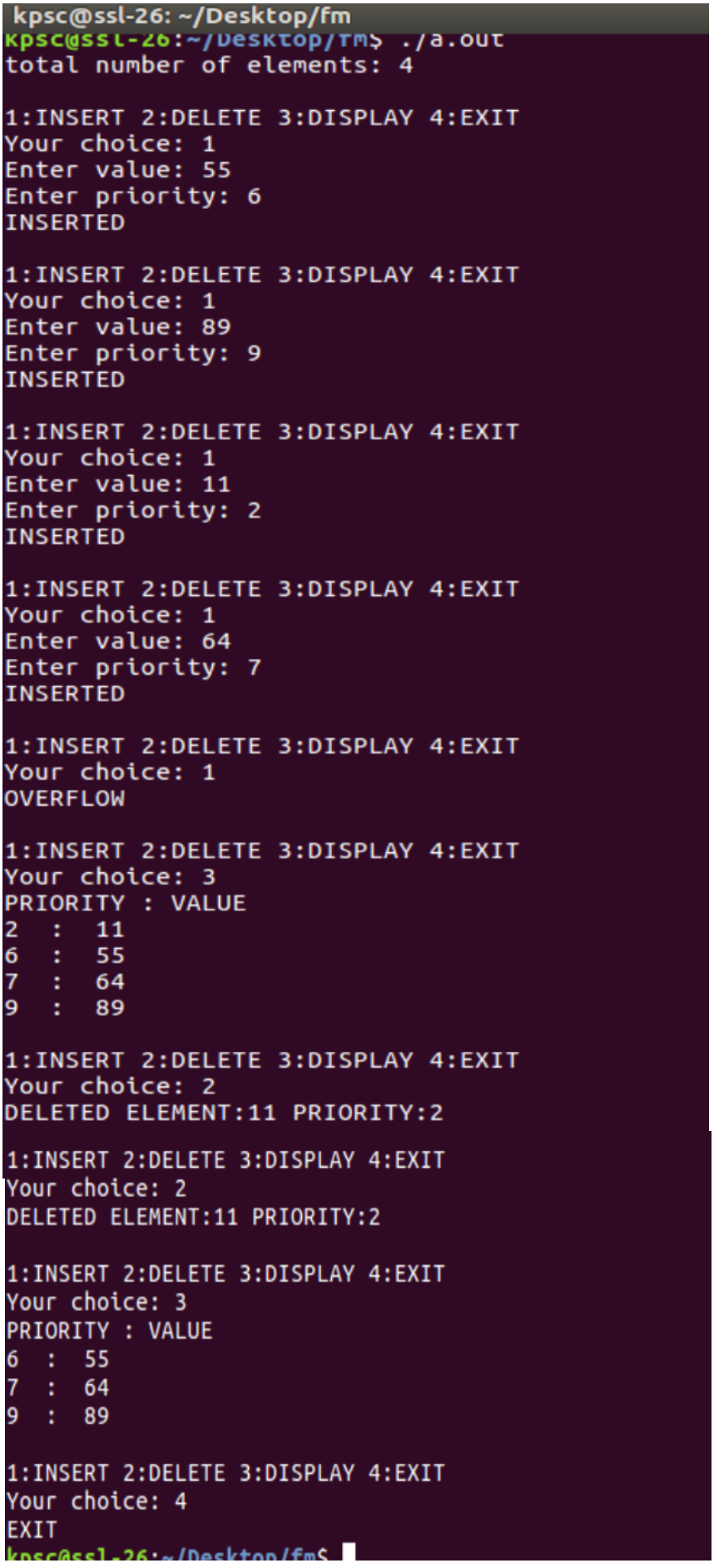
1. Start
2. Create pq[MAX_SIZE]
3. front=rear=-1
4. Case insertion :
5. If rear=size-1
6. Print overflow
7. Else
8. Read value and priority
9. If front=-1
10. Front=0
11. i=rear++
12. While pq[i].prio > newprio And i>=0
13. pq[i+1] = pq[i]
14. i--
15. End while
16. pq[i+1].el = value,
17. pq[i+1] prio= newprio
18. End if
19. Case deletion :
20. If front=-1
21. Print empty queue
22. Else
23. If front==rear
24. Front =rear=-1
25. Else front++
26. End if
27. Case display :
28. If front ==-1
29. Print empty queue
30. Else
31. For i=front to rear
32. Print pq[i]
33. End for
34. End if
35. Stop

Program code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20

int front=-1,rear=-1;
struct priority
{
        int el,pr;
}pq[MAX];

void insert(int size)
{
 if(rear==size-1)
        printf("OVERFLOW\n");
 else{
        int v1,p1,i;
        printf("Enter value: ");
        scanf("%d",&v1);
        printf("Enter priority: ");
        scanf(" %d",&p1);
        if(front==-1)
                front=0;
        i=rear++;
        while(pq[i].pr>=p1 &&i>=0)
         {
                pq[i+1]=pq[i];
                i--;
        }
        pq[i+1].el=v1;
        pq[i+1].pr=p1;
        printf("INSERTED\n");
}
}
void delete()
{
 if(front==-1)
        printf("EMPTY QUEUE\n");
 else
{
        int x,y;
           x=pq[front].el;
           y=pq[front].pr;
        if(front==rear)
                front=rear=-1;
```

