**Homework 3**

**ECE 309 Fall 2019**

**Due: September 18, 2019**

Upload an electronic copy of your answers to Moodle under HW3.

*This is a shared google document. This means (1) it may change to clarify content, and (2) other people can view your comments on this file. If you have questions, you are encouraged to comment directly on this document, but* ***do not add your answers here****. Make a copy into your private Google Drive and then edit the document.*

*DO NOT ADD ANSWERS TO THE SHARED DOC! THAT’S CONSIDERED CHEATING!*

# 1. Constructors, Destructors, Copy Constructors

[30 points] When creating classes, we specify how to construct, destruct, and copy them. In some cases, these functions may do very little, as in class T below:

class T {

public:

T() {}

~T() {}

T(const T& copy) {}

};

However, what happens if we make the constructor, destructor, or copy constructor private? What influence does that have when we try to create, destroy, or copy the object. Create three different versions of class T, as shown above:

1. Private constructor, other functions public
2. Private destructor, other functions public
3. Private copy constructor, other functions public

For example, version 1 would look like this:

class T {

private:

T() {}

public:

~T() {}

T(const T& copy) {}

};

Now consider each row below in the table and explain what happens when you try to compile that code for versions 1, 2, and 3 of class T above. Does the compilation succeed or fail? If it fails, explain why.

Fill out the table with your observations. In all cases, I’ll refer to the class as T, even though the code should be different for each version.

Suggestions:

* Do not mix code snippets in the same file. It will become difficult to isolate their effects.
* Use -Wall but do not use -Werror, as some of these codes may prevent compilation.
* Make sure you include <cstddef> to obtain the definition of nullptr.
* On some systems, you may also need to pass -std=c++11 to ensure that nullptr is defined.

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Version 1 | Version 2 | Version 3 |
| Declare a local variable of that type  T tmp; |  |  |  |
| Allocate a heap object:  new T; |  |  |  |
| Pass an object to a function:  void f(T a) {  }  f(T()); |  |  |  |
| Declare a pointer and set it to nullptr.  T \* t = nullptr; |  |  |  |
| Initialize one object with another of type T:  T t1;  T t2 = t1; |  |  |  |

# 2. Copying Objects

[20 points] Consider the following object:

class Z {

public:

int size;

int \*n;

Z(int a=10) { n = new int[a]; size = a;}

};

1. [5 points] What happens when this class is copied with the default copy constructor? Draw a picture to illustrate what you mean.
2. [10 points] Implement a copy constructor for Z that makes a deep copy of the object.
3. [5 points] Is there another member function we should also implement with the copy constructor to ensure the object is used properly? If so, implement it as well.

# 3. Modify the linked list code from lecture

* (15 points) zyLab 11.24: Implement a copy constructor for the List class from Lecture05.
* (35 points) zyLab 11.25: Modify the List class to use a dummy node style implementation (for more details read ZyBook Ch. 11.10). All of the functionality of the List class must be preserved.
  + 10 points for proper implementation using dummy nodes.
  + 25 points for passing all tests. Only earned if there is some implementation.
  + -2 points: If your code prints Warning messages, you lose points.