CSCI124/MCS9124 Assignment 4

(Total 8 marks, Due by 11:59 pm sharp on Sunday, 24 May, 2015)

Aims

This assignment aims to establish a basic familiarity with C++ classes. The assignment introduces increasingly object-based, C++ style of solution to a problem.

General Requirements

- Observe the common principles of OO programming when you design your classes.
- Put your name, student number at the beginning of each source file.
- Make proper documentation and implementation comments in your codes where they are necessary.
- Logical structures and statements are properly used for specific purposes.

Objectives

On completion of these tasks you should be able to:

- Code and run C++ programs using the development environment.
- Make effective use of the on-line documentation system that supports the development environment.
- Code programs using C++ in a hybrid style (procedural code using instances of simple classes) and in a more object-based style.
- Pointers and dynamic arrays.

Problem Specification

A one-variable *polynomial* is an arithmetic expression of the form:

a_k x^k + ... + a_2 x^2 + a_1 x^1 + a_0 x^0

The highest exponent, *k*, is called the *degree* of the polynomial, and the constants *a_0, a_1, ... a_k* are the *coefficients*.
For example, here is a polynomial with degree three:

3.2 x^3 + 5.1 x^2 – 1.5 x + 7.2

Each individual *term* of a polynomial consists of a real number as a *coefficient* (such as 3.2), the variable *x,* and a *non-negative* integer as an *exponent*.
The term x^1 is usually written as an *x* rather than *x^1* and the term *x^0* is usually written with just the coefficient.

For this assignment, you should specify, design and implement a class for polynomials.
A polynomial is implemented with a dynamic array.
The index of the array represents the exponent of the variable, and the content of that array element represents the coefficient itself.
For example, for the above polynomial, the array will be of size 4, and the content of the array are as follows:

Degree:	0	1	2	3
	7.2	-1.5	5.1	3.2

Implementation

Use the following *guideline* to implement the solution to this assignment.

Step 1. Define a class Polynomial and implement simple member functions. (3.5 marks)

Define the class **Polynomial** in a file **polynomial.h** and implement the member functions in a file **polynomial.cpp**

The class consists of two data members: one double pointer can be used to allocate dynamic array to store coefficients of a polynomial; one integer is the degree of the Polynomial.

The class Polynomial consist of **constructors** and a **destructor**. The constructors include default constructor that set default values for the Polynomial. An initialization constructor that can take a (double type) array (coefficients) and the size (degree) of the array as parameters, initialize the current Polynomial instance by the given coefficients' array and degree. Copy constructor can make a deep copy of another Polynomial instance. The destructor will release the dynamic memory that has been allocated for the coefficients of a polynomial.

An **input member function** of the Polynomial class gets input degree and coefficients values for the polynomial. For example (red data means input date):

```
Degree: 4

Coefficients: 1 0 3.7 1.0 2.5
```

Then the polynomial x^4 + 0 + 3.7x^2 + x + 2.5 has been stored in the data members.

A **print member function** of the Polynomial class prints out the Polynomial in the form of

a_k x^k + ... + a_2 x^2 + a_1 x^1 + a_0 x^0

If a term's coefficient is zero, then such term should not be printed. If a term's coefficient is 1 (one) or 1.0 (one point zero), then the coefficient should not be printed unless the degree of the term is zero. If a coefficient is a negative value, the print function should only print – (not +-) for this term. For example, an array of a polynomial's coefficients contains

2.5	1.0	-3.7	0	1

The print function should print out the polynomial like following

```
x^4 - 3.7x^2 + x + 2.5
```

Step 2. main() function. (1 mark)

Implement the testing code in a file **main.cpp.** You can initialize polynomials by the default constructor, the initialization constructor, input member function, and the copy constructor. Then print out the polynomials. The output of polynomials may look like:

```
Default constructor p0:
Initialization constructor p1: 9.2x^3 + x^2 + 5.1x
Degree: 4
Coefficients: 4.5  0.0  5.1  -2.2   1.5
The input polynomial p2: 4.5x^4 + 5.1x^2 - 2.2x + 1.5
Copy constructor p3: 4.5x^4 + 5.1x^2 - 2.2x + 1.5
```

Step 3. Define and implement assignment operator (operator=). (1 mark)

Define the assignment operator in the class **Polynomial** and implement the overloading operator for the class in the file **polynomial.cpp**. The overloading operator makes a deep copy of the other Polynomial instance. The implementation of the assignment operator is similar with copy constructor. Since the object already exists, you should delete the old memory if the current object has the dynamic memory allocated when it is not the same (address) as the source object, then allocate new dynamic memory just the same size as the source object, copy the coefficients from the source array into the current array. To make it more efficient, if the size of the current coefficient array is bigger than the source polynomial size, you don't have to delete the old memory and allocate the new dynamic memory for the coefficients. You may copy the coefficients from the source polynomial, update the rest of term to zeros.

