DATA STRUCTURE ASSIGNMENT 3 3rd SEMESTER CSE- SEC D

PROBLEM STATEMENT :
To perform different operations on polynomial with two terms x, y using 2-D array and Sparse Matrix Implementations.

TEAM MEMBERS :

REG. NO.	ROLL NO.	NAME
160905058	8	Sahil Garg
160905440	20	Akshatha
160905184	32	Shounak Dey
160905372	56	Vedant

DETAILED PROBLEM STATEMENT :

a. Addition of polynomials with two terms: To perform different operations on polynomial with two terms x, y using 2-D array representations. Operations like addition and multiplication have to be implemented.

b. If the 2D array representation is sparse then optimize the memory usage by using suitable alternative representation such as sparse matrix representation. However, all the operations have to be allowed, abstracting the representation.

DATA STRUCTURE EMPLOYED :

- **Array Implementation :**

Pol : It has two instances -

1. 2-D Array (coeff[][]) which stores the coefficient of each term according to the exponents of x and y.

2. Integer (deg) which stores the degree of the polynomial.

- **Sparse Matrix Implementation :**

Addition:

Node : It has three instances –

1. Integer (coeff) to store the coefficient of the term.
2. Integer (row) to store the exponent of y.
3. Pointer (*next) for linked list implementation.

Multiplication:

SM : It has three instances –

1. Integer (row) to store the exponent of x
2. Integer (column) to store the exponent of y.
3. Integer (value) to store the entered value.

ALGORITHM :

- **Initialization:** Initializing the coefficients of two polynomials to NULL using the init() function.

- **Reading Data:** Reading the degree of polynomial and the number of terms followed by the coefficients and exponents for each term using the read() function. Polynomials are a and b.

- **Display:** Displaying the polynomials a and b using display() function.

- **Multiply:** Multiplying a and b and storing the result in polynomial x using the multiply() function.

The result x will have the coefficients as the product of coefficients of terms in a and b while the exponents will be the sum of exponents in a and b.

- **Addition:** Adding a and b and storing the result in polynomial x using the add() function.

  The result x will have all the terms present in a and b with the coefficients of terms in a and b having the same exponent added together.

MODULES (User Defined Functions) :

- **Array Implementation :**

  void init() : To initialize the coefficients to null.
  void read() : To read the details for the polynomial.
  void display() : To display the polynomial.
  void add() : To add the polynomials.
  void multiply() : To multiply the polynomials.

- **Sparse Matrix Implementation :**

  node*create(int coeff,int row) : Takes the coefficient and the exponent of y.
  node *insert(int coeff, int row, node *head) : Insert into the list and return updated head address.
  node **init(node **head,int n) : Creates a polynomial. The polynomial is saved as an array of linked lists. Row i corresponds to all terms with term x^i y^a, 0<= a <= degree of polynomial. Here we allocate space for each array location and return this newly created polynomial.

node *search(node *sum,int exp) : Searches for the term with exponent of y equal to exp and returns that node address if found otherwise NULL.

node **add(node **sum,node **p1,node **p2,int n1,int n2) : Main function which adds two polynomials. This returns the

final polynomial.

void input(SM *a) : To read values.

void multiply(SM *m,SM *a,SM *b) : To multiply the polynomials.

void display(SM *a) : To display the polynomials.

Source Codes:

Array Implementation :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100 //Maximum degree possible
#define max(a,b) (a>b?a:b)

typedef struct pol{
	int coeff[MAX][MAX];//coeff[i][j] = coefficient of x^i * y*j
	int deg;//Degree of the polynomial
}pol;

void init(pol *a)
{
	int i,j;
	for(i=0;i<MAX;i++)
	{
		for(j=0;j<MAX;j++)
			a->coeff[i][j] = 0;
	}
	return;
}

void read(pol *a)
{
	printf("Entering input\n");
	printf("Enter the degree of the polynomial\n");
```

```
        scanf("%d",&(a->deg));
        //printf("%d ",a->deg);
        printf("Enter the number of terms\n");
        int n;
        scanf("%d",&n);
        //printf("%d\n",n);
        printf("Enter the coefficients in the following manner:\n");
        printf("Coefficient\texponent of x\texponent of y:\n");
        int c,i,j;
        while(n--)
        {

                scanf("%d %d %d",&c,&i,&j);
                a->coeff[i][j] = c;
    }
}

void display(pol *x)
{
        printf("The degree is - %d.\n",x->deg);
        printf("The matrix is (The term of matrix (a,b) represents coefficient
of x^a.y^b) - \n");
        int i,j;
        for(i=0;i<=x->deg;i++)
        {
                for(j=0;j<=x->deg;j++)
                        printf("%d ",x->coeff[i][j]);
                printf("\n");
        }
        return;
}
void multiply(pol *s,pol *a,pol *b)
{
        int i,j,k,l ;
        s->deg = a->deg+b->deg;
        for(i=0;i<=a->deg;i++)
        {
                for(j=0;j<=a->deg;j++)
                        {
                            for(k=0;k<b->deg;k++)
                    {
                        for(l=0;l<b->deg;l++){
                            s->coeff[i+k][j+l]+=a->coeff[i][j]*b->coeff[k][l];
                        }
                    }
                        }
        }
        return;
}
void add(pol *s,pol *a,pol *b)
{
        int i,j;
        s->deg = max(a->deg,b->deg);
        for(i=0;i<=s->deg;i++)
        {
                for(j=0;j<=s->deg;j++)
                        s->coeff[i][j] = a->coeff[i][j] + b->coeff[i][j];
```

