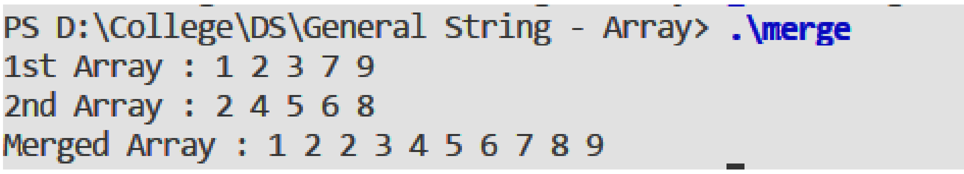
Objective:

Write a program to merge two sorted arrays.

Code :

```
#include <stdio.h>

int merge(int a[] , int b[], int c[] , int lena , int lenb);

int main() {
    int c[20];
    int a[] = {1,2,3,7,9,};
    int b[] = {2,4,5,6,8,};

    merge(a,b ,c, 5 ,5);

    printf("1st Array : ");
    for(int i = 0 ; i < 5 ; i++){
        printf("%d " , a[i]) ;
    }
    printf("\n");

    printf("2nd Array : ");
    for(int i = 0 ; i < 5 ; i++){
        printf("%d " , b[i]) ;
    }
    printf("\n");

    printf("Merged Array : ");
    for(int i = 0 ; i < 10 ; i++){
        printf("%d " , c[i]) ;
    }


}

int merge(int a[] , int b[], int c[] , int lena , int lenb){

    int c1 = 0 ;
    int c2 = 0 ;
    int i = 0 ;
    while(c1 < lena && c2 < lenb){
        if(a[c1] > b[c2]) {
            c[i++] = b[c2++] ;
        } else {
            c[i++] = a[c1++] ;
        }
    }
    while(i < lena  + lenb ) {
        if(c1<lena) {
            c[i++] = a[c1++] ;
        } else {
            c[i++] = b[c2++] ;
        }
    }
}
```

Output :



```
PS D:\College\DS\General String - Array> .\merge
1st Array : 1 2 3 7 9
2nd Array : 2 4 5 6 8
Merged Array : 1 2 2 3 4 5 6 7 8 9
```

Objective:

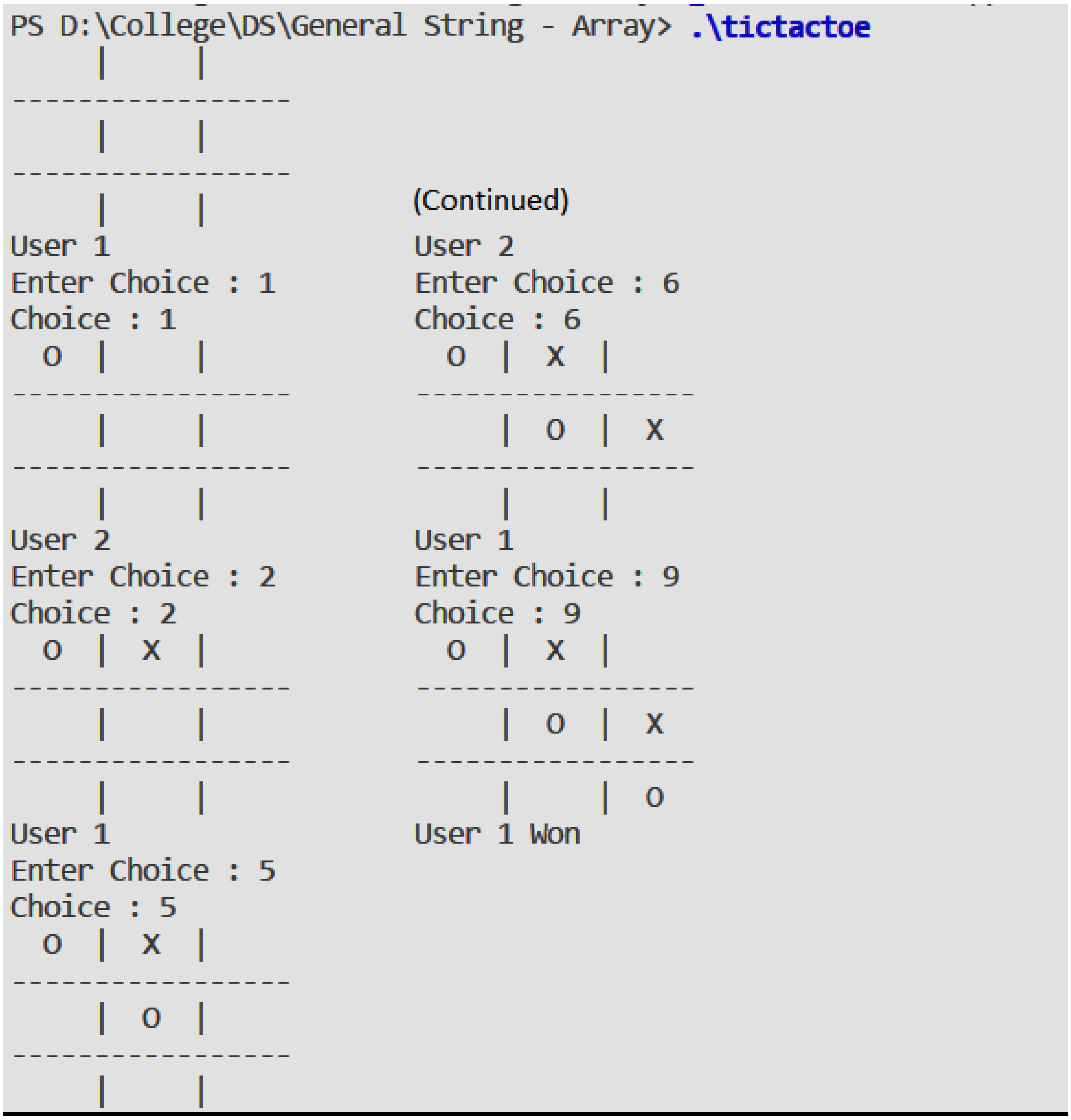
Write a program to implement Tic Tac Toe Game.

Code :

```
#include <iostream>

using namespace::std;

void init(int game[3][3]){
    for(int i=0 ; i<3; i++){
        for(int j=0 ; j<3; j++) {
            game[i][j] = -1 ;
        }
    }
}

void print(int game[3][3]){
    for(int i=0 ; i<3; i++){
        for(int j=0 ; j<3; j++) {
            if(game[i][j] == -1 ){
                cout<<"     " ;
            } else if(game[i][j] == 0 ){
                cout<<"  O  " ;
            } else if(game[i][j] == 1 ){
                cout<<"  X  " ;
            }
            if (j < 2) {
                cout << "|" ;
            }
        }
        if (i < 2) {
            cout << endl << "-----------------" ;
        }

        cout << endl ;

    }
}


int input(int x , int y , int user, int game[3][3]){
    if(game[x][y] != - 1 ) {
        return 1;
    }
    game[x][y] = (user % 2 ==  0) ? 0 : 1 ;
    return 0;
}

int check(int game[3][3]){
    for(int i=0;i<3;i++){
        if ( (game[i][0] == game[i][2] && game[i][0] == game[i][1] )
        && game[i][0] != -1 ){
            return 0 ;
        }
    }

    for(int i=0;i<3;i++){
        if ( (game[0][i] == game[1][i] && game[0][i] == game[2][i] )
        && game[0][i] != -1 ){
            return 0 ;
        }
    }

    if ( (game[0][0] == game[1][1] && game[0][0] == game[2][2] )
    && game[0][0] != -1 ){
        return 0 ;
    }

    if ( (game[0][2] == game[1][1] && game[0][2] == game[2][0] )
    && game[0][2] != -1 ){
        return 0 ;
    }

    return -1 ;

    //return 0;
}

int main()
{
    int i=0,j=0;
    int game[3][3];
    int choice ;

    int x,y ;
    int user = 0;
    int gameison = 1 ;

    init(game);

    while(gameison){
        print(game);

        cout << "User " << user%2 + 1 << endl ;
        cout << "Enter Choice : " ;
        cin >> choice ;
        cout << "Choice : " << choice << endl;

        x = (choice - 1)  / 3  ;
        y = ( choice % 3 + 2 ) % 3 ;

        int temp = input(x,y,user,game);
        if(temp){
            cout<<"Already Filled Choice" << endl;
            continue;
        }
```

```
            gameison = check(game);

            user++ ;
        }

        print(game);
        cout << "User " << user%2 << " Won" << endl ;
        return 0;
    }
```

