**ASSIGNMENT – 20**

**1.PROBLEM STATEMENT**

Write a program in C to scan polynomial using array. Implement addition, subtraction, multiplication of two polynomials.

**2.ALGORITHMS**

Algorithm **Add\_Polynomial**

**Input:** Pointers ‘poly1’ and ‘poly2’ that point to the array holding the first and second polynomial respectively with their respective lengths ‘size1’ and ‘size2’.

**Output:** The two polynomials added together and stored in an array named ‘res’, whose pointer is returned.

**Remarks:** Each element of the polynomial array has two members namely ‘coeff’ and ‘pow’ representing the coeffiecent and exponent of each term respectively.

**Steps:**

1. A polynomial array named ‘res’ is allocated with size equal to the sum of size1 and size2.
2. i=0,j=0,k=0 //initialising loop counters
3. **While**(i<size1 **AND** j<size2) **do** //Traversing both lists until one ends
4. **If**(pow🡨poly1[i]==pow🡨poly2[j]) **then**//if pow of poly1 item is equal
5. //assign sum of coefficients of poly1 and poly2 to res
6. coeff🡨res[k] = (coeff🡨poly1[i]) + (coeff🡨poly2[j])
7. pow🡨res[k] = pow🡨poly1[i] //assign pow of poly1 term to res term
8. i=i+1,j=j+1,k=k+1 //increment i,j,k by 1
9. **Else**
10. **If**(pow🡨poly1[i]>pow🡨poly2[j]) **then**//pow of poly2 term is greater
11. coeff🡨res[k] = coeff🡨poly1[i]//coeff of poly1 = coeff of res
12. pow🡨res[k] = pow🡨poly1[i]//pow of poly1 = pow of res
13. i=i+1,k=k+1 //increment i,k by 1
14. **Else**
15. coeff🡨res[k] = coeff🡨poly2[j]//coeff of poly2 = coeff of res
16. pow🡨res[k] = pow🡨poly2[j]//pow of poly2 = pow of res
17. j++;k++;//increment j,k by 1
18. **EndIf**
19. **EndIf**
20. **While**(i<size) **do** //while poly1 does not end
21. coeff🡨res[k]= coeff🡨poly1[i]//coeff of poly1 = coeff of res
22. pow🡨res[k]= pow🡨poly1[i] //pow of poly1 = pow of res
23. i=1+1,k=k+1 //increment i,k by 1
24. **EndWhile**
25. **While**(j<size2) **do** //while poly2 does not end
26. coeff🡨res[k]= coeff🡨poly2[j] //coeff of poly2 = coeff of res
27. pow🡨res[k]= pow🡨poly2[j] //pow of poly2 = pow of res
28. j=j+1;k=k+1; //increment j,k by 1
29. **EndWhile**
30. **Return** res

Algorithm **Subtract\_Polynomial**

**Input:** Pointers ‘poly1’ and ‘poly2’ that point to the array holding the first and second polynomial respectively with their respective lengths ‘size1’ and ‘size2’.

**Output:** The difference of the two polynomials stored in an array named ‘res’, whose pointer is returned.

**Remarks:** Each element of the polynomial array has two members namely ‘coeff’ and ‘pow’ representing the coeffiecent and exponent of each term respectively.

**Steps:**

1. A polynomial array named ‘res’ is allocated with size equal to the sum of size1 and size2.
2. i=0,j=0,k=0 //initialising loop counters
3. **While**(i<size1 **AND** j<size2) **do** //Traversing both lists until one ends
4. **If**(pow🡨poly1[i]==pow🡨poly2[j]) **then**//if pow of poly1 item is equal
5. //assign sum of coefficients of poly1 and poly2 to res
6. coeff🡨res[k] = (coeff🡨poly1[i]) - (coeff🡨poly2[j])
7. pow🡨res[k] = pow🡨poly1[i] //assign pow of poly1 term to res term
8. i=i+1,j=j+1,k=k+1 //increment i,j,k by 1
9. **Else**
10. **If**(pow🡨poly1[i]>pow🡨poly2[j]) **then**//pow of poly2 term is greater
11. coeff🡨res[k] = coeff🡨poly1[i]//coeff of poly1 = coeff of res
12. pow🡨res[k] = pow🡨poly1[i]//pow of poly1 = pow of res
13. i=i+1,k=k+1 //increment i,k by 1
14. **Else**
15. coeff🡨res[k] = coeff🡨poly2[j]//coeff of poly2 = coeff of res
16. pow🡨res[k] = pow🡨poly2[j]//pow of poly2 = pow of res
17. j++;k++;//increment j,k by 1
18. **EndIf**
19. **EndIf**
20. **While**(i<size) **do** //while poly1 does not end
21. coeff🡨res[k]= coeff🡨poly1[i]//coeff of poly1 = coeff of res
22. pow🡨res[k]= pow🡨poly1[i] //pow of poly1 = pow of res
23. i=1+1,k=k+1 //increment i,k by 1
24. **EndWhile**
25. **While**(j<size2) **do** //while poly2 does not end
26. coeff🡨res[k]= coeff🡨poly2[j] //coeff of poly2 = coeff of res
27. pow🡨res[k]= pow🡨poly2[j] //pow of poly2 = pow of res
28. j=j+1;k=k+1; //increment j,k by 1
29. **EndWhile**
30. **Return** res

