Date: 19-11-2020

Experiment 6
Addition Of Two Polynomials

Done By: Rohit Karunakaran **Roll no:** 58

Aim: To receive two polynomials and print their sum and product

Data Structure Used: Linked List

Operation Used: Comparisons, addition, multiplication

Algorithm for Addition (ADD_POLY):

Input: Two polynomial, A and B with the terms as the nodes of a linked list and 'a' denoting the number of terms in polynomial A and 'b' denoting the number of terms in polynomial 'B'

Output: Sum of the polynomial 'C'

Data Structure Used : Linked List

Step 1 : Start
Step 2 : Receive two polynomial in linked list
Step 3 : i = A → Header //Pointer to the header of polynomial A
Step 4 : j = B → Header //Pointer to the polynomial B
Step 5 : while i =! NULL and j=! NULL
    Step 1: new=GetNode(Node)
    Step 1 : if i→ pow == j→ pow
        Step 1: new→ pow = i→ pow
        Step 2: new → coeff = i → coeff+j → coeff
        Step 3: C.addNode(new)
        Step 4: i=i→link
        Step 5: j=j->link
    Step 2: else if i→pow < j→pow
        Step 1: new → pow = j→pow
        Step 2: new→ coeff=j→ coeff
        Step 3: C.addNode(new)
        Step 4: j=j→ link
    Step 3: else if i → pow > j → pow
        Step 1: new→coeff = i->coeff
        Step 2: new→pow = i→ pow
        Step 3: i=i → link
        Step 4: C.addNode(new)
    Step 4: Endif
Step 6 : EndWhile

Step 7 : while i!=NULL
    Step 1: new→coeff = i → coeff
    Step 2: new→pow = i → pow
    Step 3: i = i → link
    Step 4: C.addNode(new)
Step 8: EndWhile
Step 9: while j!=NULL
    Step 1: new = GetNode(Node)
    Step 2: new → pow = j→pow
    Step 3: new→ coeff=j→ coeff
    Step 4: C.addNode(new)
    Step 5: j=j→ link
Step 10 : EndWhile

Step 11 : return c
Step 12 : Stop

Description of the Algorithm:

In this algorithm the polynomials' terms are the nodes of a linked list and there are 2 pointers i and j, which points to the nodes of A and B respectively. If the powers of a term in A and B are equal then the coefficeint are added and the sum is put into a new node (new). Which is then added to the end of the resultant polynomial C. If the coefficientof the term in A is greater than the term in B then the term is added to the end of B. Likewise for B also.

Algorithm for Multiplication(MUL_POLY):

Input: A and B, two polynomials with the terms as nodes of a linked list with pow being the power of the term and coeff being the coefficient

Output: Polynomial C, with

Data Structure used: Linked list

Steps:

Step 1:Start
Step 2: receive two polynomials
Step 3:i = A→ head
Step 4: j = B→ head
Step 5: initialize C as a polynomial with 0 as the only term
Step 6: k = 0
Step 7: while(k<B→numberOfTerms)
    Step 1: j = B → head
    Step 2: while(j!=NULL)
        Step 1: new = GetNode(Node)
        Step 2; new→pow = i→pow+j→ pow
        Step 3: new → coeff = i→ coeff * i→ coeff
        Step 4: temp.addNode(new)
        Step 5: j = j→link
    Step 3: End While
    Step 4: C= ADD_POLY(C,temp)
    Step 5: i=i→ link
    Step 6: k++
Step 8: EndWhile
Step 9: return C
Step 10: Stop

Description of the Algorithm:

The polynomial product of $(6X^2+1)*(7X^2+3X+1)$ can be expressed as, $0+(6X^2+1)*(7X^2)+(6X^2+1)*(3X+(6X^2+1)*1$. Here we just need to multiply the first polynomial with one of the terms from the second and feed the result obtained before and the result obtained now to the addition function and then after the algorithm has been executed number of times as there are number of terms in B. We get the product of the polynomial.

