- Topics

1. List ADT
2. Linked list
3. Doubly
4. Circular
5. Sparse
6. Poly
7. St. Joseph

- List ADT

Program :-

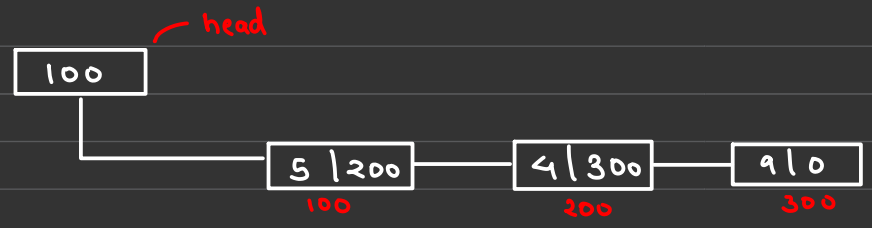
```
#include <stdio.h>
#include <stdlib>

Struct Node {                          // define structure //
    int data;
    struct Node *Nesct;
};

typeof struct Node* ptrtoNode;
typeof ptrtoNode LIST;
typeof ptrtoNode POSITION;

int Isemp (LIST L) {                   // Check it empty !
    return L->NEXT == NULL;
}

int Islast (POSITION P) {              // Check if last!!
    return P->NEXT == NULL;
}

LIST Create () {                       // Create //
    LIST L;
  L = (struct Node *) malloc (sizeof (Struct Node));
if (L == NULL) {
    printf (" error ");
    exit (-1);
}
```

```
    else {
      L -> data = -1          // initialize !
      L -> NEXT = NULL
  }
    return L;
}

void insert (int x, POSITION P) {
    ptrtoNode temp;
   temp = (struct Node *) malloc (sizeof (struct Node));
    if (temp == NULL) {
        printf ("error");                    // insert !!
        exit (-1);
    }
    else {
      temp -> data = x;
      temp -> NEXT = P -> NEXT
      P->NEXT = temp;
  }
}

POSITION findprev (int x, LISTL) {          → Position of x elem
    POSITION P = L;
  while (P->NEXT != NULL && P->NEXT->DATA != x) {
          P = P-> NEXT ;
    }
    return P;
}

void del (int x, LISTL){
    POSITION P, tempcell;
   P = findprev(x, L);
   if (!Islast (P)){
       tempcell = P-> NEXT ;
       P-> NEXT = tempcell-> NEXT;
       free (tempcell);
   }
}
```

```
void empty (LIST L){
   if (L == NULL){
        printf ("No list");
   }
   else {
      while (!isemp (L)) {
         del ( L->NEXT -> DATA , L);
     }
 }
}

POSITION find (int x, LIST L){
    POSITION temp = L;
 while ( temp != NULL) {
     if (temp -> data == x){
        return temp;
     }
     temp = temp->NEXT
  }
  return NULL
}

void disp ( LIST L) {
   L = L-NEXT;
  while (L != NULL){
     printf ("%d," , L->data );
      L = L-> NEXT ;
   }
   return ("Null \n");
}

voil delL (LIST L) {
   empty (L);
   free (L);
   L= NULL;
   return L;
  }
```

```
int main () {

  LIST   Sum =  creak ();

  insert (10, sum);
  insert (20,  "
     "    30,  "
     "    40,  "

  disp ( sum);

  del (20, sum);
  disp (sum);

  findprer ( 30, sum);

  find (40, sum);

  Sum = delL(sum);
  return 0;
}
```

- Linked list



- Creating and printing

Program :-

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

Struct node {
    int data ;
    Struct node * next ;
};
Struct node *next , * newnde , * temp;
```

```
int main () {

  head = 0 ;
  int choice ;

        do        {
newnode = (struct node*) malloc ( sizeof (struct node));

    printf (" Enter data ");
    scanf ( " %d ", & newnde -> data );
     newnde -> next = 0

                                          // Creating &
                                             adding //
if (head == 0 ) {
   head = temp = newnde ; }
  else {
    temp -> next = newnode;
     temp = newnde }

printf (" Do yow want to conti. (0,1) ? ");
   scanf ( " %d ", & choice);

