CS 235 Midterm

Version 0.12

Instructor: R. P. Burton

March 2-5, 2015 (Monday thru Thursday)

Due in the Lab on Thursday no later than 8:00 p.m.; you must be physically in line to submit by 7:45 p.m.

Penalty for submitting the midterm late:

30 points per day (including weekend days), advancing at 8:01 p.m. each day

Open Book (142 course text and your CS 235 course text only), Open Notes (including your own Lab solutions)

Open Secondary Storage Device: yours only

Open Laptop: if you wish

Open Course Website and the reference section of [www.cplusplus.com](http://www.cplusplus.com/), but **no other Internet resources** (including, but not limited to no Google)

Closed Neighbor (and everyone is thy neighbor)

**\*Instructions\***

(Please read carefully)

1. This midterm consists of a C++ programming problem. Read and understand the statement of the problem completely before beginning to design, code, and test. Produce and attach to your submission a UML diagram (see Appendix B.1) depicting an appropriate object-oriented design. Consider in advance the test cases that will establish the correctness of your solution and test your solution thoroughly before submitting it.
2. Produce a solution, which consists of your C++ code, with a comment at the beginning of each file (both .h and.cpp) which includes your name, and “CS 235 Winter 2015 Midterm.” When you are finished, go to the course website and follow the link labeled “Submit Exam” in the Exam Menu. Upload your completed project by compressing the files and submitting through Learning Suite **with TA assistance**. If a packet is not collected by a TA upon submission, your exam will not be graded and you will receive no credit for the exam. Attribute any code taken from or based on other sources (excluding the course texts and the authorized websites). Attributed code copied from or based heavily on outside sources is worth half credit. Unattributed code copied from or based heavily on outside sources is worth no credit.
3. Understanding the problem correctly is part of the examination. If something seems unclear, ask a CS 235 TA (but no one else) for clarification. You may pose questions to the CS 235 TAs at any time. However, the TAs, generally, are not permitted to answer questions related to design, C++ implementation, debugging, or testing.
4. Prior to submitting your midterm, score it using the attached scoring sheet (this will help you maximize your points and will help us grade your exam accurately). If your score is within 5 points of the TA score, you get a 3 point bonus. If your score is within 6 to 10 points of the TA score, no bonus is awarded and penalty is imposed. If your score is more than 10 points different from the TA score, you lose 3 points. Be sure that your program runs properly on the 235 lab machines before submitting your solution. **Any section that does not compile will not be graded.**
5. Your solution packet must all be stapled together before it will be accepted by a TA, even if this results in a late submission penalty. At 7:45p.m., any line which has formed to submit exams with a TA will be closed; all students in line for pass-off will be the last students to be helped. Please be sure to be in line before that time.
6. Sign the Grading Sheet to request that your midterm be graded and to certify that no unfair information related to the midterm has been received by you, either directly or indirectly, and that none will be conveyed by you. If we discover that you cheated or assisted someone in cheating, intentionally or unintentionally (including accidentally), your score for this exam may (and probably will) be rand() % 1. We’re serious.

**The Josephus Problem**

The Josephus problem is named after the historian Flavius Josephus who lived between 37 and 100 BCE. Josephus was a reluctant leader of the Jewish revolt against the occupying Roman forces. When Josephus and his band determined that they were going to be captured, they resolved to kill themselves. Josephus suggested, “Let us commit our mutual deaths to determination by lot. He to whom the first lot falls, let him be killed by him that hath the second lot, and thus fortune shall make its progress through us all; nor shall any of us perish by our own right hand, for it would be unfair if, when the rest are gone, somebody should repent and save himself.” (Flavius Josephus, *The Wars of the Jews,* Book III, Chapter 8, Verse 7, Translated by William Whiston, 1737). As fate (or scheming) would have it Josephus was the person to whom the last lot fell. He and the person he was to kill surrendered to the Romans. Josephus did not describe how the lots were assigned, but the following approach generally is believed to be the way it was done. Josephus and his band formed a circle. They counted around the circle to some predetermined number. When that number was reached, the lot “fell” to that individual, he left the circle, and met his demise at the hands of the person to whom the next lot fell. The count started over with the next person in the circle.

The purpose of this midterm is to simulate instances of the Josephus approach, in part by utilizing a Circular Double-linked List. This exam is accompanied by an interface, CircleDLLInterface.h, which describes the functionality of the Circular Double-linked List. Your data structure will be graded by a test driver, which will not be available to you.

