Assignment 1: Polynomial Multiplication

Overview

The purpose of this project is to give you practice in designing and implementing classes, and using existing templates classes. You will design and implement a class to represent polynomials and perform polynomial multiplication. You will also write a main program that is a client of the class. The main program will read a sequence of polynomial definitions and operations from a file, create polynomials, and perform the specified operations. If you need to brush up on polynomial arithmetic, refer to any elementary algebra textbook.

Polynomial Arithmetic

A ***polynomial*** (in one variable) is a function of the form

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$$

where n is a non-negative integer, $c_n \neq 0$, and the other coefficients c_k may or may not be zero. The degree of $p(x)$ is n. No exponent in a polynomial is allowed to be negative. In this assignment, all coefficients are assumed to be integers. (In this case it is called a polynomial in an integer domain.) Note that if the degree of the polynomial is 0, then the polynomial is just a constant. For the remainder of this specification, I write p as a shorthand for $p(x)$.

Given two polynomials p and q, the following binary operation is defined:

```
p * q   is defined to be the product of polynomials p and q
```

Example

If $p(x) = 2x + 3$ and $q(x) = -x + 1$ then

$$p * q = (2x + 3) * (-x + 1) = -2x^2 - x + 3$$

Project Details

The Class Interface

You are to design the interface to a `Polynomial` class. The class must contain the following member functions.

Name	Description
`eval(double x)`	Evaluate the polynomial using argument `x` and return the value. This should be implemented as an overloaded function call `operator()`.
`operator*`	Given polynomials `p` and `q`, return `p*q`.
`operator<<`	Given an output stream `out` and polynomial `p`, display `p` in symbolic form on `out`.

In addition, the class needs

- a default constructor that creates a zero polynomial,
- a constructor that takes a coefficient `c` and an exponent `e` and constructs a polynomial with the single term cx^e, e.g., `Polynomial p(c,e);`
- a copy constructor that takes a polynomial `q` and makes a new polynomial that is a copy of `q`, e.g. `Polynomial p(q);`
- an `operator=` for the class that will (copy) assign a polynomial to an existing polynomial;
- a destructor, which deletes the polynomial.

Note that this description makes no mention of the private part of the class. That is up to you. You will probably find it necessary to implement other private members of the `Polynomial` class.

Design and Implementation Requirements

A polynomial must be implemented using a list. The *C++* language includes a list template class, and you *must* use this template class instead of writing your own list class. That is my intention in giving you this assignment.

The list provides all of the functions that you will need. I suggest that your list nodes be ***terms***. Each term is completely defined by its coefficient and exponent. No two terms can have the same exponent; i.e., every node must have a unique exponent. It is a good idea to keep the nodes sorted by exponent value. When two polynomials are added, some terms may cancel. For instance, if $p = 2x + 1$ and $q = -2x + 2$, and we let$r = p + q$, then $r = 3$ because the terms $2x$ and $-2x$ canceled. We can end up with a result with fewer (and possibly no) terms. This implies that you need to check when a coefficient of the result is zero, and delete the node. There are other issues that need to be resolved regarding how to perform the arithmetic.

The main program, class implementation, and interface must each be in its own file. For this project there is no need to have more than three source code files.

Input and Output

The main program will create a vector or array of at most 100 polynomials. I will name it `Poly` here. You can name it whatever you like. The main program will then read input from a text file whose name is specified on the command line. If no file is specified or if the file that is specified does not exist or cannot be opened for one reason or another, the main program must display an error message and then exit. Each line of the file will be in one of the following four formats:

1. k : $c_1\ e_1\ c_2\ e_2\ \ldots\ c_n\ e_n$
2. `n = m * k`
3. `eval n(6)`
4. `show n`

The first format is the definition of a new polynomial. The number before the colon, `k`, is the place in the vector (array) where the polynomial should be stored. There will be whitespace separating all tokens on the line, including the colon and the coefficients and exponents. The line will be well-formed, so you do not have to do input validation in this assignment. I.e., there will be a set of pairs of numbers after the colon, each representing a term of the form $c_k x^k$. The terms are not sorted by increasing or decreasing exponents. The array index will always be a number in the range $[0, 99]$. These are both legal definitions:

```
3 : 6  5  -4  3  -2  1  2  0
4 : 1  2   3  4   -9  1
```

which define `Poly[3]` as $6x^5 - 4x^3 - 2x + 2$ and `Poly[4]` as $3x^4 + x^2 - 9x$.

The second format is an instruction to compute a product. It states that `Poly[n]` should contain the product of `Poly[m]` and `Poly[k]`, deleting anything that might have been in Poly[n] previously. You can assume all indices will be valid at the time the line is reached (both operand polynomials exist.)

The `eval` instruction specifies that the polynomial `Poly[n]` is to be evaluated with argument 6 and its value displayed in the standard output stream. The argument can be any floating-point number, such as `3.2`. The output should be something like

```
Poly[n](6) = whatever the value is
```

The last format is an output instruction. The specified polynomial, `Poly[n]`, must be displayed in the output stream in the format below. It does not need to be wrapped onto a new line if it seems long. The stream will do whatever wrapping needs to be done.

```
Poly[n] = a1 x^e1  +  a2 x^e2  +  ... +   am x^em
```

where the terms should be displayed in decreasing order of exponent.

For example, the input could look like

```
0 : 1 2 2 1 1 0
1 : 1 1 1 0
2 = 0 * 1
show 2
0 = 1 * 2
1 = 0 * 2
show 1
```

Testing Your Program

You should design your own input files and test your program using your own input. You should carefully check that the output of your program is correct for the inputs you gave to it. Try it on the simplest of polynomials to be sure. Try it on constants, on polynomials that cancel terms when multiplied, and so on.

Programming Rules

Your program must conform to the programming rules described in the Programming Rules document on the course website. It is to be your own work alone.

Grading

The program will be graded based on the following rubric.

- If the program does not compile on a cslab machine, it receives only 25%.
- For programs that compile:
  - Correctness and implementation 60%
  - Design (modularity and organization) 20%
  - Documentation: 10%
  - Style and proper naming: 10%

Submitting the Assignment

This assignment is due by the end of the day (i.e. 11:59PM, EST) on February 24, 2014. Create a directory named named *username*_hwk1. Put all project-related source-code files into that directory. ***Do not place any executable files or object files into this directory.*** You will lose 1% for each file that does not belong there, and you will lose 2% if you do not name the directory correctly. With all files in your directory, run the command

```
zip -r username_hwk1.zip ./username_hwk1
```

This will compress all of your files into the file named `username_hwk1.zip`.

Before you submit the assignment, make sure that it compiles and runs correctly on one of the `cslab` machines. Do not enhance your program beyond this specification. Do not make it do anything except what is written above.

You are to put your zip file into the directory

```
/data/biocs/b/student.accounts/cs335_sw/projects/project1
```

Give it permission 600 so that only you have access to it. To do this, `cd` to the above directory and run the command

```
chmod 600 username_hwk1.zip
```

where *username_hwk1.zip* is the name of your zip file.

If you put a file there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.