Output :



```
PS D:\College\DS\General String - Array> .\tictactoe
     |     |
-----------------
     |     |
-----------------
     |     |
User 1
Enter Choice : 1
Choice : 1
  O  |     |
-----------------
     |     |
-----------------
     |     |
User 2
Enter Choice : 2
Choice : 2
  O  |  X  |
-----------------
     |     |
-----------------
     |     |
User 1
Enter Choice : 5
Choice : 5
  O  |  X  |
-----------------
     |  O  |
-----------------
     |     |
```

```
(Continued)
User 2
Enter Choice : 6
Choice : 6
  O  |  X  |
-----------------
     |  O  |  X
-----------------
     |     |
User 1
Enter Choice : 9
Choice : 9
  O  |  X  |
-----------------
     |  O  |  X
-----------------
     |     |  O
User 1 Won
```

Objective:

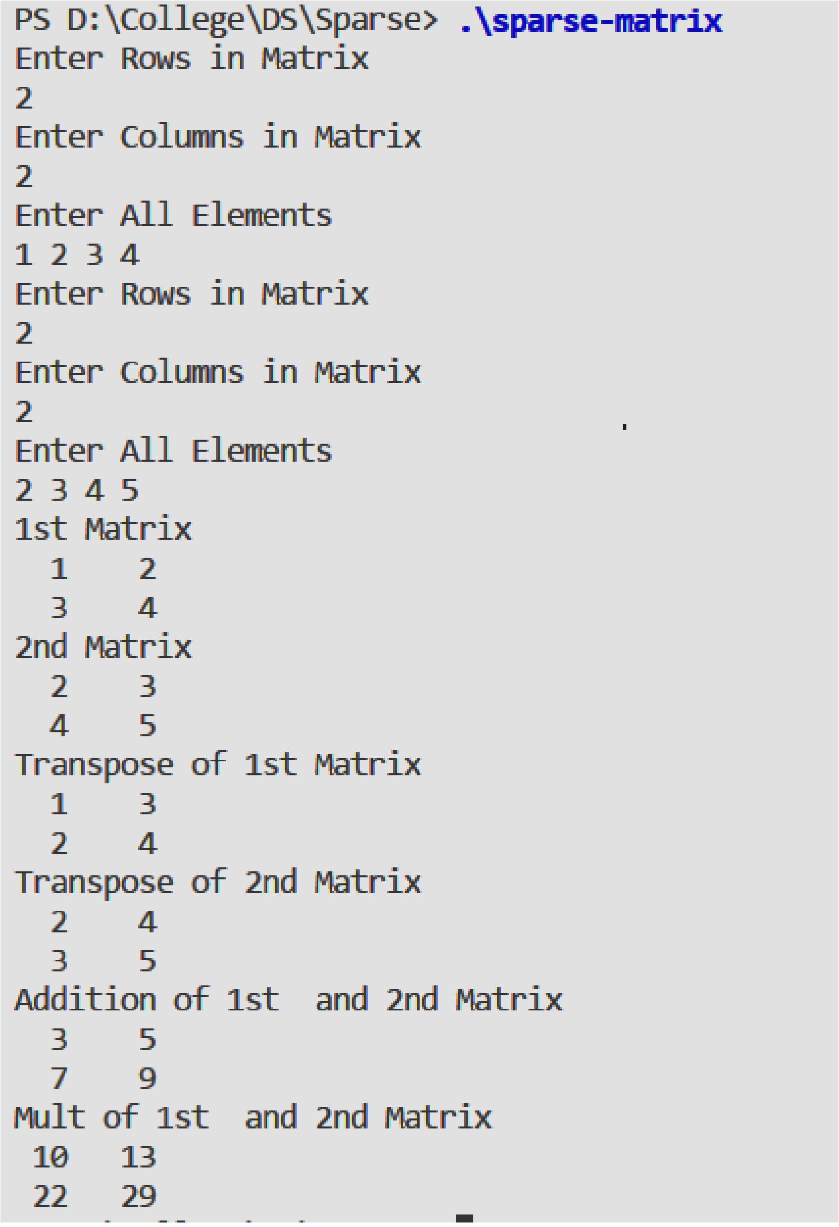
Write a program to implement the a 2D Matrix as a **Sparse Matrix** and perform transform, addition and multiplication operations on it.