Algorithm **Multiply\_Polynomial**

**Input:** Pointers ‘poly1’ and ‘poly2’ that point to the array holding the first and second polynomial respectively with their respective lengths ‘size1’ and ‘size2’.

**Output:** The product of the two polynomials stored in a polynomial array named ‘res’ whose pointer is returned.

**Remarks:** The product polynomial is stored without sorting it in decending order of the exponent of the terms.

**Steps:**

1. A polynomial array named ‘res’ is allocated with size equal to the product of size1 and size2.
2. i=j=k=0 // initialising loop counters
3. **For**(i=0 to size1) **do** //traversing poly1 till end
4. **For**(j=0 to size2) **do** //traversing poly2 till end
5. coeff🡨res[k]=(coeff🡨poly1[i])\*(coeff🡨poly2[j]//storing product
6. pow🡨res[k]= (pow🡨poly1[i])+(pow-<poly2[j])//storing exponent
7. k++ //increment k by 1
8. **EndFor**
9. **EndFor**
10. **Return** res

**3. SOURCE CODE**

#include<stdio.h>

#include<stdlib.h>

//structure to create an array for holding polynomials

typedef struct term

{

int coeff;

int pow;

}term;

int count=0;

//function to take input in a polynomial array

void getpoly(term\* poly,int len)

{

int i;

for(i=0;i<len;i++)

{

printf("Enter Coefficient: ");

scanf("%d",&poly[i].coeff);

printf("Enter Exponent: ");

scanf("%d",&poly[i].pow);

}

}

//function to display a polynomial array

void dispoly(term \*poly,int len)

{

int i;

for(i=0;i<len;i++)

if(poly[i].coeff!=0)

printf("%+dx^%d ",poly[i].coeff,poly[i].pow);

}

//function to add two polynomials together

term\* poly\_add(term \*poly1,int size1,term \*poly2,int size2)

{

term \*res=(term\*)malloc((size1+size2)\*sizeof(term));//allocate res array

int i=0,j=0,k=0; //set loop counters

while(i<size1 && j<size2) //while both arrays are not exhausted

{

if(poly1[i].pow==poly2[j].pow) //if powers of both terms are equal

{

res[k].coeff = (poly1[i].coeff) + (poly2[j].coeff);//add terms

res[k].pow = poly1[i].pow;//set power of res term to poly1 term

i++;j++;k++; //increment i,j,k by one

}

else

{

if(poly1[i].pow>poly2[j].pow) //if power of poly1 term is greater

{

res[k].coeff = poly1[i].coeff; //set res coeff to poly1 coeff

res[k].pow = poly1[i].pow;//set res pow to poly1 pow

i++;k++;//increment i,j by one

}

else

{

res[k].coeff = poly2[j].coeff; //set res coeff to poly2 coeff

res[k].pow = poly2[j].pow;//set res pow to poly1 pow

j++;k++; //increment j,k by one

}

}

}

while(i<size1) //if poly1 remains

{

res[k].coeff=poly1[i].coeff; //set res coeff to poly1 coeff

res[k].pow=poly1[i].pow; //set res pow to poly1 pow

i++;k++; //increment i,k by one

}

while(j<size2) //if poly2 remains

{

res[k].coeff=poly2[j].coeff; //set res coeff to poly1 coeff

res[k].pow=poly2[j].pow; //set res pow to poly1 pow

j++;k++;//increment j,k by one

}

count=k;

return res;

}

term\* poly\_sub(term \*poly1,int size1,term \*poly2,int size2)

