1) Write a program in C to store elements in an array and print them.

CODE:

```
#include<stdio.h>
int main(){
int arr[50],i,x;
printf("Enter the number of elements: ");
scanf("%d",&x);
printf("Enter the elements:");
for(i=0;i<x;i++)
scanf("%d",&arr[i]);
printf("The elements are :");
for(i=0;i<x;i++)
printf("%d ",arr[i]);
    return 0;
}
```

OUTPUT:

```
Enter the number of elements: 5
Enter the elements:1 2 3 4 5
The elements are :1 2 3 4 5
```

2) Write a program in c to count the total number of duplicate elements in an array.

CODE:

```
#include<stdio.h>
#define size 10
int main(){
int arr[size]={1,3,2,1,2,4,2,5,6,2},i,c=0,x,j;
for(i=0;i<size;i++){
   for(j=0;j<size;j++){
      if(arr[i]==arr[j]){
         c++;
         break;
      }}}
printf("The count of duplicate number is %d",c);
return 0;
}
```

OUTPUT:

```
The count of duplicate number is 10
```

3)Write a program in C to find the second largest element in an array.

CODE:

```
#include<stdio.h>
int main(){
   int arr[100],i,j,slar=0,temp=0,size;
   printf("Enter the size of the array:");
   scanf("%d",&size);
   printf("\nEnter the elements of array:");
   for(i=0;i<size;i++)
   scanf("%d",&arr[i]);
   for(i=0;i<size;i++)
   if(arr[i]>temp)
   temp=arr[i];
   for(i=0;i<size;i++)
   if(arr[i]>slar && arr[i]!=temp)
   slar=arr[i];
   printf("\nThe array is :");
   for(i=0;i<size;i++)
   printf("%d ",arr[i]);
   printf("\nThe second largest number is %d",slar);
   return 0;}
```

OUTPUT:

```
Enter the size of the array:5

Enter the elements of array:2 6 4 8 5

The array is :2 6 4 8 5
The second largest number is 6
```

4)Write a program to insert an element into an array.

CODE:

```
#include<stdio.h>
int main(){
   int arr[100],i,j,x,data,size;
   printf("Enter the size of the array:");
   scanf("%d",&size);
   printf("\nEnter the elements of array:");
   for(i=0;i<size;i++)
   scanf("%d",&arr[i]);
   printf("\nEnter the data and pos you want to insert: ");
   scanf("%d %d",&data,&x);
   size++;
for(i=0;i<x-1;i++)
arr[size-i]=arr[size-i-1];
printf("\nThe array before insertion is: ");
for(i=0;i<size;i++)
printf("%d ",arr[i]);
arr[x-1]=data;
printf("\n\nThe array after insertion is: ");
for(i=0;i<size;i++)
printf("%d ",arr[i]);
  return 0;
```

}

OUTPUT:

```
Enter the size of the array:5

Enter the elements of array:1 2 3 4 5

Enter the data and pos you want to insert: 3 2

The array before insertion is: 1 2 3 4 5 1
The array after insertion is: 1 3 3 4 5 1
```

5)Write a C program to delete an element from an array.

CODE:

```
#include<stdio.h>
int main(){
   int arr[100],i,j,x,data,size;
   printf("Enter the size of the array:");
   scanf("%d",&size);
   printf("\nEnter the elements of array:");
   for(i=0;i<size;i++)
   scanf("%d",&arr[i]);
   printf("\nEnter the pos you want to delete: ");
   scanf("%d",&x);
printf("\nThe array before deletion is: ");
for(i=0;i<size;i++)
printf("%d ",arr[i]);
   size--;
for(i=x-1;i<size;i++)
   arr[i]=arr[i+1];
printf("\nThe array after deletion is: ");
for(i=0;i<size;i++)
printf("%d ",arr[i]);
return 0;}
```

OUTPUT:

```
Enter the size of the array:5

Enter the elements of array:4 3 2 6 5

Enter the pos you want to delete: 4

The array before deletion is: 4 3 2 6 5
The array after deletion is: 4 3 2 5
```

6)**Write a C program to search an element in an array(Using Linear Search algorithm).**

CODE:

```
#include <stdio.h>

int search(int arr[], int N, int x)
{
    int i;
    for (i = 0; i < N; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

// Driver code
int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int N = sizeof(arr) / sizeof(arr[0]);

    // Function call
    int result = search(arr, N, x);
    (result == -1)
        ? printf("Element is not present in array")
        : printf("Element is present at index %d", result);
    return 0;
}
```

OUTPUT:

```
Element is present at index 3
```

7) Write a C program to search an element in an array(Using Binary Search algorithm).

