Module 5 : Linked List

1) Singly Linked List

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include<bits/stdc++.h>
using namespace std;
struct node
{
int data;
struct node *next;
};
struct node *start = NULL;
struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node *);
struct node *insert_after(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
struct node *sort_list(struct node *);
//int main(int argc, char *argv[]) {
int main()
{
```

```
int option;
do
{
 printf("\n\n *****MAIN MENU *****");
 printf("\n 1: Create a list");
 printf("\n 2: Display the list");
 printf("\n 3: Add a node at the beginning");
 printf("\n 4: Add a node at the end");
 printf("\n 5: Add a node before a given node");
 printf("\n 6: Add a node after a given node");
 printf("\n 7: Delete a node from the beginning");
 printf("\n 8: Delete a node from the end");
 printf("\n 9: Delete a given node");
 printf("\n 10: Delete a node after a given node");
 printf("\n 11: Delete the entire list");
 printf("\n 12: Sort the list");
 printf("\n 13: EXIT");
 printf("\n\n Enter your option : ");
 scanf("%d", &option);
 switch(option)
 {
 case 1: start = create_ll(start);
 printf("\n LINKED LIST CREATED");
 break;
 case 2: start = display(start);
 break;
 case 3: start = insert_beg(start);
 break;
 case 4: start = insert_end(start);
 break;
 case 5: start = insert_before(start);
 break;
 case 6: start = insert_after(start);
 break;
 case 7: start = delete_beg(start);
 break;
 case 8: start = delete_end(start);
 break;
 case 9: start = delete_node(start);
 break;
 case 10: start = delete_after(start);
 break;
 case 11: start = delete_list(start);
 printf("\n LINKED LIST DELETED");
 break;
 case 12: start = sort_list(start);
 break;
 }
}while(option !=13);
getch();
return 0;
}
struct node *create_ll(struct node *start)
{
struct node *new_node, *ptr;
int num;
printf("\n Enter -1 to end");
printf("\n Enter the data : ");
scanf("%d", &num);
```

```
while(num!=-1)
{
 new_node = (struct node*)malloc(sizeof(struct node));
 new_node -> data=num;
 if(start==NULL)
 {
 new_node -> next = NULL;
 start = new_node;
 }
 else
 {
 ptr=start;
 while(ptr->next!=NULL)
 ptr=ptr->next;
 ptr->next = new_node;
 new_node->next=NULL;
 }
 printf("\n Enter the data : ");
 scanf("%d", &num);
}
return start;
}
struct node *display(struct node *start)
{
struct node *ptr;
ptr = start;
while(ptr != NULL)
{
 printf("\t %d", ptr -> data);
 ptr = ptr -> next;
}
return start;
}
struct node *insert_beg(struct node *start)
{
struct node *new_node;
int num;
printf("\n Enter the data : ");
scanf("%d", &num);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
new_node -> next = start;
start = new_node;
return start;
}
struct node *insert_end(struct node *start)
{
struct node *ptr, *new_node;
int num;
printf("\n Enter the data : ");
scanf("%d", &num);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
new_node -> next = NULL;
ptr = start;
while(ptr -> next != NULL)
ptr = ptr -> next;
ptr -> next = new_node;
return start;
}
```

```
struct node *insert_before(struct node *start)
{
struct node *new_node, *ptr, *preptr;
int num, val;
printf("\n Enter the data : ");
scanf("%d", &num);
printf("\n Enter the value before which the data has to be inserted : ");
scanf("%d", &val);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
ptr = start;
while(ptr -> data != val)
{
 preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next = new_node;
new_node -> next = ptr;
return start;
}
struct node *insert_after(struct node *start)
{
struct node *new_node, *ptr, *preptr;
int num, val;
printf("\n Enter the data : ");
scanf("%d", &num);
printf("\n Enter the value after which the data has to be inserted : ");
scanf("%d", &val);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
```

```
ptr = start;
preptr = ptr;
while(preptr -> data != val)
{
 preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next=new_node;
new_node -> next = ptr;
return start;
}
struct node *delete_beg(struct node *start)
{
struct node *ptr;
ptr = start;
start = start -> next;
free(ptr);
return start;
}
struct node *delete_end(struct node *start)
{
struct node *ptr, *preptr;
ptr = start;
while(ptr -> next != NULL)
{
 preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next = NULL;
free(ptr);
```

