**CPSC 1160 SECTION 001 INSTRUCTOR: GLADYS MONAGAN**

# ASSIGNMENT #7: STRUCTS AND DYNAMIC ARRAYS

DUE DATE WITH BRIGHTSPACE: MARCH 8 AT 11:50 PM

## THE LEARNING OUTCOMES

- to work with dynamic memory including writing code without memory leaks
- to manipulate a dynamic data structure (a C++ struct) to represent **dense** univariate polynomials and to do some univariate polynomial operations

## READINGS IN THE TEXTBOOK

chapter 11 Pointers and Dynamic Memory:

§11.1, §11.2, §11.4 , §11.5, §11.6,§11.7, §11.9, §11.10

§11.11, §11.12, §11.14 and §11.15 are covered later in the course.

## PROBLEM AT HAND

How can you represent 'a dense univariate polynomial' with integer coefficients so that we can add them and multiply them? We do not want a numerical solution: we want to manipulate the polynomials symbolically. We want to read the values of the coefficients and the degree of the polynomial at run time and then create and manipulate the polynomials read.

The degree of the polynomial *p* in x is the exponent of the x in the term with the highest exponent.

The main program in univariate interacts with the user

- Reads two univariate polynomials and outputs them.
- Adds the two univariate polynomials and outputs their sum.
- Multiplies two univariate polynomials and outputs their product.
- Destroys them.
- Continues reading/adding/multiplying until the user wants to stop.

 Copyright 2023 Gladys Monagan 

## DATA STRUCTURE

We represent a univariate polynomial with integer coefficients using the following C++ structure

```
struct Univariate {
   int deg;
   int *cffs; // array of coefficients from lowest degree to highest degree
};
```

We could create the univariate polynomial $p = 2 - 7x + 4x^3$ which is of degree 3 as follows

```
   int d = 3;
   int* C;
   // create a dynamic array of coefficients
   C = new int[d + 1];
   C[0] = 2;
   C[1] = -7;
   C[2] = 0;
   C[3] = 4;
   // now create the univariate structure dynamically and assign the fields
   Univariate *p;
   p = new Univariate;
   p->deg = d;
   p->cffs = C;
```

## IMPLEMENT IN UNIVARIATE.CPP THE GIVEN FUNCTION (OPERATOR) PROTOTYPES

Use dynamic arrays for the coefficients so that they work for polynomials of any degree.
To input the polynomial $p = 2 - 7x + 4x^3$ we first read the degree
3
and store it. We now know how big to make the array of coefficients (an array of 4 elements).
Subsequently, the coefficients are read in (error free) increasing degree of x
2 -7 0 4

In your first version that prints the univariate polynomial (with the overloaded operator for output), the univariate could look like
`2*x^0 + -7*x^1 + 0*x^2 + 4*x^3`
where we have used the `^` character for powers (exponents).

Try to improve the output (for BONUS points) so that it becomes, for instance,
`2 -7*x^1 + 4*x^3`

 Copyright 2023 Gladys Monagan 

You will probably find programming `add` more difficult than `times`.
There are two difficulties with addition. One happens when the two polynomials have different degrees. The other is when the coefficient(s) of highest degree cancel as in the last example below.
If after the addition, the resulting polynomial has a highest degree term with a zero coefficient (because of cancellation), you need to allocate a new array of the right (and smaller) size and fill in the smaller size array properly (remembering that there is a zero univariate polynomial which is a special case).
Do not forget to deallocate the array that is no longer needed.
Do not just "patch" the output operator to handle highest degree terms of 0.

**Every polynomial (except for the zero polynomial) must always have a non-zero highest degree term**.

Because of this restriction of having a non-zero highest degree term, you will also have to think of how you are going to represent the zero univariate polynomial, namely, the univariate polynomial that consists of the constant `0`.

Your program should be tested on the following pairs of polynomials

```
a = 1 + 2*x + 3*x^2 + 4*x^3
b = 1 + x + x^2 + x^3

a = 1 + x + x^2
b = 1 + x + x^2 + x^3 + x^4

a = 0
b = 1 + x + x^2

a = 1 + 2*x + 3*x^2
b = 1 - 2*x - 3*x^2
```

For each pair of univariate polynomials, you should read

- The degree of the first polynomial
- The coefficients of the first polynomial
- The degree of the second polynomial
- The coefficients of the second polynomial

and print them out.
Then print out their sum and product.
And finally destroy (delete) them when no longer needed.
Allow your program to continue reading pairs of polynomials until the user wants to stop.

Use the data structure given in univariate.cpp. Do not make U\univariate a C++ class.

## TO SUBMIT WITH BRIGHTSPACE AS A SINGLE ZIP FILE:

1. Univariate.cpp that is properly documented
2. An output file that shows "the actual results" from your program
    a. for the polynomials given above
    b. other tests that illustrate how your program works
3. A README.txt
    **a. An explanation of how the zero polynomial is represented in your program**
    **b. An explanation on how the zero polynomial is read**
    c. Something else that you might need to communicate to us (if needed)
    d. Write in the README.txt that you did the BONUS of pretty printing the output