Neh Samir Joshi

19BEC1112

05/02/2022

**CSE2003**

**Data Structures and Algorithms**

**[LAB]**

**LAB – 4**

**Doubly Linked Lists and Polynomials**

**Aim:** To implement doubly linked lists (linear and circular) and implement polynomial operations using linked lists.

**Software Required:** Code editor (e.g. VS Code, Dev C++), GCC/G++ compiler

**Task 1:** To create a linear doubly linked list and perform create, insert, display, count, and delete operations.

**Code:**

*//Post lab: Doubly circular Linked List (DC-LL)*

#include <iostream>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node \*prev;

};

*//19BEC1112*

int Count(struct Node \*head)

{

    int count = 1;

    if (head == NULL)

    {

        return 0;

    }

    while (head->next != NULL)

    {

        head = head->next;

        count += 1;

    }

    return count;

}

struct Node \*InsertFront(struct Node \*head, int data)

{

    struct Node \*temp = new Node;

    if (head == NULL)

    {

        temp->data = data;

        temp->next = NULL;

        temp->prev = NULL;

        head = temp;

        return head;

    }

    temp->data = data;

    temp->prev = NULL;

    temp->next = head;

    head = temp;

    return head;

}

struct Node \*InsertMiddle(struct Node \*head, int value, int position)

{

    struct Node \*temp = new Node;

    temp->data = value;

    if (head == NULL)

    {

        temp->next = NULL;

        temp->prev = NULL;

        head = temp;

        return head;

    }

    else if (position == 1)

    {

        temp->data = value;

        temp->prev = NULL;

        temp->next = head;

        head = temp;

        return head;

    }

    else if (position == Count(head) + 1)

    {

        struct Node \*initial = head;

        temp->next = NULL;

        temp->data = value;

        while (head->next != NULL)

            head = head->next;

        temp->prev = head;

        head->next = temp;

        head = initial;

        return head;

    }

    struct Node \*initial = head;

    for (int i = 1; i < position - 1; i++)

    {

        head = head->next;

    }

    temp->prev = head;

    temp->next = head->next;

    head->next = temp;

    head = head->next;

    head = head->next;

    head->prev = temp;

    head = initial;

    return head;

}

struct Node \*InsertEnd(struct Node \*head, int value)

{

    struct Node \*temp = new Node;

    if (head == NULL)

    {

        temp->data = value;

        temp->next = NULL;

        temp->prev = NULL;

        head = temp;

        return head;

    }

    struct Node \*initial = head;

    temp->next = NULL;

    temp->data = value;

    while (head->next != NULL)

        head = head->next;

    temp->prev = head;

    head->next = temp;

    head = initial;

    return head;

}

struct Node \*DeleteLast(struct Node \*head)

{

    struct Node \*initial = head;

    while (head->next != NULL)

    {

        head = head->next;

    }

    head->prev->next = NULL;

    head = initial;

    return head;

}

struct Node \*DeleteFront(struct Node \*head)

{

    head->next->prev = NULL;

    head = head->next;

    return head;

}

struct Node \*DeleteMiddle(struct Node \*head, int position)

{

    if (Count(head) == 1)

    {

        head = NULL;

        return head;

    }

    else if (position == 1)

    {

        head->next->prev = NULL;

        head = head->next;

        return head;

    }

    else if (position == Count(head) - 1)

    {

        struct Node \*initial = head;

        while (head->next != NULL)

        {

            head = head->next;

        }

        head->prev->next = NULL;

        head = initial;

        return head;

    }

    struct Node \*initial = head;

    for (int i = 1; i < position - 1; i++)

    {

        head = head->next;

    }

    struct Node \*newNext = head->next->next;

    head->next = NULL;

    head->next = newNext;

    head = initial;

    return head;

}

void Print(struct Node \*head)

{

    struct Node \*n = head;

    while (head != NULL)

    {

        cout << head->data << endl;

        head = head->next;

    }

}

void Search(struct Node \*head, int value)

{

    if (head == NULL)

    {

        cout << "DLL is empty!" << endl;

        return;

    }

    bool found = false;

    int position = 1;

    while (head->next != NULL)

    {

        if (head->data == value)

        {

            found = true;

            break;

        }

        else

        {

            head = head->next;

            position += 1;

            continue;

        }

    }

    if (found == true)

    {

        cout << "Element is found at position: " << position << endl;

    }

    else {

        cout << "Element not found!" << endl;

    }

    return;

}

int main()

{

    struct Node \*head = new Node;

    head = NULL;

    cout << "Adding 11, 12, 13, 14, 15 and 16 to the doubly LL" << endl;

    head = InsertEnd(head, 11);

    head = InsertEnd(head, 12);

    head = InsertEnd(head, 13);

    head = InsertEnd(head, 14);

    head = InsertEnd(head, 15);

    head = InsertEnd(head, 16);

    Print(head);

    head = InsertFront(head, 10);

    cout << "Added 10 to the front" << endl;

    Print(head);

    cout << "Delete last element from doubly LL" << endl;

    head = DeleteLast(head);