Code :

```
#include <iostream>
#include <iomanip>

using namespace std ;

struct sparse {
    int row ;
    int col ;
    int value ;
} ;

void sparse_input( sparse[] );
void sparse_display( sparse[] );

void sparse_transpose(sparse[] , sparse[]);
void sparse_addition(sparse[] , sparse[] , sparse[]);
void sparse_multiplication(sparse[] , sparse[] , sparse[]);


int main() {

    sparse m1[20] , m2[20] , m3[20] , m4[20] , m5[20];

    sparse_input( m1 );
    sparse_input( m2 );

    cout << "1st Matrix" << endl ;
    sparse_display( m1 );

    cout << "2nd Matrix" << endl ;
    sparse_display( m2 );

    cout << "Transpose of 1st Matrix" << endl ;
    sparse_transpose( m1 , m3 );
    sparse_display( m3 );

    cout << "Transpose of 2nd Matrix" << endl ;
    sparse_transpose( m2 , m3 );
    sparse_display( m3 );

    cout << "Addition of 1st  and 2nd Matrix" << endl ;
    sparse_addition(m1 , m2 , m4);
    sparse_display(m4) ;

    cout << "Mult of 1st  and 2nd Matrix" << endl ;
    sparse_multiplication(m1 , m2 , m5);
    sparse_display(m5) ;

    return 0 ;
}


void sparse_input(sparse matrix[]) {

    int row , col , elem , k;

    cout << "Enter Rows in Matrix" << endl ;
    cin >> matrix[0].row ;
    cout << "Enter Columns in Matrix" << endl ;
    cin >> matrix[0].col ;

    cout << "Enter All Elements" << endl ;

    k = 1 ;

    for( int i = 0 ; i < matrix[0].row ; i++ ){

        for( int j = 0 ; j < matrix[0].col ; j++ ){
            cin >> elem ;
            if (elem != 0 ){
                matrix[k].row = i ;
                matrix[k].col = j ;
                matrix[k].value = elem ;
                k++ ;
            }
        }

    }

    matrix[0].value = k - 1 ;

}

void sparse_display(sparse matrix[]) {

    int k = 1;

    for( int i = 0 ; i < matrix[0].row ; i++ ){

        for( int j = 0 ; j < matrix[0].col ; j++ ){

            if ( k <= matrix[0].value && matrix[k].row == i && matrix[k].col == j ){
                cout << setw(3) << matrix[k].value ;
                k++ ;
            }
            else {
                cout << setw(3) << 0 ;
            }
            cout << "  " ;
        }
        cout << endl ;
    }
```

```
}

void sparse_transpose(sparse matrix[] , sparse transpose[]){
    int c[20] , d[20] , i , m , n , t ;
    m = matrix[0].row ;
    n = matrix[0].col ;
    t = matrix[0].value ;

    transpose[0].row = n ;
    transpose[0].col = m ;
    transpose[0].value = t ;

    for( i = 0 ; i < n ; i++ ) {
        c[i] = 0 ;
    }
    for( i = 1 ; i <= t ; i++ ) {
        c[ matrix[i].col ] ++ ;
    }
    d[0] = 1 ;
    for( i =1 ; i < n ; i++ ){
        d[i] = d[i-1] + c[i-1] ;
    }

    for( int i = 1 ; i <=t ; i++ ){
        transpose[ d[matrix[i].col] ].row = matrix[i].col ;
        transpose[ d[matrix[i].col] ].col = matrix[i].row ;
        transpose[ d[matrix[i].col] ].value = matrix[i].value ;

        d[matrix[i].col]++ ;
    }

}

void sparse_addition(sparse A[] , sparse B[] , sparse C[] ){

    int i = 1 ;
    int j = 1 ;
    int k = 1;

    if ( A[0].row != B[0].row || A[0].col != B[0].col  ) {
        cout << "Addition Not Possible" << endl ;
        return ;
    }

    while( i <= A[0].value && j <= B[0].value ){

        if ( ( A[i].row < B[j].row ) || ( A[i].row == B[j].row && A[i].col < B[j].col ) ){
            C[k].row = A[i].row ;
            C[k].col = A[i].col ;
            C[k].value = A[i].value ;
            i++ ;
            k++ ;
        }
        else if ( ( A[i].row > B[j].row ) || ( A[i].row == B[j].row &&
        A[i].col > B[j].col ) ){
            C[k].row = B[j].row ;
            C[k].col = B[j].col ;
            C[k].value = B[j].value ;
            j++ ;
            k++ ;
        }
        else {
            C[k].row = A[i].row ;
            C[k].col = A[i].col ;
            C[k].value = A[i].value + B[j].value;
            i++ ;
            j++ ;
            k++ ;
        }

    }

    while ( i <= A[0].value ) {
        C[k].row = A[i].row ;
        C[k].col = A[i].col ;
        C[k].value = A[i].value ;
        i++ ;
        k++ ;
    }

    while (j <= B[0].value) {
        C[k].row = B[j].row ;
        C[k].col = B[j].col ;
        C[k].value = B[j].value ;
        j++ ;
        k++ ;
    }

    C[0].row = A[0].row ;
    C[0].col = A[0].col ;
    C[0].value = k-1 ;

    return ;


}

void sparse_multiplication(sparse A[] , sparse B[] , sparse C[]){

    sparse transposeB[20] ;

    // Sqaure Matrix Only;

    sparse_transpose(B , transposeB) ;



    int c[20] , d[20] ,e[20] , i , j, m , n , t  ,temp , iter , iterB, row , col , k;
    m = transposeB[0].row ;
    n = transposeB[0].col ;
    t = transposeB[0].value ;
    k = 1 ;

    for( i = 0 ; i < n ; i++ ) {
        c[i] = 0 ;
    }
    for( i = 1 ; i <= t ; i++ ) {
        c[ transposeB[i].row ] ++ ;
    }
    d[0] = 1 ;
    for( i =1 ; i < n ; i++ ){
        d[i] = d[i-1] + c[i-1] ;
    }


    for( i = 0 ; i < n ; i++ ) {
        c[i] = 0 ;
    }
    for( i = 1 ; i <= t ; i++ ) {
        c[ A[i].row ] ++ ;
    }
    e[0] = 1 ;
    for( i =1 ; i < n ; i++ ){
        e[i] = e[i-1] + c[i-1] ;
    }


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

            iter = e[row] ;
            iterB = d[col] ;
            // cout << iter << iterB <<endl;
            temp = 0 ;
            while( A[iter].row == row && transposeB[iterB].row == col ) {

                if ( transposeB[iterB].col == A[iter].col) {
                    temp += transposeB[iterB].value * A[iter].value ;
                    iter ++ ;
                    iterB ++ ;
                }
                else if ( transposeB[iterB].col < A[iter].col ) {
                    iterB ++ ;
                } else {
                    iter++ ;
                }

            }

            if (temp != 0 ){
                C[k].row = row ;
                C[k].col = col ;
                C[k].value = temp ;
                k++ ;
            }
        }
    }

    C[0].row = n ;
    C[0].col = n ;
    C[0].value = k-1 ;
}
```

Output :



```
PS D:\College\DS\Sparse> .\sparse-matrix
Enter Rows in Matrix
2
Enter Columns in Matrix
2
Enter All Elements
1 2 3 4
Enter Rows in Matrix
2
Enter Columns in Matrix
2
Enter All Elements
2 3 4 5
1st Matrix
  1    2
  3    4
2nd Matrix
  2    3
  4    5
Transpose of 1st Matrix
  1    3
  2    4
Transpose of 2nd Matrix
  2    4
  3    5
Addition of 1st  and 2nd Matrix
  3    5
  7    9
Mult of 1st  and 2nd Matrix
 10   13
 22   29
```

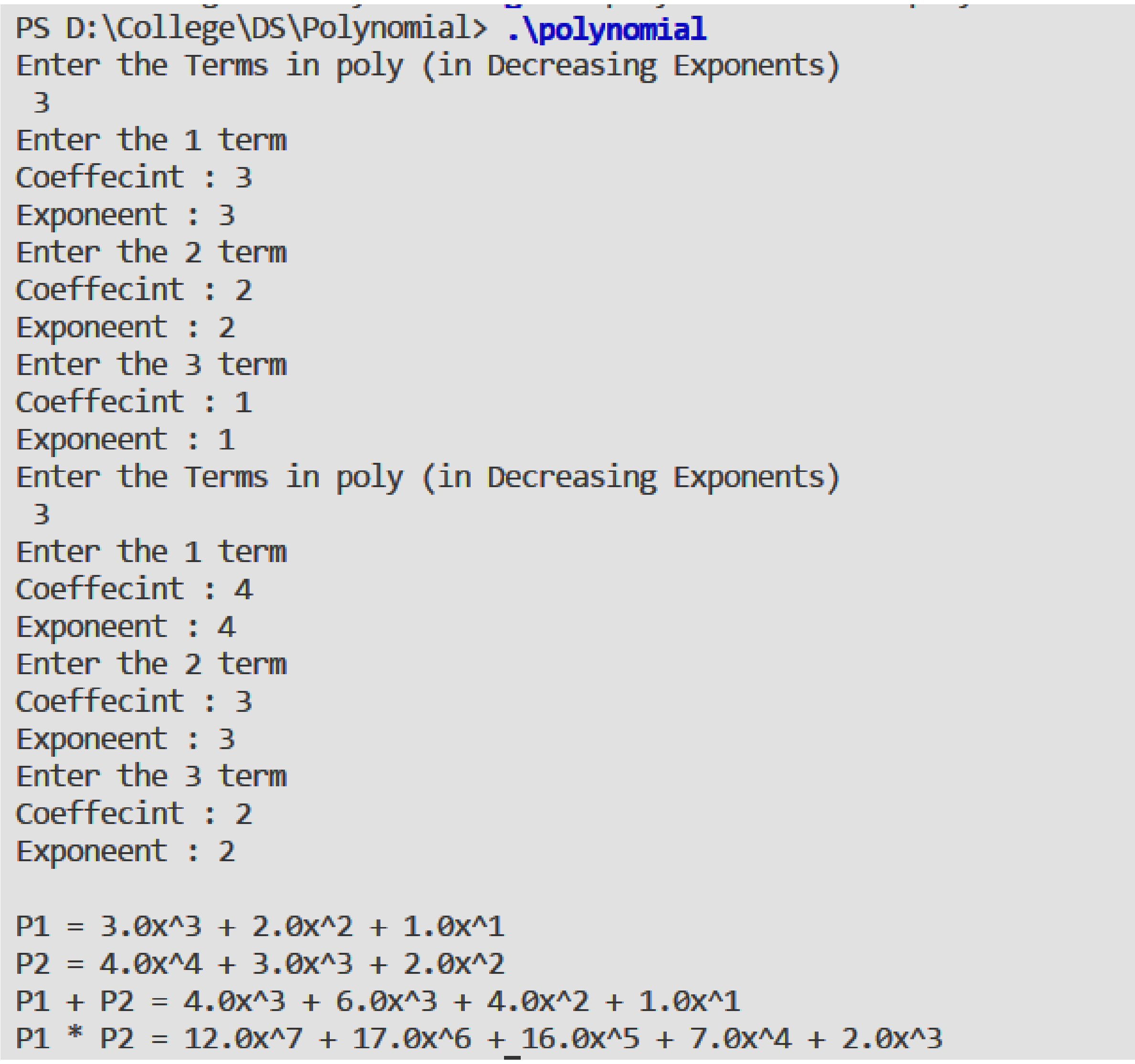
Objective:

Write a program to implement in Polynomial Addition and Multiplication

Code :

```
#include <stdio.h>

typedef struct {
    float coeff ;
    int exp ;
} poly ;

void inputPoly(poly[] , int*) ;
void displayPoly(poly[] , int) ;
void copyPoly(poly[] , poly[] , int) ;
void addPoly(poly[] ,poly[] , poly[] , int , int , int* ) ;
void multerm( poly[] , poly[] , int, int , int );
void mulPoly( poly[] , poly[] , poly[] , int, int , int* );

int main(){
    int S1 , S2 , S3  , S4;
    poly P1[20] , P2[20] , P3[20] , P4[20] ;

    inputPoly( P1 , &S1 ) ;
    inputPoly( P2 , &S2 ) ;

    printf("\n");

    printf("P1 = ") ;
    displayPoly(P1 , S1) ;
    printf("P2 = ") ;
    displayPoly(P2 , S2) ;

    printf("P1 + P2 = ");
    addPoly(P1 , P2 , P3 , S1 ,S2 , &S3) ;
    displayPoly(P3 , S3) ;

    printf("P1 * P2 = ");
    mulPoly(P1 , P2 , P4 , S1 ,S2 , &S4) ;
    displayPoly(P4 , S4) ;

    return 0;
}
void inputPoly(poly P[] , int* S){

    printf("Enter the Terms in poly (in Decreasing Exponents) \n ");
    scanf("%d" , S) ;
    for( int i = 0 ; i < *S ; i++ ){
        printf("Enter the %d term \n" , i+1);
        printf("Coeffecint : ") ;
        scanf("%f" , &P[i].coeff) ;
        printf("Exponeent : ") ;
        scanf("%d" , &P[i].exp) ;
    }
}

void displayPoly(poly P[] , int S){

    for( int i = 0 ; i < S ; i++ ){
        printf("%0.1fx^%d" , P[i].coeff, P[i].exp) ;
        if(i < S-1 )
            printf(" + ");
    }
    printf("\n");
}

void addPoly(poly P1[] , poly P2[]  ,poly P3[] ,  int S1,int S2,int* S3 ){

    int i,j,k;
    i=j=k=0;

    while( i < S1 && j < S2){
        if( P1[i].exp == P2[j].exp ){
            P3[k].exp = P1[i].exp ;
            P3[k++].coeff = P1[i++].coeff + P2[j++].coeff ;
        }
        else if( P1[i].exp > P2[j].exp ){
            P3[k].exp = P1[i].exp ;
            P3[k++].coeff = P1[i++].coeff ;
        }
        else if( P1[i].exp < P2[j].exp ){
            P3[k].exp = P1[j].exp ;
            P3[k++].coeff = P2[j++].coeff ;
        }
    }
    while( i < S1 ){
        P3[k].exp = P1[i].exp ;
        P3[k++].coeff = P1[i++].coeff ;
    }
    while( j < S2 ){
        P3[k].exp = P2[j].exp ;
        P3[k++].coeff = P2[j++].coeff ;
    }
    *S3 = k ;

}

void copyPoly( poly P1[] , poly P2[] , int s ){

    for( int i = 0 ; i < s  ; i++ ){
        P2[i].coeff = P1[i].coeff ;
        P2[i].exp = P1[i].exp ;
    }
}

// P2 = P1 * (c*x^e)

void multerm( poly P[] , poly A[] , int c, int e , int s1  ){
    for( int i = 0 ; i < s1 ; i++ ){
        A[i].exp = P[i].exp + e ;
        A[i].coeff = P[i].coeff * c ;
    }
}

void mulPoly( poly P1[] , poly P2[] , poly P3[] , int s1 , int s2 , int* s3 ){

    poly P4[20] , P5[20] ;
    int s4 , s5 ;
    *s3 = 0 ;
    s4 = s2 ;
    for( int i = 0 ; i < s1 ; i++ ){
        multerm( P2 , P4 , P1[i].coeff , P1[i].exp , s2  ) ;
        addPoly( P3 , P4 , P5 , *s3 , s2 , &s5 ) ;
        copyPoly( P5 , P3 , s5 ) ;
        *s3 = s5 ;
    }
}
```

Output :



```
PS D:\College\DS\Polynomial> .\polynomial
Enter the Terms in poly (in Decreasing Exponents)
 3
Enter the 1 term
Coeffecint : 3
Exponeent : 3
Enter the 2 term
Coeffecint : 2
Exponeent : 2
Enter the 3 term
Coeffecint : 1
Exponeent : 1
Enter the Terms in poly (in Decreasing Exponents)
 3
Enter the 1 term
Coeffecint : 4
Exponeent : 4
Enter the 2 term
Coeffecint : 3
Exponeent : 3
Enter the 3 term
Coeffecint : 2
Exponeent : 2

P1 = 3.0x^3 + 2.0x^2 + 1.0x^1
P2 = 4.0x^4 + 3.0x^3 + 2.0x^2
P1 + P2 = 4.0x^3 + 6.0x^3 + 4.0x^2 + 1.0x^1
P1 * P2 = 12.0x^7 + 17.0x^6 + 16.0x^5 + 7.0x^4 + 2.0x^3
```

Objective:

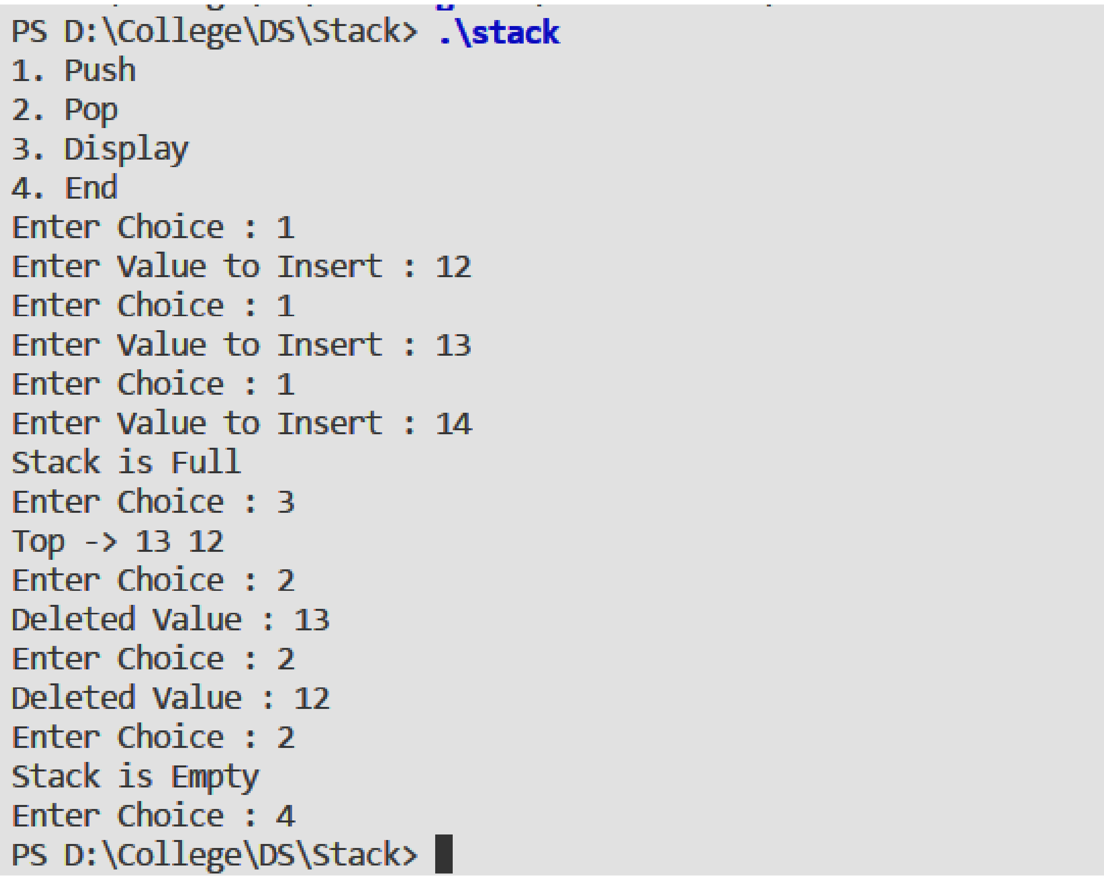
Write a program to implement a **Stack Operations** using a Array as a Stack.