} while (choice == 1);

temp = head
while (temp != 0) {
     printf (" %d", temp -> data);     // printing //
       temp = temp -> next;
}
return 0; }
```

- Insertion at beginning

Program :-

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

 struct node {
    int data ;
    struct node * next ;
}
Struct node *newnode , * temp ;
```

```
struct node * insertatbeg ( struct node * head ) {

  newnode = (struct node*) malloc (sizeof (struct node));
     printf (" Enter data ");
     scanf (" %d", & newnode -> data);

  newnode -> next = head;
     head = newnode;

  return  newnode;
}

int main () {
    struct node * head = NULL
       int choice ;

   do {
        head = insertatbeg (head);
   printf (" Do you want to insert more (0,1) ? ");
   scanf ( "%d" & choice );
} while (choice == 1 );

temp = head
while (temp != 0) {
      printf (" %d", temp -> data);     // printing //
        temp = temp -> next;
 }
 return 0; }
```

- Insert at end

Program :- (only function, coz rest everything same

```
struct node * insertatend (struct node * head ) {
  newnode = (struct node*) malloc (sizeof (struct node));
     printf (" Enter data ");
     scanf (" %d", & newnode -> data);

  newnode -> next = 0;
```

```
if (head == 0){
    return newnode;
}
else {
    temp = head;
    while (temp -> next != 0) {
        temp = temp -> next;
    }
    temp -> next = newnode;
}
return head;
}
```

- Insertion at any position

Program :-

```
int pos, i

struct node * insertatpos (struct node * head) {

newnode = (struct node*) malloc (sizeof (struct node));

        printf (" Enter pos ");
        scanf ("%d", &pos);
```

```
if (pos == 0 || head == NULL) {
    printf("Enter data: ");
    scanf("%d", &newnode->data);
    newnode->next = head;
    head = newnode;
    return head;
}

// Traverse to the node just before the insertion point
while (i < pos - 1 && temp != NULL) {
    temp = temp->next;
    i++;
}

if (temp == NULL) {
    printf("Position out of bounds\n");
    free(newnode);
    return head;
}
```

```
printf (" Enter data");
scanf ("%d" &newnode->data );

newnode -> next = temp -> next
  temp -> next = newnode;

return head;
}
```

- deleting from beginning

Program :-

```
void delfrmbeg ( ) {
  struct node * temp;

    if ( head == NULL) {
      printf (" No list");
    }
    else {
       temp = head
      head = head -> next
      free (temp) ;
   }
```

common for all three

```
struct node {
    int data;
    struct node * next ;
}
struct node *head , *temp;
```

- delete from end

Program :-

```
void delfrmend (){
    struct node * prevnode;
      temp = head
  while ( temp -> next != 0 ) {
      prevnode = temp ;
      temp = temp -> next;
   }
it ( temp == head ) {
    head = 0;
 }
else {
   prevnode -> next = 0
 }
 free (temp);

}
```

- delete at any position

Program :-

```
void delfrmpos () {
    struct node * nextnode
    int pos ,i=1 ;
   printf (" Enter pos ");
   scanf (" %d ", & pos");

  while (i< pos-1){
    temp = temp -> next ;
     i++ ;
  }
  next node = temp -> next
   temp-> next = next node -> next
    free (nextnode );
  }
```

SRNPS2858J

- To get length

```
void getlen ()
  int count =0;
  Struct node * temp ;
     temp= head;
  while (temp != 0 ) {
       count ++ ;
       temp = temp -> next;
    }
 print (" leng is : %d ", count);

}
```

- To Reverse

Program :-

```
void rev () {
 struct node * prevnode , * curnode , * nxtnode ;

  prevnode = 0;
 curnode = nxtnode = head ;
```

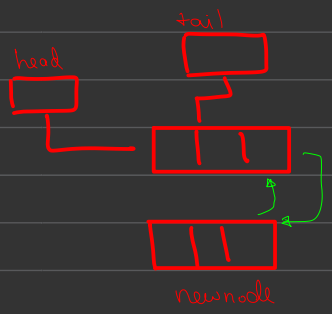
```
while (nxtnode != 0) {
    nxtnode = nxtnode -> next
    currnode -> next = prevnode
    prevnode = currnode ;
    currnod = nxtnode ;
  }
  head = prevnode ;
}
```