```
	}
	return;
}
int main()
{
	//printf("Hi!\n");
	pol a,b,x;
	printf("Starting Initialisation\n");
	init(&a);
	init(&b);
	printf("Initialising ...\n");
	read(&a);
	display(&a);
	read(&b);
	display(&b);
	int choice = 1;
	do{
	printf("\n1.Multiply  2.Add  3.Exit\n");
	scanf("%d",&choice);
	switch(choice){
	case 1 : printf("Multiplying both polynomials...\n");
		//printf("%d\n",c.deg);
		init(&x);
		multiply(&x,&a,&b);
		display(&x);
		break;

	case 2 : printf("Adding both polynomials...\n");
		//printf("%d\n",c.deg);
		init(&x);
		add(&x,&a,&b);
		display(&x);
		break;
	}
	}while(choice!=3);
	return 0;
}
```

Sparse Matrix Implementation :
Addition :

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

#define max(a,b) (a>b?a:b) //defined an inline max function

typedef struct node{
	int coeff;
	//coefficient of x^a . y^b;
	int row;
	//exponent of y
	struct node *next;
	//linked list implementation
}node;
```

```
node *create(int coeff,int row)
      //takes the coefficient and the exponent of y
{
            //and returns the node.
      node *new = (node *)malloc(sizeof(node));
      new->coeff = coeff;
      new->row = row;
      return new;
}

node *insert(int coeff,int row,node *head){
      //Insert into the list and return updated head address.
      node *new = create(coeff,row);
      if(head == NULL)
            return new;
      //return create return value if list is empty
      node *temp = head;
      while(temp->next != NULL)
      //Traverse to the last element of the linked list.
            temp = temp->next;
      temp->next = new;
      return head;
      //return updated head address.
}

void display(node *head,int column){
      //display function for the linked list.
      if(head == NULL){
            printf("No terms with exponent of x = %d\n",column);
            return;
      }
      node *temp = head;
      while(temp)
      {
            printf("Coeff=%d exp(y)=%d exp(x)=%d\n",temp->coeff,temp-
>row,column);
            temp = temp->next;
      }
}

node **init(node **head,int n){
      //Creates a polynomial. The polynomial is saved as an array of linked
      int i;
            //lists. Row i corresponds to all terms with term x^i.y^a,
      head = (node **)malloc(sizeof(node *) * n);                       //
0<= a <= degree of polynomial. Here we allocate space for each
      for(i=0;i<n;i++){
      // array location and return this newly created polynomial.
            head[i] = NULL;
      }
      return head;
}

node *search(node *sum,int exp)
      //Searches for the term with exponent of y equal to exp.
```

```
{
            //returns that node address if found otherwise NULL.
      if(sum == NULL)
            return NULL;
      while(sum != NULL)
      {
            if(sum->row == exp)
                  return sum;
            sum = sum->next;
      }
      return NULL;
}


node **add(node **sum,node **p1,node **p2,int n1,int n2)    //Main function
which adds two polynomials. This returns the
{
            //final polynomial.
      sum = init(sum,max(n1,n2));
      //initialising the resultant polynomial
      int i;

      node **temp;
      //creating a temporary polynomial
      if(n1<n2){
      //saving the smaller polynomial in p2 to simplify the code.
            temp = p1;
      //swapping p1 and p2
            p1 = p2;
            p2 = temp;
            n1 = n1+n2;
      //swapping n1 and n2
            n2 = n1-n2;
            n1 = n1-n2;
      }
      for(i=0;i<n1;i++)
      //copying the larger polynomial to the resultant polynomial
      {
            node *t;
            t = p1[i];
            while(t!=NULL)
            {
                  sum[i] = insert(t->coeff,t->row,sum[i]);
                  t = t->next;
            }
      }
      for(i=0;i<n2;i++)
      //Adding the smaller polynomial to the resultant polynomial.
      {
            node *t;
            t = p2[i];
            while(t!=NULL)
            {
                  node *res = search(sum[i],t->row);                    //
Looks for a similar term in the resultant polynomial.
                  if(res)
            // If found , add the coefficient to the resultant polynomial.
```