In addition to the Circular Double-linked List, you will write a program which uses your list to simulate the Josephus Problem. To provide explicit instructions and minimize ambiguity, this packet contains:

1. Detailed instructions on the functionality of your program
2. Information concerning the data structure you are implementing
3. Instructions on how to navigate the terminal, compile, and run your program within Valgrind
4. Instructions on submission of your midterm
5. The Grading Sheet

**The Algorithm**

You must write a Main.cpp file which runs a main function that performs the following operations:

1. Import a file containing names to add to Josephus’ band
2. Display the band roster with indices
3. Prepend a name manually
4. Append a name manually
5. Remove a name by index
6. Randomly shuffle the roster
7. Calculate the safe index
8. Dispatch the band (Simulate the Josephus approach on the current roster)
9. Quit

Operations 1 – 6 are essentially user interface operations, setting up and taking care of the band of revolutionaries. Operations 7 and 8 are the primarily algorithms that deal directly with the Josephus approach. After an operation is complete, except for quit, return to the menu.

Operation 1: Import the file

The files provided will consist of one word names each on its own line, and will end with the extension “.txt”, but we will always give you “.txt”s so you do not need to concern yourself with that. You may assume that the contents will be formatted properly, but you must check to see if a file was successfully opened in the event that an invalid filename was given. You may assume that files will not contain duplicate names. As an example, the following is the contents of a file named “TAs.txt”

“Davis

JacobK

JacobC

Tyler

Matt

Blaine

Isaac

Trevor

Kaylee

Anthony

Levi

Jordan

Jason”

Even though this is known as Josephus’ band, he may or may not be included in the list. If his name is included, simply add him just like the other names. If he is not in the list, it is not necessary to include him.

Operation 2: Display the roster with indices

While you will not be graded on specific formatting, please make all output readable and organized. If the roster is empty, inform the user, and return to the menu. An example of printing the roster after importing “TAs.txt” would be:

0 - Davis

1 - JacobK

2 - JacobC

3 - Tyler

...

Operation 3 & 4: Prepend and Append names to the roster

Read a name from the command line for fine tuning of the Roster. If prepend was selected, add the name to the front of the roster. If append was selected add the name to the end of the roster. If the name already exists in the roster, do not add it again, instead, inform the user and return to the menu.

Operation 5: Remove a name by index

Read in an index from the command line. If the index given was not in the bounds of the roster (such as an empty list) inform the user and return to the menu. If the index is valid, remove the name associated with that index, and inform the user who was removed.

Operation 6: Randomly shuffle the roster

To add an extra layer of fate, provide the ability to randomly shuffle the roster. You must create a shuffle algorithm of your own design. You are not allowed to use any standard library shuffle function.

Operation 7: Calculate the safe index

This operation does nothing to the current roster, but computes the “safe index,” i.e. where Josephus needs to stand, replacing the person at that position, to be the person to whom the last lot falls. It expects two arguments: the number of people n in the original circle and the count m (1 <= m <= n). If the second argument is not within the valid range, reprompt the user; do not return to the menu. Use a Circular Double-linked List to store “Josephus and his band” (removing them one at a time) and calculate the “safe index.” Report m, n, and the “safe index.”

For a list of size n = 20 and a counting number m = 4, the safe index is 16 (0 being the first index as always).

Operation 8: Dispatch the band (Simulate the Josephus approach)

This operation performs Operation 7 and actually removes names from the current roster according to the same algorithm. Since the size of the roster is already known, prompt only for the counting number m. Again, if m is not in the valid range (1 <= m <= n), then reprompt; do not return to the menu.

Count around your Circular Double-linked List according to the counting number m, and remove every mth name. At each removal, report the name removed. When the roster only contains one name, report the survivor and clear the roster.

For example, if we were to run the algorithm on TAs.txt with a counting number m = 5, the output would be:

Removed: Matt

Removed: Anthony

Removed: JacobK

Removed: Trevor

Removed: Davis

Removed: Kaylee

Removed: Tyler

Removed: Jason

Removed: Jordan

Removed: JacobC

Removed: Isaac

Removed: Levi

The Survivor: Blaine

Operation 9: Quit

Exit the program with a traditional parting phrase, such as “Shalom.”

**The Data Structure**

You will implement a Circular Double-linked List for this midterm. This list should contain the names of the people in Josephus’ circle. Functionality for this list is found in CircleDLLInterface.h. The Data Structure portion of your midterm will be graded by test driver, but you will not have access to this test driver. The comments in the interface are binding as this document; please read them thoroughly. As this is not a template class, your implementation should be divided into a “.h” and a “.cpp” file.

**Implementation Notes**

Your program must compile on the CS 235 Lab machines with the g++ compiler (described below). **If a section of your code does not compile, that section cannot be graded**. For example, if your data structure passes the test driver, but your main function does not compile, then you will get credit for the test driven portion, but we will not grade your main. Points for UML diagrams and code neatness can be given independent of successful compilation.

For your convenience, a script “Compile\_and\_Run.sh” has been written to facilitate compilation on the Lab machines. Place all of your code in the same folder as the script, then double click the script. A dialogue should appear; click “Run in Terminal.” The script is set up to run your program within Valgrind. Below is a more detailed description of how to use the terminal to run your code.

