

**PROGRAM -01**  
**POLYNOMIAL MULTIPLICATION in “C”**

**THEORY:**

What is a polynomial?

“A polynomial is a sum of terms, where each term has a form  $ax^e$ , where  $x$  is the variable,  $a$  is the coefficient and  $e$  is the exponent.”

Two example polynomials are:

$$A(x) = 3x^{20} + 2x^5 + 4$$

$$B(x) = x^4 + 10x^3 + 3x^2 + 1$$

**DESIGN & ALGORITHM:**

A simple solution is to one by one consider every term of the first polynomial and multiply it with every term of the second polynomial. Following is the algorithm of this simple method.

Multiply ( $A[0..m-1]$ ,  $B[0..n-1]$ )

- 1) Create a product array  $prod[]$  of size  $m+n-1$ .
- 2) Initialize all entries in  $prod[]$  as 0.
- 3) Traverse array  $A[]$  and do following for every element  $A[i]$   
    Traverse array  $B[]$  and do following for every element  $B[j]$   
         $prod[i+j] = prod[i+j] + A[i] * B[j]$
- 4) Return  $prod[]$ .

## **PROGRAM:**

```
#include<stdio.h>

int main()
{
    int i,j,sizePoly1,sizePoly2;

    printf("Enter number of terms in Polynomial 1\n");
    scanf("%d",&sizePoly1);

    printf("Enter number of terms in Polynomial 2\n");
    scanf("%d",&sizePoly2);


    int a[sizePoly1],b[sizePoly2],prod[sizePoly1+sizePoly2];

    printf("Enter Elements of Polynomial 1\n");
    for(i=0;i<sizePoly1;i++)
    {
        printf("Enter x^%d Co-Efficient of Polynomial 1\n",i);
        scanf("%d",&a[i]);
    }


    printf("Enter Elements of Polynomial 2\n");
    for(i=0;i<sizePoly2;i++)
    {
        printf("Enter x^%d Co-Efficient of Polynomial 2\n",i);
        scanf("%d",&b[i]);
    }
```

```
for(i=0;i<sizePoly1+sizePoly2;i++)  
{  
    prod[i]=0;  
}
```

```
for(i=0;i<sizePoly1;i++)  
{  
    for(j=0;j<sizePoly2;j++)  
    {  
        if(a[i]!=0 && b[j]!=0)  
            prod[i+j]+=a[i]*b[j];  
    }  
}
```

```
for(i=sizePoly1+sizePoly2-1;i>=0;i--)  
{  
    if(prod[i]!=0){  
        if(i!=0)  
        {  
            printf("%d x^%d + ",prod[i],i);  
        }  
        else  
        {  
            printf("%d x^%d\n",prod[i],i);  
        }  
    }  
}
```

```
}  
  
}  
  
return 0;  
  
}
```

### **OUTPUT:**

```
PS C:\Users\dell\Desktop\dsa programs> cd "c:\Users\dell\Desktop\dsa  
programs\" ; if ($?)
```

```
{ gcc mul.c -o mul } ; if ($?) { .\mul }
```

