COSC-160-01

Spring 2019

Project 0 Description

List ADT

1. Summary

This project serves to prepare you for submitting more involved projects later in the semester. Your program should maintain a linked list of integers. It also provides an opportunity to practice programming again (in case it has been a while for any students). A key objective is that you must submit the project with a makefile and it must run on the class server.

1. Submission and Compilation Requirements

You will submit a zip file named **submit.zip**. The zip file includes a makeﬁle along with any .cpp and .h files for your program. The makefile will contain the necessary compilation commands for your code. The target executable will be named p0 (that’s a zero, not a capital O).

Thus, the graders should be able to compile your program by accessing it on the class server and entering the following command:

**make p0**

*Check to make sure that it works on the class server*. If If the program does not compile (using the above make command) or the program does not run, the submission will not be accepted. Budget your time well. Include signiﬁcant time for design / planning and testing / debugging. Please submit early and often (version control)! Your last submission will be graded.

1. Input Requirements

After successful compilation, the following command will run your program:

**./p0 [inputFilename]**

The one argument, written as inputFilename above, is the name of a text file (assumed to be located in the same directory as the executable). For example, if the text file’s name is myTestFile.txt, the command to run the program would be **./p0 myTestFile.txt**. The ﬁrst line of the input file is an integer that is the number of values that your program will initially add to the list (followed by a newline character). The second line contains all the values to add (followed by a newline character). We will refer to this sequence of numbers as the “Additions”. *The Additions will not contain any duplicate values.* The third line then contains another single integer number that is the number of values that your program will delete from the list (followed by a newline character). Finally, the fourth line contains a sequence of values to delete from the list. We will refer to this sequence as the “Deletions”. The Deletions might contain values that do not appear in the Additions; in such a case, your program should just continue on to the next value in the Deletions.

1. Output Requirements

The main method should construct the list structure and read in the input ﬁle populating the list with the Additions. It should then print out the list to the console. Next, it should sequentially remove all the values that appear in the Deletions. Finally, it should print out the updated list to the console. Note that the main method should also be eﬃcient.

No other outputs should be observed.

1. Structural and Operational Requirements
2. A linked list (of integers or template if you wish) structure

* Add method
* Remove method
* Print method
* Any other methods (or classes) you consider useful

1. [No other structural requirements for Project 0; future projects will be more involved]

Notes: To earn full marks for COSC-160 projects, you must implement the structures **eﬃciently** (in space and time). If you are faced with a space / time tradeoﬀ *for this project*, you will (for this project) opt to improve time complexity (if the cost of space is relatively minor).

1. Rubric

|  |  |
| --- | --- |
| **List of Requirements** | **Percentage** |
| Makefile | 0.20 |
| Compiles as specified | 0.20 |
| Runs as specified | 0.20 |
| Adds values | 0.10 |
| Removes values | 0.10 |
| Prints list | 0.10 |
| Efficiency | 0.10 |
| **TOTAL** | **1.00** |

***The following sections are boilerplate for this course, but contain valuable information:***

1. Programming Languages

You should submit your projects using C++ (unless you have spoken with me and I have approved a different programming language for your project). Take note that there are many versions of C++; you must use the version that is currently running on the course server. Keep in mind that one of the main goals of our class projects is for you to learn how to construct various data structures from the most elemental programming constructs. Thus you will not receive credit when using any pre-existing structures from programming libraries (or code that has been created or designed by others). For example in C++ you *cannot* use the pre-existing vectors, stacks, lists, etc. For some programming languages, complex data structures (non-elemental constructs) are “built-into” the language. You cannot use any built-in structure. If you have any questions as to what structures are permitted (and which are not permitted), given your language of choice, please ask me.

1. Planning and Design

Before implementation, you should plan and design your project using standard approaches, e.g. UML class diagrams, ﬂow diagrams, etc. If you have questions pertaining to your project, I will ﬁrst ask to see your designs. I will not look at your code without ﬁrst viewing your design documents. You will be faced with many design decisions during this project. It is best to spend the requisite time during the design stages to assure an appropriate and eﬃcient implementation is built. Consider your options, perform a theoretical complexity analysis of the diﬀerent options, and base your decision on the results of your analysis.

1. Testing and Debugging (Not Submitted)

You may wish to construct an interactive interface to test the functionality of your structure at intermediate stages of development. This would likely be most eﬃcient with an interactive interface that allowed you to interactively test various functionalities of your structure given diﬀerent inputs. If you do implement a testing interface, please be sure to comment it (so that it does NOT execute) before submission. I also strongly encourage you to construct and test many input ﬁles to test the functionality of your implementation on varying inputs.

1. Version Control (Not submitted, but encouraged)

I strongly recommend that you back-up your work periodically throughout the development process. This can mitigate a disaster scenario where you might accidentally delete your program ﬁles. I also recommend employing a version control strategy which records your development at diﬀerent stages (versions). If you have time, I encourage you to investigate GitHub to facilitate version control. Otherwise you can make use of a more simplistic naming scheme: each time you save a ﬁle, change the ﬁlename to indicate a version: ﬁlename v1.cpp, ﬁlename v2.cpp, ... .