    Print(head);

    cout << "Deleting front-most element from doubly LL" << endl;

    head = DeleteFront(head);

    Print(head);

    int data = 100;

    int position = 2;

    cout << "Inserting " << data << " at position " << position << " in the linked list" << endl;

    head = InsertMiddle(head, data, position);

    Print(head);

    cout << "Deleting element from position " << position << endl;

    head = DeleteMiddle(head, position);

    Print(head);

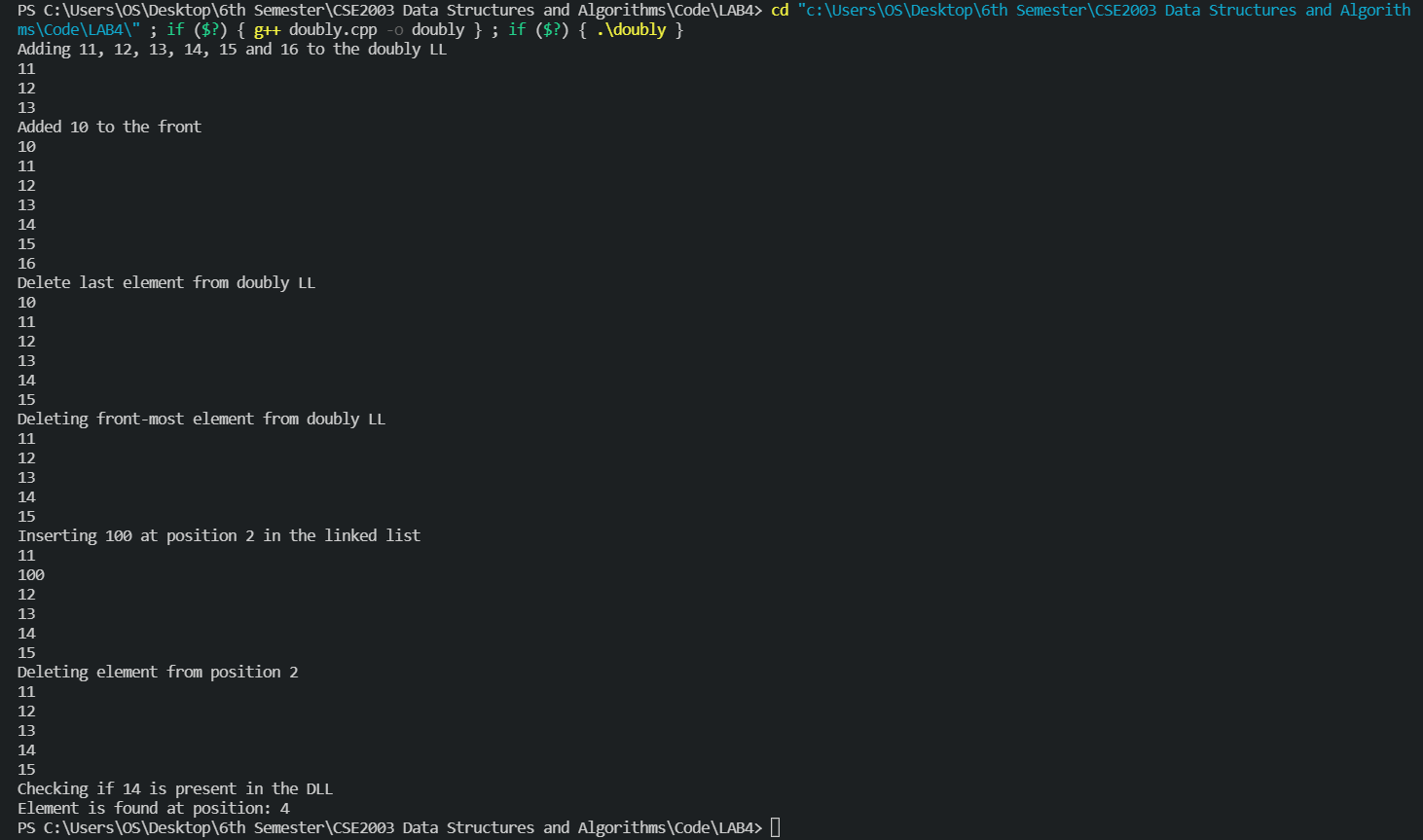
    cout << "Checking if 14 is present in the DLL" << endl;

    Search(head, 14);

    return 0;

}

**Output:**



**Task 2:** **Create a circular doubly linked list and perform the operations as done above.**

**Code:**

#include <iostream>

using namespace std;

struct Node

{

    int data;

    struct Node \*next;

    struct Node \*prev;

};

int GetLength(struct Node \*head)

{

    int length = 1;

    if (head == NULL)

        return 0;

    else if (head != NULL && head->next == NULL)

        return 1;

    struct Node \*initial = head;

    head = head->next;

    while (head != initial)

    {

        head = head->next;

        length += 1;

    }

    return length;

}

void Display(struct Node \*head)

