Introduction to Programming, PIC10A
E. Ryu
Spring 2017

Homework 9
Due 5pm, Friday, June 9, 2017

Download the starter code main.cpp, polynomial.h, polynomial.cpp, and CImg.h. You will have to modify the header file but you may not change the signatures of the provided public methods of class polynomial. (You are free to add any public and/or private members.) We have provided main.cpp to give you an idea of how we intend to use the functions. Put the implementations into the files polynomial.h and/or polynomial.cpp. All files you submit must not contain a main function.

You may not use global variables. You may not use the using-directive, using namespace std;. You may not use #pragma once. You may not use any libraries aside from cassert, cmath, iostream, algorithm, vector, string, and CImg.h. We may take off up to 20% of the total marks for poor style; make sure to name your variables reasonably, indent properly, and comment sufficiently. Submit polynomial.h and polynomial.cpp.

In hw7 and hw8, we've forced you to put all function definitions into cpp files. However, there's nothing wrong with putting very short definitions into header files and putting only the long definitions into cpp files.

Problem 1: (Polynomial)

Write a class that represents a polynomial.

The constructor

```
polynomial(double c = 0.0);
```

creates a polynomial that corresponds to $p(x) = c$.

The method

```
int degree();
```

returns the degree of the polynomial. (For the purpose of this assignment, let's say the zero polynomial $p(x) = 0$ is degree 0.)

The method

```
int nonzeroTerms();
```

returns the number of nonzero terms of the polynomial. For example, $x^4 + 1$ has 2 nonzero terms.

The procedure

```
void setCoeff(int deg, double c);
```

sets the coefficient of the term x^{deg} to c. Calling this function can increase or decrease the degree of the polynomial. assert that deg is nonnegative.

The method

```
double getCoeff(int deg);
```

returns the coefficient of the term x^{deg}. The return value can be `0.0`. In fact, it `deg` is larger than the degree of the polynomial, the return value must be `0.0`. `assert` that `deg` is nonnegative.

The method

```
double operator()(double x);
```

evaluates the polynomial at `x`.

The arithmetic operators

```
polynomial operator+(polynomial p);
polynomial operator-();
polynomial operator-(polynomial p);
polynomial operator*(polynomial p);
polynomial& operator+=(polynomial p);
polynomial& operator-=(polynomial p);
polynomial& operator*=(polynomial p);
```

perform arithmetic operations between polynomials.

The declaration and definitions of the non-member functions

```
std::ostream& operator<<(std::ostream& s, polynomial p);
void plot(polynomial p);
```

are provided. Their usage in `main.cpp` should illustrate their purposes.

The non-member operator overloads

```
polynomial operator+(double c, polynomial p);
polynomial operator*(double c, polynomial p);
```

performs arithmetic between a scalar and a polynomial.

Hint. Have a `private` object of class `vector<double>` that represents the coefficients of the polynomial.

Hint. Implementing `setCoeff` and `operator*` are probably the trickiest parts of this homework, although they both can be done with 10 lines of code or less. Remember that calling `setCoeff` can increase the polynomial degree if `deg` is larger than the current degree and can decrease the polynomial degree if `c` is equal to `0.0`. In implementing `operator*`, you may find the following formula useful: when $p(x) = \sum_{i=0}^{m} a_i x^i$ and $q(x) = \sum_{i=0}^{n} b_i x^i$ we have

$$p(x)q(x) = \sum_{i=0}^{m+n} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) x^i.$$

Problem 2: (Code management)

Put the class and function declarations within the `namespace pic10a`. Also, protect the header file `polynomial.h` with `#include` guards. You may not use `#pragma once` for this.

Remark. The library CImg.h relies on certain graphics libraries. Visual Studio users need not worry about this, since things should just work. MacOS or Linux users may have to set up a few things to get CImg.h to work. One option is to simply forget about it; simply remove the plot function and the #include "CImg.h". The plot function, which plots polynomials, is provided to you just for fun, and you can get full credit on this assignment without using it. However, if you're a MacOS or Linux user and you do want to get the plotting to work, here's how.

First check whether X11 is installed on your system. Do so by typing whereis X11 on the command line. If not installed, MacOS users can install X11 from https://www.xquartz.org/.

Then you need to provide the compiler with options that look like

```
g++ -o hw9exec main.cpp polynomial.cpp -I A -L B -l X11 -l pthread
```

A is a directory like /opt/X11/include and B is a directory like /opt/X11/lib. The precise directory of A and B will differ from system to system. Try to get it right by looking at the output of whereis X11.

The -I /opt/X11/include option tells the compiler to look in /opt/X11/include in searching for header files, in particular for X11/Xlib.h.

The -L /opt/X11/lib option tells the compiler to look in /opt/X11/lib to find the cpp files implementing the declarations of X11/Xlib.h, roughly speaking.

Using the -l X11 and -l pthread options is like providing X11.cpp and pthread.cpp to the compiler. The compiler will look in certain default directories and B to locate these library implementations.

In MacOS, you will have to open XQuartz (and keep it open) and then run your compiled cpp program for it to use X11.