PIMPRI CHINCHWAD EDUCATION TRUST's.

PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

(An Autonomous Institute)

Class : SY BTech **Acad. Yr. 2025-26** **Semester : I**

Name of the student: Om Jitendra Khalane **PRN : 124B1B040**

Department: Computer Engineering **Division : A**

Course Name : Data Structures and Laboratory **Course Code: BCE23PC02**

Completion Date : 06/08/2025

Assignment No. 05

Problem Statement:

Write a program to perform Polynomial Addition using Linked Lists

- Each term is a node (with coefficient and power).
- Add two polynomials represented by linked lists.

Source Code :

https://github.com/omkhalane/DSAL-SY-PCCOE/blob/main/lab_assignments/assignment05.cpp

```cpp
#include <iostream>
using namespace std;

class node
{
public:
    int power;
    int coeff;
    node *next;
    node(int coef, int p)
    {
        power = p;
        coeff = coef;
        next = NULL;
    }
```

```
};

class LL
{
public:
    node *head, *tail;

    LL()
    {
        head = NULL;
        tail = NULL;
    }

    void insert_var(int coef, int p)
    {
        node *temp = new node(coef, p);
        if (!head)
        {
            head = temp;
            tail = temp;
        }
        else
        {
            tail->next = temp;
            tail = temp;
        }
    }

    void print()
    {
        if (!head)
        {
            cout << "Empty Polynomial\n";
            return;
        }
        node *it = head;
        bool first = true;
        while (it)
        {
            if (!first && it->coeff >= 0)
                cout << " + ";
            cout << it->coeff << "x^" << it->power;
            it = it->next;
            first = false;
        }
```

```
        cout << endl;
    }

    void add_ply(LL &poly1, LL &poly2)
    {
        node *t1 = poly1.head;
        node *t2 = poly2.head;

        while (t1 && t2)
        {
            if (t1->power == t2->power)
            {
                insert_var(t1->coeff + t2->coeff, t1->power);
                t1 = t1->next;
                t2 = t2->next;
            }
            else if (t1->power > t2->power)
            {
                insert_var(t1->coeff, t1->power);
                t1 = t1->next;
            }
            else
            {
                insert_var(t2->coeff, t2->power);
                t2 = t2->next;
            }
        }
        while (t1)
        {
            insert_var(t1->coeff, t1->power);
            t1 = t1->next;
        }
        while (t2)
        {
            insert_var(t2->coeff, t2->power);
            t2 = t2->next;
        }
    }
};

int main()
{
    LL p1, p2, p3;
    int choice;
```

