EE 205 — Midterm

Spring 2012

Name: ____________________________

login id: __________________________

March 6, 2012

This is a open book, open notes, open computer, closed neighbor exam. You will be allowed access to the p eece e during the entire exam; however, you may only use a web browser to access the EE 205 course web site (no Google, no chat rooms, no email), and you may use SSH to login to wiliki to edit and compile files (no chats, no email etc.).

This exam consists of 8 numbered pages, including this cover sheet.
1. **Object Oriented Programming** (Short answer)
   (NOTE: since this is an open book/website exam, I expect more than quotes from the website here. I would like to see YOUR understanding of these concepts).

   (a) (5 Points). **Object**

   (b) (5 Points). **Layering** (two kinds)
(c) (10 Points). **Better C** List at least 5 features of C++ that make it a “better C” and explain why programming is “better” as a result.
(d) (10 Points). While reviewing a colleague’s code (a colleague who likes to write obscure code), you saw the following line:

    operator<<(cout, operator=(y, operator+(operator*(operator-(m), x), b)));

Rewrite this line of code in a more readable/understandable form.
2. **Iteration vs Recursion**

In your (previous) favorite class, EE 160, (some of) you wrote (and used in several applications) a function, `pos_power()`, which was given a float base, and a positive integer exponent and raised the base to the exponent power. Here is the prototype for the function:

```c
double pos_power(float base, int expon);
```

In that course, you used iteration (a loop) to compute the result and probably had code similar to that below.

(a) (10 Points). We would like to make the least number of changes to this code to be able to handle both positive and negative integer exponents, to have a new function called `power()`. Show those changes in the code below.

```c
double power(float base, int expon)
{
    double result = 1;
    if(expon <= 0)
        return result;
    for( ; expon > 0; expon--)
        result *= base;
    return result;
}
```
(b) (10 Points). Now in your (current) favorite class, we have discussed another control construct for repetitive computations: recursion. One approach to this algorithm is to use multiplication and decreasing exponent for positive exponents, and division and increasing exponent for negative exponents:

\[ x^n = \begin{cases} 
    1 & \text{if } n \text{ is 0} \\
    x^{n-1} \cdot x & \text{if } n \text{ is positive} \\
    x^{n+1} / x & \text{otherwise}
\end{cases} \]

Write the code for a recursive version of power().
3. Polynomials

We would like to build an efficient class which can represent arbitrary degree polynomials in one variable. (We want to compete with MatLab). For the purposes of this exam, we will only implement a subset of the operations that could be done on these types of objects (for now).

For example, we should be able to create and display polynomials (of type Poly) like:

\[ A(x) = 5 + 6x^3 + 2x^5 + x^7 \]

\[ B(x) = x^3 - 2x^5 + 13x^7 - 2x^8 + 26x^9 \]

and do simple operations on them like evaluate a Poly for a given \( x \) and add two polynomials:

\[ C(x) = A(x) + B(x) = 5 + 7x^3 + 14x^7 - 2x^8 + 26x^9 \]

Our container class seems like a perfect vessel to build the Poly. We will represent each term of the polynomial in a Term class which include the coefficient and exponent of the term. Poly has a container of Terms. (Hint: we can further store the Term with exponent \( n \) in position \( n \) in the container).

I have provided you with a main() to test the set of operations you should implement in the file polytest.cc. To save you some typing, I have also provided the partial class declarations in poly.h and term.h and skeleton implementations in poly.cc and term.cc. You may add anything to the private part of the either class (additional data fields or helper functions), but should not change the public interface. You should also inline any functions you think are appropriate.

You can download a tar file containing these files, and untar it using the following commands on wiliki (first change to a directory where you will create your code):

```
wget http://ee.hawaii.edu/~tep/EE205/S12/Exams/poly.tar
tar xvf poly.tar
```

I have also provided a makefile and my (partial) implementation of the Container class which should have enough functionality for this application.
Eventually, the program should compile and run with the following commands:

```shell
make
polytest
```

This code should compile and run (incorrectly, but do something) as is. When you are done, you should turn in ALL files needed to compile and run polytest using the following commands on wiliki:

```shell
make real_clean
grade -mtm,ee205 *.cc *.h makefile
```

(a) (20 Points). A couple of things you might have noticed missing in the Container class are the Copy Constructor and Assignment operator. Add these to the class.

(b) (30 Points). Finally, your task is to complete the implementation of the Poly and Term classes and get polytest to run correctly. When done, submit your files as requested above. I recommend you build this implementation incrementally; adding, compiling, and testing one feature at a time, and never be very far from a compiling program.