Name	**:** Mu In Nasif
Roll	**:** 001910501036
Class	**:** BCSE II
Sem	**:** 3
Session	**:** 2020-2021

Data Structures and Algorithms Assignment Set 2

Question - 1 : Define an ADT for Polynomials. Write C data structure representation and functions for the operations on the Polynomials in a Header file. Write a menu-driven main program in a separate file for testing the different operations and include the above header file.

Solution Approach :

In mathematics, a polynomial is an expression consisting of variables and constants (coefficients) and involves only addition, subtraction, multiplication and non-negative integer exponentiation of variables. If we consider only one variable, a polynomial P(x) is a function such that

P(x)=∑ i=0 n a x

for all x in the domain (usually real numbers, in this case each ai is a real number and are called the coefficients of the polynomial; we can compose polynomials based on any kind of set of “numbers” where addition, subtraction and multiplication are closed binary operations on the set). Each term in the given series is called a term of the polynomial and comprises only the coefficient(positive or negative) multiplied by the variable raised to some non-negative integer exponent (called the degree of the term). If a particular term of degree j is absent, we take aj to be zero for that term. The highest degree of the terms present (n) is called the degree of the entire polynomial. With this definition, we can say a polynomial of degree n is a series of n + 1 terms arranged from degree 0 to degree n, with “absent” terms assigned coefficient 0.

In mathematics, the following operations are available for polynomials:

1. Addition:
P(x) = F(x) + G(x) implies adding all the terms one after the another – P(x) will be such that terms with common degree will have their coefficients added, and “uncommon degree” terms simply listed and all the terms arranged from 0 to n.

Example: (x3 + 3x2 + 1) + (x4 + x2 – 10x) = x4 + x3 + 4x2 – 10x + 1 (terms with no listed coefficient have coefficient 1). The degree of resultant in addition operation is max(degree of F, degree of G).

2. Multiplication with a constant (Called scalar multiplication in linear algebra parlance):
P(x) = c . F(x) implies multiplying each coefficient of F(x) with c. Degree in resultant remains the same.

3. Subtraction:
P(x) = F(x) – G(x) implies P(x) = F(x) + (-1) . G(x).

4. Multiplication with a monomial:
A monomial is a special kind of polynomial with a single term having non-zero coefficient. If we take each term of P(x) as aix i and the monomial as bxc then P(x) . bxc is such that each ith term is aibxi+c . The degree in resultant increases (from the multiplicand polynomial) by c.

5. Multiplication with another polynomial:
If F(x) has degree m and G(x) has degree n, then F(x).G(x) will be a polynomial of degree m + n and can be computed as follows: take each term from G(x) from degree 0 to degree n as a monomial, multiply with F(x) to find Hi(x) (0 <= i <= m) and add all these Hi(x) so formed. For the purpose of implementing in a computer system, these operations are also defined:

4. Checking if the polynomial is zero (meaning all the coefficients are zero and the final value of the polynomial function does not depend on x).

5. Evaluating the polynomial i.e. finding its value as a (real) number given a value for x.

6. Obtaining the coefficient of any degree term – absent terms and terms with a higher degree than the degree of the polynomial have coefficient zero.

7. Finding/obtaining the degree of the polynomial.

For the purposes of implementation, we will represent a polynomial A(x) of degree n as an array A[0..n] (end indexes inclusive) of coefficients; thereby coefficient of term with degree j is A[j] if 0 <= j <= n, otherwise 0 (operation 6). An array was chosen for the advantage of obtaining coefficient of any term in constant time.

Pseudocode :

For this implementation we use two arrays of reasonable size for storing the polynomial coefficient and exponents respectively, a cursor is maintained to store the current index at the array to avoid overlapping of two or more polynomials. A structure is also maintained for each polynomial to store the starting and ending index of the polynomial inside the array.

Storing the polynomials:

Array 1 -> polynomial_exponents
Array 2 -> polynomial_coefficients

```
Struct PolynomialIndex {
    Index_start;
    Index_end;
}
```

Addition of two polynomials :

The result is stored in another polynomial (say p3)

Initialize p3 -> index_start = cursor

Initialize p3 with p1(one of the two polynomial)

```
Loop : index_start to index_end with i
    Polynomial_exponents[cursor] = polynomial_cursor[i]
    Polynomial_coefficients[cursor] = polynomial_coefficients[i]
    Increment : cursor, i
```

P3 -> index_end = cursor

Add the two polynomial:

```
Loop: p2->index_start to p2-> index_end with i
    Loop: p3-> index_start to p3-> index_end with j
        If polynomial_exponents[i] == polynomial_exponents[j]
            Polynomial_coefficients[j] += polynomial_coefficients[i]


p3->index_end = cursor-1;
```

Subtraction of two polynomials:

Similar to the addition,
p3-> index_start = cursor

Initialize with p1

Sub:
Loop and search for matching exponents, if found subtract. If not, an extra negative coefficient exponent is added to the list .

Multiplication of two polynomials:

```
Variables:  i, j, flag = 0, temporary_coefficient, temporary_exponent;

Initialize: p3-> index_start = cursor              //p3 being the output / result polynomial

Loop: p1-> index_start to p1-> index_end with i
    Loop: p2-> index_start to p2-> index_end with j
        Temporary_exponent = polynomial_exponent[i] + polynomial_exponent[j]
        Temporary_coefficient = polynomial_coefficient[i]*polynomial_coefficient[j]
        Loop p3->index_start to cursor with x
            If polynomial_exponent[x] == temporary_exponent
                Polynomial_coefficient[x] += temporary_coefficient
                    Increment: flag
                    B reak;
        If flag == 0:
            polynomial_exponent[cursor] = temporary_exponent
            Polynomial_coefficient[cursor] = temporary_coefficient
            Increment: cursor

        Flag = 0
    p3->index_end = cursor - 1
```