```
return start;
}
struct node *delete_node(struct node *start)
{
struct node *ptr, *preptr;
int val;
printf("\n Enter the value of the node which has to be deleted : ");
scanf("%d", &val);
ptr = start;
if(ptr -> data == val)
{
 start = delete_beg(start);
 return start;
}
else
{
 while(ptr -> data != val)
 {
 preptr = ptr;
 ptr = ptr -> next;
 }
 preptr -> next = ptr -> next;
 free(ptr);
 return start;
}
}
struct node *delete_after(struct node *start)
{
struct node *ptr, *preptr;
int val;
```

```
printf("\n Enter the value after which the node has to deleted : ");
scanf("%d", &val);
ptr = start;
preptr = ptr;
while(preptr -> data != val)
{
 preptr = ptr;
 ptr = ptr -> next;
}
preptr -> next=ptr -> next;
free(ptr);
return start;
}
struct node *delete_list(struct node *start)
{
            struct node *ptr;
if(start!=NULL){
 ptr=start;
 while(ptr != NULL)
 {
 printf("\n %d is to be deleted next", ptr -> data);
 start = delete_beg(ptr);
ptr = start;
 }
}
return start;
}
struct node *sort_list(struct node *start)
{
struct node *ptr1, *ptr2;
```

```
int temp;
ptr1 = start;
while(ptr1 -> next != NULL)
{
 ptr2 = ptr1 -> next;
 while(ptr2 != NULL)
 {
 if(ptr1 -> data > ptr2 -> data)
 {
 temp = ptr1 -> data;
 ptr1 -> data = ptr2 -> data;
 ptr2 -> data = temp;
 }
 ptr2 = ptr2 -> next;
 }
 ptr1 = ptr1 -> next;
 }
return start; // Had to be added
}
```

2) Doubly Linked List

```
#include<iostream>
#include<conio.h>
#include<bits/stdc++.h>
#include<stdio.h>
#include<stdlib.h>


using namespace std;

struct node
{
int data;
struct node *next;
struct node *prev;
};
```

```
struct node *start=NULL;
struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node  *);
struct node *insert_before(struct node *start);
struct node *insert_after(struct node *start);
struct node *insert_sorted(struct node *start);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *start);
struct node *delete_after(struct node *start);
struct node *delete_before(struct node *start);
struct node *delete_sorted(struct node *start);
struct node *sort_list(struct node *start);

main()
{
        int option;
        cout<<"\n\n***Main Menu***";
        cout<<"\n 1: Create a List: ";
        cout<<"\n 2: Display the list: ";
        cout<<"\n 3: Add a node in the beginning";
        cout<<"\n 4: Add a node at the end";
        cout<<"\n 5: Add a node before a given node";
        cout<<"\n 6: Add a node after a given node";
        cout<<"\n 7: Add a node in a sorted linked list";
        cout<<"\n 8: Delete a node in the beginning";
        cout<<"\n 9: Delete a node at the end";
        cout<<"\n 10: Delete a node a given node";
        cout<<"\n 11: Delete a node before a given node";
        cout<<"\n 12: Delete a node after a given node";
        cout<<"\n 13: Delete a node in a sorted linked list";
        cout<<"\n 14: Sort list";
        cout<<"\n 15: Exit ";
        cout<<"\n*****";
do
{

cout<<"\nEnter your option : ";
cin>>option;
switch(option)
{
case 1:
start = create_ll(start);
cout<<"\n Linked list created";
break;
case 2:
start = display(start);
break;
case 3:
start = insert_beg(start);
break;
case 4:
start = insert_end(start);
break;
case 5:
start = insert_before(start);
break;
case 6:
start = insert_after(start);
break;
case 7:
start = insert_sorted(start);
break;
case 8:
start = delete_beg(start);
break;
case 9:
start = delete_end(start);
break;
case 10:
start = delete_node(start);
break;
case 11: delete_before(start);
break;
case 12:
start = delete_after(start);
break;
case 13:
start = delete_sorted(start);
break;
case 14:
start = sort_list(start);
break;
}
}while(option != 15);
getch();
return 0;
}