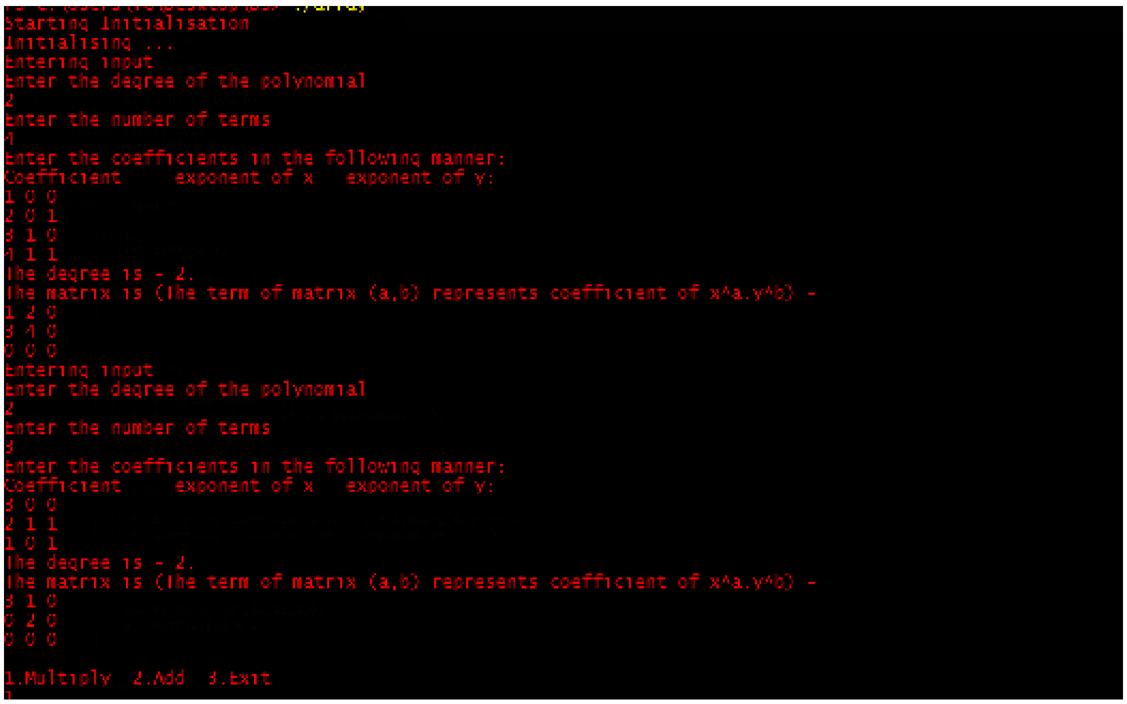
```
                        res->coeff += t->coeff;
                  else
                        sum[i] = insert(t->coeff,t->row,sum[i]);  // Else
insert this element into the resultant polynomial.
                  t = t->next;
            }
      }
      return sum;
      // Return the final polynomial
}


int main()
{
      //printf("Usage:\ndegree of first polynomial\nnumber of terms in the
first polynomial\nEnter coefficients in the format of \"Coeffiecient
exponent(x) exponent(y)\"\nDo the same for the second polynomial\n");
      int n1,t1,n2,t2;
      printf("Enter the degree and number of terms in the first polynomial :
");
      scanf("%d %d",&n1,&t1);
      node **p1,**p2;
      n1++;

      p1 = init(p1,n1);
      int coeff,x,y;
      while(t1--)
      {
            scanf("%d %d %d",&coeff, &x, &y);
            p1[x] = insert(coeff,y,p1[x]);
      }
      printf("The first polynomial is:\n");
      int i;
      for(i=0;i<n1;i++)
            display(p1[i],i);
      printf("\n\n");
      printf("\n\nEnter the degree and number of terms in the second
polynomial : ");


      scanf("%d %d",&n2,&t2);
      n2++;
      p2 = init(p2,n2);
      while(t2--)
      {
            scanf("%d %d %d",&coeff, &x, &y);
            p2[x] = insert(coeff,y,p2[x]);
      }
      printf("The second polynomial is:\n");
      for(i=0;i<n2;i++)
            display(p2[i],i);
      printf("\n\n");
      printf("The sum of the given polynomials is: \n");
      node **sum;
      sum = add(sum,p1,p2,n1,n2);
      for(i=0;i<max(n1,n2);i++)
            display(sum[i],i);
      return 0;
}
```

Multiplication :

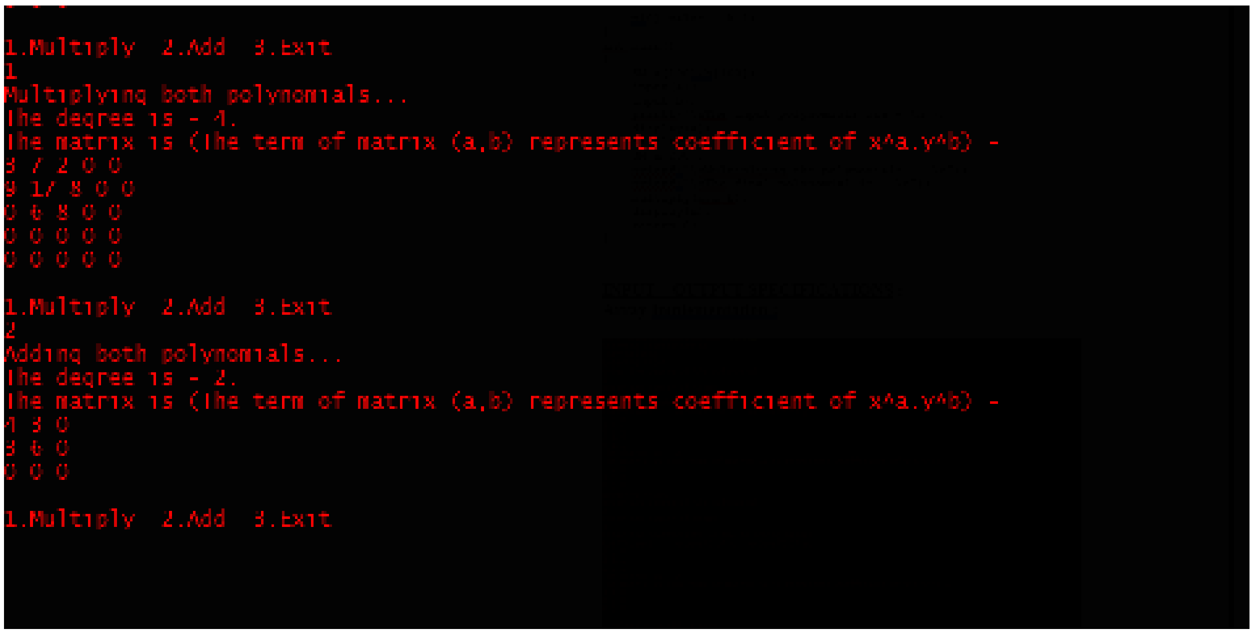
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100 //Maximum degree possible
#define max(a,b) (a>b?a:b)