OUTPUT:

```
#include<stdio.h>
int main()
{
   int n,i=0;
   printf("Enter the numbers of Elements : ");
   scanf("%d",&n);
   int arr[n];
   printf("Enter the Elements of the Array : ");
   for(int i=0;i<n;i++)
      scanf("%d",&arr[i]);
   printf("Enter the Element you want to search : ");
   int search;
   scanf("%d",&search);
//Searching----------------------------
   int low=0,mid,high=n-1;
   mid=(low+high)/2;
   while(low<=high){
      if(search==arr[mid]){
            printf("So the position of the Element : %d",mid+1);
            break;
         }
      else if(search>arr[mid])
         low=mid+1;
      else
         high=mid-1;
      mid=(low+high)/2;
   }
   if(low>high)
   printf("Invalid Search...");

   return 0;
}
```

OUTPUT:

```
Enter the numbers of Elements : 5
Enter the Elements of the Array : 1 2 3 4 5
Enter the Element you want to search : 3
So the position of the Element : 3
```

8)Write a C program to sort the elements of an array(Using Bubble sort algorithm).

CODE:

```c
#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

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

  printf("Enter %d integers\n", n);

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

  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>' */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);

  return 0;
}
```

OUTPUT:

```
Enter number of elements
5
Enter 5 integers
6 4 8 3 1
Sorted list in ascending order:
1
3
4
6
8
```

9)Write a C program to sort the elements of an array(Using Insertion sort algorithm).

CODE:

```
#include<stdio.h>
int main()
{
   int n,temp,j;
   printf("Enter the numbers of Elements : ");
   scanf("%d",&n);
   if(n==0){
      printf("Invalid Array...");
   }
   else{
      int arr[n];
      printf("Enter the Elements of the Array : ");
      for(int i=0;i<n;i++)
      scanf("%d",&arr[i]);

      //Sorting----------
      for(int i=1;i<n;i++){
         temp=arr[i];
         j=i-1;
         while(j>=0&&arr[j]>temp){
            arr[j+1]=arr[j];
            j=j-1;
         }
         arr[j+1]=temp;
      }
      printf("After Sorting----------\n");
      for(int i=0;i<n;i++)
      printf("%d\t",arr[i]);
   }
   return 0;
}
```

OUTPUT:

```
Enter the numbers of Elements : 5
Enter the Elements of the Array : 4 6 2 1 7
After Sorting----------
1       2       4       6       7
```

10) Implement a stack using n array.

CODE:

```
#include<stdio.h>
#define size 5
struct stack
{
   int top;
   int arr[size];
};
// The Stack is Empty or not
int isEmpty(struct stack s)
{
   if(s.top == -1)
      return 1;
   return 0;
}
// The Stack is Full or not
int isFull(struct stack s)
{
   if(s.top == size-1)
      return 1;
   return 0;
}
// Pushing a data to stack
void push(struct stack *s, int data)
{
   s->top++;
   s->arr[s->top] = data;
   printf("Data Pushed!");
}
// Popping a Data from Stack
int pop(struct stack *s)
{
   int val;
   val = s->arr[s->top];
   s->top--;
   return val;
}
//Displaying the Stack
void display(struct stack s)
{
   for(int i=0;i<=s.top;i++)
      printf("%d\t",s.arr[i]);
}
//Peeking the value from Stack
```

```
int peek(struct stack *s)
{
   return s->arr[s->top];
}
int main()
{
   struct stack s;
   s.top = -1;
   int n, a,flag=1;
   while (flag==1)
   {
      printf("\n\t M E N U\n");
      printf("\t 1. Push \n");
      printf("\t 2. Pop \n");
      printf("\t 3. Peek \n");
      printf("\t 4. Display the stack\n");
      printf("\t 5. Exit \n");
      printf("Enter The Case You Want To Do: ");
      scanf("%d", &a);
      switch (a)
      {
      case 1:
         if (isFull(s))
            printf("Stack Overflow");
         else
         {
            printf("Enter The Data To Push In Stack: ");
            scanf("%d", &n);
            push(&s, n);
         }
         break;
      case 2:
         if (isEmpty(s))
            printf("Stack Underflow!!");
         else
            printf("Poped Element is: %d", pop(&s));
         break;
      case 3:
         if (isEmpty(s))
            printf("Stack Underflow!!");
         else
            printf("The Top most element : %d", peek(&s));
         break;
      case 4:
         printf("The Elements Are : \n");
         display(s);
```

```
            break;
        case 5:
            printf("Exiting!!!");
            flag = 0;
            break;
        default :
            printf("Wrong Input!!");
            break;
        }
    }
    return 0;
}
```