struct node *create_ll(struct node *start)
```

```
{
            struct node *new_node;
    int num;
    cout<<"\n Enter -1 to end";
    cout<<"\n Enter the data: ";
    cin>>num;
    while (num != -1)
    {
       if (start == NULL)
       {
           start = (struct node*)malloc (sizeof (struct node*) );
           start ->prev = NULL;
           start -> data = num;
           start -> next = NULL;
       }
else
{
new_node = (struct node* )malloc (sizeof (struct node*));
new_node -> prev = NULL;
new_node->data = num;
new_node-> next = start;
start ->prev = new_node;
start = new_node;
}
cout<<"\n Enter the data : ";
cin>>num;
}
return start;
}

struct node *display (struct node *start)
{

struct node *ptr;
ptr=start;
cout<<" ";
while (ptr!=NULL)
{
cout<<" "<<ptr->data;
ptr = ptr -> next;
}
return start;
}
struct node *insert_beg(struct node *start )
{
            struct node *new_node;
```

```
        int num;
        cout<<"\n Enter the data : ";
        cin>>num;
        new_node = (struct node * )malloc(sizeof (struct node *)) ;
        start -> prev = new_node;
        new_node-> next = start;
        new_node -> prev = NULL;
        new_node-> data = num;
        start = new_node;
        return start;
}
struct node *insert_end(struct node *start)
{

        struct node *ptr, *new_node;
        int num;
        cout<<" \n Enter the data:";
        cin>>num;
        new_node = (struct node *)malloc (sizeof (struct node *) );
        new_node -> data = num;
        ptr=start;
        while(ptr -> next != NULL)
        {
                ptr = ptr -> next;
        }

                ptr ->next = new_node;
                new_node->prev = ptr;
                new_node -> next = NULL;
                return start;
}

struct node *insert_before(struct node *start)
{
struct node *new_node, *ptr;
int num, val;
cout<<"\n Enter the data : ";
cin>>num;
cout<<" \n Enter the value before which the data has to be inserted: ";
cin>>val;
new_node = (struct node *)malloc (sizeof (struct node * ));
new_node -> data = num;
ptr = start;
while (ptr->data != val)
ptr = ptr-> next;
new_node-> next = ptr;
```

```
ptr -> prev -> next = new_node;
ptr ->prev = new_node;
return start;
}
struct node *insert_after (struct node *start)
{
struct node *new_node, *ptr;
int num, val;
cout<<"\n Enter the data: ";
cin>>num;
cout<<"\n Enter the value after which the data has to be inserted: ";
cin>>val;
new_node = (struct node * )malloc (sizeof (struct node *)) ;
new_node -> data = num;
ptr = start;
while (ptr -> data != val)
ptr = ptr -> next;
new_node ->prev = ptr;
new_node -> next = ptr -> next;
ptr -> next -> prev = new_node;
ptr -> next = new_node;
return start;
}
struct node *insert_sorted(struct node *start)
{
            struct node *new_node, *ptr;
            int num;
            cout<<"\n Enter the data: ";
            cin>>num;
            new_node = (struct node *)malloc (sizeof (struct node *)) ;
            new_node -> data = num;
            ptr = start;

            while(ptr ->data<=num)
            {
                ptr = ptr -> next;
                if (ptr ->next == NULL)
                    break;
            }
            if (ptr->next == NULL)
            {
                ptr -> next = new_node;
                new_node -> prev = ptr;
                new_node -> next = NULL;
            }
            else
```

```
                    {
                              new_node->next = ptr;
                              new_node->prev = ptr->prev;
                              ptr ->prev -> next = new_node;
                              ptr -> prev = new_node;
                    }
          return start;
}
struct node *delete_beg (struct node *start)
{
struct node *ptr;
ptr = start;
start = start -> next;
start ->prev = NULL;
free (ptr);
return start;
}
struct node *delete_end(struct node *start)
{
struct node *ptr;
ptr = start;
while (ptr->next != NULL)
ptr = ptr -> next;
ptr -> prev -> next = NULL;
free (ptr);
return start;
}
struct node *delete_node (struct node *start)
{
struct node *ptr;
int val;
cout<<"\n Enter the value of the node which has to be deleted:";
cin>>val;
ptr = start;
if (ptr ->data == val)
{
start = delete_beg(start);
return start;
}
else
{
while (ptr->data!=val)
ptr = ptr -> next;
ptr ->prev -> next = ptr -> next;
ptr -> next -> prev = ptr -> prev;
free(ptr);
return start;
}
}
struct node *delete_after (struct node *start)
{
        struct node *ptr, *temp;
        int val;
        cout<<"\nIn Enter the value after which the node has to deleted: ";
        cin>>val;
        ptr = start;
        while (ptr->data != val){
                ptr = ptr -> next;
        }
        temp = ptr -> next;
        ptr -> next = temp-> next;
        temp-> next -> prev = ptr;
        free(temp);
        return start;
}
struct node *delete_before (struct node *start)
{
struct node *ptr, *temp;
int val;
cout<<"\n In Enter the value before which the node has to deleted: ";
cin>>val;
ptr = start;
while (ptr -> data != val)
ptr = ptr -> next;
temp = ptr ->prev;
if (temp == start)
start = delete_beg(start);
else
{
ptr ->prev = temp ->prev;
temp -> prev -> next = ptr;
}
free(temp) ;
return start;
}
struct node *delete_sorted (struct node *start)
{
        struct node *ptr;
        int val;
        cout<<"\n Enter the value of the node which has to be deleted: ";
        cin>>val;
        ptr = start;
```

