Homework #1

TA email: dsata at csie dot ntu dot edu dot tw

RELEASE DATE: 02/21/2012
DUE DATE: 03/07/2012 (**Thursday**), noon

As directed below, you need to submit your code to the designated place on the course website.

Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

Both English and Traditional Chinese are allowed for writing any part of your homework (if the compiler recognizes Traditional Chinese, of course). We do not accept any other languages. As for coding, ***we only allow C++ this time****. In particular, the use of* `printf`, `scanf` *and similar functions is not allowed.*

This homework set comes with 200 points. In general, every homework set of ours would come with a full credit of 200 points.

Getting Familiar with C++ Language

The purpose of this homework is to give you a quick start with the C++ language. In the language, the variables can be bundled in a *class*, which is like a C *structure* but (loosely speaking) allows you to define functions inside for manipulating the variables directly. We have shown how to define functions inside classes, and how to expose the functions to external use by the `public` keyword. In your reading assignments this week, you should learn how to construct, or initialize, an instance of the class by a special function called the *constructor*. Yes, we know that you are not totally familiar with them and the *purpose* is to force you to reach out for resources and learn. So here comes the valuable hint from your instructor: *It would be a really good time to start reading and going to your TAs!*

Another type of special function within a class is called the operator, and we have demonstrated its use as well. The operator function specifies what the operation when applying its symbol on instances of a class. This gives the operator some additional meaning, or *overloads* it. You can redefine the function of most built-in operators globally or on a class-by-class basis. Overloaded operators are implemented as functions.

In this homework set, you are asked to implement a constructor, two functions within the class, as well as operator overloading. The class you will be working on defines one-variable polynomials with complex coefficients, and you will deal with arithmetic operators, including addition, subtraction, multiplication and composition. For example, you need to overload the built-in arithmetic addition operator(+) to perform polynomial addition. You are also asked to separate definitions from implementation. In particular, you should put your definition of the class in a header file like

```
//poly.h
class poly{
  void blahblah();
};
```

and put the implementation of the `blahblah` function in the cpp file like

```
//poly.cpp
void poly::blahblah(){
  //...
}
```

Requirement

Please implement a C++ class `poly` for polynomials, with the following items:

(1) (10%) The class header file, which should include the definition for things below.

(2) (20%) The class constructor, which initializes the polynomial reasonably.

(3) (25%) The method `eval` for evaluating the polynomial with respect to an input value that is possibly complex

(4) (25%) The method `show` for printing out the polynomial, which should follow the format below:

- The coefficients are output in ascending power (i.e. starting from $x^0, x^1, \dots$)
- The output should end at the highest-degree term with nonzero coefficient. For zero polynomial, simply output "`0.00`".
- Each coefficient (which is a complex number) should be output in the format of either "$\pm$`a`", "$\pm$`bi`", or "$\pm$`a`$\pm$`bi`", where `a` and `b` are floating numbers with **exactly** two digits after the decimal point. The unary positive sign is always omitted.
- Please show the coefficients even when it is ± 1.00.
- Do not use parentheses in the coefficient.
- Examples of valid output are "`-3.14`", "`0.00`", "`2.72i`", "`1.00+1.00i`", "`3.00-4.00i`" ....

(5) (30%) The operator for polynomial addition (+)
$(ix^3 + x^2 + 1) + (x^4 + (3+i)x^2 - 2) = (x^4 + ix^3 + (4+i)x^2 - 1)$

(6) (30%) The operator for polynomial subtraction (−)
$(ix^3 + x^2 + 1) - (x^4 + (3+i)x^2 - 2) = (-x^4 + ix^3 - (2+i)x^2 + 3)$

(7) (30%) The operator for polynomial multiplication (∗)
$(x^2 + 1) * (x^4 + (3+i)x^2 - 2) = (x^6 + (4+i)x^4 + (1+i)x^2 - 2)$

(8) (30%) The operator for polynomial composition ('()')
$f(x) = ix^2, g(x) = (x+1), f(g(x)) = ix^2 + 2ix + i$

In this homework, you need to hand in a header file, `poly.h`, and a C++ program file, `poly.cpp`. You need to declare the class in `poly.h`, and implement them in `poly.cpp`. The files, when coupled with `main.cpp` provided by the TAs, can read from the sample input and produce the sample output. The TAs will use different input/output files to test your program, of course.

Please note that "complex number" is by itself something that can be implemented with a non-primitive class. You are hinted and encouraged to write a class like `ComplexNumber` with possibly operator overloading to facilitate your design of the polynomial class, but it is not required.

Input Format

The first line is the number of testing inputs. Each testing input contains three lines, the first two specify the polynomial and the third line corresponds to the operator. In the first two lines, the first number is the degree of that polynomial and the rest are the coefficients. The coefficients are in **ascending power, following the format described in the "`show`" function**; Also, the leading coefficient is non-zero, and there will not be white spaces between the complex coefficients of an exponent. For example, "`-2.00 0.00 3.00+1.00i 0.00 1.00`" corresponds to $(x^4 + (3+i)x^2 - 2)$. Finally, the third line contain an operation (one of `+`, `-`, `*`, `()`). If the operation is `e`, it means you need to evaluate the first polynomial on the second polynomial (which would be a complex number). Note that the degree of input and output polynomial will not exceed $100,000$.

Output Format

The `show` function of the class will be used to print the resulting polynomial from {(first polynomial) (operator) (second polynomial)}. The polynomial coefficients should be printed in **ascending power using at most two digits after decimal point, in one line and separated by one space.** Note that the leading coefficient can not be zero, unless all coefficients are zeros. The TAs can teach you how to print using exactly two digits after decimal point with `cout` if you go ask them, but you can surely also figure it out by yourself with Google.

Sample Input

```
5
3 1.00 0.00 1.00 1.00i
4 -2.00 0.00 3.00+1.00i 0.00 1.00
+
3 1.00 0.00 1.00 1.00i
4 -2.00 0.00 3.00+1.00i 0.00 1.00
-
2 1.00 0.00 1.00
4 -2.00 0.00 3.00+1.00i 0.00 1.00
*
2 0.00 0.00 1.00i
1 1.00 1.00
()
3 1.00 2.00 3.00 4.00+4.00i
0 -2.00+7.00i
e
```

Sample Output

```
-1.00 0.00 4.00+1.00i 1.00i 1.00
3.00 0.00 -2.00-1.00i 1.00i -1.00
-2.00 0.00 1.00+1.00i 0.00 4.00+1.00i 0.00 1.00
1.00i 2.00i 1.00i
2042.00+38.00i
```

Related Files

(1) *main.cpp*: The TAs will test your program in this way, so you need to declare the constructor/function/operators corresponding to the usage in main.cpp.

(2) *Makefile*: The TAs will compile and test your program by the command "`make`" and "`make run`". **Please make sure that your code can be compiled and run with the Makefile on CSIE R217 linux machines. Otherwise your program "fails" its most basic test and can result in ZERO!**

Submission File

Please upload your program as a single ZIP compressed file to CEIBA before noon, 03/07/2013 (Thursday). The zip file should be like `b86506054.zip`, where the file name should be changed to your own school ID. The ZIP file should contain the following items:

- `poly.cpp` and `poly.h`
- an optional text file `README`, anything you want the TAs to read before grading your code
- an optional text file `MEDAL`, in case you want to use the gold medals, with the number of medals listed as a single number in the first line of the file. Use your medals wisely—usage cannot be retracted.