NEW HORIZON
COLLEGE OF ENGINEERING

AUTONOMOUS COLLEGE Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade

POLYNOMIAL CALCULATOR

A MINI PROJECT

REPORT

Submitted by

RAHUL MUSALIYATH DINESH

In partial fulfillment for the award of
the degree of

Bachelor of Engineering

in

COMPUTER SCIENCE AND ENGINEERING

NEW HORIZON
COLLEGE OF ENGINEERING

AUTONOMOUS COLLEGE Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade

Certificate

This is to certify that the mini project work titled

POLYNOMIAL CALCULATOR

Submitted in partial fulfillment of the degree of
Bachelor of Engineering in
Computer Science and Engineering by

RAHUL MUSALIYATH DINESH
USN: 1NH18CS738

DURING

ODD SEMESTER 2019-2020

for

COURSE CODE: 19CSE39

Signature of Reviewer Signature of HOD

SEMESTER END EXAMINATION

Name of the Examiner	*Signature with date*
1. _____	_____
2. _____	_____

ABSTRACT

A polynomial is one of the most powerful yet simple forms of mathematical expressions that play a vital role in developing many of the things we use in our daily life. From mathematics to physics, electronics to computer algorithms, polynomials are almost present everywhere.

We use polynomials to describe curves of various types and to graph curves. For example, roller coaster architects may use polynomials to describe the curves in their rides. Economists and statisticians also use combinations of polynomial functions to do cost analyses. Engineers use polynomials to graph the curves of roller coasters, bridges and other structures. Trajectories of projectiles can also be described using polynomials. Polynomial integrals (i.e. the sums of many polynomials) are used to express voltage difference, inertia and energy to name a few applications.

This mini project is on a polynomial calculator which can perform basic operations on polynomials like addition, subtraction, multiplication and evaluation. This mini project has used the data structure concept of a single linked list to implement a polynomial expression and to perform the above-mentioned operations. Various user defined functions have been used throughout the project to break up the program into various sub programs for the sake of easiness and understanding.

The interface is simple to use and ensures that the user can confirm what he/she has inputted via a *display* user defined function. The user will be initially requested to choose the operation to be performed. Then he/she will be asked to enter the number of terms in the polynomial following which he/she will be asked to enter the coefficients and its corresponding power. The user will be shown the polynomial(s) he or she has entered prior to displaying the answer of the requested operation. The entire program has been written in the *C* language.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman of New Horizon Educational Institutions for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha,** Principal, New Horizon College of Engineering, for his constant support and encouragement.

I am grateful to **Dr. Prashanth C.S.R.**, Dean Academics, for his unfailing encouragement and suggestions, given to me in the course of my project work.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and Head, Department of Computer Science and Engineering, for her constant support.

I also express my gratitude to **Mr. Manjunatha Swamy C,** Senior Assistant Professor, Department of Computer Science and Engineering, my project guide, for constantly monitoring the development of the project and setting up precise deadlines. His valuable suggestions were the motivating factors in completing the work.

RAHUL MUSALIYATH DINESH

USN: 1NH18CS738

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 WHAT IS A POLYNOMIAL?

A polynomial is an expression of more than two algebraic terms, especially the sum of several terms that contain different powers of the same variable(s) as per a simple google search. Polynomial comes from poly, meaning "many" and nomial, meaning "term", hence a polynomial simply means "many terms".

An example of a polynomial of one variable x is,

$$x^2 - 5x + 8.$$

An example of a polynomial in three variables is,

$$x^3 + 6xyz^2 - 2yz + 3.$$

Polynomials can be divided into a number of *terms*, separated by each part that is being added. Polynomial terms do not have square roots of variables, fractional powers, nor does it have variables in the denominator of any fractions it may have. The polynomial terms have variables with exponents that are whole-numbers.

In general, polynomials are written with its terms being ordered in the decreasing order of exponents. The term with the highest exponent goes first, followed by the term with the next highest exponent and so on till you reach a constant term or the lowest power.

Polynomials are widely used in many areas of mathematics and science. For example, they are used to form polynomial expressions and equations, which encode a wide range of problems, from elementary school word problems to complicated scientific expressions; they are used to define polynomial functions, which appear in settings ranging from basic chemistry and physics to economics and social science; they are also used in calculus and numerical analysis to approximate other functions.

In this mini project, we will be using polynomials of a single variable, x, for performing the various operations such as addition, subtraction, multiplication and evaluation using the higher order data structure concepts like linked lists.

1.2 PROBLEM DEFINITION

Given two polynomial expressions, this mini project aims to perform the basic operations like addition, subtraction, multiplication and evaluation. The user will input the polynomial expression and the value of *x* (only for evaluation option). This mini project aims to use polynomials of a single variable to perform the above-mentioned operations.

1.3 OBJECTIVES

The mini project upon completion aims to provide the addition, subtraction, multiplication or evaluation result of two or one polynomial(s) as per the choice of the user. This mini project would use the data structure concept of a Single Linked List.

1.4 HARDWARE AND SOFTWARE REQUIREMENTS

- Processor : Intel 386 or higher
- RAM : 4 MB or higher
- Hard Disk : 25 MB or higher
- Input device : Standard Keyboard and Mouse
- Output device : VGA and High Resolution Monitor or higher
- Operating system : Microsoft DOS, Microsoft Windows 3.1 or later
- IDE : Turbo C

CHAPTER 2

IMPLEMENTING POLYNOMIALS USING LINKED LISTS

2.1 PARTS OF A POLYNOMIAL

An essential characteristic of the polynomial is that each term in the polynomial expression consists of two parts:

- one is the coefficient
- other is the exponent

For example, in the expression $10x^2 + 26x$,

here 10 and 26 are coefficients and 2, 1 is its exponential value.