```
          while (ptr ->data != val)
          ptr = ptr -> next;
          ptr ->prev -> next = ptr -> next;
          free (ptr);
          return start;
          }
          struct node *delete_list (struct node *start)
          {
          while (start != NULL)
          start = delete_beg (start);
          return start;
}

struct node *sort_list(struct node *start)
{
          struct node *ptr1, *ptr2;
          int temp;
          ptr1 = start;
          while(ptr1->next != NULL)
          {
                    ptr2 = ptr1->next;
                    while(ptr2 != NULL)
                    {
                              if(ptr1->data > ptr2->data)
                              {
                                        temp = ptr1->data;
                                        ptr1->data = ptr2->data;
                                        ptr2->data = temp;
                              }
                              ptr2 = ptr2->next;
                    }
                    ptr1 = ptr1->next;
          }
          return start;
}
/*

***Main Menu***
 1: Create a List:
 2: Display the list:
 3: Add a node in the beginning
 4: Add a node at the end
 5: Add a node before a given node
 6: Add a node after a given node
 7: Add a node in a sorted linked list
 8: Delete a node in the beginning
 9: Delete a node at the end
 10: Delete a node a given node
 11: Delete a node before a given node
 12: Delete a node after a given node
 13: Delete a node in a sorted linked list
 14: Sort list
 15: Exit
*****
Enter your option : 1

 Enter -1 to end
 Enter the data: 5

 Enter the data : 1

 Enter the data : 6

 Enter the data : 8

 Enter the data : 9

 Enter the data : 15

 Enter the data : -1

 Linked list created
Enter your option : 2
  15 9 8 6 1 5
Enter your option : 3

 Enter the data : 55

Enter your option : 2
  55 15 9 8 6 1 5
Enter your option : 4

 Enter the data:65

Enter your option : 2
  55 15 9 8 6 1 5 65
Enter your option : 5

 Enter the data : 25

 Enter the value before which the data has to be inserted: 8
Enter your option : 2
  55 15 9 25 8 6 1 5 65
Enter your option : 6

 Enter the data: 41

 Enter the value after which the data has to be inserted: 6

Enter your option : 2
  55 15 9 25 8 6 41 1 5 65
Enter your option : 8

Enter your option : 2
  15 9 25 8 6 41 1 5 65
Enter your option : 9

Enter your option : 2
  15 9 25 8 6 41 1 5
Enter your option : 10

 Enter the value of the node which has to be deleted:8

Enter your option : 2
  15 9 25 6 41 1 5
Enter your option : 11

 In Enter the value before which the node has to deleted: 41

Enter your option : 2
  15 9 25 41 1 5
Enter your option : 12

In Enter the value after which the node has to deleted: 41

Enter your option : 2
  15 9 25 41 5
Enter your option : 14

Enter your option : 7

 Enter the data: 36

Enter your option : 2
  5 9 15 25 41 36
Enter your option : 13
```

```
 Enter the value of the node which has to be deleted: 25

Enter your option : 2
  5 9 15 41 36
Enter your option : 15

*/
```

3) Circular Linked List

```
/*

 * C++ Program to Implement Circular Linked List

 */

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

/*

 * Node Declaration

 */

struct node

{

    int info;

    struct node *next;

}*last;