- **Doubly linked list**

- **Creation**

Program :-

```
struct node {
   int data ;
   struct node * next ;
   struct node * prev ;
}
struct node *newnode , * temp , * head , * tail

void create() {
 int choice ; head = 0
  do {
     newnode = (struct node *) malloc (sizeof (struct node));
       printf (" Enter data ");
       scanf ("%d", &newnode -> data);
     newnode -> prev = 0
     newnode -> next = 0

   if (head == 0) {
      head = tail = newnode ;
   }
   else {

     tail -> next = newnode
     newnode -> prev = tail
      tail = newnode
  }
```



- Insertion at start

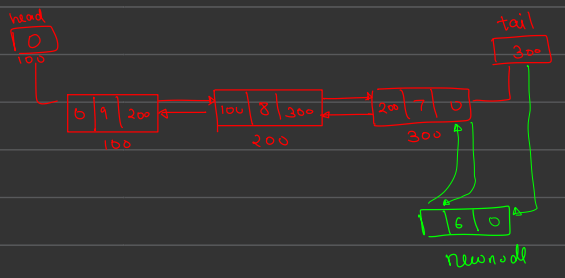
Program :-

```
void insertatbeg() {

newnode = (struct node *) malloc (sizeof (struct node));
  printf (" Enter data ");
  scanf ("%d", &newnode -> data);
newnode -> prev = 0
newnode -> next = 0

   head -> prev = newnode;
   newnode -> next = head;
      head = newnode;
}
```

- Insert at end

Program :-

```
void insertatend() {

newnode = (struct node *) malloc (sizeof (struct node));
  printf (" Enter data ");
  scanf ("%d", &newnode -> data);
newnode -> prev = 0
newnode -> next = 0

   tail -> next = newnode
   newnode -> tail = tail
      tail = newnode
}
```

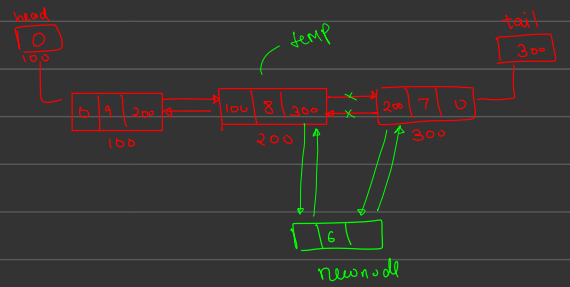


- Insert at any position

Program :-

```
void insertatpos () {
    int i=1 , pos;
   if (pos > count) {
      printf (" invalid list ");
   }
```

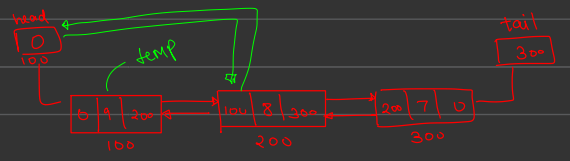
```
else if ( pos == 1 ){
    void insert at beg();
}

else {

newnode = (struct node *) malloc ( sizeof (struct node));
printf (" Enter data ");
scanf ("%d", &newnode -> data);

while ( i < pos - 1) {
    temp = temp -> next ;
    i++ ;
}

newnode -> prev = temp
newnode -> next = temp -> next;
temp -> next = newnode
newnode -> next -> prev = newnode

}
```



- delete from beg

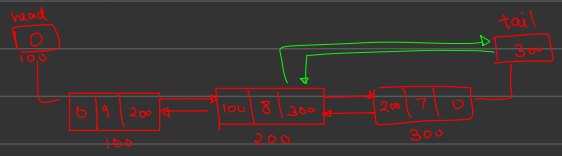
Program :-

```
void delfrmbeg () {

    if (head == 0){
        printf (" empty list ");
    }
    else {
        temp = head;
        head = head -> next
        head -> prev = 0
        free (temp);

}
```



- delete from end

- Program

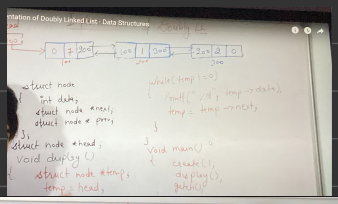


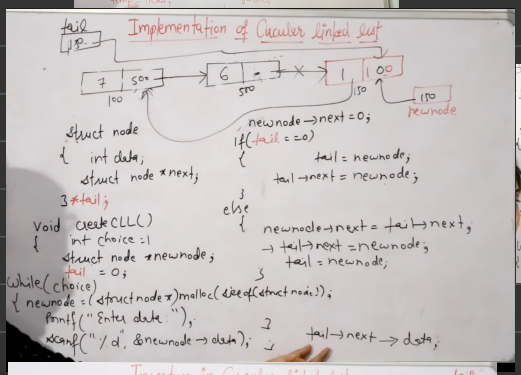

```
void delfromend (){
      temp = tail
      tail -> prev -> next = 0
       tail = tail -> prev
       free (temp);
}
```