{

    struct Node \*initial = head;

    if (head == NULL)

    {

        cout << "Circular DLL is empty!" << endl;

        return;

    }

    do

    {

        cout << head->data << endl;

        head = head->next;

    } while (head != initial);

}

struct Node \*InsertEnd(struct Node \*head, int data)

{

    struct Node \*temp = new Node;

    struct Node \*initial = head;

    temp->data = data;

    if (head == NULL)

    {

        temp->next = NULL;

        temp->prev = NULL;

        head = temp;

        return head;

    }

    else if (GetLength(head) == 1)

    {

        temp->prev = head;

        temp->next = head;

        head->next = temp;

        head->prev = temp;

        head = initial;

        return head;

    }

    temp->next = head;

    head->prev = temp;

    while (head->next != initial)

    {

        head = head->next;

    }

    temp->prev = head;

    head->next = temp;

    head = initial;

    return head;

}

struct Node \*InsertFront(struct Node \*head, int data)

{

    struct Node \*temp = new Node;

    struct Node \*initial = head;

    temp->data = data;

    if (head == NULL)

    {

        temp->next = NULL;

        temp->prev = NULL;

        head = temp;

        return head;

    }

    else if (GetLength(head) == 1)

    {

        temp->prev = head;

        temp->next = head;

        head->next = temp;

        head->prev = temp;

        head = temp;

        return head;

    }

    temp->next = head;

    head->prev = temp;

    while (head->next != initial)

    {

        head = head->next;

    }

    temp->prev = head;

    head->next = temp;

    head = temp;

    return head;

}

struct Node \*InsertAtPosition(struct Node \*head, int data, int position)

{

    struct Node \*initial = head;

    struct Node \*temp = new Node;

    temp->data = data;

    int count = 0;

    while (count < position - 2)

    {

        head = head->next;

        count += 1;

    }

    temp->prev = head;

    temp->next = head->next->next;

    head->next = temp;

    head = head->next;

    head = head->next;

    head->prev = temp;

    head = initial;

    return head;

}

struct Node \*DeleteFront(struct Node \*head)

{

    if (GetLength(head) == 1)

    {

        head = NULL;

        return head;

    }

    else if (GetLength(head) == 2)

    {

        head = head->next;

        head->prev = NULL;

        head->next = NULL;

        return head;

    }

    struct Node \*initial = head;

    struct Node \*NewInitial = head->next;

    head = head->prev;

    head->next = NewInitial;

    NewInitial->prev = head;

    head = NewInitial;

    return head;

}

struct Node \*DeleteEnd(struct Node \*head)

{

    if (GetLength(head) == 1)

    {

        head = NULL;

        return head;

    }

    else if (GetLength(head) == 2)

    {

        head->next = NULL;

        head->prev = NULL;

        return head;

    }

    struct Node \*initial = head;

    head = head->prev;

    head = head->prev;

    head->next = initial;

    initial->prev = head;

    head = initial;

    return head;

}

struct Node \*DeleteAtPosition(struct Node \*head, int position){

    struct Node \*initial = head;

    int count = 0;

    while (count < position - 2){

        head = head->next;

        count += 1;

    }

    struct Node \*temp = head->next->next;

    temp->prev = head;

    head->next = temp;

    head = initial;

    return head;

}

bool Search(struct Node \*head, int query){

    struct Node \*initial = head;

    do {

        if (head->data == query){

            return true;

        }

        head = head->next;

    } while(head != initial);

    return false;

}

int main()

{

    struct Node \*head = new Node;

    head = NULL;

    cout << "Inserting 10, 11, 12, 13 and 14 at the end of the circular DLL" << endl;

    head = InsertEnd(head, 10);

    head = InsertEnd(head, 11);

    head = InsertEnd(head, 12);

    head = InsertEnd(head, 13);

    head = InsertEnd(head, 14);

    Display(head);

    cout << "Inserting 8 and 9 at the front of the circular DLL" << endl;

    head = InsertFront(head, 9);

    head = InsertFront(head, 8);

    Display(head);

    cout << "Inserting 100 at 4th position of circular DLL" << endl;

    head = InsertAtPosition(head, 100, 4);

    Display(head);

    cout << "Deleting 3rd element from circular DLL" << endl;

    head = DeleteAtPosition(head, 3);

    Display(head);

    cout << "Deleting from front of circular DLL" << endl;

    head = DeleteFront(head);

    Display(head);

    cout << "Deleting from end of circular DLL" << endl;

    head = DeleteEnd(head);

    Display(head);

    cout << "Searching for 9 in the circular DLL" << endl;

    cout << "Searching..." << endl;