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
/*************************************
 * Sum And Product of a Polynomial
 * Done By Rohit Karunakaran
 *************************************/

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

/*  Input : 2 polynomials of the form
 *          a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an
 *  Output: First polynomial the second polynomial and there sum
 */

typedef struct Node
{
   int coeff;
   int pow;
   struct Node* link;
}PolyNode;

typedef struct Polynomial
{
    int numberOfTerms;
    PolyNode* Head; //Header contians the first polynomial, so it has to be printed
    PolyNode* Trail;
}Poly;

//UTILITY FUNCTIONS START

void initPoly(Poly **a)
{
    *a = (Poly*)malloc(sizeof(Poly));
    (*a)->Head = NULL;
    (*a)->Trail= NULL;
    (*a)->numberOfTerms=0;
}

void addNode(Poly *a,int pow, int coeff)
{
    PolyNode* n = (PolyNode*) malloc(sizeof(PolyNode));
    if(n!=NULL){
        n->coeff = coeff; n->pow = pow; n->link=NULL;
        if(a->Trail ==NULL)
        {
            a->Head = n;
        }
        else
        {
            a->Trail->link = n;
        }
        a->Trail = n;
    }
    else
```

```
    {
        return;
    }
}

void deleteNode(Poly *a,PolyNode *b)
{
    PolyNode *ptr=a->Head;
    if(ptr==NULL)return;

    while(ptr->link!=b&&ptr!=NULL){ptr=ptr->link;} //Traverse till you find the node
b

    if(ptr==NULL){return;} //If there is no such node then, return

    else
    {
        if(ptr->link->link==NULL)
        {
            free(ptr->link);
            ptr->link=NULL;
        }
        else
        {
            PolyNode *tmp = ptr->link;
            ptr->link = tmp->link;
            free(tmp);
        }
    }
}

void freePoly(Poly **poly)
{
    if(*poly !=NULL)
    {
        PolyNode *i,*tmp;
        i=(*poly)->Head;
        while(i!=NULL)
        {
            tmp=i;
            i=i->link;
            free(tmp);
        }
        free (*poly);
    }
    return;
}

//UTILITY FUNCTIONS END

/* Funtion to print the polynomials*/
void printPoly(Poly* a){
    /* Input: Polynomial stored in the structure Polynomial
     * Ouput: prints the polynomial
     */
```

```
    //int iterCount = a->numberOfTerms;
    //int i;
    PolyNode *ptr=a->Head;
    while(ptr!=a->Trail){
        printf("%d*X^%d + ",ptr->coeff,ptr->pow);
        ptr = ptr->link;
    }
    printf("%d*X^%d",ptr->coeff,ptr->pow);
}

/* Funtion to convert the polynomial into tuple*/
Poly* createPolyFromString(char* s){
    /* Input: String of charecters
     *
     * Output: the Head node of the linked list contating the polynomial
     * */

    Poly* a=NULL;initPoly(&a);
    int i;

    int count = 0;
    int numberStack[2];
    int numberStackTop = -1;

    int number = 0,pow,coeff;
    int negative = 0;

    //parsing the string
    for(i = 0; s[i]!='\0'; i++){
        if(s[i] == '-'){
            negative = 1;
        }

        if(s[i]>='0'&&s[i]<='9'){
            while((s[i]!= 'X'||s[i]!='x'||s[i]!=' '||s[i]!='^') &&
(s[i]>='0'&&s[i]<='9')){
                 // here s[i] will only be numbers
                number = number*10+(s[i]-'0');
                i++;
            }

            if(negative) numberStack[++numberStackTop] = -1*number;
            else numberStack[++numberStackTop] = number;

            i--;
            negative = 0;
            number = 0;
        }

        if(i!=0&&(s[i]=='-'||s[i]=='+'||s[i]=='\0')){//&&s[i-1]!='^'){
            if(numberStackTop==0)
            {
                if(s[i-1]=='X')
                    numberStack[++numberStackTop] = 1;
                else
                    numberStack[++numberStackTop] = 0;
```

```
            }

                count++;
                pow = numberStack[numberStackTop--];
                coeff = numberStack[numberStackTop--];
                addNode(a,pow,coeff);
        }
    }

    if(numberStackTop==0)
    {
        if(s[i-1]=='X')
            numberStack[++numberStackTop] = 1;
        else
            numberStack[++numberStackTop] = 0;
    }
    count++;
    pow = numberStack[numberStackTop--];
    coeff = numberStack[numberStackTop--];
    addNode(a,pow,coeff);

    a->numberOfTerms = count;

    return a;
}

/*Funtion to find the sum of the polynomials*/
Poly* sumOfPoly(Poly* a, Poly* b)
{
    Poly* c = (Poly*)malloc(sizeof(Poly));
    initPoly(&c);