```
        else
                front++;
        printf("DELETED ELEMENT:%d PRIORITY:%d \n",x,y);
}
}
void display()
{
if(front==-1)
        printf("EMPTY QUEUE\n");
else
{
        printf("PRIORITY : VALUE\n");
        for(int i=front;i<=rear;i++)
                printf("%d  :  %d\n",pq[i].pr,pq[i].el);
}
}
int main()
{
int n,op;
printf("total number of elements: ");
scanf("%d",&n);
printf("\n1:INSERT 2:DELETE 3:DISPLAY 4:EXIT\nYour choice: ");
scanf("%d",&op);
while(op)
{
        switch(op)
        {
                case 1:insert(n);
                        break;
                case 2:delete();
                        break;
                case 3:display();
                        break;
                case 4:printf("EXIT\n");
                        exit(0);
                        break;
                default:printf("INVALID CHOICE!\n");
        }
        printf("\n1:INSERT 2:DELETE 3:DISPLAY 4:EXIT\nYour choice: ");
        scanf("%d",&op);
}
}
```

Sample output:



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 15

Deque using Array

Aim : - Write a C program to implement Double ended queue using array with the operation insertion, Deletion and display the elements.

Algorithm DeQueue(q, front, rear,)
Input Specification : q[MAX_SIZE], Front, rear,value
Output Specification: q[] after push,pop, Inject and eject
Data Structures Used : Array

1. Start
2. Create Deque[max]
3. Case Push_dq :
4. if front= (rear+1)%size
5. print QUEUE IS FULL
6. Else
7. Read value
8. if rear==-1 and front==-1
9. front = rear 0
10. dq[front] =value
11. Else
12. If front =0
13. front=size
14. End if
15. front=front-1%size
16. Dq[front] = value
17. End if
18. Case pop_dq :
19. if(front==-1)
20. printf("EMPTY!!\n");
21. else
22. if(front==rear)
23. front=rear=-1;
24. else
25. front=(front+1)%size
26. End if
27. End if
28. Case inject:
29. if front==(rear+1)%size
30. print FULL
31. else
32. Read value
33. if(front==-1)
34. front=0;
35. End if

36. rear=(rear+1)%size;
37. dq[rear]=v;
38. End if
39. Case eject :
40. if(front==-1)
41. print "EMPTY"
42. else
43. if(front==rear)
44. front=rear=-1;
45. else
46. rear=(rear-1)%size;
47. End if
48. End if
49. Stop

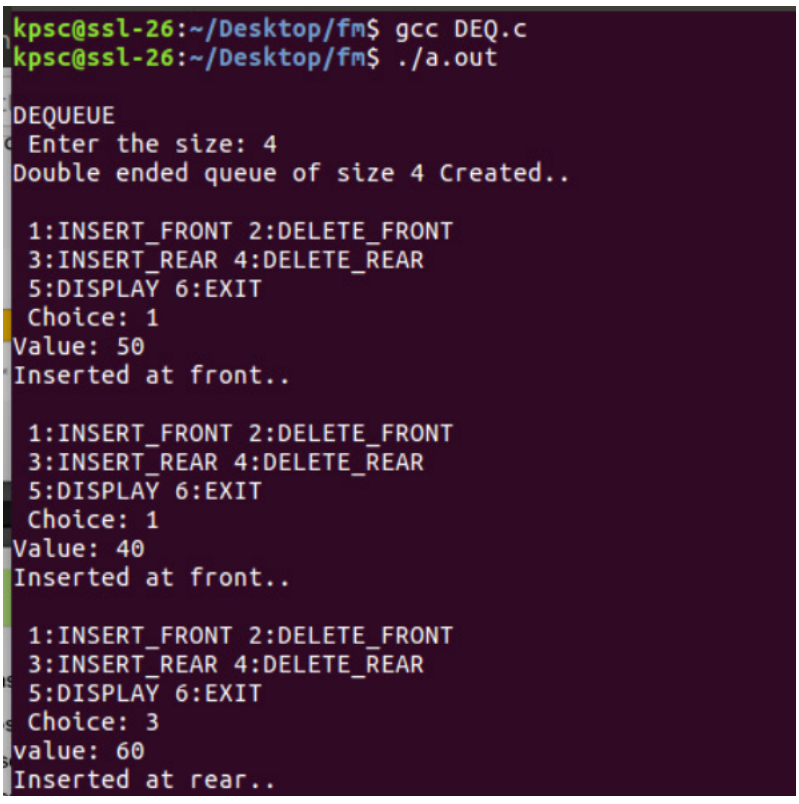
Program code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 30

int dq[MAX],size,front=-1,rear=-1;
void push_DQ()  //INSERT_AT_FRONT
{
        if(front==(rear+1)%size)
                printf("FULL!!\n");
        else{
                int v;
                printf("Value: ");
                scanf("%d",&v);
                if(rear==-1&&front==-1){
                        front=rear=0;
                        dq[front]=v;
                }
                else{
                        if(front==0)
                                front=size;
                        front=(front-1)%size;
                        dq[front]=v;
                }
                printf("Inserted at front..\n");
        }
}
void pop_DQ()  //DELET_FROM_FRONT
{
        if(front==-1)
                printf("EMPTY!!\n");
        else{
                int x= dq[front];
                printf("deleted %d \n",x);
```

```
                if(front==rear)
                        front=rear=-1;
                else
                        front=(front+1)%size;
                printf("Deleted from front..\n");
        }
}
void inject_DQ()  //INSERT_AT_REAR
{
        if(front==(rear+1)%size)
                printf("FULL!!\n");
        else{
                int v;
                printf("value: ");
                scanf("%d",&v);
                if(front==-1)
                        front=0;
                rear=(rear+1)%size;
                dq[rear]=v;
                printf("Inserted at rear..\n");
        }

}
void eject_DQ() //REMOVE_FROM_REAR
{
        if(front==-1)
                printf("EMPTY!!\n");
        else{
                if(front==rear)
                        front=rear=-1;
                else
                        rear=(rear-1)%size;
                printf("Deleted from rear..\n");
        }
}
void display()
{
                if(front==-1)
                        printf("EMPTY!!\n");
                else{
                        int i;
                        for(i=front;i!=rear;i=(i+1)%size)
                                printf(" %d ",dq[i]);
                        printf(" %d \n",dq[i]);
                }
}
int main()
{
        int op;
        printf("\nDEQUEUE\n Enter the size: ");
```

```
        scanf("%d",&size);
        printf("Double ended queue of size %d Created..\n",size);
        printf("\n 1:INSERT_FRONT 2:DELETE_FRONT\n 3:INSERT_REAR 4:DELETE_REAR\n
5:DISPLAY 6:EXIT\n Choice: ");
        scanf("%d",&op);
        while(op)
        {
                switch(op)
                {
                        case 1: push_DQ();  //insert_front
                                break;
                        case 2: pop_DQ();   //delete_front
                                break;
                        case 3: inject_DQ(); //insert_rear
                                break;
                        case 4: eject_DQ(0);  //delet_rear
                                break;
                        case 5:display();
                                break;
                        case 6:printf("EXIT.. \n");
                                exit(0);
                        default: printf("INVALID CHOICE\n");
                                break;

                }
                printf("\n 1:INSERT_FRONT 2:DELETE_FRONT\n 3:INSERT_REAR
4:DELETE_REAR\n 5:DISPLAY 6:EXIT\n Choice: ");
                scanf("%d",&op);
        }
}
```

Sample Output:



```
 1:INSERT_FRONT 2:DELETE_FRONT
 3:INSERT_REAR 4:DELETE_REAR
 5:DISPLAY 6:EXIT
 Choice: 3
value: 70
Inserted at rear..

 1:INSERT_FRONT 2:DELETE_FRONT
 3:INSERT_REAR 4:DELETE_REAR
 5:DISPLAY 6:EXIT
 Choice: 1
FULL!!

 1:INSERT_FRONT 2:DELETE_FRONT
 3:INSERT_REAR 4:DELETE_REAR
 5:DISPLAY 6:EXIT
 Choice: 2
deleted 40
Deleted from front..

 1:INSERT_FRONT 2:DELETE_FRONT
 3:INSERT_REAR 4:DELETE_REAR
 5:DISPLAY 6:EXIT
 Choice: 5
 50  60  70

 1:INSERT_FRONT 2:DELETE_FRONT
 3:INSERT_REAR 4:DELETE_REAR
 5:DISPLAY 6:EXIT
 Choice: 4
Deleted from rear..

 1:INSERT_FRONT 2:DELETE_FRONT
 3:INSERT_REAR 4:DELETE_REAR
 5:DISPLAY 6:EXIT
 Choice: 5
 50  60

 1:INSERT_FRONT 2:DELETE_FRONT
 3:INSERT_REAR 4:DELETE_REAR
 5:DISPLAY 6:EXIT
 Choice: 6
EXIT..
kpsc@ssl-26:~/Desktop/fm$
```

Result :
The program was excuted on an array of numbers and the output was obtained.

Expt.No: 16

Single Linked List

Aim : - Write a menu driven program for performing the following operations on a Linked List: Display, Insert at Beginning, Insert at End, Insert at a specified Position, Delete from Beginning, Delete from End, Delete from a specified Position

Algorithm SingleLL(Node)
Input Specification : Head, Link, ptr, value
Output Specification: Linked list after insertion and deletion
Data Structures Used : structure

1. Start
2. Case insertion at front :
3. Read data, ptr->data
4. if (head == NULL)
5. head = ptr
6. else
7. ptr ->link = head
8. head = ptr
9. End if
10. Break
11. Case insertion at end :
12. Read data, ptr->data
13. if head == NULL
14. head = ptr
15. else
16. ptr1 = head
17. while (ptr1 ->link != NULL)
18. ptr1 = ptr1-> link
19. ptr1-> link = ptr
20. End if
21. ptr ->link = NULL
22. Break
23. Case insert at a position :
24. Read data, ptr->data
25. Read position, pos
26. if(pos==1)
27. ptr->link=ptr1;
28. head=ptr;
29. return;
30. End if
31. for i=1 to < pos-1
32. ptr1=ptr1->link
33. End for
34. ptr->link=ptr1->link
35. ptr1->link= ptr
36. Break
37. Case delete from front :

38. if(head==NULL)
39. print "EMPTY LIST"
40. else
41. ptr=head
42. head=ptr->link
43. free(ptr)
44. Break
45. Case delete from end :
46. ptr=head;
47. if(head==NULL)
48. print "EMPTY LIST"
49. else if(head->link==NULL)
50. head=NULL
51. free(ptr)
52. else
53. while(ptr->link!=NULL)
54. prev=ptr
55. ptr=ptr->link
56. End while
57. prev->link=NULL
58. free(ptr)
59. End if
60. Break
61. Case delete from a position :
62. Read position
63. ptr=head
64. for i=1 to i<pos
65. if(head==NULL)
66. print "List is empty"
67. return
68. End uf
69. prev=ptr
70. ptr=ptr->link
71. End for
72. prev->link=ptr->link
73. free(ptr)
74. Break
75. Stop

Program code:

```
#include <stdio.h>
#include <stdlib.h>
int pos;
struct node {
        int data;
        struct node *link;
}*head=NULL, *ptr, *ptr1;

int insert_front()
{
```