/*

 * Class Declaration

 */

class circular_llist

{

    public:

        void create_node(int value);

        void add_begin(int value);
        void add_after(int value, int position);
        void delete_element(int value);
        void search_element(int value);
        void display_list();
        void update();
        void sort();
        circular_llist()
        {
            last = NULL;
        }
};

/*
 * Main :contains menu
 */
int main()
{
    int choice, element, position;
    circular_llist cl;
    while (1)
    {
        cout<<endl<<"---------------------------"<<endl;
        cout<<endl<<"Circular singly linked list"<<endl;
        cout<<endl<<"---------------------------"<<endl;
        cout<<"1.Create Node"<<endl;
        cout<<"2.Add at beginning"<<endl;
        cout<<"3.Add after"<<endl;
        cout<<"4.Delete"<<endl;
        cout<<"5.Search"<<endl;
        cout<<"6.Display"<<endl;
```

```
cout<<"7.Sort"<<endl;
cout<<"8.Quit"<<endl;
cout<<"Enter your choice : ";
cin>>choice;
switch(choice)
{
case 1:
   cout<<"Enter the element: ";
   cin>>element;
   cl.create_node(element);
   cout<<endl;
   break;
case 2:
   cout<<"Enter the element: ";
   cin>>element;
   cl.add_begin(element);
   cout<<endl;
   break;
case 3:
   cout<<"Enter the element: ";
   cin>>element;
   cout<<"Insert element after position: ";
   cin>>position;
   cl.add_after(element, position);
   cout<<endl;
   break;
case 4:
   if (last == NULL)
   {
      cout<<"List is empty, nothing to delete"<<endl;
```

```
        break;
    }
    cout<<"Enter the element for deletion: ";
    cin>>element;
    cl.delete_element(element);
    cout<<endl;
    break;
case 5:
    if (last == NULL)
    {
        cout<<"List Empty!! Can't search"<<endl;
        break;
    }
    cout<<"Enter the element to be searched: ";
    cin>>element;
    cl.search_element(element);
    cout<<endl;
    break;
case 6:
    cl.display_list();
    break;
case 7:
    cl.sort();
    break;
case 8:
    exit(1);
    break;
default:
    cout<<"Wrong choice"<<endl;
}
```

```
    }
    return 0;
}

/*
 * Create Circular Link List
 */
void circular_llist::create_node(int value)
{
    struct node *temp;
    temp = new(struct node);
    temp->info = value;
    if (last == NULL)
    {
        last = temp;
        temp->next = last;
    }
    else
    {
        temp->next = last->next;
        last->next = temp;
        last = temp;
    }
}

/*
 * Insertion of element at beginning
 */
void circular_llist::add_begin(int value)
{
    if (last == NULL)
    {
        cout<<"First Create the list."<<endl;
        return;
    }
    struct node *temp;
    temp = new(struct node);
    temp->info = value;
    temp->next = last->next;
    last->next = temp;
}

/*
 * Insertion of element at a particular place
 */
void circular_llist::add_after(int value, int pos)
{
    if (last == NULL)
    {
        cout<<"First Create the list."<<endl;
        return;
    }
    struct node *temp, *s;
    s = last->next;
    for (int i = 0;i < pos-1;i++)
    {
        s = s->next;
        if (s == last->next)
        {
            cout<<"There are less than ";
            cout<<pos<<" in the list"<<endl;
            return;
        }
    }
    temp = new(struct node);
    temp->next = s->next;
    temp->info = value;
    s->next = temp;
    /*Element inserted at the end*/
    if (s == last)
    {
        last=temp;
    }
}

/*
 * Deletion of element from the list
 */
void circular_llist::delete_element(int value)
{
    struct node *temp, *s;
    s = last->next;
      /* If List has only one element*/
    if (last->next == last && last->info == value)
    {
        temp = last;
        last = NULL;
        free(temp);
        return;
    }
```

```
if (s->info == value)  /*First Element Deletion*/
{
    temp = s;
    last->next = s->next;
    free(temp);
    return;
}
while (s->next != last)
{
    /*Deletion of Element in between*/
    if (s->next->info == value)
    {
        temp = s->next;
        s->next = temp->next;
        free(temp);
        cout<<"Element "<<value;
        cout<<" deleted from the list"<<endl;
        return;
    }
    s = s->next;
}
/*Deletion of last element*/
if (s->next->info == value)
{
    temp = s->next;
    s->next = last->next;
    free(temp);
    last = s;
    return;
}
```