    PolyNode *i=a->Head;
    PolyNode *j=b->Head;

    while(i!=NULL&&j!=NULL)
    {

        if(i->pow==j->pow)
        {
            if(i->coeff+j->coeff!=0)
                addNode(c,i->pow,i->coeff+j->coeff);

            i=i->link;
            j=j->link;
        }
        else if(i->pow>j->pow)
        {
            addNode(c,i->pow,i->coeff);
            i=i->link;
        }
        else if(i->pow<j->pow)
        {
            addNode(c,j->pow,j->coeff);
            j=j->link;
        }
```

```
        c->numberOfTerms++;
    }

    while(i!=NULL)
    {
        addNode(c,i->pow,i->coeff);
        i=i->link;
        c->numberOfTerms++;
    }
    while(j!=NULL)
    {
        addNode(c,j->pow,j->coeff);
        j=j->link;
        c->numberOfTerms++;
    }

    return c;
}

Poly* productOfPolynomials(Poly* a,Poly*b)
{
    Poly *c=NULL;
    Poly *temp=NULL;
    //intiPoly(Temp);

    int k = 0;
    PolyNode *i = a->Head;
    PolyNode *j = b->Head;

    while(k<a->numberOfTerms)
    {
        //i=a->Head;
        j=b->Head;
        if(c==NULL)
        {
            initPoly(&c);
            while(j!=NULL)
            {
                addNode(c,i->pow+j->pow,i->coeff*j->coeff);
                j=j->link;
            }
        }
        else
        {
            initPoly(&temp);
            while(j!=NULL)
            {
                addNode(temp,i->pow+j->pow,i->coeff*j->coeff);
                j=j->link;
            }
            c=sumOfPoly(c,temp);
        }
        i=i->link;
        freePoly(&temp);
        k++;
    }
```

```
    return c;
}

int main(){
    Poly* a;
    Poly* b;
    Poly* c;
    int strLength = 100;
    char* polyString = (char*) malloc(strLength*sizeof(char));

    /*Read the polynomials*/
        fflush(stdin);
        printf("Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 .....
an*X^0 --> ");
        scanf("%[^\n]",polyString);
        scanf("%*c"); //remove the \n charecter from the input stream
        a  = createPolyFromString(polyString);
        free(polyString);

        fflush(stdin);
        fflush(stdout);

        polyString = (char*) malloc(strLength*sizeof(char));

        printf("Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 .....
an*X^0 --> ");
        scanf("%[^\n]",polyString);
        b  = createPolyFromString(polyString);
        free(polyString);
    /*Finish reading Polynomials*/

    printf("\nPolynomial 1 is: ");
    printPoly(a);
    printf("\nPolynomial 2 is: ");
    printPoly(b);

    c = sumOfPoly(a,b); //Find the sum of the polynomials

    printf("\nSum is ");
    printPoly(c);

    c = productOfPolynomials(a,b);
    printf("\nProduct is ");
    printPoly(c);
    printf("\n");

    freePoly(&a);
    freePoly(&b);
    freePoly(&c);
    return 0;
}
```

Sample Input/Output:

```
..ograming/C/CSL201/2020-11-16❯ ./polynomial.o
Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> 4X^2+5X+1
Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> 5X+4

Polynomial 1 is: 4*X^2 + 5*X^1 + 1*X^0
Polynomial 2 is: 5*X^1 + 4*X^0
Sum is 4*X^2 + 10*X^1 + 5*X^0
Product is 20*X^3 + 41*X^2 + 25*X^1 + 4*X^0
..ograming/C/CSL201/2020-11-16❯
```

```
..ograming/C/CSL201/2020-11-16❯ ./polynomial.o
Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> 12X^100+1
Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> 7X

Polynomial 1 is: 12*X^100 + 1*X^0
Polynomial 2 is: 7*X^1
Sum is 12*X^100 + 7*X^1 + 1*X^0
Product is 84*X^101 + 7*X^1
..ograming/C/CSL201/2020-11-16❯
```