typedef struct{
int row;
int column;
int value;
}SM;
void input(SM *a)
{
    printf("Enter the degree: ");
    int m,n,k=1,item;
    int i,j;
    scanf("%d",&m);
    n=m;
    m++;
    n++;
    a[0].row=m;
    a[0].column=n;
    printf("\nEnter the elements as a 2-D matrix of size %d: \n",m);
    printf("The matrix row number is equal to power of x and column number is
the power of y. Enter the value for each term as the matrix element.\n");
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            scanf("%d",&item);
            if(item==0)
                continue;
            a[k].row=i;
            a[k].column=j;
            a[k].value=item;
            k++;
        }
    }
    a[0].value=k-1;
}
void display(SM *a)
{
    printf("Polynomial is : \nx power\ty power\tCoeff\t\n");
    int i;
    for(i=1;i<=a[0].value;i++)
    {
        printf("%d\t%d\t%d\n",a[i].row,a[i].column,a[i].value);;
    }
}
void multiply(SM *m,SM *a,SM *b){
    int i,j,k=1,l;
    for(i=0;i<200;i++) m[i].column = m[i].row = m[i].value = 0;
    m[0].row = a[0].row+b[0].row;
    m[0].column = a[0].column+b[0].column;
    for(i=1;i<=a[0].value;i++){
        for(j=1;j<=b[0].value;j++){
        for(l=1;l<k;l++){
            if(m[l].row == a[i].row+b[j].row &&
m[l].column==a[i].column+b[j].column){
                m[l].value+=a[i].value*b[j].value;