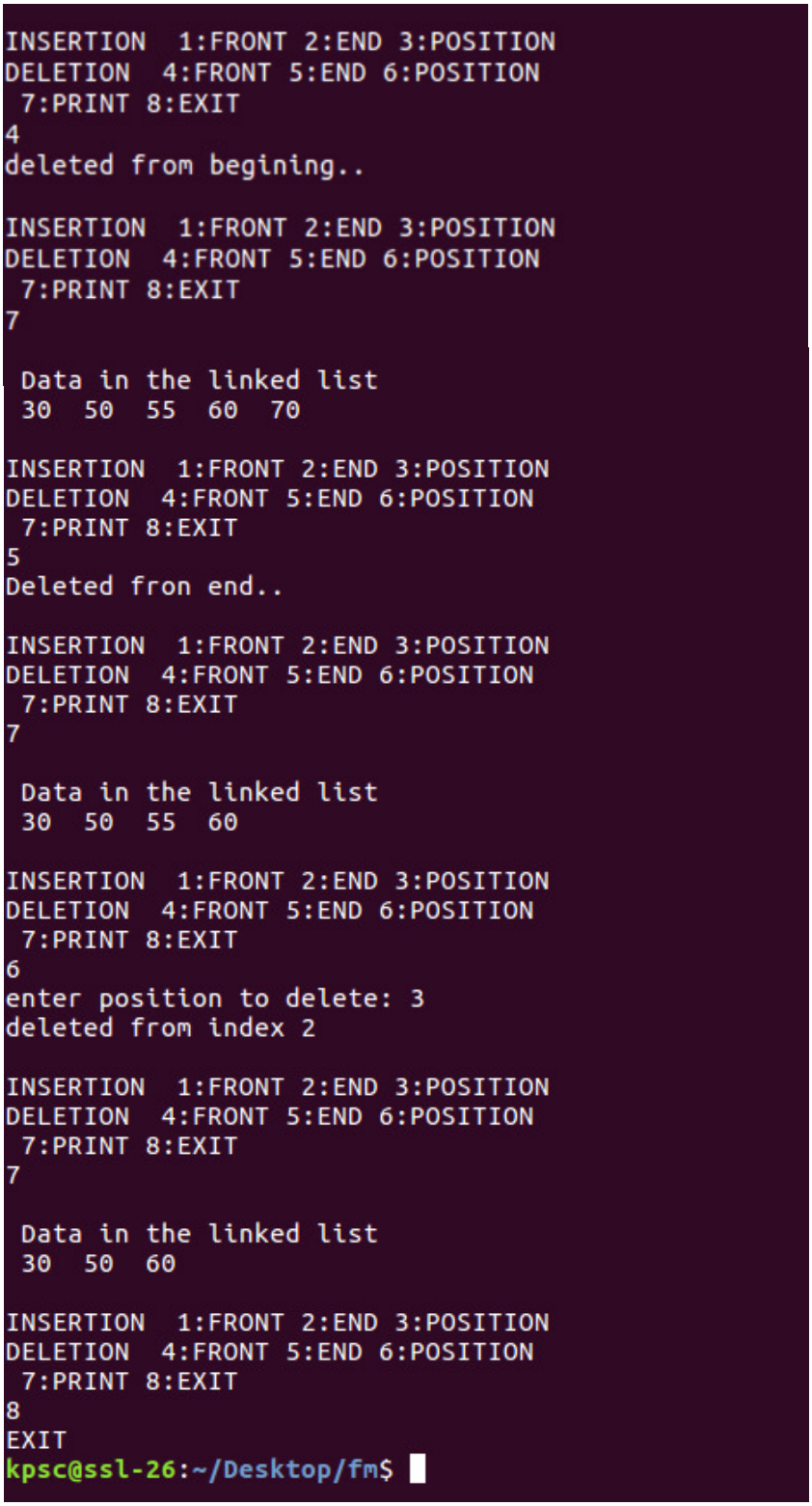
```
	printf("\n Insertion at front\n Enter your data : ");
	ptr =(struct node*) malloc(sizeof(struct node*));
	scanf("%d", &ptr->data);
	if (head == NULL)
		head = ptr;
	else{
		ptr ->link = head;
		head = ptr;
	}
}
int insert_end() //To Insert at End
{
	printf("\nInsertion at End\nEnter your data : ");
	ptr =(struct node*) malloc(2*sizeof(struct node*));
	scanf("%d", &ptr->data);
	if (head == NULL)
		head = ptr;
	else{
		ptr1 = head;
		while (ptr1 ->link != NULL)
			ptr1 = ptr1-> link;
		ptr1-> link = ptr;
	}
	ptr ->link = NULL;
}
void insert_pos(){
	printf("Enter position: ");
	scanf("%d",&pos);
	ptr =(struct node*) malloc(sizeof(struct node*));
	if(ptr==NULL)
		printf("OVERFLOW\n");
	else{
		printf("\nData : ");
		scanf("%d", &ptr->data);
		ptr1= head;
		if(pos==1){
			ptr->link=ptr1;
			head=ptr;
			return;
		}
		for(int i=1;i<pos-1;i++)
			ptr1=ptr1->link;

		ptr->link=ptr1->link;
		ptr1->link= ptr;
	}
}
void delete_front()
{
	if(head==NULL)
```

```
                printf("EMPTY LIST\n");
        else{
                ptr=head;
                head=ptr->link;
                free(ptr);
                printf("deleted from begining..\n");
        }
}
void delete_end()
{
        ptr=head;
        if(head==NULL)
                printf("EMPTY LIST\n");
        else if(head->link==NULL){
                head=NULL;
                free(ptr);
                printf("only one node.. deleted it..\n");
        }
        else{
                struct node* prev;
                while(ptr->link!=NULL){
                        prev=ptr;
                        ptr=ptr->link;
                }
                prev->link=NULL;
                free(ptr);
                printf("Deleted fron end..\n");
        }
}
void delete_pos()
{
        printf("enter position to delete: ");
        scanf("%d",&pos);
        struct node* prev;
        ptr=head;
        for(int i=1;i<pos;i++) {
                if(head==NULL){
                        printf("Can't delete!!List is empty\n");
                        return;
                }
                prev=ptr;
                ptr=ptr->link;
        }
        prev->link=ptr->link;
        free(ptr);
        printf("deleted from index %d\n",pos-1);
}
void print()      //To Print Linked List Data
{
        printf("\n Data in the linked list \n");
```

```
        ptr= head;
        while(ptr != NULL)
        {
                printf(" %d ", ptr->data);
                ptr= ptr->link;
        } printf("\n");
}
void main()
{
        int op = 1;
        printf("\nINSERTION  1:FRONT 2:END 3:POSITION\nDELETION  4:FRONT 5:END
6:POSITION\n 7:PRINT 8:EXIT\n");
        scanf("%d",&op);
        while (op)
        {
                ptr =(struct node*) malloc(20*sizeof(struct node*));
                   switch(op)
                   {
                        case 1:insert_front();
                                break;
                        case 2:insert_end();
                                break;
                        case 3:insert_pos();
                                break;
                        case 4:delete_front();
                                break;
                        case 5:delete_end();
                                break;
                        case 6:delete_pos();
                                break;
                        case 7:print();
                                break;
                        case 8:printf("EXIT\n");
                                exit(0);
                                break;
                        default:printf("Invalid Choice\n");
                     }
                printf("\nINSERTION  1:FRONT 2:END 3:POSITION\nDELETION  4:FRONT 5:END
6:POSITION\n 7:PRINT 8:EXIT\n");
                scanf("%d",&op);
        }
}
```

Sample output:

```
kpsc@ssl-26: ~/Desktop/fm
kpsc@ssl-26:~/Desktop/fm$ ./a.out

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
1

 Insertion at front
 Enter your data : 50

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
2

Insertion at End
Enter your data : 60

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
2

Insertion at End
Enter your data : 70

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
1

 Insertion at front
 Enter your data : 20

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
7

 Data in the linked list
 20  50  60  70

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
3
Enter position: 2

Data : 30

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
7

 Data in the linked list
 20  30  50  60  70

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
3
Enter position: 4

Data : 55

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
7

 Data in the linked list
 20  30  50  55  60  70
```



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 17

Doubly Linked List

Aim : - Write a menu driven program for performing the following operations on a Doubly Linked List: Display, Insert at Beginning, Insert at End, Insert at a specified Position, Delete from Beginning, Delete from End, Delete from a specified Position

Algorithm DoublyLL(ptr, RLink, LLink)
Input Specification : Head, RLink, LLink, ptr, value
Output Specification: Linked list after insertion and deletion
Data Structures Used : structure