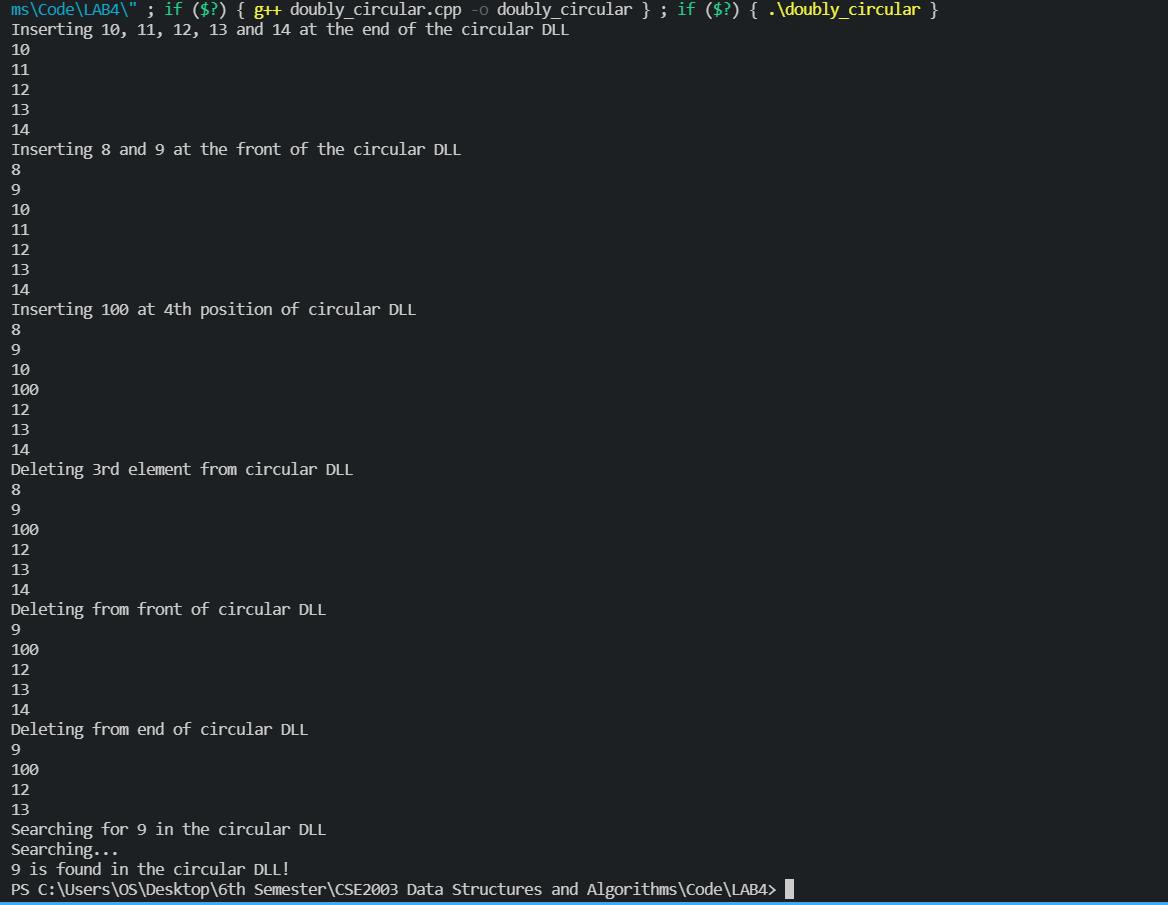
    if (Search(head, 9)) cout << "9 is found in the circular DLL!" << endl;

    else cout << "9 is NOT found!" << endl;

    return 0;

}

**Output:**



**Task 3:** **Using linked lists, perform addition, subtraction, multiplication and division of polynomials.**

**Code for addition:**

*//Post lab: Polynomial MUL and DIV*

#include <iostream>

#include <string.h>

using namespace std;

struct Node

{

    float coeff;

    int power;

    struct Node \*next;

};

string Display(struct Node \*head){

    string result = "";

    while (head != NULL){

        result += to\_string(head->coeff);

        result += "x^";

        result += to\_string(head->power);

        if (head->next != NULL) result += " + ";

        head = head->next;

    }

    return result;

}

int GetLength(struct Node \*head)

{

    int length = 1;

    if (head == NULL)

        return 0;

    while (head->next != NULL)

    {

        head = head->next;

        length += 1;

    }

    return length;

}

struct Node \*Create(struct Node \*head, float coeff, int power)

{

    struct Node \*temp = new Node;

    if (head == NULL)

    {

        temp->next = NULL;

        temp->power = power;

        temp->coeff = coeff;

        head = temp;

        return head;

    }

    struct Node \*initial = head;

    while (head->next != NULL)

    {

        head = head->next;

    }

    temp->coeff = coeff;

    temp->power = power;

    temp->next = NULL;

    head->next = temp;

    head = initial;

    return head;

}

struct Node \*AddPoly(struct Node \*f1, struct Node \*f2)

{

    int l1 = GetLength(f1);

    int l2 = GetLength(f2);

    struct Node \*initialf2 = f2;

    struct Node \*result = new Node;

    result = NULL;

    while (f1 != NULL)

    {

        bool commonPower = false;

        while (f2 != NULL)

        {

            if (f1->power == f2->power)

            {

                float resultCoeff = f1->coeff + f2->coeff;

                result = Create(result, resultCoeff, f1->power);

                commonPower = true;

                break;

