CS256 — Homework 4

February 10, 2016

Due: Monday, February 22, 2015 before midnight (100 points)

Like homework 3, you may work with a partner for this assignment. Make sure to place comments at the top of the source code for each file listing both names and add the second person as a member on your project on `codebank.xyz`.

Preparing the Project

1. Go to https://codebank.xyz and you should see a project in the CS256 group named `CS256-HW4`. Fork this project for your own user.
2. Next, clone the repository so you have a local copy:

   ```
   $ git clone https://codebank.xyz/username/CS256-HW4.git
   ```

3. The repository should have three files: `Polynomial.cpp`, `Polynomial.h`, and `main.cpp`.

Description

In this assignment we will create a class called `Polynomial` for handling polynomials of a variable x.

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. You will not receive credit if you 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 4 would store the coefficient for the x^4 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.

You must implement the following functions:

1. A default constructor that allocates space for 1 and assigns value 0 to coefficient 0
2. A constructor that takes an array of doubles and a size and copies their contents to the object being constructed
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 working `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). If future tests fail, it might just be because your `str` function does not handle some special case.
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. Overload the combined assignment operators `+=`, `-=`, and `*=`
16. Overload `==` and `!=`, where two `Polynomials` are considered equal if they have the same degree and all coefficients match
17. Outside of the `Polynomial class`, overload the `<<` operator for `ostream` to work for `Polynomials`

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 and Second Person
// Homework 4
// Completed 2/22/2016
```

Submission

You can use `git add`, `git commit`, and `git push` to push your code to https://codebank.xyz. You can make as many commits and push as many times as desired until the deadline.