Due Date
Wednesday, September 25, 2013

Program objectives
The objectives of this assignment are as follows.
An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution (ABET b).

Point value
This program is worth 15 points. The distribution of points will be as follows.

Criterion	**Value**
Global functions	1
Poly class	5
Program style (see below)	3
Correct output with annotation	6

Delivery method
Archive your files using the tar command (see below). Attach the tar file which will be named hw3.tar to an email that you send to the class at csc2421@orion.ucdenver.edu In the Subject field, type HW3. In the body of the email type your name, then send the mail. Archive your files as follows. Notice that all file names are in lowercase.

```
tar -cvf hw3.tar hw3.cpp hw3functions.h hw3functions.cpp polynomial.h polynomial.cpp
```

Background
A polynomial is a function of the following form:

$p(n) = \sum_{i=0}^{d} a_i n^i$, where d is an integer and the a_i are the coefficients with $a_d \neq 0$.

$p(n)$ is said to be a *polynomial* in n of degree d.

Problem
Create a class named polynomial that uses a dynamic array to perform the following arithmetic for polynomials, p_1 and p_2.

1. $p_1 + p_2$
2. $p_1 * p_2$
3. $p_1 (n), n \in Z$

Input

1. A text file consisting of two rows of space-delimited integers. Each row represents the coefficients, a_i for a polynomial p. The first number on each row represents a_0 and the last number a_d. Notice that any a_i (except a_d) may be 0. No coefficient will be missing, even if it is 0. The name of the text file will be available in positional parameter, argv[1].
2. A number, $n \in Z$ input on the command line in positional parameter argv[2] that is used to evaluate polynomial p_1. For example, if p_1 = (1, 0, 3, 0, 0) and n=2, then $p_1(2)$ = 28. Notice that argv[2] is type char*, so it must be converted to type int by your program (a global function).

Class requirements

1. Use a dynamic array to implement your polynomial.
2. Define a constant member to evaluate a polynomial at a value *n*.
3. Define a "setter" to set any coefficient requested.
4. Define a "getter" to return any coefficient requested.
5. Overload operators + and * for the polynomial class.
6. Overload operator << for polynomial class.
    a. Operator << should display the polynomial as an *n*-tuple in the order highest term to lowest term. For example,
    $(3, -2, 0, 0, 7) \equiv 3x^4 - 2x^3 + 7$

Driver

1. Greet the user.
2. Instantiate p_1 and p_2.
3. Read the text file and set the coefficients of p_1 and p_2.
4. Perform the polynomial arithmetic per the problem specification and display the annotated results.
5. Convert argv[2] to an int, evaluate p_1 at this value, and display the annotated result.

Notes

1. Use an appropriate annotation with all output.
2. Make sure that main is short. It should basically be making top-level function calls.
3. Use good program style that includes, but is not limited to, header comments, pre and post conditions, macro guards, whitespace and indentation, self-documenting names, symbolic constants where appropriate (use all caps for the name of each symbolics), separation of interface and implementation, correctly submitted program file (tar archive), and a greeting to start the program.
4. You are not restricted to the polynomial members listed above; that is, you may define other members that you feel necessary.
5. Be sure to use error checking where appropriate.