2.1 REPRESENTATION OF A POLYNOMIAL USING LINKED LIST

A polynomial can be thought of as an ordered list of non-zero terms. It is implemented using dynamic memory allocation via a Single Linked List. Each non zero term can be thought of as a tuple which holds three pieces of information:

- The exponent part
- The coefficient part
- The link to the next tuple

For example,

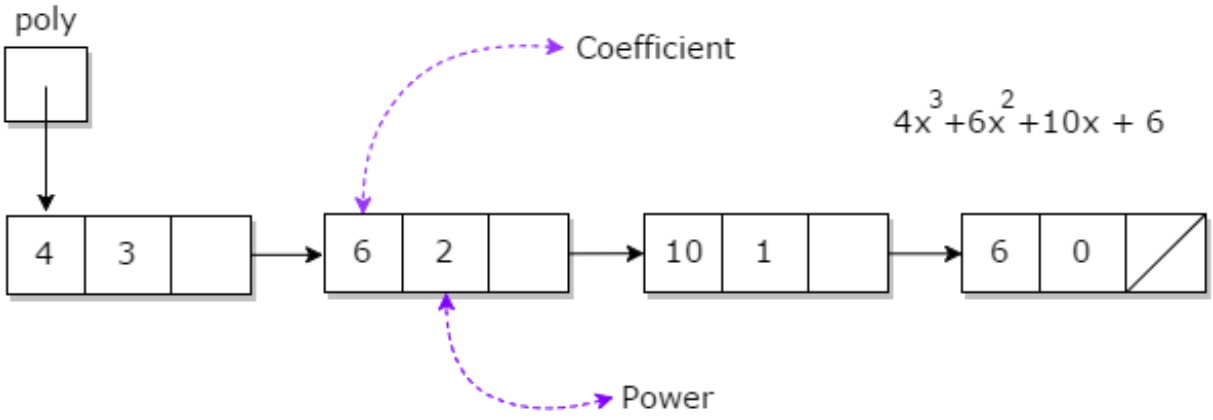


Figure 1: Implementation of a Polynomial using Single Linked List

CHAPTER 3

DESIGN

3.1 DESIGN GOALS

This mini project has ensured that the user has an interactive and explorable environment. The interface is user friendly, simple to understand and has tried to ensure that there are no bugs.

I have tried to ensure that the logics / algorithms used are simple yet logical without compromising on the final output.

Various user-defined functions have been used across the mini project so that the number of lines in the program is reduced and also for sake of simplicity and reading.

3.1 ALGORITHM – PSEUDOCODE

Creation of Polynomial

```
void create (struct node *first)
{
      int coef, pow, i, n;
      struct node *curr;

      curr = first;

      printf ("Enter the number of terms in the polynomial: ");
      scanf ("%d", &n);

      for (i = 0; i < n; i++)
      {
          printf ("\tEnter %d term's coefficient & power: ",i+1);
          scanf ("%d%d", &coef, &pow);
```

```
          curr->coef = coef;
          curr->pow = pow;

          if (i < n-1)
          {
            curr->link = (struct node*)malloc(sizeof (struct node));
            curr = curr->link;
          }
          else
          {
               curr->link = NULL;
          }

      }
}
```

Sorting Algorithm for arranging terms in Descending order

I have used the *bubble sort* technique to sort out the terms of the polynomial entered by the user in the descending order of their respective powers.

I have also defined a swap function, to exchange the data of two polynomial terms, when they are passed as arguments to it.

A flag named 'swapped' is used in the do-while loop. A while loop is used within the do-while loop to compare the powers of two adjacent terms.

```
void sort (struct node *first)
{
    int swapped, i;
    struct node *temp;

    if (first == NULL)
      return;
```

```
    do
    {
      swapped = 0;
      temp = first;

      while (temp->link != NULL)
      {
          if (temp->pow < temp->link->pow)
          {
            swap (temp, temp->link);
            swapped = 1;
          }

          temp = temp->link;
      }
    }
    while (swapped);
}

void swap (struct node *a, struct node *b)

{
    int temp1, temp2;

    temp1 = a->pow;
    a->pow = b->pow;
    b->pow = temp1;

    temp2 = a->coef;
    a->coef = b->coef;
    b->coef = temp2;
}
```

Displaying the Polynomial

In this function, the *first* node of a polynomial is passed to it as an argument. It then assigns *first* to a temporary pointer *temp* which will traverse the linked list till the last node.

While traversing, there are two conditions; one is when *temp* is at the last node and the other one when *temp* is at all other nodes except the last.

```
void display (struct node *first)
{
      struct node* temp;
      temp = first;
      while (1)
      {
          if (temp->link == NULL)
          {
            if (temp->pow == 0)
            {
                printf ("%d", temp->coef);
                break;
            }

            if (temp->pow == 1)
            {
                printf ("%dx", temp->coef);
                break;
            }
            else
            {
                printf ("%dx^%d", temp->coef, temp->pow);
                break;
            }
          }
```

```
          else if (temp->pow == 1)
            printf ("%dx + ", temp->coef);

          else if (temp->pow == 0)
            printf("%d + ", temp->coef);

          else
            printf ("%dx^%d + ", temp->coef, temp->pow);

          temp = temp->link;

      }
}
```

Addition of two Polynomials

In this function, the *first* nodes (*poly1* and *poly2*) of two polynomials are passed to it as arguments. It then assigns *poly1* to a current pointer *curr1* which will traverse through the first polynomial. Likewise, *poly2* is assigned to *curr2*.