                break;
            }
        }
        if(l==k){
            m[k].value = a[i].value*b[j].value;
            m[k].row = a[i].row+b[j].row;
            m[k].column=a[i].column+b[j].column;
            k++;
        }
    }
    }
    m[0].value = k-1;
}
int main()
{
    SM a[100],b[100];
    input(a);
    input(b);
    printf("\nThe input polynomials are - \n");
    display(a);
    display(b);
    SM m[200];
    printf("\nMultiplying the polynomials... \n");
    printf("\nThe final polynomial is - \n");
    multiply(m,a,b);
    display(m);
    return 0;
}
```

<u>RESULTS – INPUT OUTPUT SNAPSHOTS :</u>

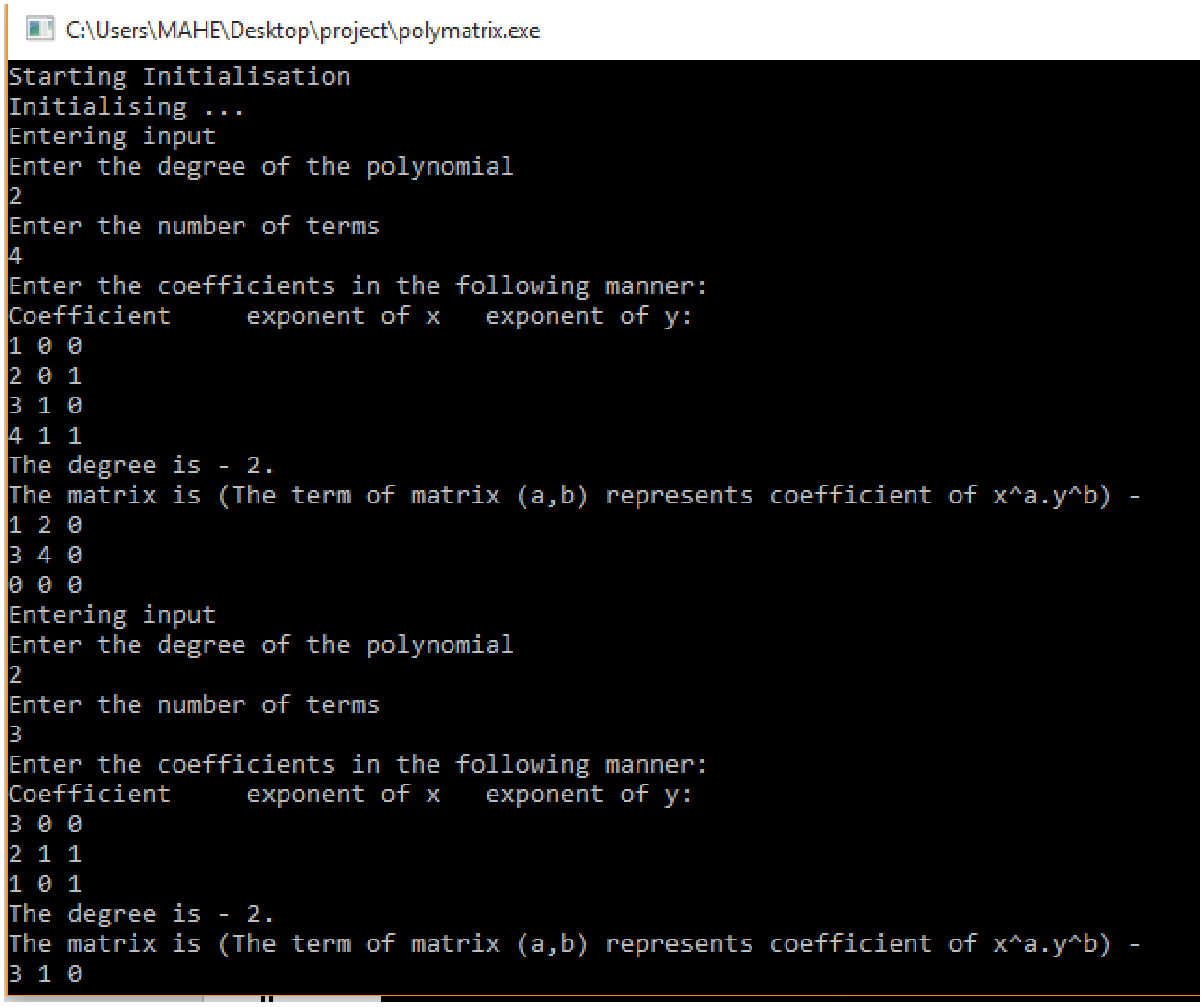
Array Implementation :



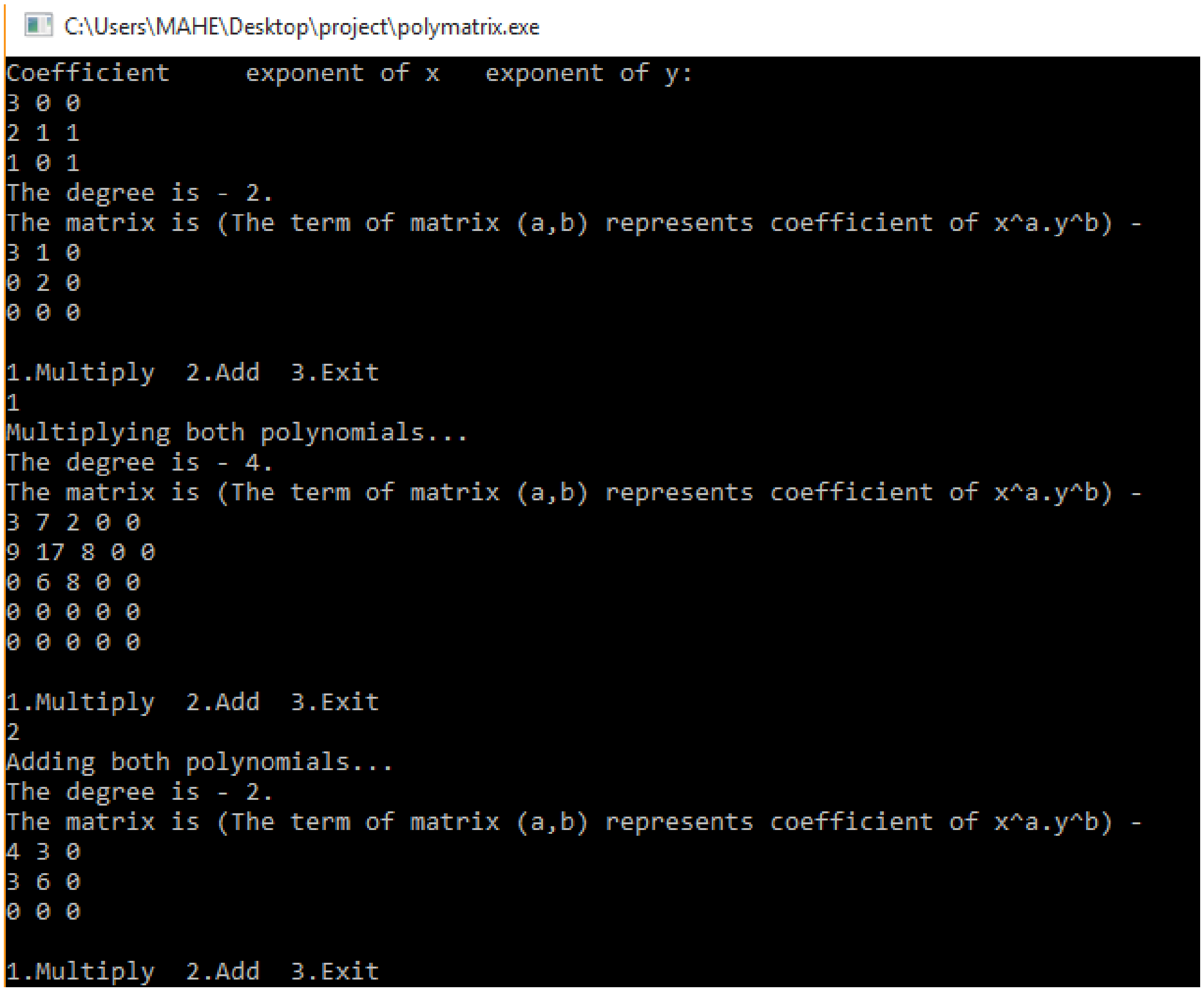
```
Starting Initialisation
Initialising ...
Entering input
Enter the degree of the polynomial
2
Enter the number of terms
4
Enter the coefficients in the following manner:
Coefficient     exponent of x   exponent of y:
1 0 0
2 0 1
3 1 0
4 1 1
The degree is - 2.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
1 2 0
3 4 0
0 0 0
Entering input
Enter the degree of the polynomial
2
Enter the number of terms
3
Enter the coefficients in the following manner:
Coefficient     exponent of x   exponent of y:
3 0 0
2 1 1
1 0 1
The degree is - 2.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
3 1 0
0 2 0
0 0 0