            }

            else {

                f2 = f2->next;

            }

        }

        if (commonPower == false){

            result = Create(result, f1->coeff, f1->power);

        }

        f2 = initialf2;

        f1 = f1->next;

    }

    return result;

}

int main()

{

    struct Node \*head1 = new Node;

    struct Node \*head2 = new Node;

    head1 = NULL;

    head2 = NULL;

    cout << "---------- FIRST POLYNOMIAL ----------" << endl;

    char c1;

    do

    {

        float coeff;

        int power;

        cout << "Enter coefficient of node" << endl;

        cin >> coeff;

        cout << "Enter power of node (highest first)" << endl;

        cin >> power;

        head1 = Create(head1, coeff, power);

        cout << "Do you want to add more nodes? (y/n)" << endl;

        cin >> c1;

    } while (c1 != 'n');

    cout << "---------- SECOND POLYNOMIAL ----------" << endl;

    char c2;

    do

    {

        float coeff;

        int power;

        cout << "Enter coefficient of node" << endl;

        cin >> coeff;

        cout << "Enter power of node (highest first)" << endl;

        cin >> power;

        head2 = Create(head2, coeff, power);

        cout << "Do you want to add more nodes? (y/n)" << endl;

        cin >> c2;

    } while (c2 != 'n');

    struct Node \*result = new Node;

    result = NULL;

    result = AddPoly(head1, head2);

    cout << "First polynomial: " << Display(head1) << endl;

    cout << "Second polynomial: " << Display(head2) << endl;

    cout << "Resultant polynomial: " << Display(result) << endl;

}

**Output for addition:**



**Code for subtraction:**

*//Post lab: Polynomial MUL and DIV*

#include <iostream>

#include <string.h>

using namespace std;

struct Node

{

    float coeff;

    int power;

    struct Node \*next;

};

string Display(struct Node \*head){

    string result = "";

    while (head != NULL){

        result += to\_string(head->coeff);

        result += "x^";

        result += to\_string(head->power);

        if (head->next != NULL) result += " + ";

        head = head->next;

    }

    return result;

}

int GetLength(struct Node \*head)

{

    int length = 1;

    if (head == NULL)

        return 0;

    while (head->next != NULL)

    {

        head = head->next;

        length += 1;

    }

    return length;

}

struct Node \*Create(struct Node \*head, float coeff, int power)

{

    struct Node \*temp = new Node;

    if (head == NULL)

    {

        temp->next = NULL;

        temp->power = power;

        temp->coeff = coeff;

        head = temp;

        return head;

    }

    struct Node \*initial = head;

    while (head->next != NULL)

    {

        head = head->next;

    }

    temp->coeff = coeff;

    temp->power = power;

    temp->next = NULL;

    head->next = temp;

    head = initial;

    return head;

}

struct Node \*SubPoly(struct Node \*f1, struct Node \*f2)

{

    int l1 = GetLength(f1);

    int l2 = GetLength(f2);

    struct Node \*initialf2 = f2;

    struct Node \*result = new Node;

    result = NULL;

    while (f1 != NULL)

    {

        bool commonPower = false;

        while (f2 != NULL)

        {

            if (f1->power == f2->power)

            {

                float resultCoeff = f1->coeff - f2->coeff;

                result = Create(result, resultCoeff, f1->power);

                commonPower = true;

                break;

            }

            else {

                f2 = f2->next;

            }

        }

        if (commonPower == false){

            result = Create(result, f1->coeff, f1->power);

        }

        f2 = initialf2;

        f1 = f1->next;

    }

    return result;

}

int main()

{

    struct Node \*head1 = new Node;

    struct Node \*head2 = new Node;

    head1 = NULL;

    head2 = NULL;

    cout << "---------- FIRST POLYNOMIAL ----------" << endl;

    char c1;

    do

    {

        float coeff;

        int power;

        cout << "Enter coefficient of node" << endl;

        cin >> coeff;

        cout << "Enter power of node (highest first)" << endl;

        cin >> power;

        head1 = Create(head1, coeff, power);

        cout << "Do you want to add more nodes? (y/n)" << endl;

        cin >> c1;

    } while (c1 != 'n');

    cout << "---------- SECOND POLYNOMIAL ----------" << endl;

    char c2;

    do

    {

        float coeff;

        int power;

        cout << "Enter coefficient of node" << endl;

        cin >> coeff;

        cout << "Enter power of node (highest first)" << endl;

        cin >> power;

        head2 = Create(head2, coeff, power);

        cout << "Do you want to add more nodes? (y/n)" << endl;

        cin >> c2;

    } while (c2 != 'n');

    struct Node \*result = new Node;

    result = NULL;

    result = SubPoly(head1, head2);