1. Start
2. Create Node
3. Case Insertion at front :
4. Read value, ptr->data
5. if (head == NULL)
6. head = ptr
7. else
8. head->Llink=ptr
9. ptr ->Rlink = head
10. head = ptr
11. End if
12. Break
13. Case Insertion at end :
14. Read value, ptr->data
15. if (head == NULL)
16. head = ptr
17. else
18. head->Llink=ptr
19. ptr ->Rlink = head
20. head = ptr
21. End if
22. Break
23. Case Insertion at a position :
24. Read value, ptr->data
25. Read position, pos
26. ptr1= head
27. if(pos==1)
28. ptr->Rlink=ptr1
29. ptr->Llink=NULL
30. head=ptr
31. return
32. for i=1 to <pos-1
33. ptr1=ptr1->Rlink
34. End for
35. ptr->Rlink=ptr1->Rlink

```
    ptr->Llink=ptr1
   ptr1->Rlink= ptr
Break
Case deletion from front :
if(head==NULL)
        print " EMPTY LIST "
else
        ptr=head
        head=ptr->Rlink
        head->Llink=NULL
        free(ptr)
End if
Break
Case deletion from end :
 ptr=head
if(head==NULL)
        print EMPTY LIST
else if  head->Rlink==NULL
        head=NULL
        free(ptr)
else
        while (ptr->Rlink!=NULL)
                prev=ptr
                ptr=ptr->Rlink
        End while
        prev->Rlink=NULL
        free(ptr)
End if
Break
Case Deletion from a position :
Read position , pos
ptr=head
for  i=1  to <pos
        if(head==NULL)
                print List is empty
                return
            End if
        prev=ptr
        ptr=ptr->Rlink
End for
temp=ptr
ptr=ptr->Rlink
prev->Rlink=ptr
ptr->Llink=prev
free(temp)
Break
```

82. Case Display:
83. ptr= head
84. while(ptr != NULL)
85. print ptr->data
86. ptr= ptr->Rlink;
87. End while
88. Break
89. Stop

Program code:

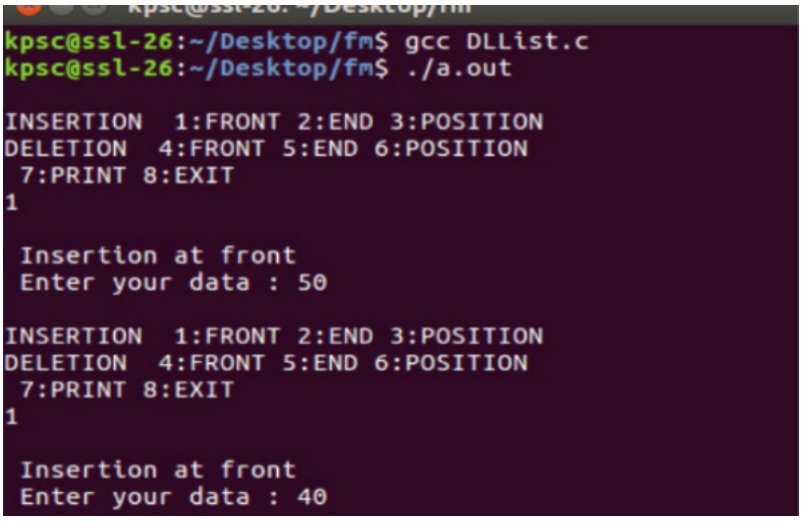
```
#include <stdio.h>
#include <stdlib.h>
int pos;
struct node {
	int data;
	struct node *Llink, *Rlink;
}*head=NULL, *ptr, *ptr1;

int insert_front()
{
	printf("\n Insertion at front\n Enter your data : ");
	ptr =(struct node*) malloc(sizeof(struct node*));
	scanf("%d", &ptr->data);
	if (head == NULL)
		head = ptr;
	else{
		head->Llink=ptr;
		ptr ->Rlink = head;
		head = ptr;
	}
}
int insert_end() //To Insert at End
{
	printf("\nInsertion at End\nEnter your data : ");
	ptr =(struct node*) malloc(2*sizeof(struct node*));
	scanf("%d", &ptr->data);
	if (head == NULL){
		head = ptr;
		head->Llink=NULL;
	}
	else{
		ptr1 = head;
		while (ptr1 ->Rlink != NULL)
			ptr1 = ptr1->Rlink;
		ptr1-> Rlink = ptr;
		ptr->Llink=ptr1;
```

```
        }
        ptr ->Rlink = NULL;
}
void insert_pos(){
        printf("Enter position: ");
        scanf("%d",&pos);
        ptr =(struct node*) malloc(sizeof(struct node*));
        if(ptr==NULL)
                printf("OVERFLOW\n");
        else{
                printf("\nData : ");
                scanf("%d", &ptr->data);
                ptr1= head;
                if(pos==1){
                        ptr->Rlink=ptr1;
                        ptr->Llink=NULL;
                        head=ptr;
                        return;
                }
                for(int i=1;i<pos-1;i++)
                        ptr1=ptr1->Rlink;

                ptr->Rlink=ptr1->Rlink;
                ptr->Llink=ptr1;
                ptr1->Rlink= ptr;
        }
}
void delete_front()
{
        if(head==NULL)
                printf("EMPTY LIST\n");
        else{
                ptr=head;
                head=ptr->Rlink;
                head->Llink=NULL;
                free(ptr);
                printf("deleted from begining..\n");
        }
}
void delete_end()
{
        ptr=head;
        if(head==NULL)
                printf("EMPTY LIST\n");
        else if(head->Rlink==NULL){
                head=NULL;
```

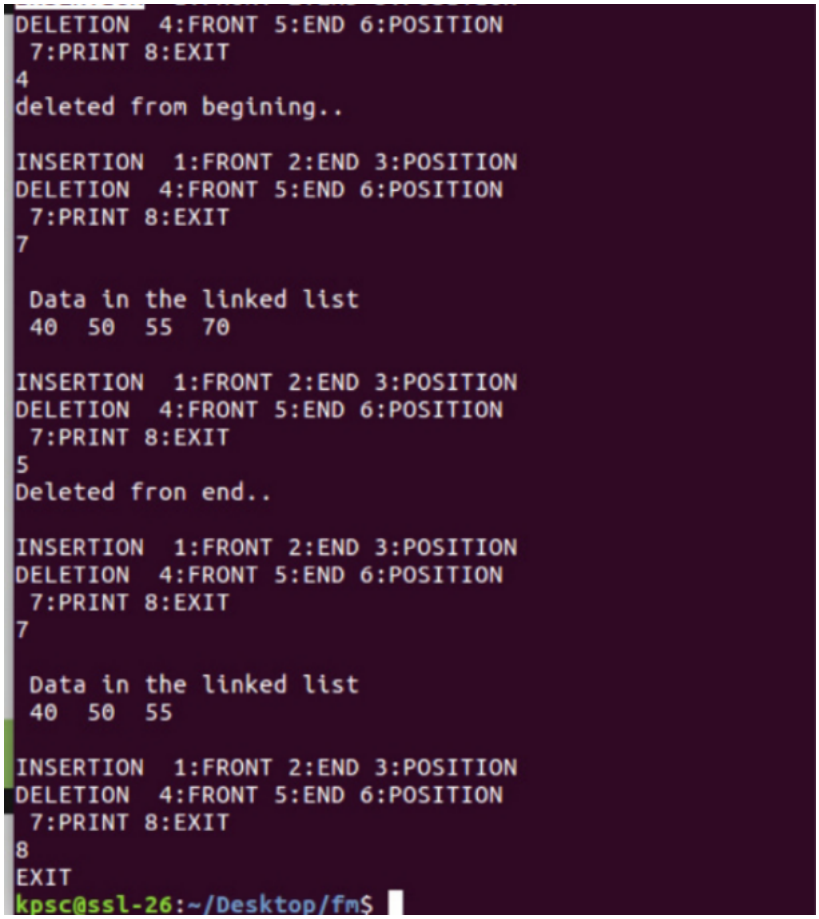
```
                free(ptr);
                printf("only one node.. deleted it..\n");
        }
        else{
                struct node* prev;
                while(ptr->Rlink!=NULL){
                        prev=ptr;
                        ptr=ptr->Rlink;
                }
                prev->Rlink=NULL;
                free(ptr);
                printf("Deleted fron end..\n");
        }
}
void delete_pos()
{
        printf("enter position to delete: ");
        scanf("%d",&pos);
        struct node * prev,*temp;
        ptr=head;
        for(int i=1;i<pos;i++) {
                if(head==NULL){
                        printf("Can't delete!!List is empty\n");
                        return;
                }
                prev=ptr;
                ptr=ptr->Rlink;
        }
        temp=ptr;
        ptr=ptr->Rlink;
        prev->Rlink=ptr;
        ptr->Llink=prev;
        free(temp);
        printf("deleted from index %d\n",pos-1);
}
void print() //To Print Linked List Data
{
        printf("\n Data in the linked list \n");
        ptr= head;
        while(ptr != NULL)
        {
                printf(" %d ", ptr->data);
                ptr= ptr->Rlink;
        } printf("\n");
}
void main()
```

```
{
        int op = 1;
        printf("\nINSERTION  1:FRONT 2:END 3:POSITION\nDELETION  4:FRONT 5:END
6:POSITION\n 7:PRINT 8:EXIT\n");
        scanf("%d",&op);
        while (op)
        {
                ptr =(struct node*) malloc(20*sizeof(struct node*));
                  switch(op)
                  {
                        case 1:insert_front();
                                break;
                        case 2:insert_end();
                                break;
                        case 3:insert_pos();
                                break;
                        case 4:delete_front();
                                break;
                        case 5:delete_end();
                                break;
                        case 6:delete_pos();
                                break;
                        case 7:print();
                                break;
                        case 8:printf("EXIT\n");
                                exit(0);
                                break;
                        default:printf("Invalid Choice\n");
                  }
                printf("\nINSERTION  1:FRONT 2:END 3:POSITION\nDELETION  4:FRONT
5:END 6:POSITION\n 7:PRINT 8:EXIT\n");
                scanf("%d",&op);
        }
}
```

