CS368 Programming Assignment 3

Section 1, Fall 2015

Due by 11:59 pm on Friday, November 20

In this page: Announcements | Overview | Specifications | Handing in Related pages: Assignments

Announcements

Check here periodically.

11/9/2015

- Program released. To ask questions about the homework and see questions posed by other students and their answers, go to: http://www.piazza.com/wisc/fall2015/cs3681 and sign-in using your wisc.edu account.
- Note completing this assignment is critical to developing a good working understanding of memory managment in C++ classes. See the `valgrind` tutorial to get started using `valgrind`.

Overview

Why are we doing this program?

Description

In programming assignment 2 the issues of proper memory management for the `Student` and `SortedList` classes were ignored. The compiler defaults for the destructors, copy constructors, and copy assignment operators were assumed to be good enough. For this assignment you'll write your own to ensure that memory dynamically allocated in the classes is correctly managed. You'll also write a main function that demonstrates the usefulness of the added functions.

Goals

The goals of this assignment are to give you experience:

- writing a non-trivial destructor, copy constructor, and copy assignment operator.
- writing and using private auxiliary functions.
- using `valgrind`.

Specifications

What are the program requirements?

Updating the `Student` Class

Add the following to the `Student` class you wrote for Programming Assignment 2:

- a destructor,
- a copy constructor, and
- a copy assignment operator (`operator=` that defines the assignment operator for two `Student` objects)

Make sure to modify both `Student.h` and `Student.cpp`.

Updating the `SortedList` class

Add the following to the `SortedList` class you wrote for Programming Assignment 2:

- a destructor,
- a copy constructor, and
- a copy assignment operator (`operator=` that defines the assignment operator for two `SortedList` objects)

Make sure to modify both `SortedList.h` and `SortedList.cpp`.

`SortedList` Class Auxiliary Functions

Both the destructor and copy assignment operator of the `SortedList` class must deallocate the nodes of a linked list returning them to the free storage (i.e., the heap). Both the copy constructor and copy assignment operator make a deep copy of an existing linked list. Therefore, it is a good idea to write two auxiliary functions to handle these two tasks.

```
private:
    static void      freeList (ListNode *L);
    static ListNode *copyList (ListNode *L);
```

Implement an auxilliary function, named `freeList`, having one parameter, a pointer to a `ListNode`. This function traverses the linked list, returning each node to free storage (i.e., heap). It is used by both the destructor and copy assignment operator.

Also implement an auxilliary function, named `copyList`, having one parameter, a pointer to a `ListNode`. This function returns a pointer to the first node of a copy of original linked list. It is used by both the copy constructor and the copy assignment operator.

Note both auxilliary functions are implemented as `static` functions of the `SortedList` class and should be used as such. They are also `private` intended to only be used within the class. Also note in `SortedList.cpp` the the modifier "`static`" is *not* included.

Demonstrating the Difference

Write a `main` function, in the file `main.cpp` that tests your new version of the `Student` and `SortedList` classes. This function should fully exercise the destructors, copy constructors, and copy assignment operators of each class.

Next, create two executables: one using the class implementations from programming assignment 2 and your `main.cpp` file and another using your updated classes and your `main.cpp` file. When `valgrind` is run on the first executable, it should report storage leaks, but when `valgrind` is run on the second executable, there should be no storage leaks.

Finally, compare the differences between the `Student` class and the `SortedList` class. Determine for which class(es) the updates mattered. **Include your conclusions in the header comments of the `main.cpp` file.**

Handing in

What should be handed in?

Make sure your code follows the style and commenting standards used in CS 302 and CS 367. Note: the

commenting standards use javadoc comments for class, method, and constructor headers. You do not need to use javadoc comments in your programs for CS 368; however, your comments should include the same information as the javadoc comments. For example, your function header comments should include a description of what the function does, the name and a short description of each parameter, and a description of the return value.

Electronically submit the following files to your **`In`** "handin" directory by the due date and time (or refer to the late policy):

- **`main.cpp`** containing your `main` function (including, in the header comments, your conclusions from the Demonstrating the Difference section),
- **`Student.h`** containing your updated header file for the `Student` class,
- **`Student.cpp`** containing your updated source code for the `Student` class,
- **`SortedList.h`** containing your updated header file for the `SortedList` class, and
- **`SortedList.cpp`** containing your updated source code for the `SortedList` class.

- Pair Programming students must also submit a `README` file: All students working in pairs must read the collaboration policy and submit a `README` file. Each student working in a pair must hand-in this file to his/her own hand-in directory. Copy, complete, and hand-in the file that is linked here:
  - `README.txt`

Please turn in only the file named above. Extra files clutter up the "handin" directories.

Last Updated: 4/9/2015