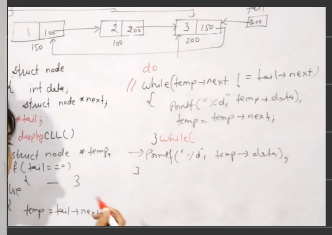
- delete from any position

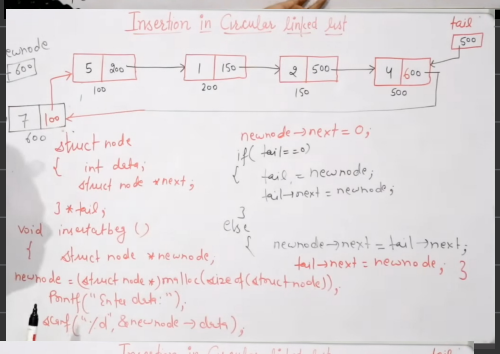
Program :-

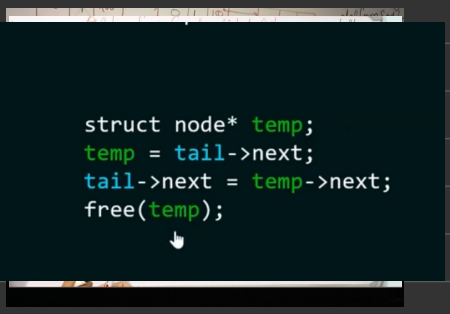
```
void delfrmpos (){
    int i=1 , pos;
   if (pos > count) {
     printf ("invalid list");
   }

else {

    while ( i< pos) {
      temp = temp -> next;
       i++;
   }
   temp -> prev -> next  = temp -> prev;
   temp -> next -> prev = temp -> next;
}
```

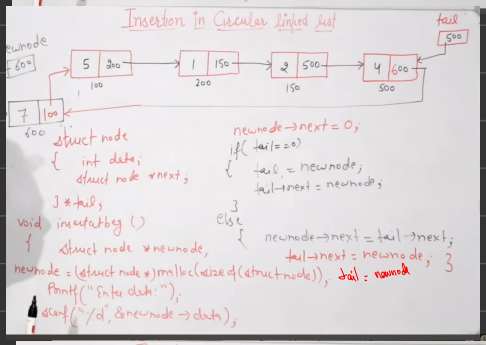



Implementation of Circular Linked List
struct node
int data;
struct node *next;
*tail;
Void createCLLC()
int choice =1
struct node *newnode;
tail = 0;
while(choice)
newnode = (struct node*)malloc(sizeof(struct node));
Printf("Enter data");
scanf("%d", &newnode->data);
newnode->next = 0;
if(tail == 0)
tail = newnode;
tail->next = newnode;
else
newnode->next = tail->next;
tail->next = newnode;
tail = newnode;
tail->next->data;

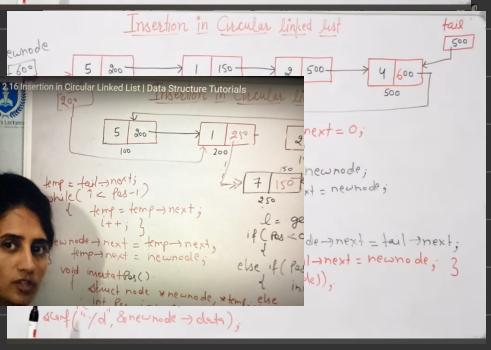



Insertion in Circular Linked List
struct node
int data;
struct node *next;
*tail;
Void insertatbeg()
struct node *newnode,
newnode = (struct node*)malloc(sizeof(struct node));
Printf("Enter data:");
scanf("%d", &newnode->data);
newnode->next = 0;
if(tail==0)
tail = newnode;
tail->next = newnode;
else
newnode->next = tail->next;
tail->next = newnode;


struct node* temp;
temp = tail->next;
tail->next = temp->next;
free(temp);