Your entire program, and specifically the List, should manage its memory well. We will grade your midterm using Valgrind, which your program must pass for you to receive full credit.

Common commands for navigation in the Terminal:

|  |  |
| --- | --- |
| Command | What it does |
| pwd | “print working directory” – Prints the absolute file path to the directory you are currently in. |
| cd filepath | “change directory” – changes to the directory specified by the given file path.  If a directory or file name has spaces, put the file path in quotes. |
| cd .. | Goes up one level in the directory. |
| ls | “listing” – displays the list of files and folders of the current directory. |

How to manually compile C++ code with g++:

**g++ \*.h \*.cpp –o NameOfPogramToCreate**

“g++” is the command to compile, “\*.h” and “\*.cpp” mean to collect all .h’s and .cpp’s of the current directory, and “-o” is a flag that means the following word will be the name of your executable.

How to run your program normally and with Valgrind:

A simple run of your program, after compilation, is done with “./”, for example if you named your program “nameOfProgram,” you can run it by navigating to its directory and typing the following and hitting enter:

**./nameOfProgram**

To run it with valgrind, use the following command:

**valgrind --tool=memcheck --leak-check=yes ./nameOfProgram**

Ideally, you should resolve all memory errors, but the most important part is the heap summary at the end. If it says:

**All heap blocks were freed -- no leaks are possible**

Then you are doing fine.

**Submission Procedures**

When you are ready to submit your midterm, make sure that the following activities have been completed:

1. You have verified that your solution compiles on the Lab machines with the g++ compiler.
2. You have printed out the grading sheet (not necessary to print entire packet), filled out the student column, and signed the bottom.
3. You have attached a UML diagram of your implementation to the grading sheet or specified an electronic one is included with your code.
4. You have properly set up your factory class.
5. You have compressed your code, interface files, factory, and main function all into preferably a “.zip”. When you compress a file, there is a drop down menu of common archive types. Most lab computers default to “.tar.gz” but you can select “.zip” explicitly.

Once those things are in place, ***you must access a TA to watch you submit your “.zip” to Learning Suite.*** Verify that the TA initials that they have entered on your grading sheet. If you submit to Learning Suite without observation or permission, then late penalties may apply.

**Grading Sheet**

Student’s Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ TA Initials\_\_\_\_

Days Late\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **Student:** | **TA:** | **The Circular Double-linked List:** |
| \_\_/18pts | \_\_/18pts | 1. Insert functions properly add to the list (6 points each). |
| \_\_/18pts | \_\_/18pts | 2. Remove functions properly remove from the list (6 points each). |
| \_\_/18pts | \_\_/18pts | 3. Accessor functions (atFromHead 5 points, atFromTail 5 points, and size 3 points) and clear (5 points). |
|  |  | **The Main Program:** |
| \_\_/5pts | \_\_/5pts | 1. Menu follows required format. After any operation except quit, the user is returned to the menu. |
| \_\_/6pts | \_\_/6pts | 2. Reads an input file and successfully creates the roster. Rejects unacceptable file names entered by the user and returns to the menu. |
| \_\_/6pts | \_\_/6pts | 3. Prints the contents of the roster with indices beginning at 0. |
| \_\_/6pts | \_\_/6pts | 4. Properly appends, prepends, and removes names to and from the roster. Rejects any invalid input and returns to the menu. |
| \_\_/6pts | \_\_/6pts | 5. Randomly shuffles the order of the names in the roster. |
| \_\_/15pts | \_\_/15pts | 6. Successfully calculates and reports the safe index for two numbers: n: size, n > 0, m: counting number, 1 <= m <= n. Rejects any invalid input and repompts the user. |
| \_\_/15pts | \_\_/15pts | 7. Successfully simulates the Josephus approach on the current roster. Rejects and returns to the menu and reprompts for any counting number out of range. Prints the names of the dispatched members in the order of their execution with the survivor last. Clears the roster when complete. |
|  |  | **Other:** |
| \_\_/15pts | \_\_/15pts | 1. Appropriate UML diagram attached. |
| \_\_/15pts | \_\_/15pts | 2. Your code successfully passes Valgrind and does not crash. (No partial credit). |
| \_\_/7pts | \_\_/7pts | 3. You have neat and consistent code with comments where appropriate. |
| \_\_/Subtotal | \_\_/150pts |  |

**For TA use only:**

|  |  |  |
| --- | --- | --- |
|  | \_\_/Late Day Penalty |  |
|  | \_\_/Accurate Grading Modifier  +3 if |TA – Student| <= 5  -3 if |TA – Student| > 10 |  |
|  | **\_\_/150 TOTAL** |  |

|  |  |  |
| --- | --- | --- |
| Student’s Signature |  | Grader’s Printed Name |