OUTPUT:

```
M E N U
        1. Push
        2. Pop
        3. Peek
        4. Display the stack
        5. Exit
Enter The Case You Want To Do: 1
Enter The Data To Push In Stack: 10
Data Pushed!
        M E N U
        1. Push
        2. Pop
        3. Peek
        4. Display the stack
        5. Exit
Enter The Case You Want To Do: 1
Enter The Data To Push In Stack: 20
Data Pushed!
        M E N U
        1. Push
        2. Pop
        3. Peek
        4. Display the stack
        5. Exit
Enter The Case You Want To Do: 4
The Elements Are :
10      20
```

M E N U

1. Push
2. Pop
3. Peek
4. Display the stack
5. Exit

Enter The Case You Want To Do: 2
Poped Element is: 20

M E N U

1. Push
2. Pop
3. Peek
4. Display the stack
5. Exit

Enter The Case You Want To Do: 3
The Top most element : 10

M E N U

1. Push
2. Pop
3. Peek
4. Display the stack
5. Exit

Enter The Case You Want To Do: 4
The Elements Are :
10

M E N U

1. Push
2. Pop
3. Peek
4. Display the stack
5. Exit

Enter The Case You Want To Do: 5
Exiting!!!

11)Implement two stacks in a single array.

CODE:

```
#include <stdio.h>
#define SIZE 20
 int array[SIZE];  // declaration of array type variable.
int top1 = -1;
int top2 = SIZE;

//Function to push data into stack1
void push1 (int data)
{
// checking the overflow condition
  if (top1 < top2 - 1)
  {
     top1++;
    array[top1] = data;
  }
  else
  {
    printf ("Stack is full");
  }
}
// Function to push data into stack2
void push2 (int data)
{
// checking overflow condition
if (top1 < top2 - 1)
  {
    top2--;
    array[top2] = data;
  }
  else
  {
    printf ("Stack is full..\n");
  }
}

//Function to pop data from the Stack1
void pop1 ()
{
```

```
// Checking the underflow condition
 if (top1 >= 0)
  {
    int popped_element = array[top1];
    top1--;

    printf ("%d is being popped from Stack 1\n", popped_element);
  }
  else
  {
    printf ("Stack is Empty \n");
  }
}
// Function to remove the element from the Stack2
void pop2 ()
{
// Checking underflow condition
if (top2 < SIZE)
  {
      int popped_element = array[top2];
    top2--;

    printf ("%d is being popped from Stack 1\n", popped_element);
  }
  else
  {
    printf ("Stack is Empty!\n");
  }
}

//Functions to Print the values of Stack1
void display_stack1 ()
{
  int i;
  for (i = top1; i >= 0; --i)
  {
    printf ("%d ", array[i]);
  }
  printf ("\n");
}
// Function to print the values of Stack2
void display_stack2 ()
```

```
{
  int i;
  for (i = top2; i < SIZE; ++i)
  {
    printf ("%d ", array[i]);
  }
  printf ("\n");
}

int main()
{
  int ar[SIZE];
  int i;
  int num_of_ele;

  printf ("We can push a total of 20 values\n");

  //Number of elements pushed in stack 1 is 10
  //Number of elements pushed in stack 2 is 10

// loop to insert the elements into Stack1
for (i = 1; i <= 10; ++i)
  {
    push1(i);
    printf ("Value Pushed in Stack 1 is %d\n", i);
  }
// loop to insert the elements into Stack2.
for (i = 11; i <= 20; ++i)
  {
    push2(i);
    printf ("Value Pushed in Stack 2 is %d\n", i);
  }

  //Print Both Stacks
  display_stack1 ();
 display_stack2 ();

  //Pushing on Stack Full
  printf ("Pushing Value in Stack 1 is %d\n", 11);
  push1 (11);

  //Popping All Elements from Stack 1
```

```
  num_of_ele = top1 + 1;
  while (num_of_ele)
  {
    pop1 ();
    --num_of_ele;
  }

  // Trying to Pop the element From the Empty Stack
  pop1 ();

  return 0;
}
```