Enter number of terms in Polynomial 1

4

Enter number of terms in Polynomial 2

3

Enter Elements of Polynomial 1

Enter x<sup>0</sup> Co-Efficient of Polynomial 1

3

Enter x<sup>1</sup> Co-Efficient of Polynomial 1

4

Enter x<sup>2</sup> Co-Efficient of Polynomial 1

5

Enter x<sup>3</sup> Co-Efficient of Polynomial 1

6

Enter Elements of Polynomial 2

Enter x<sup>0</sup> Co-Efficient of Polynomial 2

1

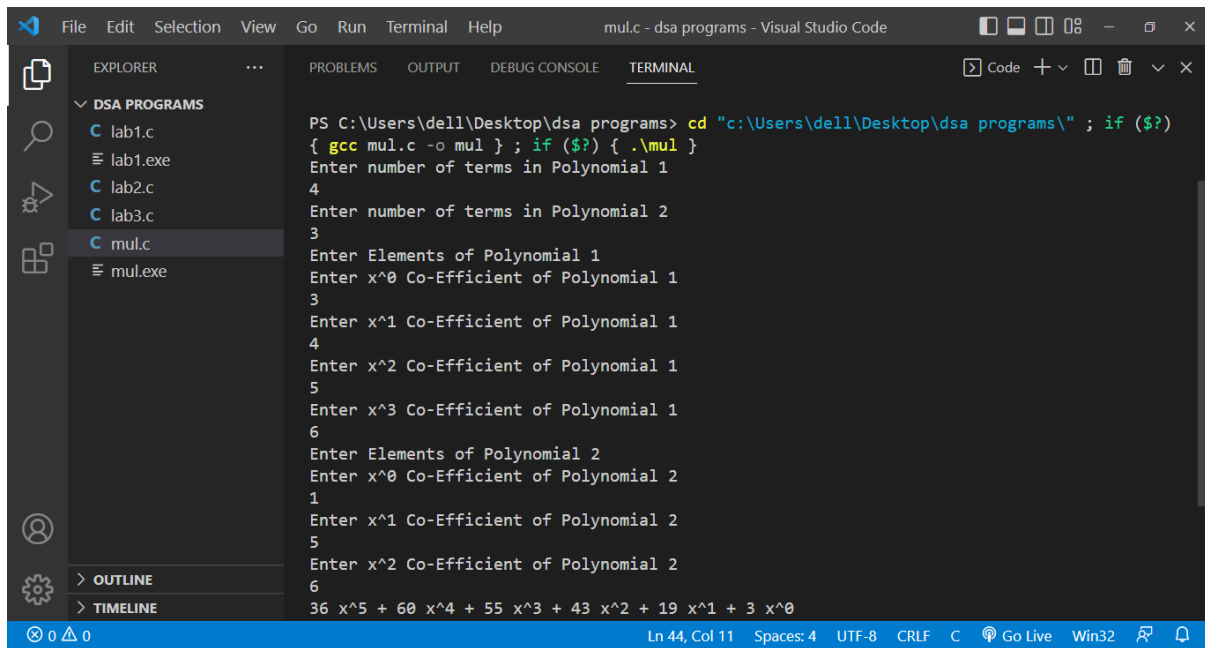
Enter x<sup>1</sup> Co-Efficient of Polynomial 2

5

Enter  $x^2$  Co-Efficient of Polynomial 2

6

$36x^5 + 60x^4 + 55x^3 + 43x^2 + 19x^1 + 3x^0$



The screenshot shows the Visual Studio Code interface with the 'TERMINAL' panel active. The terminal displays the execution of a C program named 'mul.c' which calculates the product of two polynomials. The program prompts the user for the number of terms and coefficients for two polynomials. The input provided is 4 terms for Polynomial 1 and 3 terms for Polynomial 2. The resulting polynomial is displayed at the bottom of the terminal output.

```
PS C:\Users\dell\Desktop\dsa programs> cd "c:\Users\dell\Desktop\dsa programs\" ; if ($?) { gcc mul.c -o mul } ; if ($?) { .\mul }
Enter number of terms in Polynomial 1
4
Enter number of terms in Polynomial 2
3
Enter Elements of Polynomial 1
Enter x^0 Co-Efficient of Polynomial 1
3
Enter x^1 Co-Efficient of Polynomial 1
4
Enter x^2 Co-Efficient of Polynomial 1
5
Enter x^3 Co-Efficient of Polynomial 1
6
Enter Elements of Polynomial 2
Enter x^0 Co-Efficient of Polynomial 2
1
Enter x^1 Co-Efficient of Polynomial 2
5
Enter x^2 Co-Efficient of Polynomial 2
6
36 x^5 + 60 x^4 + 55 x^3 + 43 x^2 + 19 x^1 + 3 x^0
```

Fig 1: Output Screenshot

## **PROGRAM -02**

### **TRANSPOSE OF A SPARSE MATRIX in “C”**

#### **THEORY:**

- *What is Sparse Matrix?*

A matrix which contains many zero entries or very few non-zero entries is called as Sparse matrix.

- *Transposing a Matrix*

To transpose a matrix, interchange the rows and columns. This means that each element  $a[i][j]$  in the original matrix becomes element  $a[j][i]$  in the transpose matrix.

- *Sparse Matrix Representation*

An element within a matrix can characterize by using the triple  $\langle \text{row}, \text{col}, \text{value} \rangle$ , an array of triples is used to represent a sparse matrix. Organize the triples so that the row indices are in ascending order. The operations should terminate, so we must know the number of rows and columns, and the number of nonzero elements in the matrix.

$a[0].\text{row}$  contains the number of rows,

$a[0].\text{col}$  contains the number of columns

$a[0].\text{value}$  contains the total number of nonzero entries.