The function then finds the highest power among the two polynomials by traversing through it with the help of a variable, *highestPower* using a while loop.

Then using a for loop, all the coefficients of like powers in both polynomials are added and stored in another linked list *polysum* with the help of a current pointer called *currentNode*.

Then the resultant linked list, *polysum* is displayed by passing it as an argument to the display function.

```
void add (struct node *poly1, struct node *poly2)
{
      int highestPower = 0, coeff, i;
      struct node *polysum = NULL, *currentNode, *curr1, *p2;

      printf ("\nFirst Polynomial: ");
      display(poly1);
      printf ("\nSecond Polynomial: ");
      display(poly2);

      curr1 = poly1;
      curr2 = poly2;

      while (1)
      {
            if (curr1->pow > highestPower)
            {
                  highestPower = curr1->pow;
            }

            if (curr1->link == NULL)
            {
                  break;
            }

            curr1 = curr1->link;
      }

      while (1)
      {
            if (curr2->pow > highestPower)
            {
```

```
                highestPower = curr2->pow;
        }

        if (curr2->link == NULL)
        {
                break;
        }

        curr2 = curr2->link;
    }

    for (i = highestPower; i >= 0; i--)
    {
        coeff = 0;

        curr1 = poly1;
        curr2 = poly2;

        while (1)
        {
                if (curr1->pow == i)
                {
                        coeff = coeff + curr1->coef;
                }

                if (curr1->link == NULL)
                {
                        break;
                }

                curr1 = curr1->link;
        }
```

```
while (1)
{
      if (curr2->pow == i)
      {
            coeff = coeff + curr2->coef;
      }
      if (curr2->link == NULL)
      {
            break;
      }
      curr2 = curr2->link;
}
if (coeff != 0)
{
      if (polysum == NULL)
      {
            polysum = (struct node*) malloc (sizeof
            (struct node));
            polysum->coef = coeff;
            polysum->pow = i;
            polysum->link = NULL;
            currentNode = polysum;
      }
      else
      {
            currentNode->link = (struct node*) malloc
            (sizeof (struct node));
            currentNode = currentNode->link;
            currentNode->coef = coeff;
            currentNode->pow = i;
            currentNode->link = NULL;
      }
}
```

```
      }
      printf ("\nThe polynomial sum is: ");
      display (polysum);
}
```

Subtraction of two Polynomials

In this function, like we saw in the add function, the *first* nodes (*poly1* and *poly2*) of two polynomials are passed to the subtract function as arguments. It then assigns *poly1* to a current pointer *curr1* which will traverse through the first polynomial. Likewise, *poly2* is assigned to *curr2*.

The function then finds the highest power among the two polynomials by traversing through it with the help of a variable, *highestPower* using a while loop.

Then using a for loop, all the coefficients of like powers is evaluated by subtracting *poly2'*s coefficient from *poly1*'s coefficient and the resultant difference of the coefficients is stored in another linked list *polydiff* with the help of a current pointer called *currentNode*.

Then the resultant linked list, *polydiff* is displayed by passing it as an argument to the display function.

```
void subtract (struct node *poly1, struct node *poly2)
{
      int highestPower = 0, coeff, i;
      struct node *polydiff = NULL, *currentNode, *curr1, *curr2;

      printf ("\nFirst Polynomial: ");
      display(poly1);
      printf ("\nSecond Polynomial: ");
```

```
	display(poly2);

	curr1 = poly1;
	curr2 = poly2;

	while (1)
	{
		if (curr1->pow > highestPower)
		{
			highestPower = curr1->pow;
		}

		if (curr1->link == NULL)
		{
			break;
		}

		curr1 = curr1->link;
	}

	while (1)
	{
		if (curr2->pow > highestPower)
		{
			highestPower = curr2->pow;
		}
		if (curr2->link == NULL)
		{
			break;
		}

		curr2 = curr2->link;
	}
```

```
	for (i = highestPower; i >= 0; i--)
	{
		coeff = 0;
		curr1 = poly1;
		curr2 = poly2;
		while (1)
		{
			if (curr1->pow == i)
			{
				coeff = coeff + curr1->coef;
			}

			if (curr1->link == NULL)
			{
				break;
			}

			curr1 = curr1->link;
		}
		while (1)
		{
			if (curr2->pow == i)
			{
				coeff = coeff - curr2->coef;
			}

			if (curr2->link == NULL)
			{
				break;
			}

			curr2 = curr2->link;
		}
```

```
      if (coeff != 0)
            {
                  if (polydiff == NULL)
                  {
                        polydiff = (struct node*) malloc (sizeof
                        (struct node));
                        polydiff->coef = coeff;
                        polydiff->pow = i;
                        polydiff->link = NULL;
                        currentNode = polydiff;
                  }
                  else
                  {
                        currentNode->link = (struct node*) malloc
                        (sizeof (struct node));
                        currentNode = currentNode->link;
                        currentNode->coef = coeff;
                        currentNode->pow = i;
                        currentNode->link = NULL;
                  }
            }
      }

      printf ("\nThe polynomial difference is: ");
      display (polydiff);
}
```

Multiplication of two Polynomials

In this function, like we saw in the add function, the *first* nodes (*poly1* and *poly2*) of two polynomials are passed to the subtract function as arguments. It then assigns *poly1* to a current pointer *curr1* which will traverse through the first polynomial. Likewise, *poly2* is assigned to *curr2*.