OUTPUT:

```
We can push a total of 20 values
Value Pushed in Stack 1 is 1
Value Pushed in Stack 1 is 2
Value Pushed in Stack 1 is 3
Value Pushed in Stack 1 is 4
Value Pushed in Stack 1 is 5
Value Pushed in Stack 1 is 6
Value Pushed in Stack 1 is 7
Value Pushed in Stack 1 is 8
Value Pushed in Stack 1 is 9
Value Pushed in Stack 1 is 10
Value Pushed in Stack 2 is 11
Value Pushed in Stack 2 is 12
Value Pushed in Stack 2 is 13
Value Pushed in Stack 2 is 14
Value Pushed in Stack 2 is 15
Value Pushed in Stack 2 is 16
Value Pushed in Stack 2 is 17
Value Pushed in Stack 2 is 18
Value Pushed in Stack 2 is 19
Value Pushed in Stack 2 is 20
10 9 8 7 6 5 4 3 2 1
20 19 18 17 16 15 14 13 12 11
Pushing Value in Stack 1 is 11
Stack is full10 is being popped from Stack 1
9 is being popped from Stack 1
8 is being popped from Stack 1
7 is being popped from Stack 1
6 is being popped from Stack 1
5 is being popped from Stack 1
4 is being popped from Stack 1
3 is being popped from Stack 1
2 is being popped from Stack 1
1 is being popped from Stack 1
Stack is Empty
```

12)Reverse a stack using recursion.

CODE:

```
#include <stdio.h>

#define MAXSIZE 7
#define TRUE 1
#define FALSE 0
//Structure defining Stack data structure
struct Stack {
   int top;
   int array[MAXSIZE];
} st;


//Initializes the top index to -1

void initialize() {
 st.top = -1;
}
//Checks if Stack is Full or not

int isFull() {
   if(st.top >= MAXSIZE-1)
      return TRUE;
   else
      return FALSE;
}
//Checks if Stack is Empty or not
int isEmpty() {
 if(st.top == -1)
    return TRUE;
 else
    return FALSE;
}
//Adds an element to stack and then increment top index
void push(int num) {
   if (isFull())
      printf("Stack is Full...\n");
   else {
      st.array[st.top + 1] = num;
      st.top++;
   }
}
//Removes top element from stack and decrement top index
```

```
int pop() {
   if (isEmpty())
      printf("Stack is Empty...\n");
   else {
    st.top = st.top - 1;
      return st.array[st.top+1];
   }
}
//Prints elements of stack using recursion
void printStack(){
 if(!isEmpty()){
    int temp = pop();
    printStack();
    printf(" %d ", temp);
    push( temp);
   }
}
void insertAtBottom(int item) {
   if (isEmpty()) {
      push(item);
   } else {
      int top = pop();
      insertAtBottom(item);
      push(top);
   }
}

void reverse() {
   if (!isEmpty()) {
      int top = pop();
      reverse();
      insertAtBottom(top);
   }
}
//Returns the number of elements in Stack
int getSize(){
 return st.top+1;
}

int main() {
   initialize(st);
   push(1);
   push(2);
   push(3);
   push(4);
   push(5);
```

```
    printf("Original Stack\n");
    printStack();
    reverse();
    printf("\nReversed Stack\n");
    printStack();
    return 0;
}
```

OUTPUT:

```
Original Stack
 1 2 3 4 5
Reversed Stack
 5 4 3 2 1
```

13)Write a C Program to Create a Linked List and Display it.

CODE:

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

struct Node
{
   int data;
   struct Node *next;
};

void linkedListTraversal(struct Node *ptr)
{
   while (ptr != NULL)
   {
      printf("%d\t", ptr->data);
      ptr = ptr->next;
   }
}

int main()
{
   struct Node *head;
   struct Node *second;
   struct Node *third;
   struct Node *fourth;

   // Allocate memory for nodes in the linked list in Heap
   head = (struct Node *)malloc(sizeof(struct Node));
   second = (struct Node *)malloc(sizeof(struct Node));
   third = (struct Node *)malloc(sizeof(struct Node));
   fourth = (struct Node *)malloc(sizeof(struct Node));

   // Link first and second nodes
   head->data = 7;
   head->next = second;

   // Link second and third nodes
   second->data = 11;
   second->next = third;

   // Link third and fourth nodes
   third->data = 41;
```

```
    third->next = fourth;