{

term \*res=(term\*)malloc((size1+size2)\*sizeof(term));//allocate res array

int i=0,j=0,k=0; //set loop counters

while(i<size1 && j<size2) //while both arrays are not exhausted

{

if(poly1[i].pow==poly2[j].pow) //if powers of both terms are equal

{

res[k].coeff = (poly1[i].coeff) - (poly2[j].coeff);//add terms

res[k].pow = poly1[i].pow;//set power of res term to poly1 term

i++;j++;k++; //increment i,j,k by one

}

else

{

if(poly1[i].pow>poly2[j].pow) //if power of poly1 term is greater

{

res[k].coeff = poly1[i].coeff; //set res coeff to poly1 coeff

res[k].pow = poly1[i].pow;//set res pow to poly1 pow

i++;k++;//increment i,j by one

}

else

{

res[k].coeff = poly2[j].coeff; //set res coeff to poly2 coeff

res[k].pow = poly2[j].pow;//set res pow to poly1 pow

j++;k++; //increment j,k by one

}

}

}

while(i<size1) //if poly1 remains

{

res[k].coeff=poly1[i].coeff; //set res coeff to poly1 coeff

res[k].pow=poly1[i].pow; //set res pow to poly1 pow

i++;k++; //increment i,k by one

}

while(j<size2) //if poly2 remains

{

res[k].coeff=poly2[j].coeff; //set res coeff to poly1 coeff

res[k].pow=poly2[j].pow; //set res pow to poly1 pow

j++;k++;//increment j,k by one

}

count=k;

return res;

}

term\* poly\_mult(term \*poly1,int size1,term \*poly2,int size2)

{

term \*res=(term\*)malloc((size1\*size2)\*sizeof(term));//allocate res array

int i,j,k=0;//set loop counters

for(i=0;i<size1;i++) //traversing poly1

for(j=0;j<size2;j++)//traversing poly2

{

res[k].coeff=(poly1[i].coeff)\*(poly2[j].coeff);

res[k].pow=(poly1[i].pow)+(poly2[j].pow);

k++;

}

count=k;

return res;

}

//function for input validation

void validate(term \*poly,int size)

{

int i;

for(i=0;i<size-1;i++)

{

if(poly[i].pow<poly[i+1].pow)

{

printf("The terms must be in decending order of their exponents\nPlease Retry");

exit(1);

}

}

}

int main(void)

{

int size1,size2,ch;

term \*poly1,\*poly2,\*res;

printf("To perfrom arithmetic operations on two polynomials \n");

printf("Enter the number of terms in first polynomial: ");

scanf("%d",&size1);

printf("Enter the number of terms in second polynomial: ");

scanf("%d",&size2);

poly1=(term\*)malloc(size1\*sizeof(term));

poly2=(term\*)malloc(size2\*sizeof(term));

printf("\nPlease enter terms in decending order of their exponents: \n");

printf("For the first polynomial: \n");

getpoly(poly1,size1);

validate(poly1,size1);

printf("For the second polynomial: \n");

getpoly(poly2,size2);

validate(poly2,size2);

printf("First polynomial: ");

dispoly(poly1,size1);

printf("\nSecond polynomial: ");

dispoly(poly2,size2);

printf("\nMENU:\n");

printf("1.Addition\n2.Subtraction\n3.Multiplication");

printf("\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

res=poly\_add(poly1,size1,poly2,size2);

printf("SUM: ");

dispoly(res,count);

break;

case 2:

res=poly\_sub(poly1,size1,poly2,size2);

printf("DIFFERENCE: ");

dispoly(res,count);

break;

case 3:

res=poly\_mult(poly1,size1,poly2,size2);

printf("PRODUCT: ");

dispoly(res,count);

break;

free(poly1);free(poly2);free(res);