Two while loops which iterate until the last node of each polynomial linked list is used to evaluate the given polynomial inputs. Hence, in the first iteration, the first term of *poly1* is multiplied with each term of *poly2* by multiplying the coefficients and adding the powers. This is stored in the resultant polynomial linked list *polyprod.*

Then, *poly2* is once again assigned to *curr2* and *curr1* is advanced to the next term of *poly1*. The above-mentioned process for the second term of *poly1* happens and this continues till the last nodes of both polynomial linked lists are encountered.

Next, a function called GroupLikePowers is called by passing *polyprod* as argument. This function will group the coefficients of all the like powers in *polyprod*. After this, *polyprod* is displayed using the display function.

```
void multiply (struct node* poly1, struct node* poly2)
{
    struct node *curr1, *curr2, *currentNode = NULL, *polyprod =
    NULL;
    int coef, pow;
    curr1 = poly1;
    curr2 = poly2;
    while (curr1 != NULL)
    {
      while (curr2 != NULL)
      {
          coef = curr1->coef * curr2->coef;
          pow = curr1->pow + curr2->pow;

          if (coef != 0)
            {
                  if (polyprod == NULL)
                  {
                        polyprod = (struct node*) malloc (sizeof
                        (struct node));
```

```
                        polyprod->coef = coef;
                        polyprod->pow = pow;
                        polyprod->link = NULL;

                        currentNode = polyprod;
                  }
                  else
                  {
                        currentNode->link = (struct node*) malloc
                        (sizeof (struct node));

                        currentNode = currentNode->link;
                        currentNode->coef = coef;
                        currentNode->pow = pow;
                        currentNode->link = NULL;
                  }
            }
          curr2 = curr2->link;
      }
      curr2 = poly2;
      curr1 = curr1->link;
    }

    GroupLikePowers (polyprod);

    printf("\nPolynomial 1: ");
    display(poly1);
    printf("\nPolynomial 2: ");
    display(poly2);

    printf("\nThe polynomial product is: ");
    display(polyprod);
}
```

```
void GroupLikePowers(struct node* first)
{
    struct node *curr1, *curr2, *temp;
    curr1 = first;

    while (curr1 != NULL && curr1->link != NULL)
    {
      curr2 = curr1;

      while (curr2->link != NULL)
      {
          if (curr1->pow == curr2->link->pow)
          {
              curr1->coef = curr1->coef + curr2->link->coef;
              temp = curr2->link;
              curr2->link = curr2->link->link;
              free (temp);
          }
          else
              curr2 = curr2->link;
      }
      curr1 = curr1->link;
    }
}
```

Evaluation of a Polynomial

In this function, a polynomial linked list, *poly* is passed as an argument. A temporary pointer, *temp* is assigned *poly* which will traverse the linked list.

The value of *x* is taken from the user for evaluating the polynomial. A while loop is used which will run until the last node. A variable *sum* initialized with zero is used to find the sum, i.e. *f(x)*. The resultant sum is displayed.

```
void evaluate (struct node *poly)
{
      int sum=0, x;
      struct node *temp = NULL;

      printf("\nPolynomial entered: ");
      display(poly);

      temp = poly;
      printf("\nEnter value for x: ");
      scanf("%d", &x);

      while(temp!=NULL)
      {
            sum = sum + ((temp->coef)*(pow(x,temp->pow)));
            temp = temp->link;
      }

      printf("\nThe value of f(x) for x = %d is: %d ", x, sum);
}
```

CHAPTER 4

IMPLEMENTATION

4.1 EVALUATION

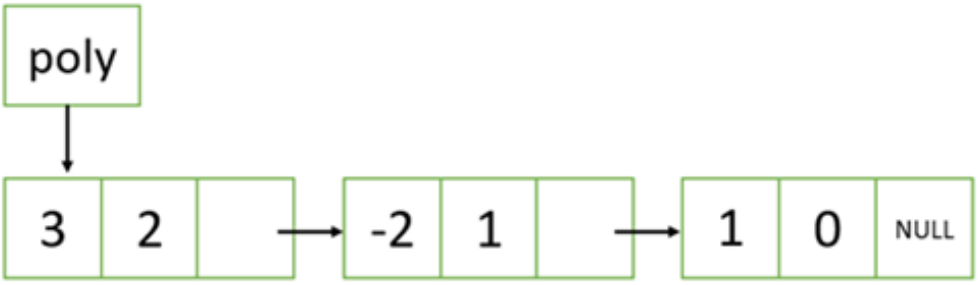
Manual Implementation

To evaluate an expression for a value of the variable, you substitute the value for the variable every time it appears. Then use the order of operations to find the resulting value for the expression. Example:

Table 1: Manual Implementation of Evaluation of a Polynomial

Example 1		
Problem	Evaluate $3x^2 - 2x + 1$ for x = -1.	
	3(-1)2 – 2(-1) + 1	Substitute -1 for each x in the polynomial.
	3(1) – 2(-1) + 1	Following the order of operations, evaluate exponents first.
	3 + (-2)(-1) + 1	Multiply 3 times 1, and then multiply -2 times -1.
	3 + 2 + 1	Change the sign.
Answer	$3x^2 - 2x + 1 = 6$, for x = −1	Find the sum.