    // Terminate the list at the third node
    fourth->data = 66;
    fourth->next = NULL;

    //  For Trversal
    printf("Output : ");
    linkedListTraversal(head);
    return 0;
}
```

OUTPUT:

```
Output : 7      11      41      66
```

14)Write a C program to insert a new Node

a. At the beginning of a Singly linked list.
b. At the middle of a Singly linked list.
c. At the end of a Singly linked list.

CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
   int data;
   struct Node * next;
};
void linkedListTraversal(struct Node *ptr)
{
   while (ptr != NULL)
   {
      printf("Element: %d\n", ptr->data);
      ptr = ptr->next;
   }
}
// A) Insert at Beginning
struct Node * insertAtFirst(struct Node *head, int data){
   struct Node * ptr = (struct Node *) malloc(sizeof(struct Node));
   ptr->data = data;

   ptr->next = head;
   return ptr;
}
// B) Insert At the Middle Index
struct Node * insertAtIndex(struct Node *head, int data, int index){
   struct Node * ptr = (struct Node *) malloc(sizeof(struct Node));
   struct Node * p = head;
   int i = 0;

   while (i!=index-1)
   {
      p = p->next;
      i++;
   }
   ptr->data = data;
   ptr->next = p->next;
   p->next = ptr;
```

```
    return head;
}

// C) Insert At the End
struct Node * insertAtEnd(struct Node *head, int data){
    struct Node * ptr = (struct Node *) malloc(sizeof(struct Node));
    ptr->data = data;
    struct Node * p = head;

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

int main(){
    struct Node *head;
    struct Node *second;
    struct Node *third;
    struct Node *fourth;

    // Allocate memory for nodes in the linked list in Heap
    head = (struct Node *)malloc(sizeof(struct Node));
    second = (struct Node *)malloc(sizeof(struct Node));
    third = (struct Node *)malloc(sizeof(struct Node));
    fourth = (struct Node *)malloc(sizeof(struct Node));

    // Link first and second nodes
    head->data = 7;
    head->next = second;

    // Link second and third nodes
    second->data = 11;
    second->next = third;

    // Link third and fourth nodes
    third->data = 41;
    third->next = fourth;

    // Terminate the list at the third node
    fourth->data = 66;
    fourth->next = NULL;

    printf("Linked list before insertion\n");
```

```
    linkedListTraversal(head);
    head = insertAtFirst(head, 56);              (Insert at first activate)
    // head = insertAtIndex(head, 56, 1);        (When Insert at middle Activate just
                                                  comment out)
    // head = insertAtEnd(head, 56);             (When Insert at end Activate just comment
                                                  out)
    printf("\nLinked list after insertion\n");
    linkedListTraversal(head);


    return 0;
}
```

OUTPUT:

a)

```
Linked list before insertion
Element: 7
Element: 11
Element: 41
Element: 66

Linked list after insertion
Element: 56
Element: 7
Element: 11
Element: 41
Element: 66
```

b)

```
Linked list before insertion
Element: 7
Element: 11
Element: 41
Element: 66

Linked list after insertion
Element: 7
Element: 56
Element: 11
Element: 41
Element: 66
```

c)

```
Linked list before insertion
Element: 7
Element: 11
Element: 41
Element: 66

Linked list after insertion
Element: 7
Element: 11
Element: 41
Element: 66
Element: 56
```

16) Write a program to delete a new Node in a singly linked list

a) At the beginning
b) At the middle
c) At the end

CODE:

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

struct node {
   int raz;
   struct node *baad;
};

void display(struct node *ptr)
{
   while(ptr!=NULL)
   {
      printf("%d ",ptr->raz);
      ptr=ptr->baad;
   }
}

struct node * insertatbeg(struct node *ptr,int data)
{
   struct node *p=(struct node*)malloc(sizeof(struct node));
   p->raz=data;
   p->baad=ptr;
   ptr=p;
   return ptr;
}
```

```
struct node *insertatend(struct node *ptr,int data)
{
   struct node *p,i,*q=ptr;
   p=(struct node*)malloc(sizeof(struct node));
while(q->baad!=NULL)
       q=q->baad;
       q->baad=p;
       p->raz=data;
       p->baad=NULL;
       return ptr;
}
struct node *delfirst(struct node* ptr)
{
   struct node *p;
   p=ptr;
   ptr=ptr->baad;
   free(p);
   return ptr;