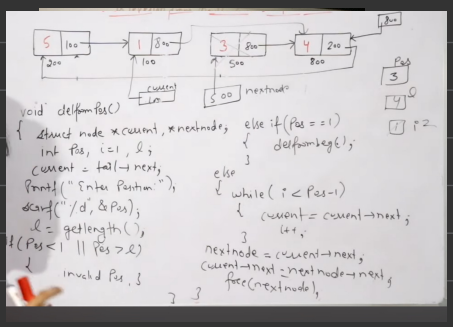

Insertion in Circular Linked List
struct node
int data;
struct node *next;
*tail;
Void insertatbeg()
struct node *newnode,
newnode = (struct node*)malloc(sizeof(struct node));
Printf("Enter data:");
scanf("%d", &newnode->data);
newnode->next = 0;
if(tail==0)
tail = newnode;
tail->next = newnode;
else
newnode->next = tail->next;
tail->next = newnode;


2000
temp
temp->next = tail->next;
free(tail);
tail = temp;


Insertion in Circular Linked List
struct node
int data;
struct node *next;
*tail;
Void insertatbeg()
struct node *newnode,
newnode = (struct node*)malloc(sizeof(struct node));
Printf("Enter data:");
scanf("%d", &newnode->data);
newnode->next = 0;
if(tail==0)
tail = newnode;
tail->next = newnode;
else
newnode->next = tail->next;
tail->next = newnode;

Insertion in Circular Linked List
struct node
int data;
struct node *next;
*tail;
Void insertatbeg()
struct node *newnode,
newnode = (struct node*)malloc(sizeof(struct node));
Printf("Enter data:");
scanf("%d", &newnode->data);
newnode->next = 0;
if(tail==0)
tail = newnode;
tail->next = newnode;
else
newnode->next = tail->next;
tail->next = newnode;

2.16 Insertion in Circular Linked List | Data Structure Tutorials
Insertion in Circular Linked List
struct node
int data;
struct node *next;
*tail;
Void insertatbeg()
struct node *newnode,
newnode = (struct node*)malloc(sizeof(struct node));
Printf("Enter data:");
scanf("%d", &newnode->data);
newnode->next = 0;
if(tail==0)
tail = newnode;
tail->next = newnode;
else
newnode->next = tail->next;
tail->next = newnode;


void delfromPos()
struct node *current, *nextnode,
int Pos, i=1, l,
current = tail->next,
Printf("Enter Position:");
scanf("%d", &Pos);
l = getlength();
if(Pos<1 || Pos>l)
Invalid Pos.
else if(Pos==1)
delfrombeg();
else
while(i<Pos-1)
current = current->next;
i++;
nextnode = current->next,
current->next = nextnode->next;
free(nextnode);

- Sparse Matrix

1. Using Arrays

Program :-

```
#include <stdio.h>

int main () {

   int sparsem [4] [5] = {

          { 0 , 0, 3, 0, 4 },
          { 0, 0, 5 , 7, 0 },
          { 0, 0, 0, 0, 0 },
          { 0, 2 , 6 , 0 , 0 }
      };

int   size  = 0;

   for ( int i = 0 ;  i < 4  ; i++){
         for (int j = 0 ; j < 8  ; j++ ){

              if ( sparsem [i ][j ] != 0){
                     size ++;
              }
         }
    }

   int   compactm [3][size];
   int   k=0

 for ( int i = 0 ;   i <4 ; i++) {
         for  ( int j=0 ; j <5   ; j++) {

              if ( sparsem [i][j]  != 0 ) {

                   compactm [0][k]  = i
                   compactm [1 ][k] = j
                   compact m [2][k]= sparsem [i][j];
                  k ++  ;
              }
         }
 }
```

```
    printf (" Compact m :");
     for (int i=0 ; i<3  ; i++) {
         for (int j=0 ; j<size ; j++) {

             printf ("%d", compact m[i][i]);
         }
         print ("\n");
  }
  return 0;
}
```

2. Using Linked list

Program :-

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

 Struct node {

     int row;
     int col;
     int val;
    Struct node * next;
};
Struct node * head , * temp , * newnode ;

struct node * create ( int row , int col, int val ) {

     newnode = malloc .....