Linked List Implementation



For x = -1

$3 \times (-1)^2 + -2 \times (-1)^1 + 1 \times (-1)^0 = 6$

Figure 2: Linked List implementation of Evaluation operation

4.2 ADDITION

Manual Implementation

To add polynomials, you first need to identify the like terms in the polynomials and then combine them according to the correct integer operations. Like terms must have the same exact variables raised to the same exact power. Example:

Table 2: Manual Implementation of Addition of two Polynomials

Example 2		
Problem	**Add $4x^2 - 12x + 9$ and $25x^2 + 4x - 32$**	
	$[4x^2 + 25x^2] + [(-12x) + 4x] + [9 + (-32)]$	Group like terms using commutative and associative properties.
	$29x^2 + (-8x) + (-23)$	Combine like terms.
Answer	The sum is $29x^2 - 8x - 23$.	Rewrite the answer.

Linked List Implementation

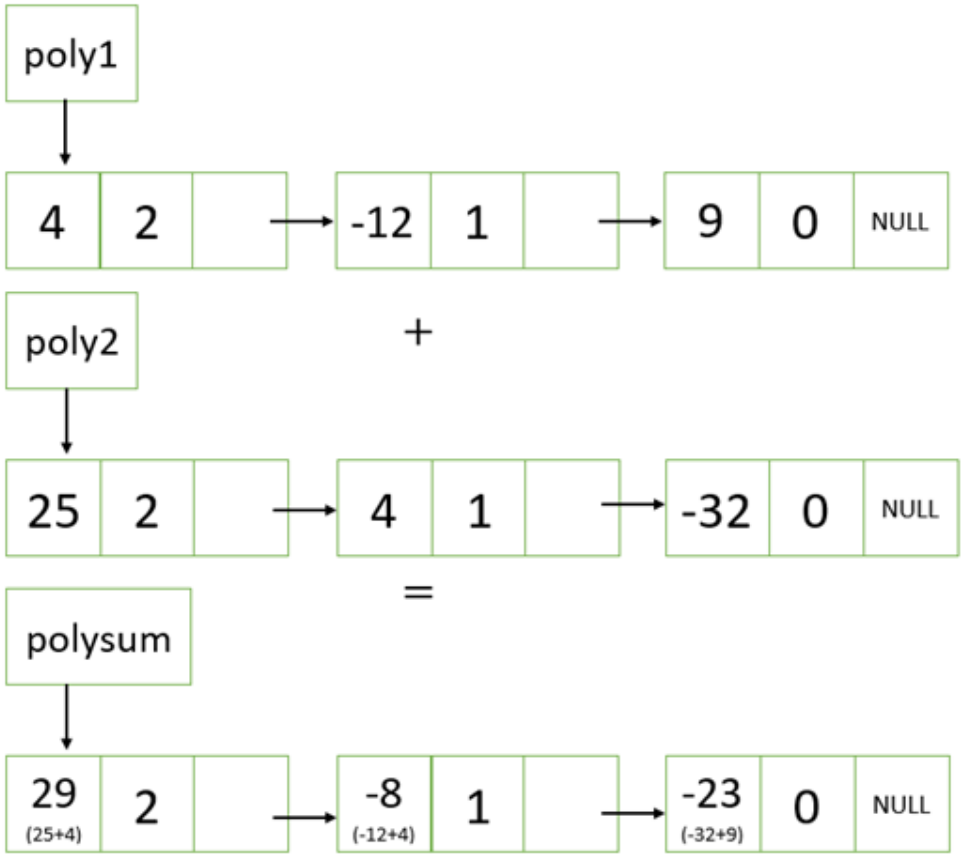


Figure 3: Linked List implementation of Addition operation

4.3 SUBTRACTION

Manual Implementation

We can apply the same process used for addition operation to subtract polynomials with one variable or to subtract polynomials with more than one variable. Example:

Table 3: Manual Implementation of Subtraction of two Polynomials

Example 3		
Problem	**Subtract $7x^2 - 4x + 17$ from $2x^3 + 14x^2 - 5x + 9$**	
	$2x^3 + 14x^2 - 5x + 9 - (7x^2 - 4x + 17)$	Rewrite the question.
	$[2x^3] + [14x^2 - 7x^2] + [(-5x) - (-4x)] + [9 - 17]$	Group like terms using commutative and associative properties.
	$2x^3 + 7x^2 + (-x) + (-8)$	Combine like terms.
Answer	The difference is $2x^3 + 7x^2 - x - 8$	Rewrite the answer.