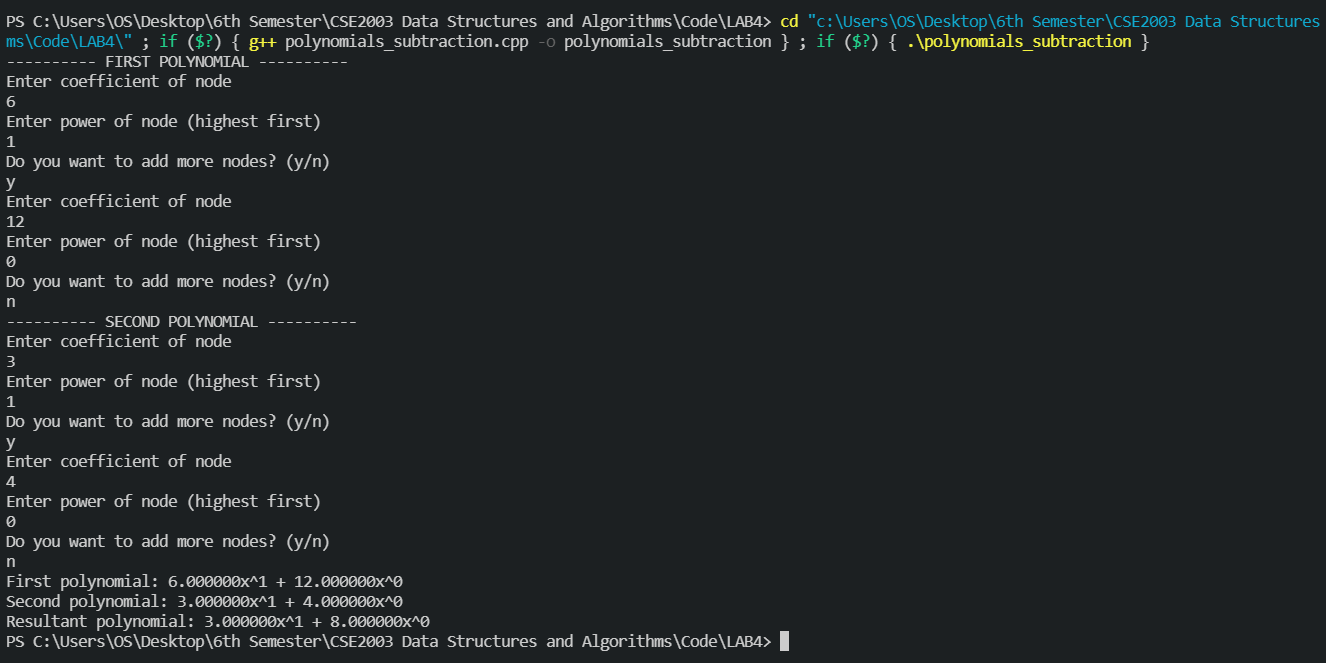
    cout << "First polynomial: " << Display(head1) << endl;

    cout << "Second polynomial: " << Display(head2) << endl;

    cout << "Resultant polynomial: " << Display(result) << endl;

}

**Output for subtraction:**



**Code for multiplication:**

*// Post lab: Polynomial MUL and DIV*

#include <iostream>

#include <string.h>

using namespace std;

struct Node

{

    float coeff;

    int power;

    struct Node \*next;

};

string Display(struct Node \*head)

{

    string result = "";

    while (head != NULL)

    {

        result += to\_string(head->coeff);

        result += "x^";

        result += to\_string(head->power);

        if (head->next != NULL)

            result += " + ";

        head = head->next;

    }

    return result;

}

int GetLength(struct Node \*head)

{

    int length = 1;

    if (head == NULL)

        return 0;

    while (head->next != NULL)

    {

        head = head->next;

        length += 1;

    }

    return length;

}

struct Node \*Create(struct Node \*head, float coeff, int power)

{

    struct Node \*temp = new Node;

    if (head == NULL)

    {

        temp->next = NULL;

        temp->power = power;

        temp->coeff = coeff;

        head = temp;

        return head;

    }

    struct Node \*initial = head;

    while (head->next != NULL)

    {

        head = head->next;

    }

    temp->coeff = coeff;

    temp->power = power;

    temp->next = NULL;

    head->next = temp;

    head = initial;

    return head;

}

struct Node \*MulPoly(struct Node \*f1, struct Node \*f2)

{

    int l1 = GetLength(f1);

    int l2 = GetLength(f2);

    struct Node \*initialf2 = f2;

    struct Node \*result = new Node;

    result = NULL;

    while (f1 != NULL)

    {

        while (f2 != NULL)

        {

            float resultCoeff = f1->coeff \* f2->coeff;

            result = Create(result, resultCoeff, f1->power + f2->power);

            f2 = f2->next;

        }

        f2 = initialf2;

        f1 = f1->next;

    }

    return result;

}

int main()

{

    struct Node \*head1 = new Node;

    struct Node \*head2 = new Node;

    head1 = NULL;

    head2 = NULL;

    cout << "---------- FIRST POLYNOMIAL ----------" << endl;

    char c1;

    do

    {

        float coeff;

        int power;

        cout << "Enter coefficient of node" << endl;

        cin >> coeff;

        cout << "Enter power of node (highest first)" << endl;

        cin >> power;

        head1 = Create(head1, coeff, power);

        cout << "Do you want to add more nodes? (y/n)" << endl;

        cin >> c1;