Sample output:



```
INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
1

 Insertion at front
 Enter your data : 20

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
2

Insertion at End
Enter your data : 60

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
2

Insertion at End
Enter your data : 70

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
7

 Data in the linked list
 20  40  50  60  70

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
3
Enter position: 4

Data : 55

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
7

 Data in the linked list
 20  40  50  55  60  70

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
6
enter position to delete: 5
deleted from index 4

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
7

 Data in the linked list
 20  40  50  55  70

INSERTION  1:FRONT 2:END 3:POSITION
```



Result:
The program was excuted on an array of numbers and the output was obtained.

Expt.No: 18

Circular Linked List

Aim : - Write a menu driven program for performing the following operations on a circular Linked List: Display, Insert at Beginning, Insert at End, Insert at a specified Position, Delete from Beginning, Delete from End, Delete from a specified Position

Algorithm DoublyLL(ptr, Link)
Input Specification : Head, LLink, ptr, value
Output Specification: Linked list after insertion and deletion
Data Structures Used : structure

```
1.  Start
2.  Case insertion at front :
3.   ptr->data=value
4.   if (head == NULL)
5.              head= ptr
6.              ptr->link=head
7.      else
8.              ptr->link = head
9.              ptr1 = head
10.             while (ptr1->link!= head)
11.                     ptr1 =ptr1->link
12.            End while
13.             head = ptr
14.             ptr1->link=head
15.  End if
16. Break
17. Case insertion at end :
18.  ptr->data=value
19. if (head == NULL) Then
20.             head= ptr
21.             ptr->link=head
22. else
23.             ptr1 = head
24.             while (ptr1->link!= head)
25.                     ptr1 =ptr1->link
26.             End while
27.             ptr1->link=ptr
28.             ptr->link=head
29.  End if
30. Break
31. Case insertion at a position :
32. ptr->data = value
33.  ptr1= head
34.     if(pos==1)
35.             ptr->link=ptr1
36.             head=ptr
37.             return
38.  End if
```

```
    for  i=1 to <pos-1
            ptr1=ptr1->link
   End for
    ptr->link=ptr1->link
    ptr1->link= ptr
End if
Break
Case deletion from front :
if(head==NULL)
      print EMPTY LIST
else
            ptr=ptr1=head
            while (ptr1->link!= head)
                  ptr1 =ptr1->link
             End while
            head = ptr->link
            ptr1->link=head
            free(ptr)
End if
Break
Case deletion from end :
ptr=head
    if(head==NULL)
       print EMPTY LIST
    else if(head->link==NULL)
            head=NULL
            free(ptr)
    else
    while(ptr->link!=head)
            prev=ptr
            ptr=ptr->link
    End while
    prev->link=head
    free(ptr)
 End if
Break
Case deletion from a position :
Read position ,pos
ptr=head
    for i=1 To <pos
            if(head==NULL)
                  print List is empty
                  return
            End if
            prev=ptr
            ptr=ptr->link
   End for
    prev->link=ptr->link
    free(ptr)
Break
```

87. Case display :
88. ptr= head
89. while(ptr->link !=head)
90. print ptr->data
91. ptr= ptr->link
92. End while
93. print ptr->data
94. Break
95. Stop

Program Code :

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

int pos;
struct node {
   int data;
   struct node *link;
}*head=NULL, *ptr, *ptr1, *end=NULL;

int insert_front()
{
   printf("\n Insertion at front\n Enter your data : ");
   ptr =(struct node*) malloc(sizeof(struct node*));
   scanf("%d", &ptr->data);
   if(ptr==NULL)
            printf("OVERFLOW!!\n");
   else if (head == NULL){
            head= ptr;
            ptr->link=head;
   }
   else{
            ptr->link = head;
            ptr1 = head;
            while (ptr1->link!= head)
                     ptr1 =ptr1->link;
            head = ptr;
            ptr1->link=head;
            printf("Inserted at front..\n");
   }
}
int insert_end() //To Insert at End
{
   printf("\nInsertion at End\nEnter your data : ");
   end =(struct node*) malloc(2*sizeof(struct node*));
   scanf("%d", &ptr->data);
```

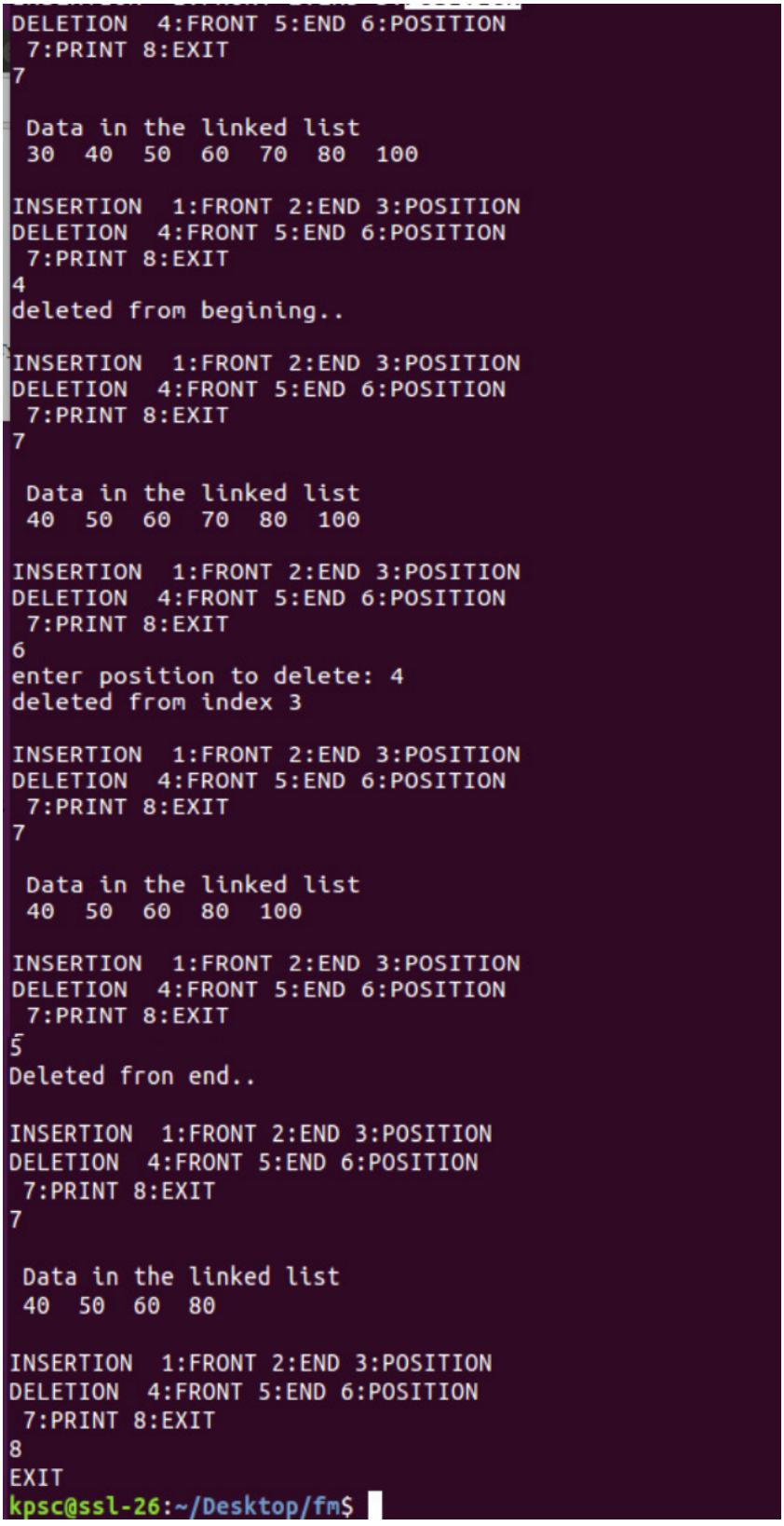
```
    if (head == NULL){
            head= ptr;
            ptr->link=head;
    }
    else{
            ptr1 = head;
            while (ptr1->link!= head)
                    ptr1 =ptr1->link;
            ptr1->link=ptr;
            ptr->link=head;
            printf("Inserted at end..\n");
    }
}
void insert_pos(){
    printf("Enter position: ");
    scanf("%d",&pos);
    ptr =(struct node*) malloc(sizeof(struct node*));
    if(ptr==NULL)
            printf("OVERFLOW\n");
    else{
            printf("\nData : ");
            scanf("%d", &ptr->data);
            ptr1= head;
            if(pos==1){
                    ptr->link=ptr1;
                    head=ptr;
                    return;
            }
            for(int i=1;i<pos-1;i++)
                    ptr1=ptr1->link;