Linked List Implementation

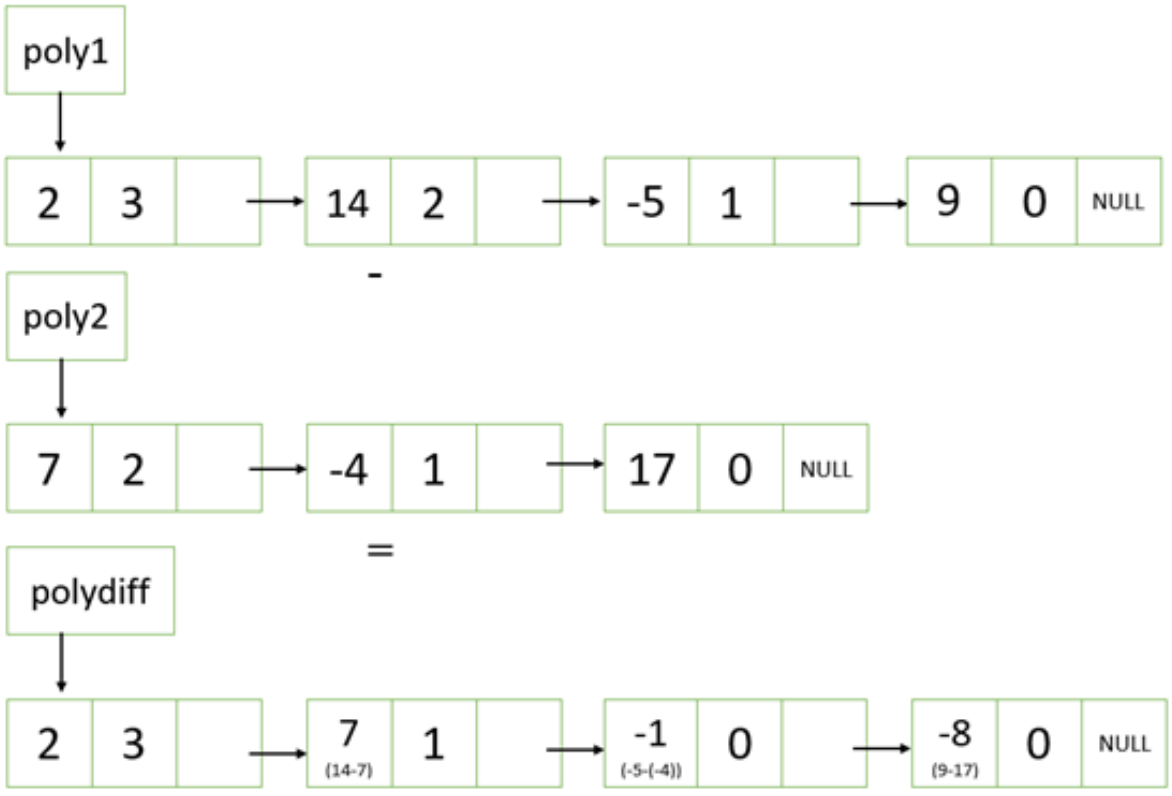


Figure 4: Linked List implementation of Subtraction operation

4.4 MULTIPLICATION

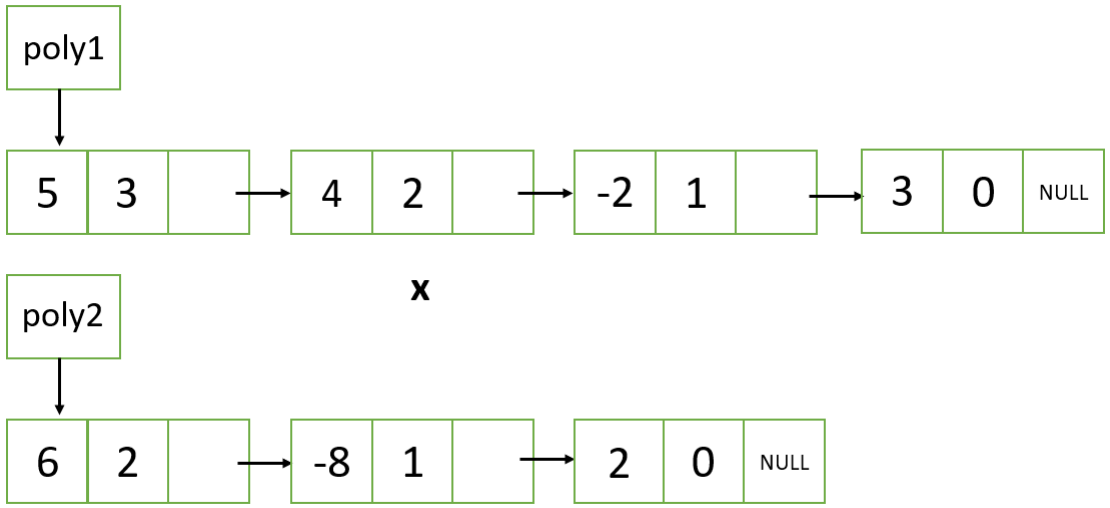
Manual Implementation

To multiply two polynomials, we have to multiply each term of the first polynomial to each term of the second polynomial and then group the like terms. Example:

Table 4: Manual Implementation of Multiplication of two Polynomials

Example 4		
Problem	**Multiply $6x^2 - 8x + 2$ with $5x^3 + 4x^2 - 2x + 3$**	
	$[5x^3 + 4x^2 - 2x + 3] * [6x^2 - 8x + 2]$	Rewrite the question.
	$[5x^3(6x^2) + 5x^3(-8x) + 5x^3(2)] + [4x^2(6x^2) + 4x^2(-8x) + 4x^2(2)] + [(-2x)(6x^2) + (-2x)(-8x) + (-2x)(2)] + [3(6x^2) + 3(-8x) + 3(2)]$	Multiply each term of the first polynomial with each term of the second polynomial.
	$[30x^5 + (-40x^4) + 10x^3] + [24x^4 + (-32x^3) + (8x^2)] + [(-12x^3) + 16x^2 + (-4x)] + [18x^2 + (-24x) + 6]$	Expand the expression.
	$30x^5 + [(-40x^4) + 24x^4] + [10x^3 + (-32x^3) + (-12x^3)] + [8x^2 + 16x^2 + 18x^2] + [(-4x) + (-24x)] + 6$	Group the like terms.
Answer	The product is $30x^5 - 16x^4 - 34x^3 + 42x^2 - 28x + 6$	Rewrite to get the answer.