```
    cout<<"Element "<<value<<" not found in the list"<<endl;
}

/*
 * Search element in the list
 */
void circular_llist::search_element(int value)
{
    struct node *s;
    int counter = 0;
    s = last->next;
    while (s != last)
    {
        counter++;
        if (s->info == value)
        {
            cout<<"Element "<<value;
            cout<<" found at position "<<counter<<endl;
            return;
        }
        s = s->next;
    }
    if (s->info == value)
    {
        counter++;
        cout<<"Element "<<value;
        cout<<" found at position "<<counter<<endl;
        return;
    }
    cout<<"Element "<<value<<" not found in the list"<<endl;
```

```
}

/*
 * Display Circular Link List
 */
void circular_llist::display_list()
{
    struct node *s;
    if (last == NULL)
    {
        cout<<"List is empty, nothing to display"<<endl;
        return;
    }
    s = last->next;
    cout<<"Circular Link List: "<<endl;
    while (s != last)
    {
        cout<<s->info<<"->";
        s = s->next;
    }
    cout<<s->info<<endl;
}


/*
 * Sort Circular Link List
 */
void circular_llist::sort()
{
    struct node *s, *ptr;
    int temp;
    if (last == NULL)
    {
        cout<<"List is empty, nothing to sort"<<endl;
        return;
    }
    s = last->next;
    while (s != last)
    {
        ptr = s->next;
        while (ptr != last->next)
        {
            if (ptr != last->next)
            {
                if (s->info > ptr->info)
                {
                    temp = s->info;
                    s->info = ptr->info;
                    ptr->info = temp;
                }
            }
            else
            {
                break;
            }
            ptr = ptr->next;
        }
        s = s->next;
    }
}
```

4) Polynomial Addition

```
#include<bits/stdc++.h>
using namespace std;
struct Node
{
int coeff;
int pow;
struct Node *next;
};
void create_node(int x, int y, struct Node **temp)
{
struct Node *r, *z;
z = *temp;
if(z == NULL)
{
r =(struct Node*)malloc(sizeof(struct Node));
r->coeff = x;
r->pow = y;
*temp = r;
r->next = (struct Node*)malloc(sizeof(struct Node));
r = r->next;
r->next = NULL;
}
else
{
r->coeff = x;
r->pow = y;
r->next = (struct Node*)malloc(sizeof(struct Node));
r = r->next;
r->next = NULL;
}
}
void polyadd(struct Node *p1, struct Node *p2, struct Node *result)
{
while(p1->next && p2->next)
{
if(p1->pow > p2->pow)
{
result->pow = p1->pow;
result->coeff = p1->coeff;
p1 = p1->next;
}
else if(p1->pow < p2->pow)
{
result->pow = p2->pow;
result->coeff = p2->coeff;
p2 = p2->next;
}
else
{
result->pow = p1->pow;
result->coeff = p1->coeff+p2->coeff;
p1 = p1->next;
p2 = p2->next;
}
result->next = (struct Node *)malloc(sizeof(struct Node));
result = result->next;
result->next = NULL;
}
while(p1->next || p2->next)
```

```
{
if(p1->next)
{
result->pow = p1->pow;
result->coeff = p1->coeff;
p1 = p1->next;
}
if(p2->next)
{
result->pow = p2->pow;
result->coeff = p2->coeff;
p2 = p2->next;
}
result->next = (struct Node *)malloc(sizeof(struct Node));
result = result->next;
result->next = NULL;
}
}
void printpoly(struct Node *node)
{
while(node->next != NULL)
{
printf("%dx^%d", node->coeff, node->pow);
node = node->next;
if(node->next != NULL)
printf(" + ");
}
}
int main()
{
```

```
struct Node *p1 = NULL, *p2 = NULL, *result = NULL;
create_node(41,7,&p1);
create_node(12,5,&p1);
create_node(65,0,&p1);
create_node(21,5,&p2);
create_node(15,2,&p2);
printf("polynomial 1: ");
printpoly(p1);
printf("\npolynomial 2: ");
printpoly(p2);
result = (struct Node *)malloc(sizeof(struct Node));
polyadd(p1, p2, result);
printf("\npolynomial after adding p1 and p2 : ");
printpoly(result);
return 0;
}
```