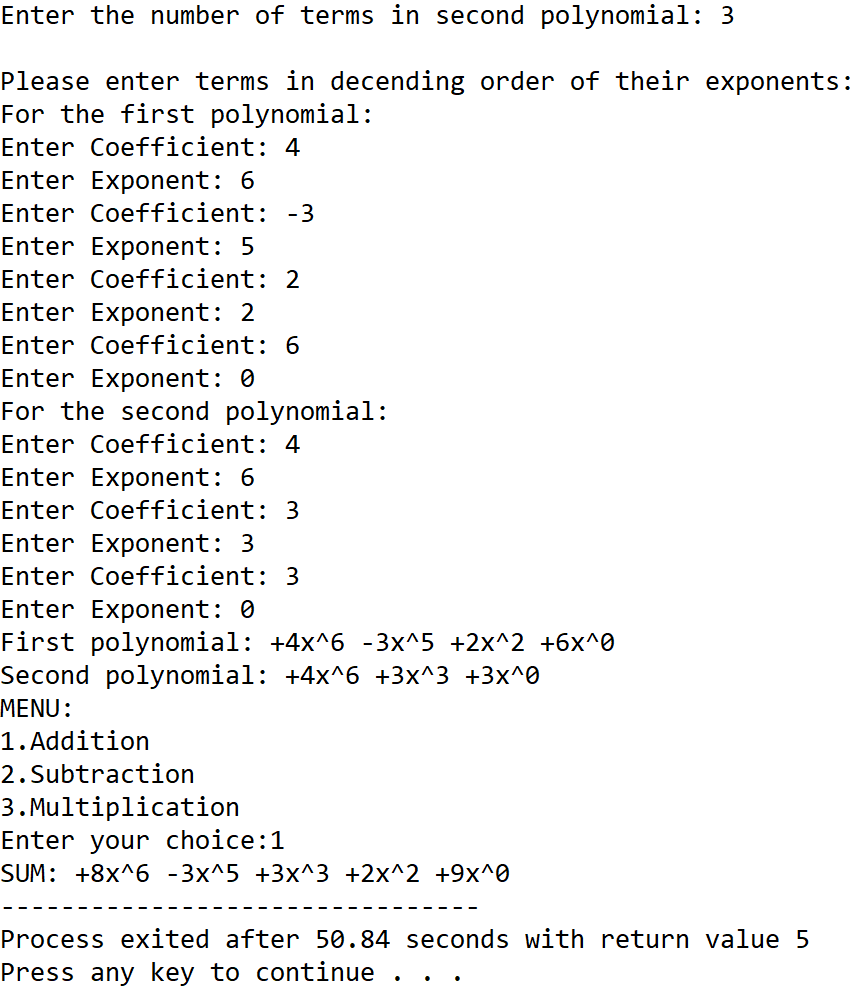
return 0;