1.Multiply  2.Add  3.Exit
1
```



```
1.Multiply  2.Add  3.Exit
1
Multiplying both polynomials...
The degree is - 4.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
3 7 2 0 0
9 17 8 0 0
0 6 8 0 0
0 0 0 0 0
0 0 0 0 0

1.Multiply  2.Add  3.Exit
2
Adding both polynomials...
The degree is - 2.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
4 3 0
3 6 0
0 0 0

1.Multiply  2.Add  3.Exit
```


C:\Users\MAHE\Desktop\project\polymatrix.exe

```
Starting Initialisation
Initialising ...
Entering input
Enter the degree of the polynomial
2
Enter the number of terms
4
Enter the coefficients in the following manner:
Coefficient     exponent of x   exponent of y:
1 0 0
2 0 1
3 1 0
4 1 1
The degree is - 2.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
1 2 0
3 4 0
0 0 0
Entering input
Enter the degree of the polynomial
2
Enter the number of terms
3
Enter the coefficients in the following manner:
Coefficient     exponent of x   exponent of y:
3 0 0
2 1 1
1 0 1
The degree is - 2.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
3 1 0
```

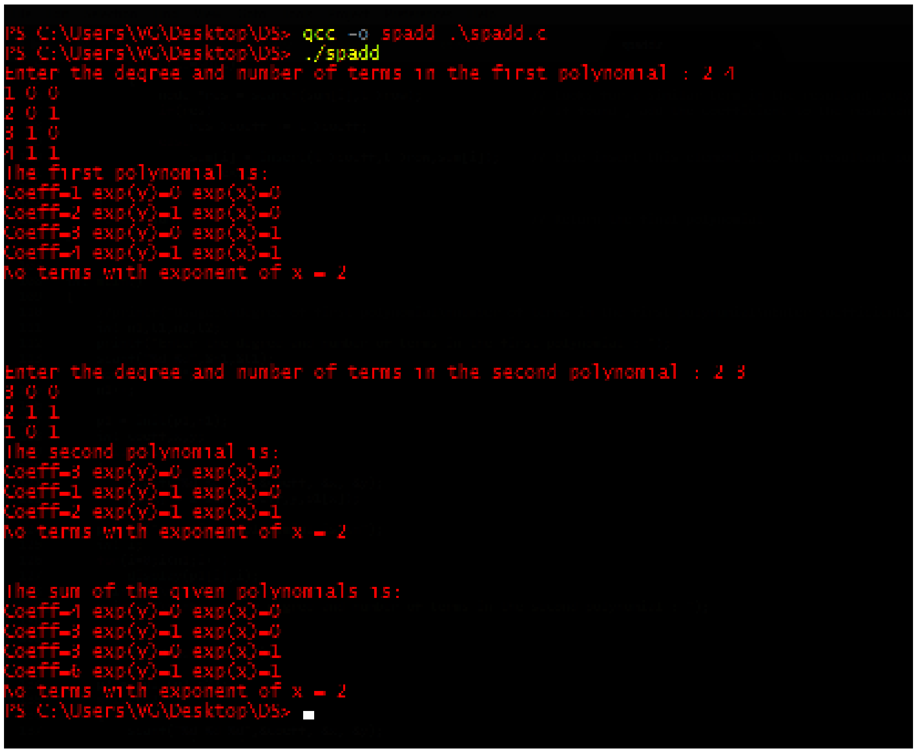


C:\Users\MAHE\Desktop\project\polymatrix.exe

```
Coefficient     exponent of x   exponent of y:
3 0 0
2 1 1
1 0 1
The degree is - 2.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
3 1 0
0 2 0
0 0 0

1.Multiply  2.Add  3.Exit
1
Multiplying both polynomials...
The degree is - 4.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
3 7 2 0 0
9 17 8 0 0
0 6 8 0 0
0 0 0 0 0
0 0 0 0 0

1.Multiply  2.Add  3.Exit
2
Adding both polynomials...
The degree is - 2.
The matrix is (The term of matrix (a,b) represents coefficient of x^a.y^b) -
4 3 0
3 6 0
0 0 0