            ptr->link=ptr1->link;
            ptr1->link= ptr;
            printf("Inserted..\n");
    }
}
void delete_front()
{
    if(head==NULL)
            printf("EMPTY LIST\n");
    else{
            ptr=ptr1=head;
            while (ptr1->link!= head)
                    ptr1 =ptr1->link;
            head = ptr->link;
            ptr1->link=head;
```

```
            free(ptr);
            printf("deleted from begining..\n");
    }
}
void delete_end()
{
    ptr=head;
    if(head==NULL)
            printf("EMPTY LIST\n");
    else if(head->link==NULL){
            head=NULL;
            free(ptr);
            printf("only one node.. deleted it..\n");
    }
    else{
            struct node* prev;
            while(ptr->link!=head){
                    prev=ptr;
                    ptr=ptr->link;
            }
            prev->link=head;
            free(ptr);
            printf("Deleted fron end..\n");
    }
}
void delete_pos()
{
    printf("enter position to delete: ");
    scanf("%d",&pos);
    struct node* prev;
    ptr=head;
    for(int i=1;i<pos;i++) {
            if(head==NULL){
                    printf("Can't delete!!List is empty\n");
                    return;
            }
            prev=ptr;
            ptr=ptr->link;
    }
    prev->link=ptr->link;
    free(ptr);
    printf("deleted from index %d\n",pos-1);
}
void print() //To Print Linked List Data
{
    printf("\n Data in the linked list \n");
```

```
   ptr= head;
   while(ptr->link !=head)
   {
            printf(" %d ", ptr->data);
            ptr= ptr->link;
   } printf(" %d\n",ptr->data);
}
void main()
{
   int op = 1;
   printf("\nINSERTION  1:FRONT 2:END 3:POSITION\nDELETION  4:FRONT 5:END
6:POSITION\n 7:PRINT 8:EXIT\n");
   scanf("%d",&op);
   while (op)
   {
            ptr =(struct node*) malloc(20*sizeof(struct node*));
              switch(op)
              {
                    case 1:insert_front();
                              break;
                    case 2:insert_end();
                              break;
                    case 3:insert_pos();
                              break;
                    case 4:delete_front();
                              break;
                    case 5:delete_end();
                              break;
                    case 6:delete_pos();
                              break;
                    case 7:print();
                              break;
                    case 8:printf("EXIT\n");
                              exit(0);
                              break;
                    default:printf("Invalid Choice\n");
              }
            printf("\nINSERTION  1:FRONT 2:END 3:POSITION\nDELETION  4:FRONT
5:END 6:POSITION\n 7:PRINT 8:EXIT\n");
            scanf("%d",&op);
   }
}
```

Sample output:

```
kpsc@ssl-26:~/Desktop/fm$ ./a.out

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
1

 Insertion at front
 Enter your data : 50

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
1

 Insertion at front
 Enter your data : 40
Inserted at front..

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
1

 Insertion at front
 Enter your data : 30
Inserted at front..

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
2

Insertion at End
Enter your data : 60
Inserted at end..

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
2

Insertion at End
Enter your data : 70
Inserted at end..

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
7

 Data in the linked list
 30  40  50  60  70

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
2

Insertion at End
Enter your data : 100
Inserted at end..

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
7

 Data in the linked list
 30  40  50  60  70  100

INSERTION  1:FRONT 2:END 3:POSITION
DELETION  4:FRONT 5:END 6:POSITION
 7:PRINT 8:EXIT
3
Enter position: 6

Data : 80
Inserted..

INSERTION  1:FRONT 2:END 3:POSITION
```



Result:

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 19

Queue Using Linked List

Aim : - Write a menu driven program to implement queue data strcture using linked list.

Algorithm QueueLL(ptr, Link)
Input Specification : Head, LLink, ptr, value
Output Specification: queue after insertion and deletion
Data Structures Used : structure

1. Start
2. Case Insertion :
3. ptr->data = value
4. if (head == NULL)
5. head = ptr
6. else
7. ptr1 = head
8. while (ptr1 ->link != NULL)
9. ptr1 = ptr1-> link
10. ptr1-> link = ptr
11. End if
12. ptr ->link = NULL
13. Break
14. Case deletion :
15. if(head==NULL)
16. print EMPTY LIST
17. else
18. ptr=head
19. head=ptr->link
20. free(ptr)
21. End if
22. Break
23. Case display :
24. if(head==NULL)
25. print EMPTY LIST
26. return
27. End if
28. ptr= head
29. while(ptr != NULL)
30. print ptr->data
31. ptr= ptr->link
32. End while
33. Break
34. Stop

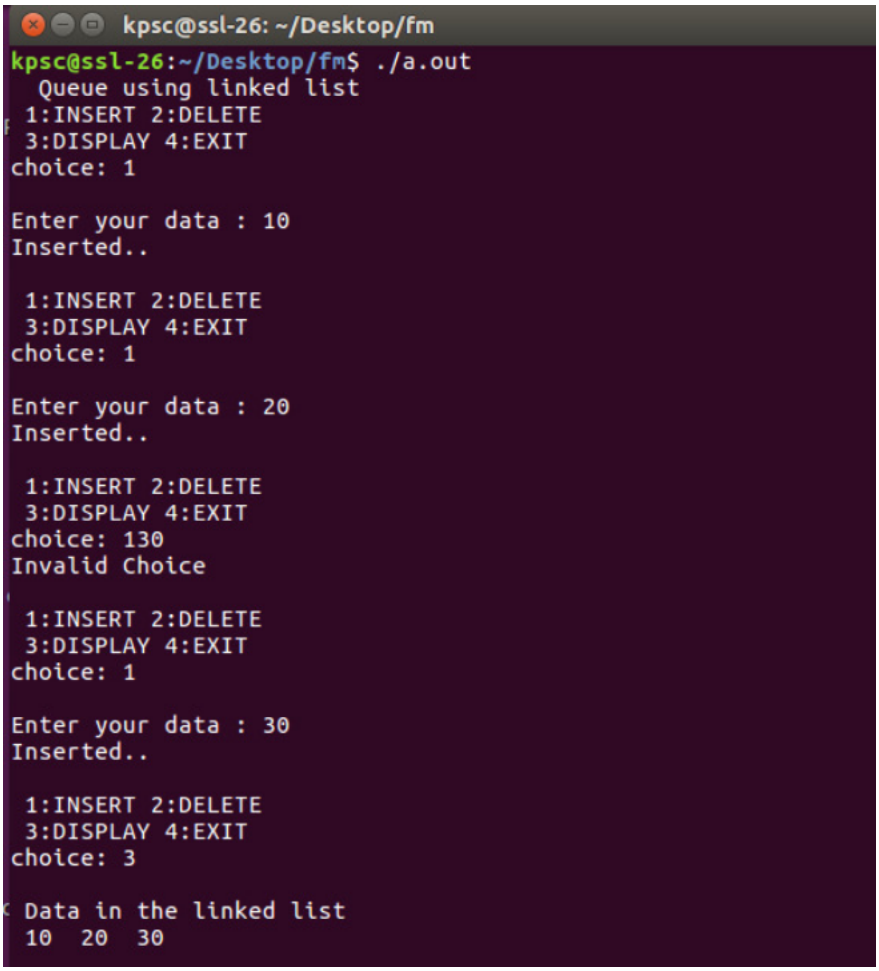
Program code:

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