Code :

```
#include <stdio.h>
#define MAXQ 2
typedef struct stack {
    int A[MAXQ] ;
    int top;
} stack ;

void inserts( stack* , int);
int deletes( stack* );
void displays( stack);
void initialize( stack* );

int main(){
    stack s ;
    int ch , n ;
    initialize(&s);

    printf("1. Push \n");
    printf("2. Pop \n");
    printf("3. Display \n");
    printf("4. End \n");

    do{
        printf("Enter Choice : ");
        scanf("%d" , &ch );
        switch(ch) {
            case 1 :
            printf("Enter Value to Insert : ");
            scanf("%d" , &n);
            inserts(&s , n);
            break;
            case 2 :
            n = deletes(&s);
            if(n == -1) break;
            printf("Deleted Value : %d\n" , n );
            break;
            case 3 :
            displays(s);
            break;
        }

    } while ( ch != 4) ;
}

void initialize(stack *S){
    S->top = -1;
}
```

```
void inserts( stack *S , int x ){

    if( S->top == MAXQ -1 ){
        printf("Stack is Full \n");
        return;
    }
    S->A[++S->top] = x ;
}

int deletes( stack *S ){

    int x ;
    if( S->top == -1 ){
        printf("Stack is Empty \n");
        return(-1);
    }
    x = S->A[S->top--] ;
    return(x);

}

void displays( stack S ){
    printf("Top -> ");
    for( int i = S.top ; i >= 0 ; i--){
        printf("%d " , S.A[i] );
    }
    printf("\n");
}
```

Output :



```
PS D:\College\DS\Stack> .\stack
1. Push
2. Pop
3. Display
4. End
Enter Choice : 1
Enter Value to Insert : 12
Enter Choice : 1
Enter Value to Insert : 13
Enter Choice : 1
Enter Value to Insert : 14
Stack is Full
Enter Choice : 3
Top -> 13 12
Enter Choice : 2
Deleted Value : 13
Enter Choice : 2
Deleted Value : 12
Enter Choice : 2
Stack is Empty
Enter Choice : 4
PS D:\College\DS\Stack> █
```

Objective:

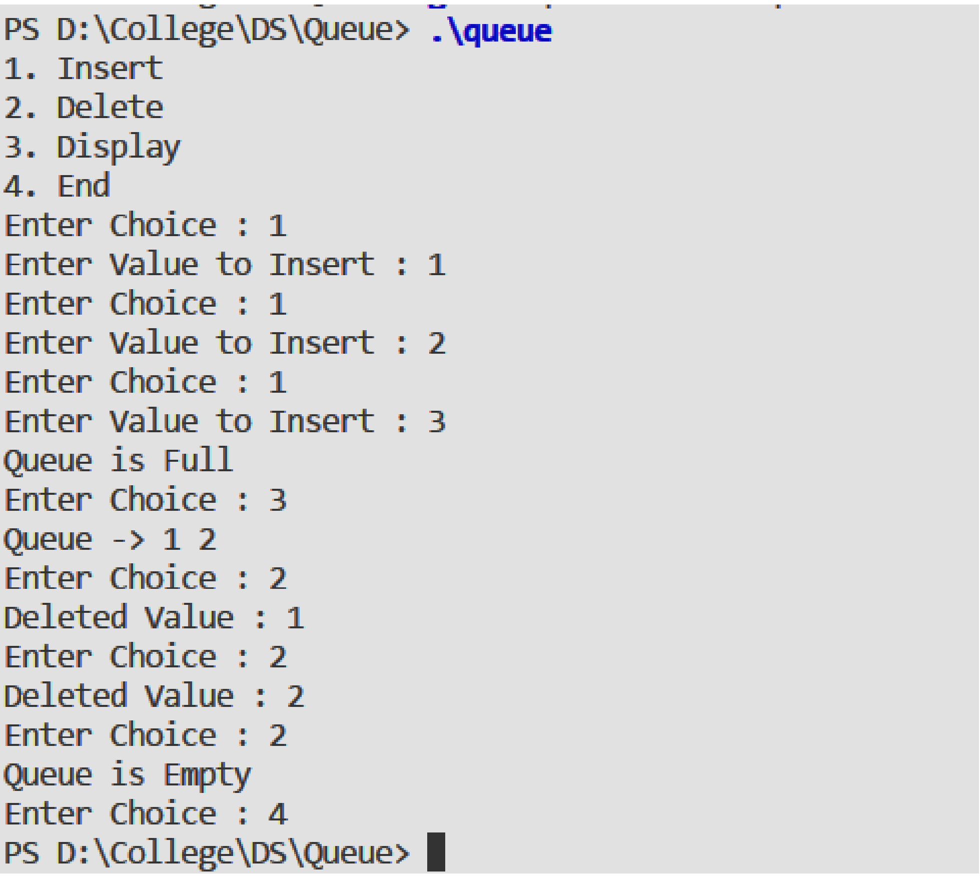
Write a program to implement a **Queue Operations** using a Array as a Queue.

Code :

```
#include <stdio.h>
#define MAXQ 2

typedef struct queue {
    int A[MAXQ] ;
    int front , rear ;
} queue ;

void insertq( queue* , int);
int deleteq( queue* );
void displayq( queue);
void initialize( queue* );

int main(){
    queue q ;
    int ch , n ;
    initialize(&q);
    printf("1. Insert \n");
    printf("2. Delete \n");
    printf("3. Display \n");
    printf("4. End \n");
    do{
        printf("Enter Choice : ");
        scanf("%d" , &ch );
    switch(ch) {
            case 1 :
            printf("Enter Value to Insert : ");
            scanf("%d" , &n);
            insertq(&q , n);
            break;
            case 2 :
            n = deleteq(&q);
            if(n == -1) break;
            printf("Deleted Value : %d \n" , n );
            break;
            case 3 :
            displayq(q);
            break;
        }
    } while ( ch != 4) ;
}

void initialize(queue *Q){
    Q->front = 0;
    Q->rear = 0;
}
```

```
void insertq( queue *Q , int x ){

    if( Q->rear == MAXQ ){
        printf("Queue is Full \n");
        return;
    }
    Q->A[Q->rear++] = x ;
}

int deleteq( queue *Q ){
    int x ;
    if( Q->front == Q->rear ){
        printf("Queue is Empty \n");
        return(-1);
    }
    x = Q->A[Q->front++] ;
    return(x);
}

void displayq( queue Q ){
    printf("Queue -> ");
    for(int i = Q.front ; i < Q.rear ; i++ ){
        printf("%d " , Q.A[i]);
    }
    printf("\n");

}
```