1.Multiply  2.Add  3.Exit
```

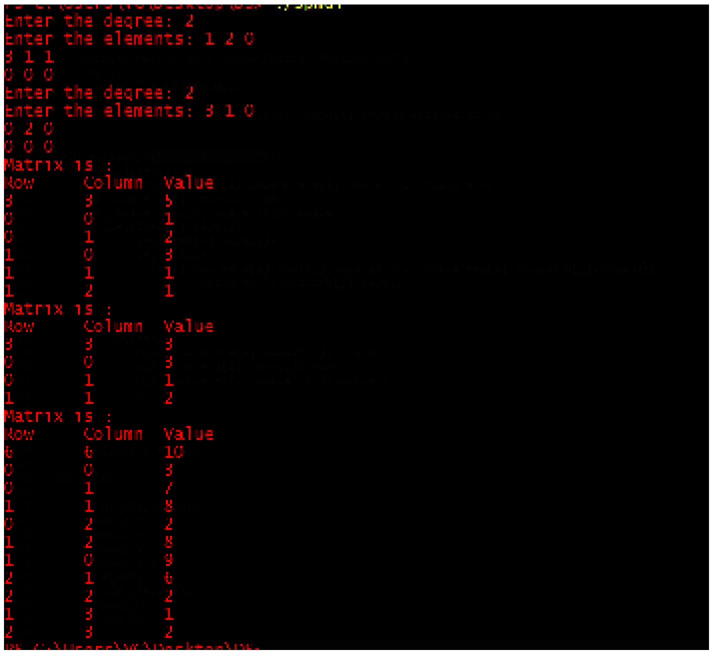
Sparse Matrix Implementation :

Addition:

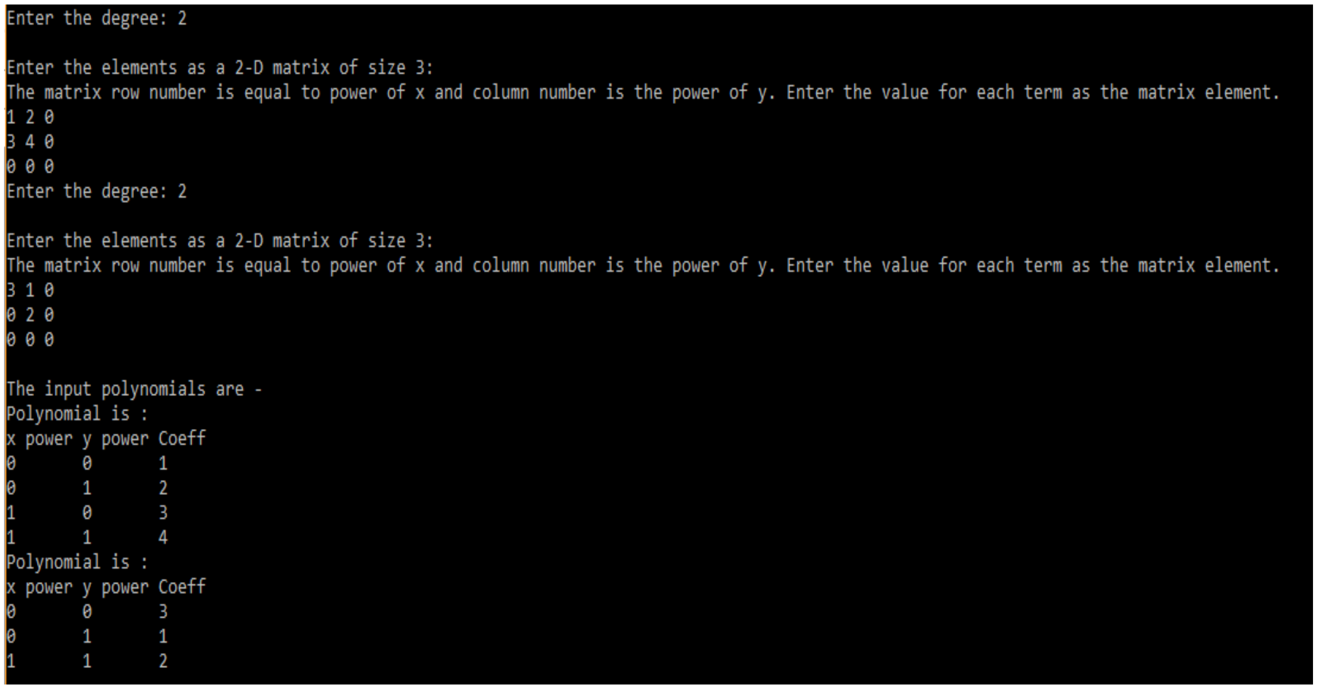


```
PS C:\Users\WG\Desktop\DS> gcc -o spadd .\spadd.c
PS C:\Users\WG\Desktop\DS> ./spadd
Enter the degree and number of terms in the first polynomial : 2 4
1 0 0
2 0 1
3 1 0
4 1 1
The first polynomial is:
Coeff=1 exp(y)=0 exp(x)=0
Coeff=2 exp(y)=1 exp(x)=0
Coeff=3 exp(y)=0 exp(x)=1
Coeff=4 exp(y)=1 exp(x)=1
No terms with exponent of x = 2

Enter the degree and number of terms in the second polynomial : 2 3
3 0 0
2 1 1
1 0 1
The second polynomial is:
Coeff=3 exp(y)=0 exp(x)=0
Coeff=1 exp(y)=1 exp(x)=0
Coeff=2 exp(y)=1 exp(x)=1
No terms with exponent of x = 2