    } while (c1 != 'n');

    cout << "---------- SECOND POLYNOMIAL ----------" << endl;

    char c2;

    do

    {

        float coeff;

        int power;

        cout << "Enter coefficient of node" << endl;

        cin >> coeff;

        cout << "Enter power of node (highest first)" << endl;

        cin >> power;

        head2 = Create(head2, coeff, power);

        cout << "Do you want to add more nodes? (y/n)" << endl;

        cin >> c2;

    } while (c2 != 'n');

    struct Node \*result = new Node;

    result = NULL;

    result = MulPoly(head1, head2);

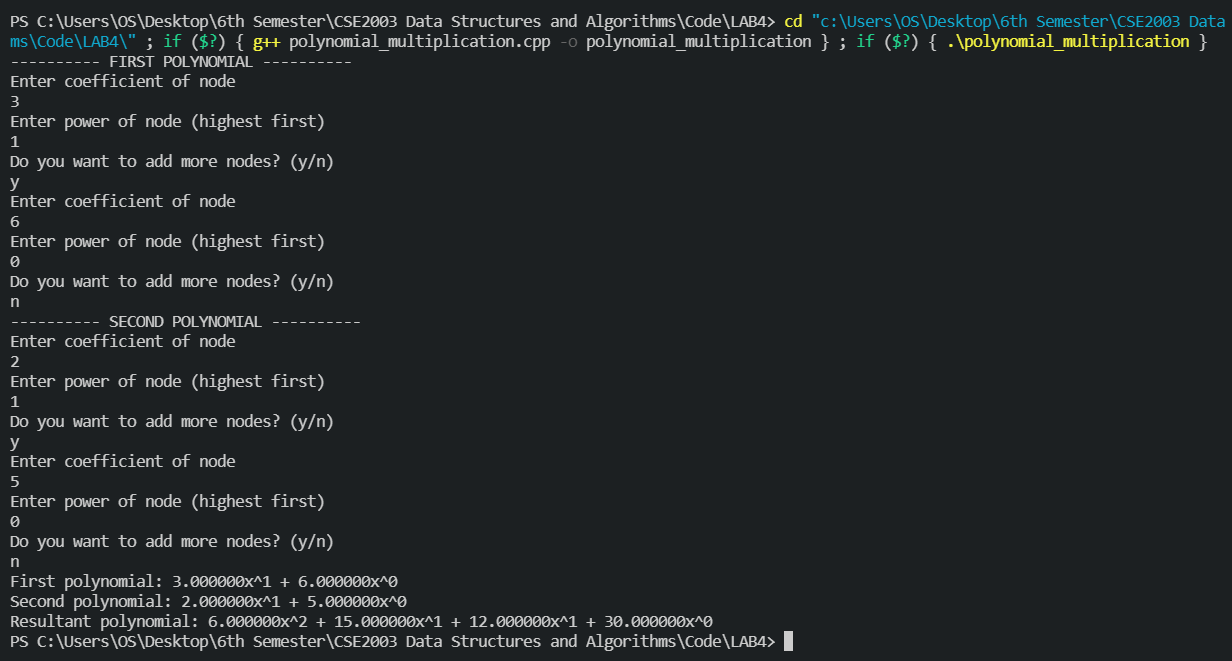
    cout << "First polynomial: " << Display(head1) << endl;

    cout << "Second polynomial: " << Display(head2) << endl;

    cout << "Resultant polynomial: " << Display(result) << endl;

}

**Output for multiplication:**



**Code for division:**

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

using namespace std;

struct Node {

    float coeff;

    int pow;

    struct Node\* next;

};

void create\_node(float 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 store\_quotient(float mul\_c, int diff,

                    struct Node\* quo)

{

    while (quo->next != NULL) {

        quo = quo->next;

    }

    quo->pow = diff;

    quo->coeff = mul\_c;

    quo->next = (struct Node\*)malloc(

        sizeof(struct Node));

    quo = quo->next;

    quo->next = NULL;

}

void formNewPoly(int diff, float mul\_c,

                struct Node\* poly)

{

    while (poly->next != NULL) {

        poly->pow += diff;

        poly->coeff \*= mul\_c;

        poly = poly->next;

    }

}

void copyList(struct Node\* r,

            struct Node\*\* copy)

{

    while (r != NULL) {

        struct Node\* z

            = (struct Node\*)malloc(

                sizeof(struct Node));

        z->coeff = r->coeff;

        z->pow = r->pow;

        z->next = NULL;

        struct Node\* dis = \*copy;

        if (dis == NULL) {

            \*copy = z;

        }

        else {

            while (dis->next != NULL) {

                dis = dis->next;

            }

            dis->next = z;

        }

        r = r->next;

    }

}

void polySub(struct Node\* poly1,

            struct Node\* poly2,

            struct Node\* poly)