#### **DESIGN:**

To Transpose a matrix, we can simply change every column value to the row value and vice-versa, however, in this case, the resultant matrix won't be sorted as we require. Hence, we initially determine the number of elements less than the current element's column being inserted in order to get the exact index of the resultant matrix where the current element should be placed. This is done by maintaining an array  $\text{index}[]$  whose  $i$ th value indicates the number of elements in the matrix less than the column  $i$ .

## PROGRAM:

```
/* C program to read sparse matrix and find the transpose of a matrix*/
```

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

```
#define MAX 20
```

```
void printsparse(int[][3]);
```

```
void readsparse(int[][3]);
```

```
void transpose(int[][3],int[][3]);
```

```
int main()
```

```
{
```

```
int b1[MAX][3], b2[MAX][3], m, n;
```

```
printf("Enter the size of matrix (rows,columns):");
```

```
scanf("%d%d",&m,&n);
```

```
b1[0][0]=m;
```

```
b1[0][1]=n;
```

```
readsparse(b1);
```

```
transpose(b1,b2);
```

```
printsparse(b2);
```

```
}
```

```
void readsparse(int b[MAX][3])
```

```
{
```

```
int i,t;
```

```
printf("\nEnter no. of non-zero elements:");
```

```
scanf("%d",&t);
```

```
b[0][2]=t;
```

```
for(i=1;i<=t;i++)
```

```
{
```

```

printf("\nEnter the next triple(row,column,value):");
scanf("%d%d%d",&b[i][0],&b[i][1],&b[i][2]);
}
}
void printspase(int b[MAX][3])
{
int i,n;
n=b[0][2];
//no of 3-triples
printf("\nAfter Transpose:\n");
printf("\nrow\t\tcolumn\t\tvalue\n");
for(i=0;i<=n;i++)
printf("%d\t\t%d\t\t%d\n",b[i][0],b[i][1],b[i][2]);
}
void transpose(int b1[][3],int b2[][3])
{
int i,j,k,n;
b2[0][0]=b1[0][1];
b2[0][1]=b1[0][0];
b2[0][2]=b1[0][2];
k=1;
n=b1[0][2];
for(i=0;i<b1[0][1];i++)
for(j=1;j<=n;j++)
if(i==b1[j][1])
{

```



```
b2[k][0]=b1[j][1];
```

```
b2[k][1]=b1[j][0];
```

```
b2[k][2]=b1[j][2];
```

```
k++;
```

```
}
```

```
}
```

### **OUTPUT:**

```
PS C:\Users\dell\Desktop\dsa programs> cd "c:\Users\dell\Desktop\dsa  
programs\" ; if ($?)
```

```
{ gcc sparse.c -o sparse } ; if ($?) { .\sparse }
```

```
Enter the size of matrix (rows,columns):6 6
```

```
Enter no. of non-zero elements:8
```

```
Enter the next triple(row,column,value):0 0 15
```

```
Enter the next triple(row,column,value):0 3 22
```

```
Enter the next triple(row,column,value):0 5 -15
```

```
Enter the next triple(row,column,value):1 1 11
```

```
Enter the next triple(row,column,value):1 2 3
```

```
Enter the next triple(row,column,value):2 3 -6
```

```
Enter the next triple(row,column,value):4 0 91
```

Enter the next triple(row,column,value):5 2 28

After Transpose:

row	column	value
6	6	8
0	0	15
0	4	91
1	1	11
2	1	3
3	0	22
3	2	-6
5	0	-15

```
{ gcc sparse.c -o sparse } ; if ($?) { .\sparse }
Enter the size of matrix (rows,columns):6 6

Enter no. of non-zero elements:8

Enter the next triple(row,column,value):0 0 15
Enter the next triple(row,column,value):0 3 22
Enter the next triple(row,column,value):0 5 -15
Enter the next triple(row,column,value):1 1 11
Enter the next triple(row,column,value):1 2 3
Enter the next triple(row,column,value):2 3 -6
Enter the next triple(row,column,value):4 0 91
Enter the next triple(row,column,value):5 2 28

After Transpose:

row      column  value
6         6      8
0         0     15
0         4     91
1         1     11
2         1      3
2         5     28
3         0     22
3         2     -6
5         0    -15

PS C:\Users\dell\Desktop\dsa programs>
```

Fig 2:Output Screenshot

