CS256 — Homework 5

October 29, 2015

Due: Thursday, November 5, 2015 before midnight (80 points)

Description

In this assignment we will create a class called `Polynomial` for handling polynomials of a variable x. You should implement this with two files: `Polynomial.h` and `Polynomial.cpp`.

We normally see a polynomial as something of the form

$$9x^4 + 2x^3 - 6x^2 + 41x - 3$$

We could rewrite this as:

$$(-3)x^0 + (41)x^1 + (-6)x^2 + (2)x^3 + (9)x^4$$

The important thing to note is that, when written backwards from how we normally do it and explicitly noting the exponents on the first and second terms, the value of the exponent is the same as the index of an array.

Because of this, we will store our polynomial as a *dynamically allocated* array of `double`s called `coeff`, where the contents of the array are the coefficients of the polynomial. You must use a dynamically allocated array of `double`s for credit. Do not use a class like `vector` or a fixed size array.

Therefore, position 0 of our array stores the coefficient for the x^0 term, position 12 would store the coefficient for the x^{12} term, etc.

You must also have an `int` variable named `size` that tracks the size of the allocated space so you know how many coefficients your polynomial has.

Make sure your variables are named `coeff` and `size` so that the test driver class works properly!

You must implement the following functions:

1. A default constructor that allocates space for 1 and assigns coefficient 0
2. A constructor that takes an array of doubles and a size and copies their contents to our object
3. A copy constructor
4. A constructor that takes an `int` to allow converting from `int` to `Polynomial`. It should make space for 1 coefficient and store the passed `int` as the value for that coefficient.
5. A constructor that takes a `double` to allow converting from `double` to `Polynomial`. It should make space for 1 coefficient and store the passed `double` as the value for that coefficient.
6. Overload the assignment operator for copying
7. A destructor that properly deallocates the memory we allocated for our object

8. A function `int getSize() const;` that returns the value of the `size` variable
9. A function `int degree() const;` that returns the degree of the polynomial
10. A function `std::string str() const;` that returns a `std::string` representation of the polynomial. This should display the polynomial in the way we would expect to see it written, from highest exponent to lowest. For example, the polynomial above should be displayed as:

    ```
    9x^4 + 2x^3 - 6x^2 + 41x - 3
    ```

    The test driver relies on a properly function `str()` function, so be very careful. There are a lot of little tricks to make sure it is displayed properly (e.g., not showing coefficients of 1, not showing exponent for 1 or 0 term, properly using minus and plus signs, not displaying terms with coefficient of 0)
11. A function `double solve(double x) const;` that solves the polynomial for the value `x` passed. For example, calling `solve(3)` on our polynomial above should return 849 because $9(3)^4 + 2(3)^3 - 6(3)^2 + 41(3) - 3 = 849$
12. Overload `operator[]` to allow access to the coefficients in the polynomial based on the exponent (index) provided.

    If the user provides an index value above your allocated space, your operator should allocate new space for your polynomial, move the contents to the new space, fill the newly allocated space with 0s, and then return the value for the index requested. This will allow a user to increase the size of a `Polynomial` by trying to assign to a higher exponent value than previously allocated without needing to expand it manually.

    Example:

    ```
    double coeff[5] = {-3, 41, -6, 2, 9}; // above example
    Polynomial p(coeff, 5); // p.coeff is now {-3, 41, -6, 2, 9}
    p[0] = 12; // p.coeff is now {12, 41, -6, 2, 9}
    p[8] = 5; // p.coeff is now {12, 41, -6, 2, 9, 0, 0, 0, 5}
    ```
13. Overload the following arithmetic operators for polynomials: `+`, `-`, `*`
14. Overload the `*` operator allowing you to multiply your polynomial by a `double`
15. Outside of the `Polynomial class`, overload the `<<` operator for `ostream` to work for `Polynomial`s
16. Overload the combined assignment operators `+=`, `-=`, and `*=`
17. Overload `==` and `!=`, where two `Polynomial`s are considered equal if they have the same degree and all coefficients match

I will provide a test driver program `main.cpp` that you can use to test your class. This is just a sample of tests. When grading, I may use a different test driver. Make sure you test your class on more possible cases than what are provided.

In order to use the test driver class, we will have to make it a friend of `Polynomial` because it checks internal state during the tests. You can do this by starting your `Polynomial` class declaration in `Polynomial.h` as follows:

```
// needed because we define the class elsewhere
class PolyTester;

class Polynomial
{
    friend class PolyTester; // now test class can access private members
private:
    ...
public:
    ...
};
```

Include comments at the beginning of your source code file that contain your name, the homework assignment number, and the date that you completed the assignment. For example, my submission's comments might look like this:

```
// Nick Pantic
// Homework 5
// Completed 10/29/2015
```

Submission

Create a file in your local project directory called `Makefile` and paste the following in to it:

```
all: Polynomial.cpp main.cpp Polynomial.h; g++ -o Polynomial Polynomial.cpp main.cpp
clean: Polynomial; rm Polynomial
```

1. Create a project on https://codebank.xyz named `bronconame-hw5`. Follow this naming convention precisely including case (all lowercase).
2. On the project page, click *Settings* to the left
3. From the *Settings* page, click *Members* to the left
4. On the Members page, click *Add Members*.
5. Add me (and only me) as a member. My user name is nmpantic.
6. For project access, choose *Developer*
7. Locally, use `git` to add a reference to the remote repository with the `git remote add` command.
8. After committing your code locally, use `git push` to push the code to `codebank.xyz`. You can make as many commits as you want and continue to commit changes up until the deadline.

For this and future assignments, submissions will not be accepted if they are not submitted via `git`.