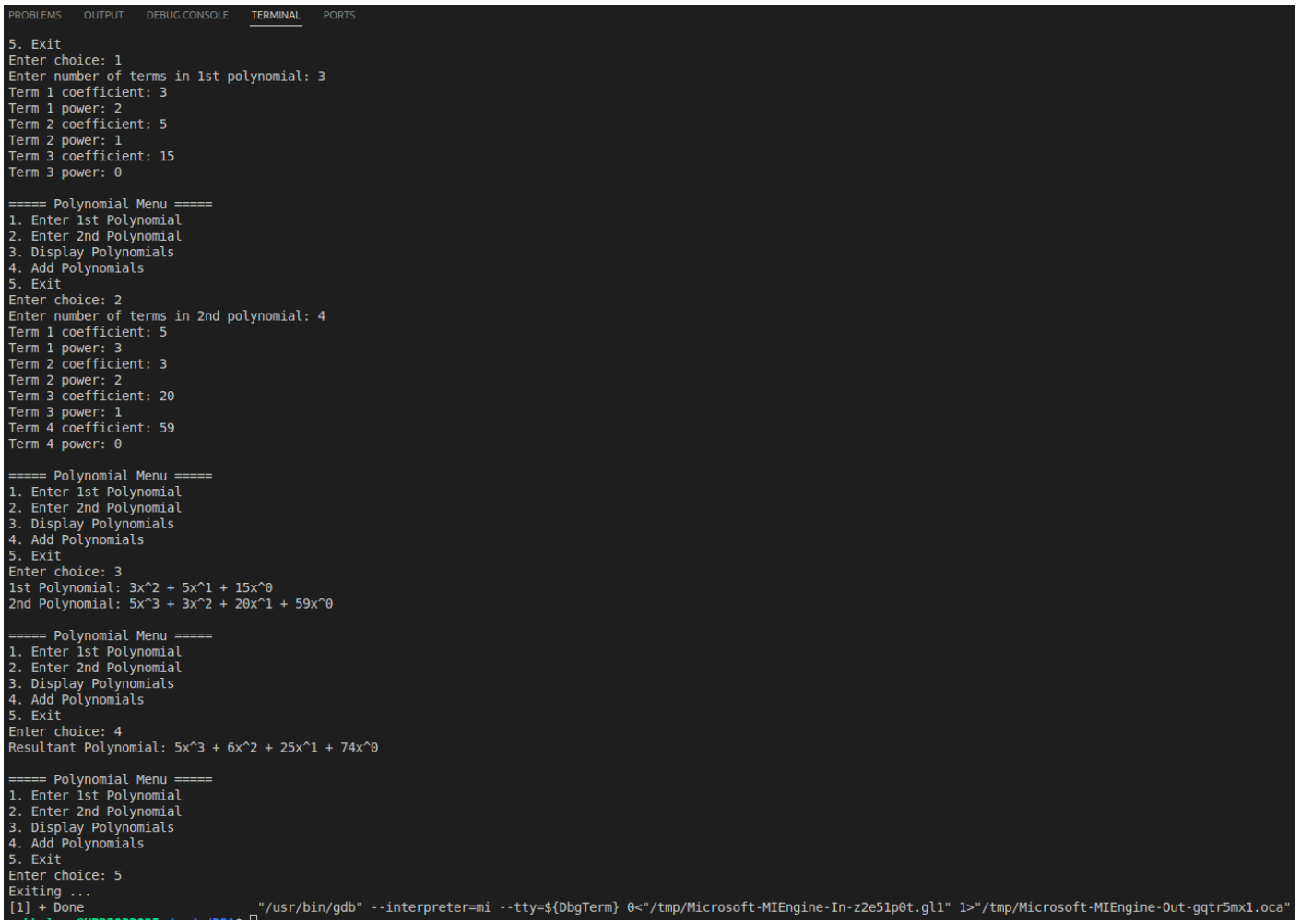
```
        do
        {
            cout << "\n===== Polynomial Menu =====\n";
            cout << "1. Enter 1st Polynomial\n";
            cout << "2. Enter 2nd Polynomial\n";
            cout << "3. Display Polynomials\n";
            cout << "4. Add Polynomials\n";
            cout << "5. Exit\n";
            cout << "Enter choice: ";
            cin >> choice;

            switch (choice)
            {
            case 1:
            {
                int t;
                cout << "Enter number of terms in 1st polynomial: ";
                cin >> t;
                p1 = LL();
                for (int i = 0; i < t; i++)
                {
                    int coef, pow;
                    cout << "Term " << i + 1 << " coefficient: ";
                    cin >> coef;
                    cout << "Term " << i + 1 << " power: ";
                    cin >> pow;
                    p1.insert_var(coef, pow);
                }
                break;
            }
            case 2:
            {
                int t;
                cout << "Enter number of terms in 2nd polynomial: ";
                cin >> t;
                p2 = LL();
                for (int i = 0; i < t; i++)
                {
                    int coef, pow;
                    cout << "Term " << i + 1 << " coefficient: ";
                    cin >> coef;
                    cout << "Term " << i + 1 << " power: ";
                    cin >> pow;
                    p2.insert_var(coef, pow);
                }
```

```
            break;
        }
        case 3:
            cout << "1st Polynomial: ";
            p1.print();
            cout << "2nd Polynomial: ";
            p2.print();
            break;
        case 4:
            p3 = LL();
            p3.add_ply(p1, p2);
            cout << "Resultant Polynomial: ";
            p3.print();
            break;
        case 5:
            cout << "Exiting ..." << endl;
            break;
        default:
            cout << "Invalid choice" << endl;
        }
    } while (choice != 5);

    return 0;
}
```

Screen Shot of Output :



Conclusion:

The implemented program performs polynomial addition using a singly linked list representation, where each term is stored as a node containing its coefficient and power. This approach allows efficient handling of polynomials with varying degrees and numbers of terms without requiring fixed-size arrays.

- Time Complexity:
    - Insertion: O(1)O(1)O(1) when using the tail pointer for adding terms sequentially.
    - Polynomial Addition: O(m+n)O(m + n)O(m+n), where mmm and nnn are the number of terms in the two polynomials, as each term is processed exactly once.
    - Display Polynomial: O(k)O(k)O(k), where kkk is the number of terms in the polynomial.
- Space Complexity:

- O(m+n)O(m + n)O(m+n) — Each term of both polynomials is stored in a node, requiring memory proportional to the total number of terms.

- Observations:
  - The linked list representation provides dynamic memory allocation, allowing polynomials of arbitrary size and degree.
  - Terms are added in descending order of powers, ensuring correct representation of the result.
  - This approach avoids shifting elements, unlike array-based polynomial storage.
  - For improved efficiency, especially in operations like subtraction or multiplication, the same structure can be extended.

This program effectively demonstrates the application of linked lists to represent and manipulate mathematical expressions in a structured and memory-efficient manner.