}

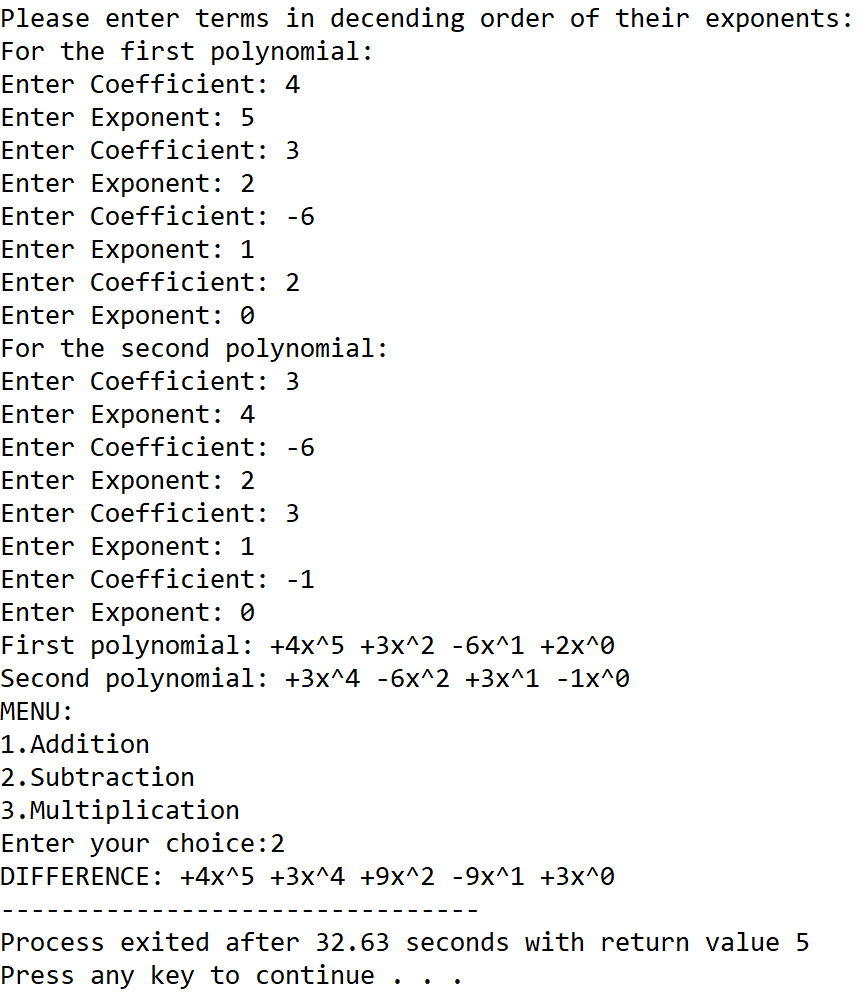
}

**4.OUTPUT**

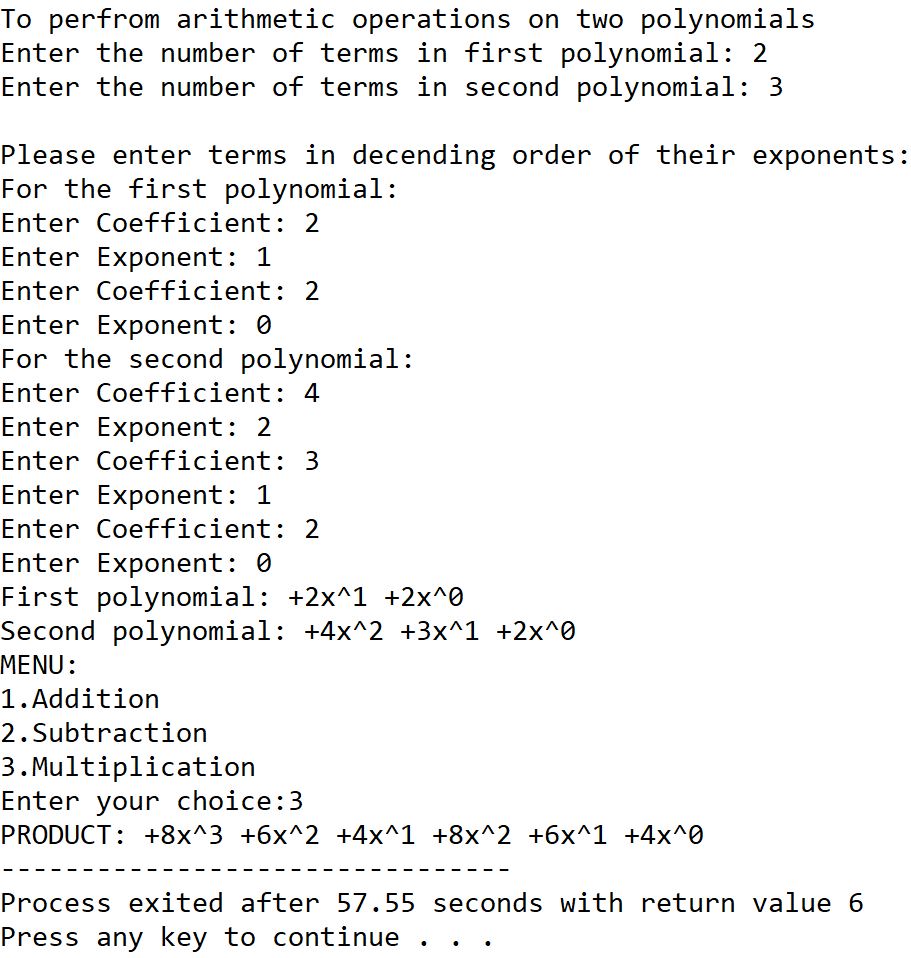
**SET 1:** Addition of two polynomials

****

**SET 2:** Subtraction of two polynomials

****

**SET 3:** Multiplication of two polynomials



**5.DISCUSSIONS**

**Variable Description**

* **size1:** Number of terms in first polynomial.
* **size2:** Number of terms in second polynomial.
* **ch:** variable to take user input in switch-case-default.
* **\*poly1:** pointer to the array of structures holding the first polynomial.
* **\*poly2:** pointer to the array of structures holding the second polynomial.
* **\*res:** pointer to the array of structures holding the result polynomial after an operation is performed.
* **count:** global variable to track the size of res array
* **i,j,k:** loop counters

**Limitations**

* The program needs the user to enter the polynomials with the exponents sorted in decending order since it applies the concept of merging of two sorted lists into a single list.
* The program can handle polynomials with one variable only.
* The program uses an array of structures which is a static data structure and whose size cannot be manipulated once it is constructed in the memory.

**Uses**

* The program can be used to perfrom addition,subtraction and multiplication of two polynomials having one variable.The program can find implementation in a calculator application.

**Future Scope:**

* The logic of the program can be changed to ask the uses the number of variables present in the polynomial and the operations are done accordingly.
* The data stucture used in the program can be changed to linkedlists making it more memory efficient and removing dependency on contiguous memory locations.