Output :



```
PS D:\College\DS\Queue> .\queue
1. Insert
2. Delete
3. Display
4. End
Enter Choice : 1
Enter Value to Insert : 1
Enter Choice : 1
Enter Value to Insert : 2
Enter Choice : 1
Enter Value to Insert : 3
Queue is Full
Enter Choice : 3
Queue -> 1 2
Enter Choice : 2
Deleted Value : 1
Enter Choice : 2
Deleted Value : 2
Enter Choice : 2
Queue is Empty
Enter Choice : 4
PS D:\College\DS\Queue>
```

Objective:

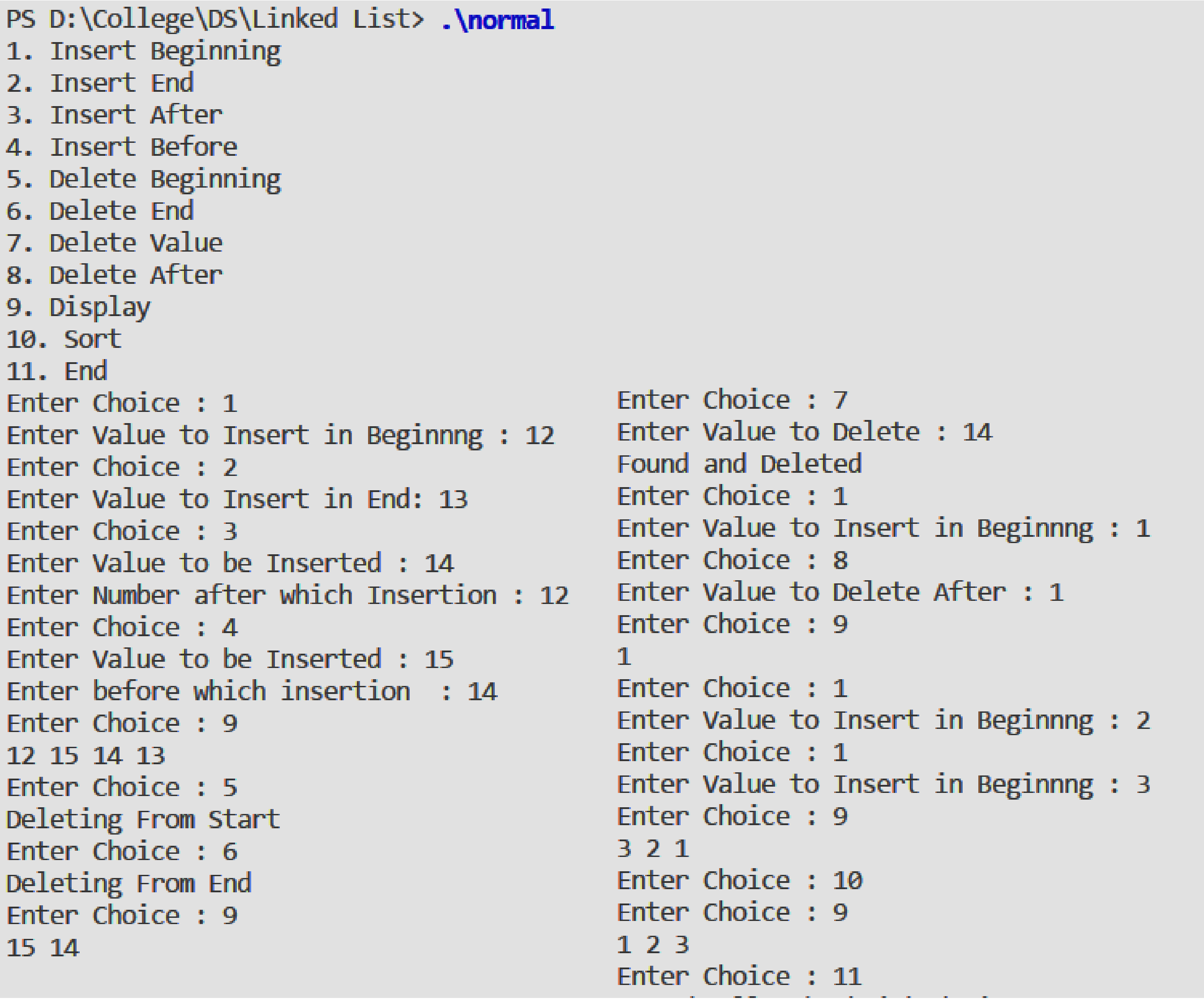
To implement Linked List and perform various insertion and deletion operations on it.

Code :

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

typedef struct node {
    int value ;
    struct node* next ;
} node ;

node* getnewnode() ;
node* insertbeg( node* , int);
node* insertend( node* , int);
node* insertafter( node* , int , int);
node* insertbefore( node* , int , int);
node* deletebeg( node* );
node* deleteend( node* );
node* deletevalue( node*, int);
node* deleteafter( node*, int );;
node* sortll( node* );;

void displayll( node* );

int main(){

    int ch , n , temp ;
    node* start = NULL ;
    printf("1. Insert Beginning \n");
    printf("2. Insert End \n");
    printf("3. Insert After \n");
    printf("4. Insert Before \n");
    printf("5. Delete Beginning \n");
    printf("6. Delete End \n");
    printf("7. Delete Value \n");
    printf("8. Delete After \n");
    printf("9. Display \n");
    printf("10. Sort \n");
    printf("11. End \n");

    do{
        printf("Enter Choice : ");
        scanf("%d" , &ch );
        switch(ch) {
            case 1 :
                printf("Enter Value to Insert in Beginnng : ");
                scanf("%d" , &n);
                start  = insertbeg(start , n);
                break;
            case 2 :
                printf("Enter Value to Insert in End: ");
                scanf("%d" , &n);
                start  = insertend(start , n);
                break;
            case 3 :
                printf("Enter Value to be Inserted : ");
                scanf("%d" , &n);
                printf("Enter Number after which Insertion : ");
                scanf("%d" , &temp);
                start  = insertafter(start , n ,temp );
                break;
            case 4 :
                printf("Enter Value to be Inserted : ");
                scanf("%d" , &n);
                printf("Enter before which insertion  : ");
                scanf("%d" , &temp);
                start  = insertbefore(start , n , temp);
                break;
            case 5 :
                printf("Deleting From Start\n");
                start = deletebeg(start);
                break;
            case 6 :
                printf("Deleting From End\n");
                start = deleteend(start);
                break;
            case 7 :
                printf("Enter Value to Delete : ");
                scanf("%d" , &n);
                start = deletevalue(start , n);
                break;
            case 8 :
                printf("Enter Value to Delete After : ");
                scanf("%d" , &n);
                start = deleteafter(start , n);
                break;
            case 9 :
                displayll(start);
                break;
            case 10 :
                start = sortll(start);
                break;
        }

    } while ( ch != 11) ;

}

node* getnewnode(){
    node* new = malloc(sizeof( node ) );
    return new;
}

node* insertbeg( node* start  , int x){

    node* q = getnewnode() ;
    q->value = x ;
    q->next = start ;
    start = q ;
    return start ;

}

node* insertend( node* start  , int x){

    node* q = getnewnode() ;
    q->value = x ;
    node* p = start ;

    if( start == NULL ){
        start = q ;
        start->next = NULL ;
        return start;
    }
    while(  p->next != NULL ){
        p = p->next ;
    }
    q->next = p->next ;
    p->next = q ;

    return start ;

}

node* insertafter( node* start  , int x , int y){

    node* q = getnewnode() ;
    q->value = x ;
    node* p = start ;

    if( start == NULL ){
        printf("Empty Linked List. Inserting at Beginning\n");
        start = q ;
        start->next = NULL ;
        return start;
    }

    while(  p->next != NULL && p->value != y ){
        p = p->next ;
    }

    if(p->value != y ){
        printf("Not Found. Inserting at End\n");
    }

    q->next = p->next ;
    p->next = q ;

    return start ;