Linked List Implementation



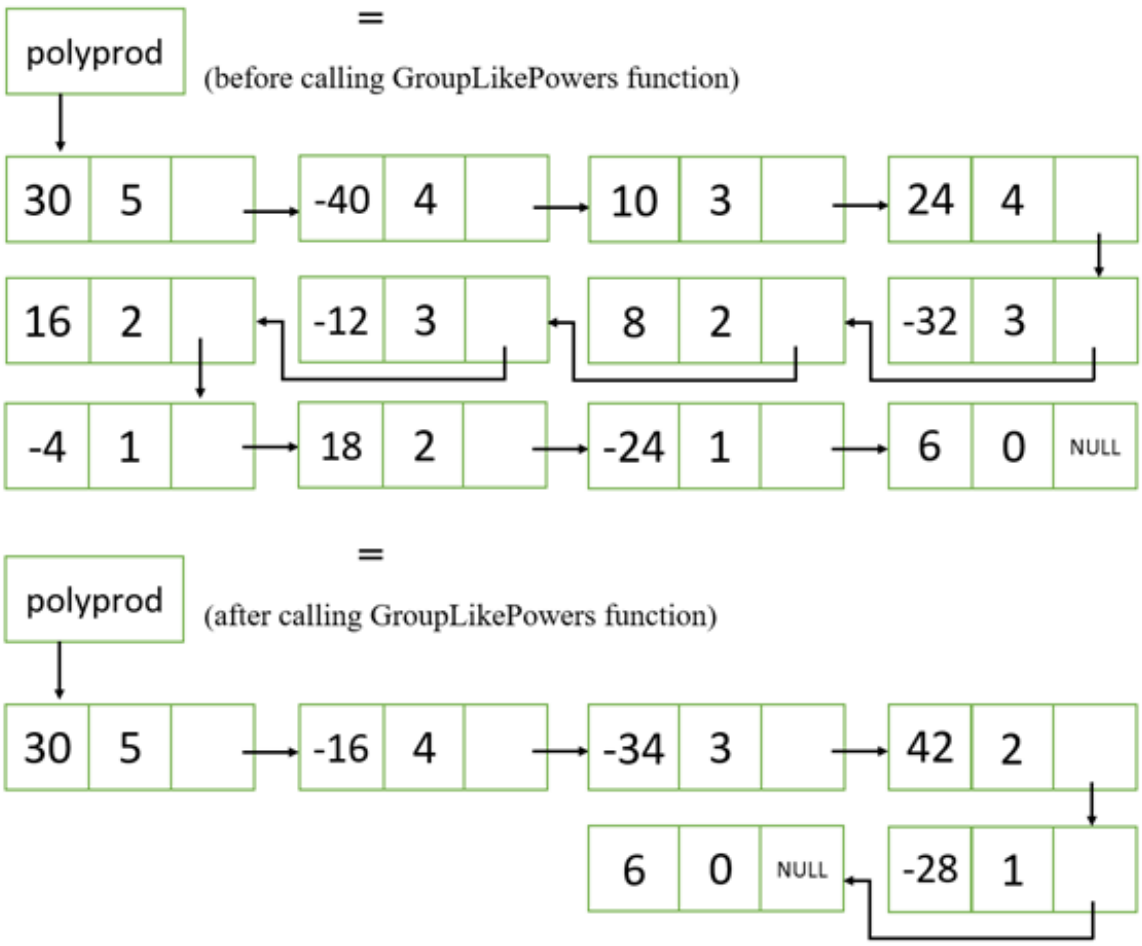


Figure 5: Linked List implementation of Multiplication operation

In the linked list implementation part, the program initially sends the *struct node* * variable *polyprod* to the *multiply* user defined function as an argument when called from the main function.

Inside the *multiply* function, each term of the first input polynomial is multiplied with each term of the second polynomial. When we perform this, a problem arises; there will be multiple terms with the same power in the *polyprod* output polynomial.

To fix this, we have another user defined function called *GroupLikePowers* which takes *polyprod* as an argument when called from the *multiply* function. This function traverses the entire *polyprod* expression and combines all the like powers into one term and removes the extra terms from the *polyprod* expression. The final product expression is then displayed to the user.

CHAPTER 5

RESULTS

The sample results of the manually implemented problems seen in the previous section when executed using the program are shown here.