{

    while (poly1->next && poly2->next) {

        if (poly1->pow > poly2->pow) {

            poly->pow = poly1->pow;

            poly->coeff = poly1->coeff;

            poly1 = poly1->next;

            poly->next

                = (struct Node\*)malloc(

                    sizeof(struct Node));

            poly = poly->next;

            poly->next = NULL;

        }

        else if (poly1->pow < poly2->pow) {

            poly->pow = poly2->pow;

            poly->coeff = -1 \* poly2->coeff;

            poly2 = poly2->next;

            poly->next

                = (struct Node\*)malloc(

                    sizeof(struct Node));

            poly = poly->next;

            poly->next = NULL;

        }

        else {

            if ((poly1->coeff

                - poly2->coeff)

                != 0) {

                poly->pow = poly1->pow;

                poly->coeff = (poly1->coeff

                            - poly2->coeff);

                poly->next = (struct Node\*)malloc(

                    sizeof(struct Node));

                poly = poly->next;

                poly->next = NULL;

            }

            poly1 = poly1->next;

            poly2 = poly2->next;

        }

    }

    while (poly1->next || poly2->next) {

        if (poly1->next) {

            poly->pow = poly1->pow;

            poly->coeff = poly1->coeff;

            poly1 = poly1->next;

        }

        if (poly2->next) {

            poly->pow = poly2->pow;

            poly->coeff = -1 \* poly2->coeff;

            poly2 = poly2->next;

        }

        poly->next

            = (struct Node\*)malloc(

                sizeof(struct Node));

        poly = poly->next;

        poly->next = NULL;

    }

}

void show(struct Node\* node)

{

    int count = 0;

    while (node->next != NULL

        && node->coeff != 0) {

        if (count == 0)

            cout << node->coeff;

        else

            cout << abs(node->coeff);

        count++;

        if (node->pow != 0)

            cout << "x^" << node->pow;

        node = node->next;

        if (node->next != NULL)

            if (node->coeff > 0)

                cout << " + ";

            else

                cout << " - ";

    }

    cout << "\n";

}

void divide\_poly(struct Node\* poly1,

                struct Node\* poly2)

{

    struct Node \*rem = NULL, \*quo = NULL;

    quo = (struct Node\*)malloc(

        sizeof(struct Node));

    quo->next = NULL;

    struct Node \*q = NULL, \*r = NULL;

    copyList(poly1, &q);

    copyList(poly2, &r);

    while (q != NULL

        && (q->pow >= poly2->pow)) {

        int diff = q->pow - poly2->pow;

        float mul\_c = (q->coeff

                    / poly2->coeff);

        store\_quotient(mul\_c, diff,

                    quo);

        struct Node\* q2 = NULL;

        copyList(r, &q2);

        formNewPoly(diff, mul\_c, q2);

        struct Node\* store = NULL;

        store = (struct Node\*)malloc(

            sizeof(struct Node));

        polySub(q, q2, store);

        q = store;

        free(q2);

    }

    cout << "Quotient: ";

    show(quo);

    cout << "Remainder: ";

    rem = q;

    show(rem);

}

int main()

{

    struct Node\* poly1 = NULL;

    struct Node \*poly2 = NULL, \*poly = NULL;

*// Create 1st Polynomial (Dividend):*

*// 5x^2 + 4x^1 + 2*

    cout << "--------First polynmomial--------" << endl;

    cout << "5x^2 + 4x^1 + 2" << endl;

    create\_node(5.0, 2, &poly1);

    create\_node(4.0, 1, &poly1);

    create\_node(2.0, 0, &poly1);

    cout << "--------Second polynmomial--------" << endl;

    cout << "2x^1 + 2" << endl;

    create\_node(2.0, 1, &poly2);

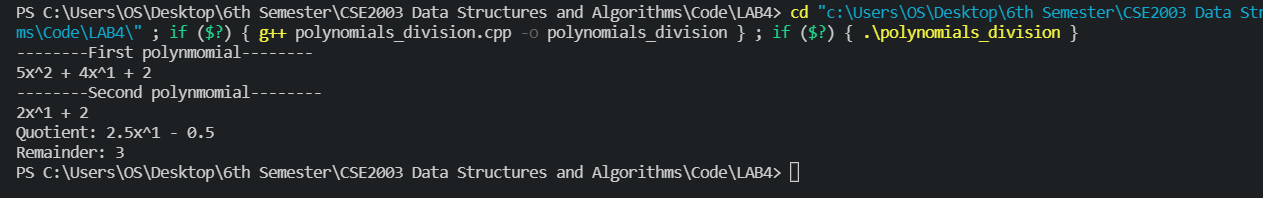
    create\_node(2.0, 0, &poly2);

    divide\_poly(poly1, poly2);

    return 0;

}

**Output for division:**



**Conclusion**

Thus, we have implemented linear and circular doubly linked lists, and also seen how they can be used for simple applications such as polynomial addition, subtraction, multiplication and division.

*Neh Samir Joshi*

*19BEC1112*