    newnode -> row = row ;
        "        col  = c  ;
        "        val  = val;
        "    -> next  = NULL;

  return  newnode ;
}

Struct node * insert ( head , row , col , val ) {
     newnode = create ( rol , col, val);
if ( head == 0) {
     return newnode;
```

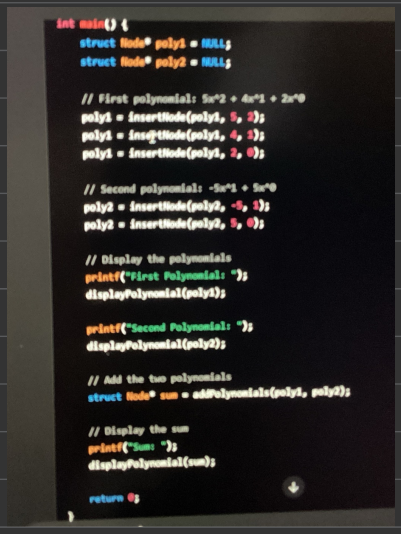
```
    }
  else {
      temp = head;
    while ( temp -> next != 0){
        temp = temp -> next
    }
    temp -> next = newnode
     return head;
  }
}

void disp ( head ) {
     temp = head
     printf ( " Row \t Col \t Val \n");
      while ( temp != 0) {
        printf(" %d \t %d \t %d \n " , & temp->row , col, va );
       temp = temp -> next
     }
}

int main () {

  int sparsem [4] [5] ={

      . - - -:
      - - - -:

  }
  head = 0;

for ( int i=0 ; i<4 ; i++){
     for (int j = 0 ; j<5 ; j++) {

       if ( sparsm [i] [j] != 0 ) {

          head= insert ( head , i , j , sparsm [i] [i]);
       }
     }
  }
  disp (head);
 return 0;
}
```

- Polynomials

Program :-

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

struct node {
    int coeff;
    int exp;
    struct node * next;
};
struct node * head, * temp, * newnode;

struct node * create (int coeff, int exp) {
    newnode = malloc....

    newnode -> coeff = coeff;
            -> exp = exp;
            -> next = 0;

    return newnode;

}

struct void * insert (head, int coeff, int exp) {
    newnode = malloc..

    if (head == 0) {
        return newnode;
    }
    else {
        temp = head;
    while (temp -> next != 0){
        temp = temp -> next;
        }
    temp -> next = newnode;

    return head;
    }
}
```

```
Struct node* add ( struct node *poly1 , .... poly2){

    Struct node* res = 0;
      "     "    temp1 = poly1;
      "     "    temp2 = poly 2;

  while ( temp1 != 0 && temp 2 != 0){

 if ( temp1 -> exp  > temp2 -> exp ) {
    res = insert (res , temp1-> coeff , temp1-> exp );
    temp = temp -> next;
   }

else if (  2-> ep  >  1 -> exp) {
     res = insert ( res , temp 2-> coeff ,   2-> exp);
     temp = temp -> next
 }

 else {

    int sum = temp1-> coeff + temp2 -> coeff;
     if (sum != 0) {
         res = insert ( res , sum, temp1-> exp);
      }
      temp1 = temp1-> next
          2        2-> next
   }
 }

   return res;

 }

void  disp ( head) {
     temp = hedd

  while ( temp != 0) {
    printf ( " %d ^ %d ", temp -> coeff , temp -> exp);
     temp = temp -> next;
   if ( temp != 0){
         print (" + ") }
    else { printf (" "); }
  }
print ("\n"); }
```



```c
int main() {
    struct Node* poly1 = NULL;
    struct Node* poly2 = NULL;

    // First polynomial: 5x^2 + 4x^1 + 2x^0
    poly1 = insertNode(poly1, 5, 2);
    poly1 = insertNode(poly1, 4, 1);
    poly1 = insertNode(poly1, 2, 0);

    // Second polynomial: -5x^1 + 5x^0
    poly2 = insertNode(poly2, -5, 1);
    poly2 = insertNode(poly2, 5, 0);

    // Display the polynomials
    printf("First Polynomial: ");
    displayPolynomial(poly1);

    printf("Second Polynomial: ");
    displayPolynomial(poly2);

    // Add the two polynomials
    struct Node* sum = addPolynomials(poly1, poly2);

    // Display the sum
    printf("Sum: ");
    displayPolynomial(sum);

    return 0;
}
```