}
```

```
node* insertbefore( node* start  , int x , int y ){

    node* q = getnewnode() ;
    q->value = x ;
    node* p = start ;

    if( start == NULL ){
        printf("Empty Linked List. Inserting at Beginning\n");
        start = q ;
        start->next = NULL ;
        return start;
    }

    if( start->value == y ){
        q->next = start ;
        start = q ;
    } else {
        while(  p->next != NULL && p->next->value != y ){
            p = p->next ;
        }

        if(p->next == NULL ){
            printf("Not Found. Inserting at End\n");
        }

        q->next = p->next ;
        p->next = q ;

    }

    return start ;

}


node* deletebeg( node* start){

    if( start == NULL ){
        printf("Empty\n");
        return start;
    }

    node* q = start ;
    start = start->next ;
    free(q) ;

    return start ;

}

node* deleteend( node* start){

    node* q  ;
    node* p = start ;

    if( start == NULL ){
        printf("Empty\n");
        return start;
    }

    while(  p->next->next != NULL ){
        p = p->next ;
    }

    q = p->next ;
    p->next = p->next->next ;
    free(q) ;

    return start ;

}


node* deletevalue( node* start  , int x){


    if( start == NULL ){
        printf("List is Empty\n");
        return start;
    }

    node* q ;

    if( start->value == x ){
        q = start ;
        start = start->next ;
        return start ;
    }

    node* p = start;

    while( p->next != NULL ){
        if(p->next->value == x){
            printf("Found and Deleted\n");
            q = p->next ;
            p->next = q->next ;
            free(q);
            return(start) ;
        }
        p = p->next ;
    }

    printf("Not Found\n") ;

    return start ;

}

node* deleteafter( node* start  , int x){


    if( start == NULL ){
        printf("List is Empty\n");
        return start;
    }

    node* q ;

    if( start->value == x ){
        start->next = start->next->next ;
        return start ;
    }

    node* p = start;

    while( p->next != NULL ){
        if(p->value == x){
            printf("Found and Deleted\n");
            q = p->next ;
            p->next = q->next ;
            free(q);
            return(start) ;
        }
        p = p->next ;
    }

    printf("Not Found\n") ;

    return start ;

}


void displayll(node* start){

    while(start != NULL ){

        printf("%d " , start->value);
        start = start->next ;

    }
    printf("\n") ;

}

node* sortll(node* start){

    node *q , *p ;
    int temp ;

    if(start == NULL ){
        return start ;
    }

    p = start ;
    while(p->next != NULL ){

        q = p->next ;
        while( q != NULL ){

            if ( p->value > q->value ){

                int temp = p->value ;
                p->value = q->value ;
                q->value = temp ;
            }
            q = q->next ;
        }
        p = p->next ;
    }
    return start ;
}
```

Output :



```
PS D:\College\DS\Linked List> .\normal
1. Insert Beginning
2. Insert End
3. Insert After
4. Insert Before
5. Delete Beginning
6. Delete End
7. Delete Value
8. Delete After
9. Display
10. Sort
11. End
Enter Choice : 1
Enter Value to Insert in Beginnng : 12
Enter Choice : 2
Enter Value to Insert in End: 13
Enter Choice : 3
Enter Value to be Inserted : 14
Enter Number after which Insertion : 12
Enter Choice : 4
Enter Value to be Inserted : 15
Enter before which insertion  : 14
Enter Choice : 9
12 15 14 13
Enter Choice : 5
Deleting From Start
Enter Choice : 6
Deleting From End
Enter Choice : 9
15 14
```

```
Enter Choice : 7
Enter Value to Delete : 14
Found and Deleted
Enter Choice : 1
Enter Value to Insert in Beginnng : 1
Enter Choice : 8
Enter Value to Delete After : 1
Enter Choice : 9
1
Enter Choice : 1
Enter Value to Insert in Beginnng : 2
Enter Choice : 1
Enter Value to Insert in Beginnng : 3
Enter Choice : 9
3 2 1
Enter Choice : 10
Enter Choice : 9
1 2 3
Enter Choice : 11
```

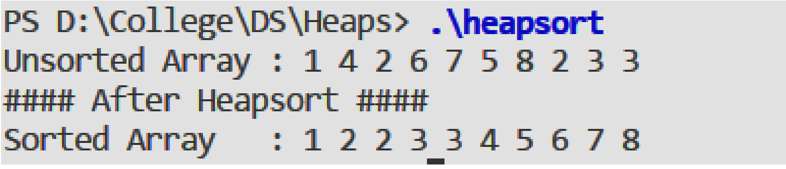
Objective:

To implement Heap Sort Algorithm on a unsorted array.

Code :

```
#include <stdio.h>
#define LENGTH 10

int heapSize;

void Heapify(int* A, int i)
{
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    int largest;

    if(l <= heapSize && A[l] > A[i]) {
        largest = l;
    }
    else {
        largest = i;
    }
    if(r <= heapSize && A[r] > A[largest]) {
        largest = r;
    }

    if(largest != i) {
        int temp = A[i];
        A[i] = A[largest];
        A[largest] = temp;
        Heapify(A, largest);
    }
}

void BuildHeap(int* A)
{
    heapSize = LENGTH - 1;
    for(int i = (LENGTH - 1) / 2; i >= 0; i--)
    Heapify(A, i);
}

void HeapSort(int* A)
{
    BuildHeap(A);
    int i;
    for(i = LENGTH - 1; i > 0; i--)
    {
        int temp = A[heapSize];
        A[heapSize] = A[0];
        A[0] = temp;
        heapSize--;
        Heapify(A, 0);
    }
}

int main()
{
    int tab[LENGTH] = {1,4,2,6,7,5,8,2,3,3};
    printf("Unsorted Array : ");
    for(int i = 0; i < LENGTH; i++){
        printf("%d ",tab[i]);
    }
    printf("\n#### After Heapsort #### \n");
    HeapSort(tab);
    printf("Sorted Array   : ");
    for(int i = 0; i < LENGTH; i++){
        printf("%d ",tab[i]);
    }
    return 0;
}
```

Output :



```
PS D:\College\DS\Heaps> .\heapsort
Unsorted Array : 1 4 2 6 7 5 8 2 3 3
#### After Heapsort ####
Sorted Array   : 1 2 2 3 3 4 5 6 7 8
```

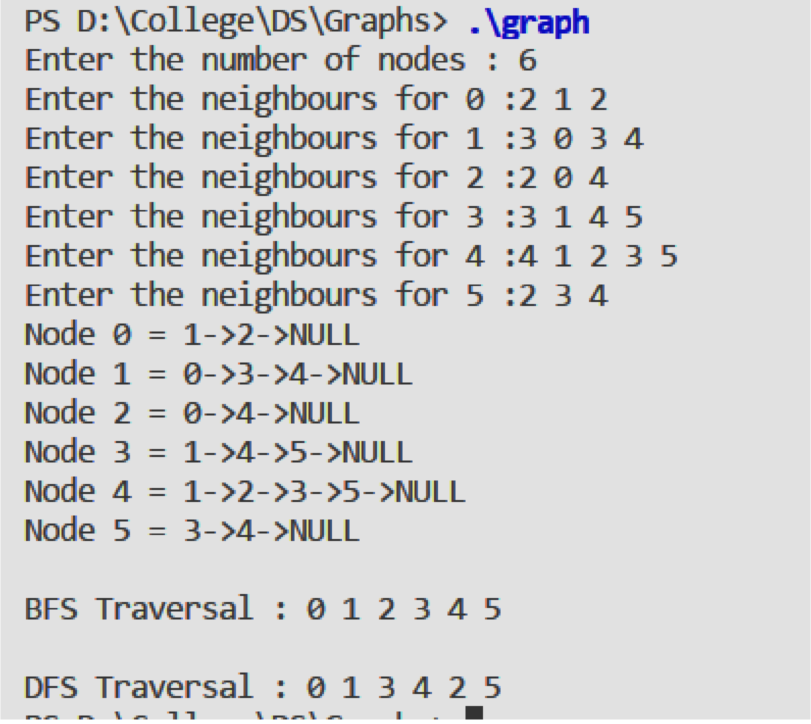
Objective:

To implement a Graph using Adjacency List and perform Breadth First and Depth First Search

Code :