5.1 EVALUATION

```
POLYNOMIAL CALCULATOR
by Rahul M Dinesh (1NH18CS738)
CSE 3-D, New Horizon College of Engineering

Which operation would you like to perform?
1. Evaluation
2. Addition
3. Subtraction
4. Multiplication

Choice: 1

Enter Polynomial: Enter the number of terms in the polynomial: 3
        Enter term 1's coeficient & power: 3 2
        Enter term 2's coeficient & power: -2 1
        Enter term 3's coeficient & power: 1 0

Polynomial entered: 3x^2 + -2x + 1
Enter value for x: -1

The value of f(x) for x = -1 is: 6
```

Figure 6: Output Screen of Evaluation of a Polynomial

5.2 ADDITION

```
POLYNOMIAL CALCULATOR
by Rahul M Dinesh (1NH18CS738)
CSE 3-D, New Horizon College of Engineering

Which operation would you like to perform?
1. Evaluation
2. Addition
3. Subtraction
4. Multiplication

Choice: 2

Enter Polynomial 1: Enter the number of terms in the polynomial: 3
        Enter term 1's coeficient & power: 4 2
        Enter term 2's coeficient & power: -12 1
        Enter term 3's coeficient & power: 9 0

Enter Polynomial 2: Enter the number of terms in the polynomial: 3
        Enter term 1's coeficient & power: 25 2
        Enter term 2's coeficient & power: 4 1
        Enter term 3's coeficient & power: -32 0

First Polynomial: 4x^2 + -12x + 9
Second Polynomial: 25x^2 + 4x + -32
The polynomial sum is: 29x^2 + -8x + -23
```

Figure 7: Output Screen of Addition of two Polynomials

5.3 SUBTRACTION

```
POLYNOMIAL CALCULATOR
by Rahul M Dinesh (1NH18CS738)
CSE 3-D, New Horizon College of Engineering

Which operation would you like to perform?
1. Evaluation
2. Addition
3. Subtraction
4. Multiplication

Choice: 3

Enter Polynomial 1: Enter the number of terms in the polynomial: 4
        Enter term 1's coeficient & power: 2 3
        Enter term 2's coeficient & power: 14 2
        Enter term 3's coeficient & power: -5 1
        Enter term 4's coeficient & power: 9 0

Enter Polynomial 2: Enter the number of terms in the polynomial: 3
        Enter term 1's coeficient & power: 7 2
        Enter term 2's coeficient & power: -4 1
        Enter term 3's coeficient & power: 17 0

First Polynomial: 2x^3 + 14x^2 + -5x + 9
Second Polynomial: 7x^2 + -4x + 17
The polynomial difference is: 2x^3 + 7x^2 + -1x + -8
```

Figure 8: Output Screen of Subtraction of two Polynomials

5.4 MULTIPLICATION

```
POLYNOMIAL CALCULATOR
by Rahul M Dinesh (1NH18CS738)
CSE 3-D, New Horizon College of Engineering

Which operation would you like to perform?
1. Evaluation
2. Addition
3. Subtraction
4. Multiplication

Choice: 4

Enter Polynomial 1: Enter the number of terms in the polynomial: 4
        Enter term 1's coeficient & power: 5 3
        Enter term 2's coeficient & power: 4 2
        Enter term 3's coeficient & power: -2 1
        Enter term 4's coeficient & power: 3 0

Enter Polynomial 2: Enter the number of terms in the polynomial: 3
        Enter term 1's coeficient & power: 6 2
        Enter term 2's coeficient & power: -8 1
        Enter term 3's coeficient & power: 2 0

Polynomial 1: 5x^3 + 4x^2 + -2x + 3
Polynomial 2: 6x^2 + -8x + 2
The polynomial product is: 30x^5 + -16x^4 + -34x^3 + 42x^2 + -28x + 6
```

Figure 9: Output Screen of Multiplication of two Polynomials

CHAPTER 6

CONCLUSION

The mini project has successfully accomplished the goals it had set out in the objectives and design sections of this report.

The various operations namely, addition, subtraction, multiplication and evaluation of polynomials were successfully implemented using the data structure concept of a single linked list.

The project has also achieved the target of setting a simple and easy to use interface for the user as was seen in the results section.

The evaluation operation has been demonstrated successfully with the user inputting a polynomial and the value of *x.* The evaluated result *f(x)* was displayed for the given value of *x.*

The addition operation took two polynomials by asking the number of terms in each polynomial, followed by the user inputting the terms and coefficients. The addition result of the inputted polynomials was shown along with the polynomials entered by the user.

The subtraction operation was also demonstrated successfully, similar to the addition operation.

The multiplication operation which had the most complex logic in this mini project was also implemented successfully with the user inputting two polynomials followed by the function calling the *GroupLikePowers* user defined function to eliminate the terms with like powers. The final polynomial product was displayed along with the input polynomials.

In the future, I look forward to expanding this mini project to include the division operation on two polynomials, which I found rather difficult to develop and algorithm for.

This project has helped me expand my knowledge about linked lists and made me think in unconventional ways to produce the required logics and algorithms.

REFERENCES

[1] *Data Structures with C,* by SEYMOUR LIPSCHUTZ, Special Indian Edition, Thirteenth Reprint 2015, McGraw Hill Education

[2] *Data Structures using C*, by Aaron M. Tanenbaum, Yedidyah Langsam & Moshe J Augenstein, Thirteenth Impression 2014, Pearson Education

[3] *Data Structures – A Pseudocode Approach with C*, by Richard F Gilberg and Behrouz A Forouzan, Second Edition, Fifth Indian Reprint 2015, Cengage Learning

[4] Polynomial Representation, Addition and Multiplication:
http://www.manojagarwal.co.in/polynomial-representation-addition-multiplication/

[5] International Institute of Information Technology: Virtual Labs:
http://cstar.iiit.ac.in/~kkishore/DSVL/exp4/index.html

[6] Polynomial Representation using Linked List:
https://www.youtube.com/watch?v=hM-rvbVJ4Po

[7] Linked List Data Structure, by GeeksForGeeks:
https://www.geeksforgeeks.org/data-structures/linked-list/