You can test the assignment operator to make it work.

Step 3. Adding and Subtracting. (2 marks)

Define and implement member functions to do the polynomials adding and subtracting. Both member functions return Polynomial objects of adding or subtracting results. Two polynomials adding and subtracting following the same rule: Add (or subtract) the like terms in the two polynomials. The like terms in here means they have the same exponents.
For example:
Polynomial p1=15.3x^5 + 11.2 x^3 + 8.6 x^2 - 7.1x + 1
Polynomial p2=5.2x^4 + 4.3x^3 + 2.5 x^2 - 2.7
The addition of polynomials p1 and p2 will be:
15.3x^5 + 5.2x^4 + 15.5x^3+11.1x^2 - 7.1x - 1.7

Note: The adding and subtracting functions do not change the operands' values. E.g. p1 and p2's values remain the same when the functions return.

Step 4: makefile (0.5 marks)

Define a **makefile** for the assignment to make your assignment 4 program. The program name must be **poly**. It should run by using the command make to make **poly**.

Testing

Your testing code in the file **main.cpp** should be able to get input of polynomials as and test all the member functions and assignment operator that defined for this assignment. The testing results will look like following (red values means the input data):

```
Default constructor p0:
Initialization constructor p1: 9.2x^3 + x^2 + 5.1x
Degree: 4
Coefficients: 4.5  0.0  5.1  -2.2   1.5
The input polynomial p2: 4.5x^4 + 5.1x^2 - 2.2x + 1.5
Copy constructor p3: 4.5x^4 + 5.1x^2 - 2.2x + 1.5
```

Degree: 5
Coefficients: -1.7 2.4   -3.6 -10.2 0 -8.3
The input polynomial p4: -1.7x^5 + 2.4x^4 - 3.6x^3 - 10.2x^2 - 8.3
p5 = addition of p3 and p4: -1.7x^5 + 6.9x^4 - 3.6x^3 - 5.1x^2 - 2.2x - 6.8
p2=subtraction of p1 and p4: 1.7x^5 - 2.4x^4 + 12.8x^3 + 11.2x^2 + 5.1x + 8.3

Use command **becheck** on banshee to verify your program if there is any memory leak exists.
To use becheck, you should copy the source code (include makefile) into a working directory on your banshee account. Use the command make to make the program **poly**.
Run the bechck like:
bcheck ./poly
You should input polynomials like the Testing above. After the program finished, a report for actual memory leaks and possible memory leaks will be provided. Your program should have both actual memory leak and possible memory leak equal to zeros.

Submission

This assignment is due by 11.59 pm (sharp) on Sunday 24 May, 2015.

Assignments are submitted electronically via the ***submit*** system.

For this assignment you must submit the files via the command:

```
$ submit -u your_user_name -c CSCI124 -a 4 main.cpp polynomial.h polynomial.cpp
makefile
```

and input your password.

Make sure that you use the correct file names. The Unix system is case sensitive. You must submit all files in one ***submit*** command line.

Remember the submit command scripts in the file should be in one line.

Your program code must be in a good programming style, such as good names for variables, methods, classes, and keep indentation.

Submission via e-mail is NOT acceptable.

After submit your assignment successfully, please check your email of confirmation. **You would loss 50%~100% of the marks if your program codes could not be compiled correctly.**

Late submissions do not have to be requested. Late submissions will be allowed for a few days after close of scheduled submission (up to 3 days). Late submissions attract a mark penalty (25% off for each day late); this penalty may be waived if an appropriate request for Academic Consideration (for medical or similar problem) is made via the university SOLS system *before* the close of the late submission time. No work can be submitted after the late submission time.

A policy regarding late submissions is included in the course outline.

The assignment is an **individual assignment** and it is expected that all its tasks will be solved **individually without any cooperation** with the other students. If you have any doubts, questions, etc. please consult your lecturer or tutor during tutorial classes or office hours. Plagiarism will result in a **<u>FAIL</u>** grade being recorded for that assessment task.

End of specification