```
#include<stdio.h>
#include<stdlib.h>
struct node {
    int data;
    int status; struct node *next;
};

struct node *root=NULL;

void createGraph(struct node *adj[10],int n) {
    struct node *last;
    int m,val,d;
    for(int i=0; i<n; i++) {
        printf("Enter the neighbours for %d :",i);
        scanf("%d",&m);
        for(int j=0; j<m; j++) {
            scanf("%d",&val);
            struct node *temp= malloc(sizeof(struct node));
            temp->data=val;
            temp->next=NULL;
            if(adj[i]==NULL) {
                adj[i]=temp;
            } else {
                last->next=temp;
            }
            last=temp;
        }
    }
}

void Display(struct node *adj[10],int n) {
    for(int i=0; i<n; i++) {
        struct node *temp; temp=adj[i];
        printf("Node %d = ",i); while(temp!=NULL)
        {
            printf("%d->",temp->data); temp=temp->next;
        }
        printf("NULL\n");
    }
}

void bfs(struct node *adj[10],int n,int *parent) {
    int queue[n];
    int visited[n];
    for(int i=0; i<n; i++) {
        visited[i]=0;
    }
    int front,rear;
    front=rear=-1;
    queue[rear++]=0;
    visited[0]=1;
    parent[0]=-1;

    while(rear!=front) {
        int x=queue[front++];
        printf("%d ",x);
        struct node *temp=adj[x];
        while(temp!=NULL) {
            if(visited[temp->data]==0) {
                queue[rear++]=temp->data;
                visited[temp->data]=1;
                parent[temp->data]=x;
            }
            temp=temp->next;
        }
    }
    printf("\n");

}
void dfs(struct node *adj[10],int n,int *visited) {
    struct node *temp;
    visited[n]=1;
    printf("%d ",n);
    temp=adj[n];
    while(temp!=NULL) {
        if(visited[temp->data]==0) {
            dfs(adj,temp->data,visited);
        }
        else {
            temp=temp->next;
        }
    }
}
int main() {
    int v;
    printf("Enter the number of nodes : ");
    scanf("%d",&v);
    struct node *adj[10];
    for(int i=0; i<v; i++) {
        adj[i]=NULL;
    }
    createGraph(adj,v);
    int p[v];

    Display(adj,v);

    printf("\nBFS Traversal : ");
    bfs(adj,v,p);

    int visited[10]= {0};
    printf("\n");
    printf("DFS Traversal : ");
    dfs(adj,0,visited);
    return 0;
}
```

Output :



```
PS D:\College\DS\Graphs> .\graph
Enter the number of nodes : 6
Enter the neighbours for 0 :2 1 2
Enter the neighbours for 1 :3 0 3 4
Enter the neighbours for 2 :2 0 4
Enter the neighbours for 3 :3 1 4 5
Enter the neighbours for 4 :4 1 2 3 5
Enter the neighbours for 5 :2 3 4
Node 0 = 1->2->NULL
Node 1 = 0->3->4->NULL
Node 2 = 0->4->NULL
Node 3 = 1->4->5->NULL
Node 4 = 1->2->3->5->NULL
Node 5 = 3->4->NULL

BFS Traversal : 0 1 2 3 4 5

DFS Traversal : 0 1 3 4 2 5
PS D:\College\DS\Graphs>
```

Objective:

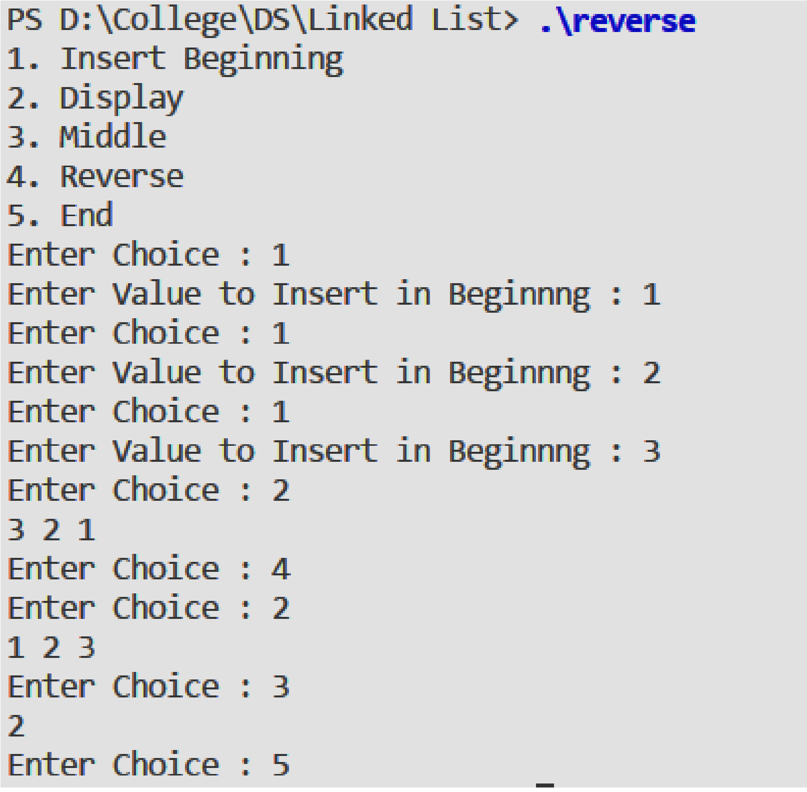
To perform following operation on linked list :

- Reverse the Given Linked List
- Find the Middle Element in Linked List

Code :

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

typedef struct node {
    int value ;
    struct node* next ;
} node ;

node* getnewnode() ;
node* insertbeg( node* , int);
void displayll( node* );
node* reverse( node* );
int middle( node* );

int main(){

    int ch , n , temp ;
    node* start = NULL ;

    printf("1. Insert Beginning \n");
    printf("2. Display \n");
    printf("3. Middle \n");
    printf("4. Reverse \n");
    printf("5. End \n");

    do{
        printf("Enter Choice : ");
        scanf("%d" , &ch );

        switch(ch) {
            case 1 :
                printf("Enter Value to Insert in Beginnng : ");
                scanf("%d" , &n);
                start  = insertbeg(start , n);
                break;
            case 2 :
                displayll(start);
                break;
            case 3 :
                temp = middle(start);
                printf("%d\n" , temp );
            case 4 :
                start = reverse(start);
                break;
        }

    } while ( ch != 5) ;

}


node* getnewnode(){

    node* new = malloc(sizeof( node ) );
    return new;

}

node* insertbeg( node* start  , int x){

    node* q = getnewnode() ;
    q->value = x ;

    q->next = start ;
    start = q ;

    return start ;

}

void displayll(node* start){

    while(start != NULL ){

        printf("%d " , start->value);
        start = start->next ;

    }
    printf("\n") ;

}

node* reverse(node* start){

    node *q , *p ;

    if(start == NULL || start->next == NULL ){
        return start ;
    }

    p = start->next ;
    start->next =NULL ;

    while(p != NULL ){

        q = p->next ;
        p->next = start ;
        start = p ;
        p = q ;
    }
    return start ;
}


int middle(node* start){
    node *p , *q ;
    p = start ;
    q = start ;
    while(p != NULL && p->next != NULL ){
        p = p->next->next ;
        q = q->next ;
    }

    return q->value ;
}
```

Output :



```
PS D:\College\DS\Linked List> .\reverse
1. Insert Beginning
2. Display
3. Middle
4. Reverse
5. End
Enter Choice : 1
Enter Value to Insert in Beginnng : 1
Enter Choice : 1
Enter Value to Insert in Beginnng : 2
Enter Choice : 1
Enter Value to Insert in Beginnng : 3
Enter Choice : 2
3 2 1
Enter Choice : 4
Enter Choice : 2
1 2 3
Enter Choice : 3
2
Enter Choice : 5
```