```
#include <stdlib.h>
int n=0,pos;
struct node {
    int data;
    struct node *link;
}*head=NULL,*ptr,*ptr1;

int insert()
{
    printf("\nEnter your data : ");
    ptr =(struct node*) malloc(2*sizeof(struct node*));
    scanf("%d", &ptr->data);
    if (head == NULL)
            head = ptr;
    else{
            ptr1 = head;
            while (ptr1 ->link != NULL)
                    ptr1 = ptr1-> link;
            ptr1-> link = ptr;
    }
    ptr ->link = NULL;
    printf("Inserted..\n");
}

void delete()
{
    if(head==NULL)
            printf("EMPTY LIST\n");
    else{
            ptr=head;
            head=ptr->link;
            free(ptr);
            printf("deleted..\n");
    }
}

void print()
{
    if(head==NULL){
            printf("EMPTY LIST!!\n");
            return;
    }
    printf("\n Data in the linked list \n");
    ptr= head;
    while(ptr != NULL) {
            printf(" %d ", ptr->data);
            ptr= ptr->link;
    } printf("\n");
}
void main()
```

```
{
    int op = 1;
    printf("  Queue using linked list");
    printf("\n 1:INSERT 2:DELETE\n 3:DISPLAY 4:EXIT\nchoice: ");
    scanf("%d",&op);
    while (op)
    {
                switch(op)
                {
                      case 1:insert();
                               break;
                      case 2:delete();
                               break;
                      case 3:print();
                               break;
                      case 4:printf("EXIT\n");
                               exit(0);
                               break;
                      default:printf("Invalid Choice\n");
                 }
               printf("\n 1:INSERT 2:DELETE\n 3:DISPLAY 4:EXIT\nchoice: ");
               scanf("%d",&op);
    }
}
```

Sample Output:



Result :

The program was excuted on an array of numbers and the output was obtained.

Expt.No: 20

Polynomial Addition Using Linked List

Aim : - Write a program to read two polynomials and store them using linked list.Calculate the sum of the two polynomials and display the first polynomial, second polynomial and the resultant polynomial.

Algorithm Read_polynomial (poly *start)

1.Start
2.Enter no of terms
3.open for loop
4.Enter coefficients and powers
5.call insert to assign to a pointer
6.close for loop
7.stop

Algorithm PolynomialsLL(ptr, Link)

Input Specification : Head,Head1, Head2, LLink, ptr, value
Output Specification: polynomials sum
Data Structures Used : structure

1. Start
2. Read_polynomial1(head1)
3. Read_polynomial2(head2)
4. Define method to add
5. a=head1 , b=head2
6. while (a and b!=NULL)
7. if(head==NULL)
8. head=t
9. ptr1=head
10. End if
11. if(a->expo > b->expo)
12. t->expo=a->expo
13. t->coef=a->coef
14. a=a->link
15. else if (a->expo < b->expo)
16. t->expo=b->expo
17. t->coef=b->coef
18. b=b->link
19. else
20. t->expo=a->expo
21. t->coef=(a->coef)+(b->coef)
22. a=a->link
23. b=b->link
24. End if

25. ptr1->link=t
26. ptr1=ptr1->link
27. End while
28. while(a!=NULL)
29. t->expo=a->expo
30. t->coef=a->coef
31. a=a->link
32. ptr1->link=t
33. ptr1=ptr1->link
34. End while
35. while (b!=NULL)
36. t->expo=b->expo
37. t->coef=b->coef
38. b=b->link
39. ptr1->link=t
40. ptr1=ptr1->link
41. End while
42. ptr1->link=NULL
43. End add() method
44. Stop

Program Code

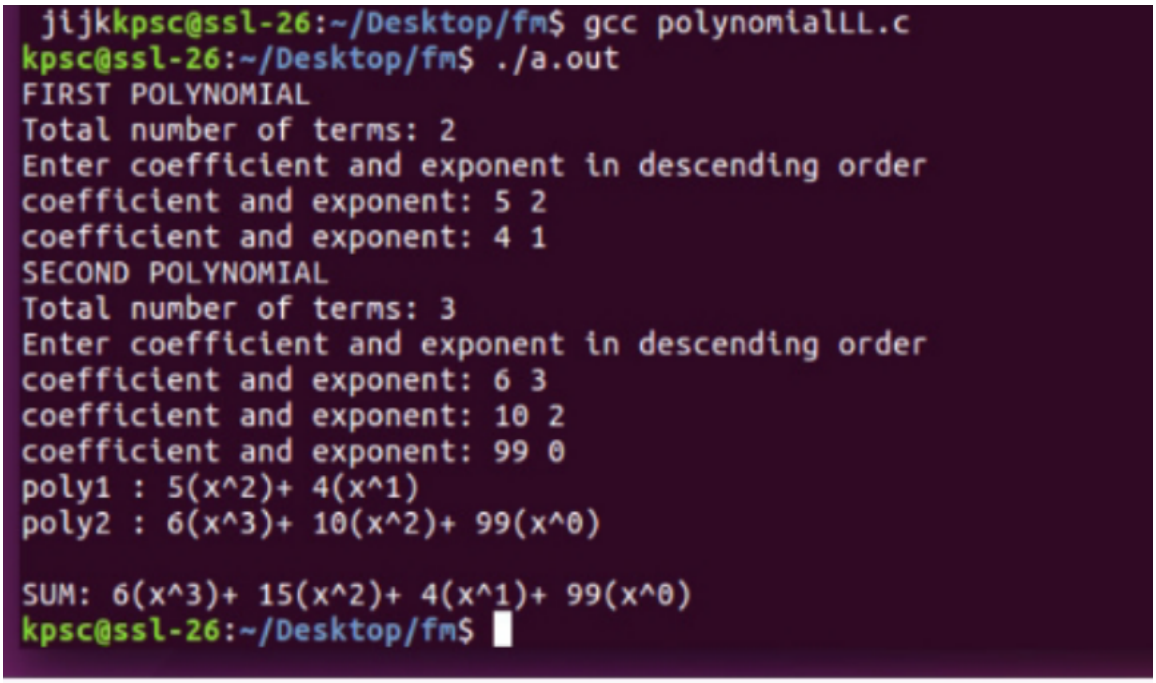
```
#include <stdio.h>
#include <stdlib.h>

struct node {
        int coef,expo;
        struct node *link;
}*head1=NULL,*head2=NULL, *head=NULL, *ptr1;

void read_poly1()
{
        struct node *ptr,*ptr1;
        ptr=(struct node*)malloc(sizeof(struct node*));
        if(ptr==NULL){
                printf("OverFlow\n");
                return;
        }
        printf("coefficient and exponent: ");
        scanf("%d%d",&ptr->coef,&ptr->expo);
        if(head1==NULL)
                head1=ptr;
        else{
                ptr1=head1;
                while(ptr1->link!=NULL)
```

```
                ptr1=ptr1->link;
            ptr1->link=ptr;
        }
        ptr->link=NULL;
}
void read_poly2()
{
        struct node *ptr,*ptr1;
        ptr=(struct node*)malloc(sizeof(struct node*));
        if(ptr==NULL){
            printf("OverFlow\n");
            return;
        }
        printf("coefficient and exponent: ");
        scanf("%d%d",&ptr->coef,&ptr->expo);
        if(head2==NULL)
            head2=ptr;
        else{
            ptr1=head2;
            while(ptr1->link!=NULL)
                ptr1=ptr1->link;
            ptr1->link=ptr;
        }
        ptr->link=NULL;
}
void display1()
{
        printf("poly1 : ");
        struct node *ptr1;
        ptr1=head1;
        while(ptr1->link!=NULL){
            printf("%d(x^%d)+ ",ptr1->coef,ptr1->expo);
            ptr1=ptr1->link;
        }
        printf("%d(x^%d)\n",ptr1->coef,ptr1->expo);
}
void display2()
{
        printf("poly2 : ");
        struct node *ptr1;
        ptr1=head2;
        while(ptr1->link!=NULL){
            printf("%d(x^%d)+ ",ptr1->coef,ptr1->expo);
            ptr1=ptr1->link;
        }
        printf("%d(x^%d)\n",ptr1->coef,ptr1->expo);
```