The sum of the given polynomials is:
Coeff=4 exp(y)=0 exp(x)=0
Coeff=3 exp(y)=1 exp(x)=0
Coeff=3 exp(y)=0 exp(x)=1
Coeff=6 exp(y)=1 exp(x)=1
No terms with exponent of x = 2
PS C:\Users\WG\Desktop\DS>
```

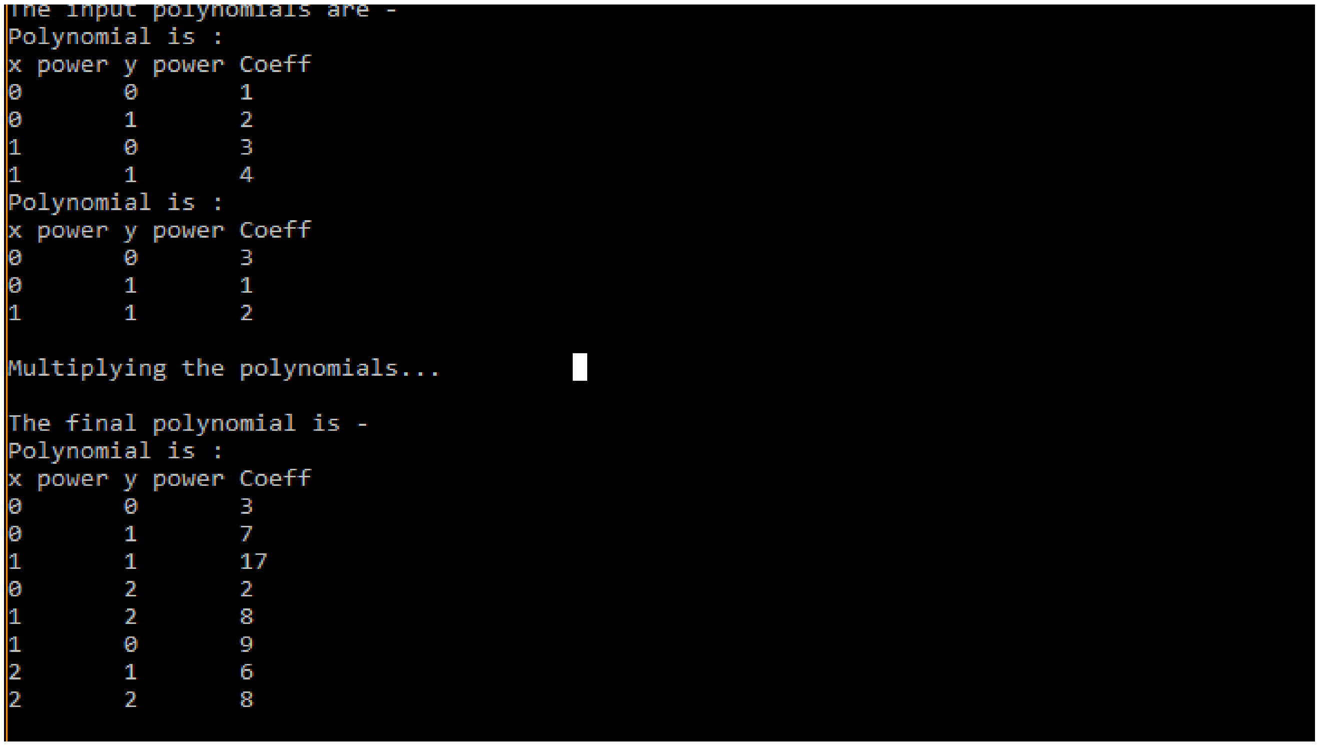
Multiplication:



```
Enter the degree: 2
Enter the elements: 1 2 0
3 1 1
0 0 0
Enter the degree: 2
Enter the elements: 3 1 0
0 2 0
0 0 0
Matrix is :
Row     Column  Value
3       3       5
0       0       1
0       1       2
1       0       3
1       1       1
1       2       1
Matrix is :
Row     Column  Value
3       3       3
0       0       3
0       1       1
1       1       2
Matrix is :
Row     Column  Value
6       6       10
0       0       3
0       1       7
1       1       8
0       2       2
1       2       8
1       0       9
2       1       6
2       2       2
1       3       1
2       3       2
PS C:\Users\WG\Desktop\DS>
```



```
Enter the degree: 2

Enter the elements as a 2-D matrix of size 3:
The matrix row number is equal to power of x and column number is the power of y. Enter the value for each term as the matrix element.
1 2 0
3 4 0
0 0 0
Enter the degree: 2

Enter the elements as a 2-D matrix of size 3:
The matrix row number is equal to power of x and column number is the power of y. Enter the value for each term as the matrix element.
3 1 0
0 2 0
0 0 0

The input polynomials are -
Polynomial is :
x power y power Coeff
0       0       1
0       1       2
1       0       3
1       1       4
Polynomial is :
x power y power Coeff
0       0       3
0       1       1
1       1       2
```



```
The input polynomials are -
Polynomial is :
x power y power Coeff
0       0       1
0       1       2
1       0       3
1       1       4
Polynomial is :
x power y power Coeff
0       0       3
0       1       1
1       1       2

Multiplying the polynomials...

The final polynomial is -
Polynomial is :
x power y power Coeff
0       0       3
0       1       7
1       1       17
0       2       2
1       2       8
1       0       9
2       1       6
2       2       8
```

Limitations:

- Evaluating only two polynomials.
- Array implementation use up a lot of space and also allocates space for non existent terms in the polynomial. The sparse matrix implementation allocates memory only for the required terms in the polynomial.
- Time complexity for sparse matrix is large due to unnecessary traversals as opposed to array implementation.