}
struct node *delmid(struct node *ptr,int pos)
{
int i;
struct node *p,*q=ptr;
for(i=1;i< pos-1;i++)
ptr=ptr->baad;
p=ptr->baad;
ptr->baad=ptr->baad->baad;
free(p);
return q;
}
struct node *delend(struct node *ptr)
```

```
{

   struct node*p=ptr,*q;
   q=p->baad;
   while(q->baad!=NULL)
   {
      q=q->baad;
      p=p->baad;
      }
p->baad=NULL;
free(q);
return ptr;
}

struct node * creatlist(struct node *ptr)
{
   int i,data,node;
   printf("Enter how many nodes: ");
   scanf("%d",&node);
   if(node==0)
   exit(0);
   printf("Enter the element: ");
   scanf("%d",&data);
   ptr=insertatbeg(ptr,data);
   for(i=2;i<=node;i++)
   {

   scanf("%d",&data);
      ptr=insertatend(ptr,data);
      }
   return ptr;
}
```

```
int main(){
int data,pos,i,choice;
struct node *dada=NULL;
dada=creatlist(dada);

printf("\nBefore Deletion Linked list: ");
display(dada);
printf("\n\nAfter deletion Linked list: ");
display(dada=delfirst(dada));
//display(dada=delmid(dada,3));
//display(dada=delend(dada));
   return 0;
}
```

OUTPUT:

a)

```
Enter how many nodes: 5
Enter the element: 1 3 2 4 5

Before Deletion Linked list: 1 3 2 4 5

After deletion Linked list: 3 2 4 5
```

b)

```
Enter how many nodes: 5
Enter the element: 1 2 3 4 5

Before Deletion Linked list: 1 2 3 4 5

After deletion Linked list: 1 2 4 5
```

c)

```
Enter how many nodes: 5
Enter the element: 1 2 4 6 7

Before Deletion Linked list: 1 2 4 6 7

After deletion Linked list: 1 2 4 6
```

16)Add two Polynomial using Linked list.

OUTPUT:

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

struct Node {
   int coef;
   int exp;
   struct Node* next;
};

typedef struct Node Node;

void insert(Node** poly, int coef, int exp) {
   Node* temp = (Node*) malloc(sizeof(Node));
   temp->coef = coef;
   temp->exp = exp;
   temp->next = NULL;

   if (*poly == NULL) {
      *poly = temp;
      return;
   }

   Node* current = *poly;

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

   current->next = temp;
}

void print(Node* poly) {
   if (poly == NULL) {
      printf("0\n");
      return;
   }
```

```
    Node* current = poly;

    while (current != NULL) {
        printf("%dx^%d", current->coef, current->exp);
        if (current->next != NULL) {
            printf(" + ");
        }
        current = current->next;
    }

    printf("\n");
}

Node* add(Node* poly1, Node* poly2) {
    Node* result = NULL;

    while (poly1 != NULL && poly2 != NULL) {
        if (poly1->exp == poly2->exp) {
            insert(&result, poly1->coef + poly2->coef, poly1->exp);
            poly1 = poly1->next;
            poly2 = poly2->next;
        } else if (poly1->exp > poly2->exp) {
            insert(&result, poly1->coef, poly1->exp);
            poly1 = poly1->next;
        } else {
            insert(&result, poly2->coef, poly2->exp);
            poly2 = poly2->next;
        }
    }

    while (poly1 != NULL) {
        insert(&result, poly1->coef, poly1->exp);
        poly1 = poly1->next;
    }
    while (poly2 != NULL) {
        insert(&result, poly2->coef, poly2->exp);
        poly2 = poly2->next;
    }
```

```
    return result;
}

int main() {
    Node* poly1 = NULL;
    insert(&poly1, 5, 4);
    insert(&poly1, 3, 2);
    insert(&poly1, 1, 0);

    Node* poly2 = NULL;
    insert(&poly2, 4, 4);
    insert(&poly2, 2, 2);
    insert(&poly2, 1, 1);

    printf("First polynomial: ");
    print(poly1);

    printf("Second polynomial: ");
    print(poly2);

    Node* result = add(poly1, poly2);
    printf("Result: ");
    print(result);

    return 0;
}
```

OUTPUT:

```
First polynomial: 5x^4 + 3x^2 + 1x^0
Second polynomial: 4x^4 + 2x^2 + 1x^1
Result: 9x^4 + 5x^2 + 1x^1 + 1x^0
```