```
}
void add()
{
	struct node *a,*b,*t,*ptr1;
	a=head1;
	b=head2;
	while(a!=NULL&&b!=NULL)
	{
		t=(struct node*)malloc(sizeof(struct node*));
		if(head==NULL){
			head=t;
			ptr1=head;
		}
		if(a->expo > b->expo){
			t->expo=a->expo;
			t->coef=a->coef;
			a=a->link;
		}
		else if(a->expo < b->expo){
			t->expo=b->expo;
			t->coef=b->coef;
			b=b->link;
		}
		else
		{
			t->expo=a->expo;
			t->coef=(a->coef)+(b->coef);
			a=a->link;
			b=b->link;
		}
		ptr1->link=t;
		ptr1=ptr1->link;
	}
	while(a!=NULL){
		t=(struct node*)malloc(sizeof(struct node*));
		t->expo=a->expo;
		t->coef=a->coef;
		a=a->link;

		ptr1->link=t;
		ptr1=ptr1->link;
	}
	while(b!=NULL){
		t=(struct node*)malloc(sizeof(struct node*));
		t->expo=b->expo;
		t->coef=b->coef;
```

```
                b=b->link;

                ptr1->link=t;
                ptr1=ptr1->link;
        }
        ptr1->link=NULL;
}

void display()
{
        if(head==NULL)
                printf("EMPTY LIST\n");
        else{
                ptr1=head;
                while(ptr1->link!=NULL){
                        printf("%d(x^%d)+ ",ptr1->coef,ptr1->expo);
                        ptr1=ptr1->link;
                }
                printf("%d(x^%d)\n",ptr1->coef,ptr1->expo);
        }
}

void main()
{
        int n1,n2;
        printf("FIRST POLYNOMIAL\nTotal number of terms: ");
        scanf("%d",&n1);
        printf("Enter coefficient and exponent in descending order\n");
        for(int i=0;i<n1;i++)
                read_poly1();

        printf("SECOND POLYNOMIAL\nTotal number of terms: ");
        scanf("%d",&n2);
        printf("Enter coefficient and exponent in descending order\n");
        for(int i=0;i<n2;i++)
                read_poly2();
        display1();
        display2();
        add();
        printf("\nSUM: ");
        display();
}
```

Sample output:



Result:
The program was excuted on an array of numbers and the output was obtained.

Expt.No: 21

Polynomial Multiplication Using Linked List

Aim : - Write a program to read two polynomials and store them using linked list.Calculate the product of the two polynomials and display the first polynomial, second polynomial and the resultant polynomial.

Algorithm create(poly *start)

1.Start
2.Enter no of terms
3.open for loop
4.Enter coefficients and powers
5.call insert to assign to a pointer
6.close for loop
7.stop

Algorithm insert(poly *start,cf,pow)

1.Start
2.Assign memory for temp od type poly
3.if(start==NULL)
4.temp->next = start; start = temp;
5.else
6.ptr = start;
7.while (ptr->next != NULL && ptr->next->pow >= pow)
8. ptr = ptr->next;
9.temp->next = ptr->next;
10. ptr->next = temp;
11.return start
12.stop

Algorithm mul(*p1,*p2)

Input specification: number of terms of polynomial
Output Specification: multiplication result of the input polynomials

1.Start
2.declare *start3 of type struct poly.*p2_b=p2
3.if (p1 == NULL || p2 == NULL)
4.return
5.else
6.while (p1 != NULL)
7. p2 = p2_b
8.while (p2 != NULL)

9. start3 = insert(start3, p1->cf * p2->cf, p1->pow + p2->pow);
10. p2 = p2->next;
11.close while
12.p1 = p1->next;
13.close while
14.stop

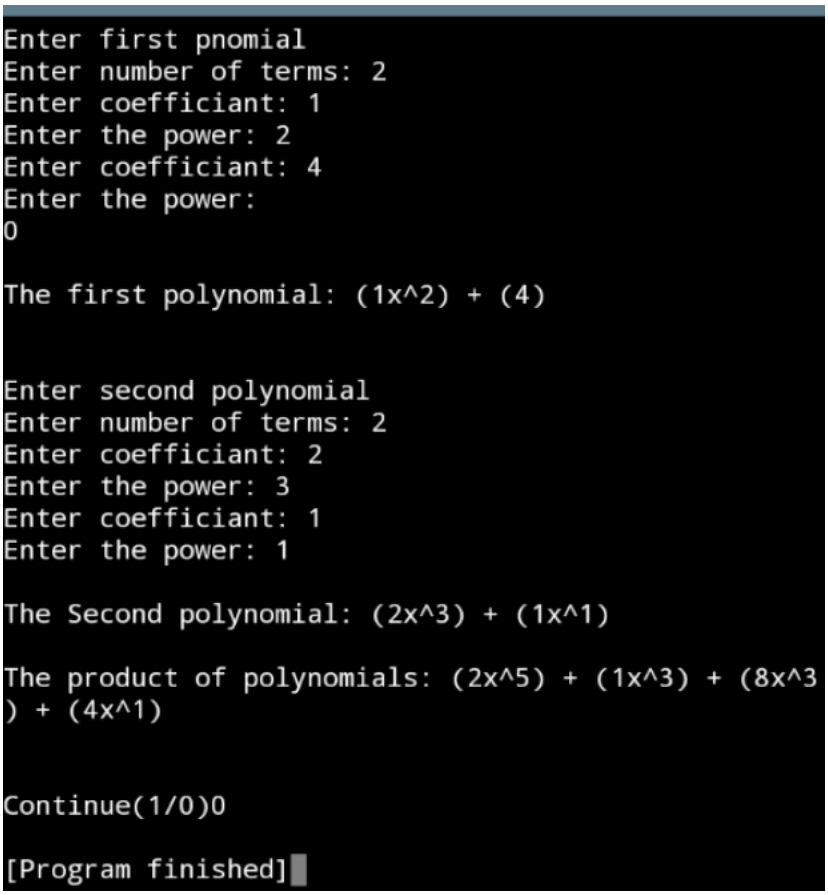
Program code :

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

struct poly {
        int cf;
        struct poly *next;
        int pow;
};

struct poly *insert(struct poly *start, int cf, int pow)
 {
      struct poly *ptr, *temp;
      temp = (struct poly *)malloc(sizeof(struct poly));

      temp->cf = cf;
      temp->pow = pow;
      if (start == NULL || pow > start->pow)
      {
            temp->next = start;
            start = temp;
      }
      else {
      ptr = start;
      while (ptr->next != NULL && ptr->next->pow >= pow)
            ptr = ptr->next;
      temp->next = ptr->next;
      ptr->next = temp;
      }
      return start;
}

struct poly *create(struct poly *start)
{
      int n, cf, pow;
      printf("Enter number of terms: ");
      scanf("%d", &n);
```

```
    for (int i = 0; i < n; i++)
     {
    printf("Enter coefficiant: ");
    scanf("%d", &cf);
    printf("Enter the power: ");
    scanf("%d", &pow);
    start = insert(start, cf, pow);
    }
    return start;
}

void display(struct poly *ptr)
 {
     if (ptr == NULL) {
     printf("Zero polynomial\n");
     return;
     }
     else {
     while (ptr != NULL) {
     if (ptr->pow == 0)
          printf("(%d)", ptr->cf);
     else
           printf("(%dx^%d)", ptr->cf, ptr->pow);
     ptr = ptr->next;
     if (ptr != NULL)
           printf(" + ") ;
     else
          printf("\n");
     }
     }
}

void mul(struct poly *p1, struct poly *p2)
 {

     struct poly *start3;
     struct poly *p2_b = p2;
     start3 = NULL;
     if (p1 == NULL || p2 == NULL)
      {
     printf("Zero polynomial\n");
     return;
     }
     else
     {
```

```
        while (p1 != NULL) {
        p2 = p2_b;

        while (p2 != NULL)
         {
        start3 = insert(start3, p1->cf * p2->cf, p1->pow + p2->pow);
        p2 = p2->next;
        }
        p1 = p1->next;
        }
        }
printf("The product of polynomials: ");
display(start3);
}

void main()
{
         struct poly *p1 = NULL, *p2 = NULL;
         int n = 1;
         while (n != 0)
         {
         printf("Enter first pnomial\n");
         p1 = create(p1);
         printf("\nThe first polynomial: ");
         display(p1);
         printf("\n");
         printf("\nEnter second polynomial\n");
         p2 = create(p2);
         printf("\nThe Second polynomial: ");
         display(p2);
         printf("\n");
         mul(p1, p2);
         printf("\n\nContinue(1/0)");
         scanf("%d", &n);
         }
}
```

Sample Output :



Result :
The program was excuted on an array of numbers and the output was obtained.

END PAGE