**CSE-381: Operating Systems**

**Exercise #5**

Max Points: 20

**Note: If you are using your personal machine then prior to commencing work on this exercise, you may need to install XMing, Putty, and WinScp as illustrated in LinuxEnvironment.pdf (and shown in the videos in the Handouts folder).**

|  |
| --- |
| **Objective**: The objective of this exercise is to:* Review developing classes in C++.
* Gain familiarity with operator overloading.
* Explore interactions between various methods.

**Submission**: Once you have completed this exercise, upload just the completed **User.cpp** file associated this exercise.You may discuss the questions with your instructor. |

# Preliminaries

1. Log onto the Linux server for this course via the following steps (that were covered in the previous exercises and as illustrated in the [LinuxEnvironment.pdf](https://niihka.muohio.edu/access/content/group/b05b7d30-d8c6-4312-9559-4d980565cbaf/Handouts%20_%20Video%20Tutorials/LinuxEvironment.pdf)):
	1. Run the X-Server Xming.
	2. Use PuTTY to log into the Linux server cse381-f12.csi.muohio.edu.
	3. When you log onto the server, you will be presented with a shell (**$**) prompt. You need to perform various tasks by typing commands at the shell prompt and pressing the enter (↵) key.
	4. Start emacs and ensure you see the graphical screen for emacs.
2. Download the following starter code and data file from Niihka into the same directory:
	1. User.h (Do not modify): This is the header file that contains the definition for a simple User class. This class encapsulates the following information about a user: userID, firstName, lastName.
	2. User.cpp: This is the source file in which all the methods associated with the User class are implemented.
	3. exercise5.cpp (Do not modify): This file contains simple test code for exercising various methods in the User class. It is educational to review this file as well once you have completed the exercise.
	4. user\_list.txt (Do not modify): This file contains user data for our course and is used by the test code in exercise5.cpp.

# Part #1: Complete methods in User.cpp [20 points]

|  |  |
| --- | --- |
| C:\Users\Victoria\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\MARLET97\MP900314351[1].jpg | Do not modify User.h and exercise5.cpp. You only need to modify User.cpp and implement various methods in User.cpp. A couple of methods are already implemented to illustrate an example. |

**Background**: C++ supports a more comprehensive set of standard methods that constitute the API of a class. Although the complier automatically generates many default implementations, it is often necessary to suitably implement various methods and operators to ensure correct functionality.

**Exercise**: This exercise requires you to implement various methods declared in User class. Review the declaration and documentation for the methods in User.h. Using the declaration, standard conventions, and documentation implement each method in User.cpp. Here are some notes/tips:

1. Until all the methods are implemented you will not be able to run the program.
2. However, each time you implement a method ensure your program continues to compile without any warnings or errors using the command line:

|  |
| --- |
| g++ **-c** -std=c++0x -g -Wall User.cpp exercise5.cpp |

1. Once you have implemented all the methods use the following command line to generate the final executable:

|  |
| --- |
| g++ -std=c++0x -g -Wall User.cpp exercise5.cpp **–o exercise5** |

**Sample Output**:

Once the various methods in User class have been successfully implemented the output from running the program multiple times is shown in the sample outputs below. Outputs shown below must exactly match your outputs. User inputs in the different runs of the program are shown in red color in the sample outputs below:

|  |
| --- |
| Test of constructors, operator<(), and operator>>(): |
| raodm@cse381-f12:~/hackArea$ ./exercise5 Loading user data...Done.Loaded data for 50 users.Query (list, quit, <userID>):quit |

|  |
| --- |
| Test of operator==() and operator<<(): |
| raodm@cse381-f12:~/hackArea$ ./exercise5 Loading user data...Done.Loaded data for 50 users.Query (list, quit, <userID>):**raodm**Information for userID: 'raodm': Dhananjai Rao (raodm)Query (list, quit, <userID>):**007**Information for userID: '007': James Bond (007)Query (list, quit, <userID>):**krypton1**Information for userID: 'krypton1': Super Man (krypton1)Query (list, quit, <userID>):**quit** |

|  |
| --- |
| Test of operator()(): |
| raodm@cse381-f12:~/hackArea$ ./exercise5 Loading user data...Done.Loaded data for 50 users.Query (list, quit, <userID>):**Super**Information for userID: 'Super': \*not found\* ()Here are results from generic search:Super Girl (krypton2)Super Man (krypton1)Query (list, quit, <userID>):**Blah**Information for userID: 'Blah': \*not found\* ()Here are results from generic search:Query (list, quit, <userID>):**quit** |

# Part 2: Submit files to Niihka

Upload just the following file to Nihhka:

* ***User*.cpp**: The C++ source file that you completed as part of this exercise.

# Part 3: Understanding calls to various methods

|  |  |
| --- | --- |
| C:\Users\Victoria\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\MARLET97\MP900314351[1].jpg | You should successfully complete earlier parts of this exercise prior to proceeding to this part. Note that there are no submissions required for this part. However, expect questions in exam(s) based on the assumption that you have completed this part of the exercise. |

**Background**: C++ supports a more comprehensive set of standard methods that constitute the API of a class. The complier automatically generates calls to these methods and consequently it is important to understand when calls to these methods occur so that these methods can be appropriately implemented.

**Exercise**: This part of the exercise focuses on self-learning about calls to various methods in a C++ class. In this exercise, the User.cpp class will be used as an example. Perform this exercise in the following manner and answer the questions further below:

1. Modify the constructors in User.cpp and add a line to print the fact that the constructor method has been called in each of the four constructors. Refer to the commented-line in the default constructor for example.
2. Modify the destructor in User.cpp and print a line of output to indicate that the destructor has been called.
3. Now compile and run exercise5. Simply enter quit to immediately exit out of the program. You should notice a large number of calls to various constructors and destructors.
4. Observe the sequence of calls to various constructors and answer the following questions:
	1. Why are there so many calls to constructors and destructors?

|  |
| --- |
|  |

* 1. Are there calls to the default constructor? Why do you think they are happening?

|  |
| --- |
|  |

* 1. Are there calls to the copy constructor? Why do you think they are happening?

|  |
| --- |
|  |

* 1. Are there calls to the move constructor? Why do you think they are happening?

|  |
| --- |
|  |

* 1. Why are the calls to the constructors occurring in batches?

|  |
| --- |
|  |

* 1. Why are the initial batch of calls smaller in size but the batch size seems to be larger further on?

|  |
| --- |
|  |

* 1. Why are there a large batch of destructor calls at the end without any calls to constructors (similar to the startup-time of the program)?

|  |
| --- |
|  |