CS 235 Midterm
Version 0.7
Instructor: R. P. Burton

February 29 - March 4, 2016 (Monday through Friday)
Due in the Lab on Friday 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 (not the forums), but **no other Internet resources** (including 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 the test cases in advance 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 2016 Midterm." When you are finished, go to the course website and follow the link labeled "Submit" in the Midterm section of the Assignments 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.
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, your score is unaffected. 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.
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 CE. 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 Problem, 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 Doubly-linked List. You must implement your roster using a Circular Double-Linked List. Your data structure will be graded by a test driver.

In addition to the Circular Double-linked List, you will write a program which uses your list to simulate the Josephus Problem. To give 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 Problem on the current roster)
9. Quit

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

Operation 1: Import the file

This operation should import a list of names into the roster. If the roster is not empty, clear the roster and then import the names. The files provided will consist of one-word names each on its own line. We will include the file extension in the file name so you do not need to append any file extension to the file names that we give you. You may assume that the contents will be formatted properly, but you must check to see if a file was opened successfully 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 content of a file named "NoEvolution.txt"

Farfetch'd
Kangaskhan
Pinsir
Tauros
Lapras
Ditto
Aerodactyl
Articuno
Zapdos
Moltres
Mewtwo
Mew

Operation 2: Display the roster with indices

This operation should display each member of the roster and his index, beginning at index 0. 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 "NoEvolution.txt" would be:

0 - Farfetch'd
1 - Kangaskhan
2 - Pinsir
3 - Tauros
...

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

These operations should read in 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

This operation should 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. See CircularDLLInterface.h for further explanation regarding indexing with the at() function.

Operation 6: Randomly shuffle the roster

To add an extra layer of fate, this option should randomly shuffle the players in the roster. You must create a shuffle algorithm of your own design. Although you may use the rand() and srand() functions, 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 would need to stand in order to be the person to whom the last lot falls. It expects two arguments which should be read in from the command line: The number of people n in the circle and the count c (1 <= |c| <= n where |c| is the absolute value of c). If c is less than zero, you should count backwards around your list (starting at index zero). If the second argument is not within the valid range, reprompt the user; do not return to the menu. You must make sure that the given number of people (the variable n) is valid, i.e. greater than 0. Note that the number of people given may be different from the size of your roster. One way to solve this problem is to use a Circular Double-linked List to store "Josephus and his band" (removing them one at a time) and calculating the "safe index." Report n, c, and the "safe index."

A few examples of what this operation should output follow:

n = 10 c = 1 safe index = 9

n = 50 c = 6 safe index = 44

n = 100 c = 3 safe index = 90

n = 10 c = -2 safe index = 6

n = 20 c = -7 safe index = 18

Note that in all cases the person standing at index 0 is always counted first. Thus, if c = 1 or c = -1 the first person to be removed is the person at index 0. Walking through one of the smaller examples above by hand will give some insight into how the safe index is calculated.

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

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

Count around your Circular Double-linked List according to the counting number c (backward or forward depending on the sign), and remove every c^{th} name. At each removal, report the name removed. When the roster contains only one name, report the survivor and clear the roster.

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' band. Functionality for this list is found in CircleDLLInterface.h. The Data Structure portion of your midterm will be graded by test driver. The comments in the interface are as binding as this document, so read them thoroughly. As this is not a template class, your implementation should be divided into a ".h" and a ".cpp" file. You are given access to this test driver, but understand that **passing the test driver does not guarantee a passing score for the midterm**. The test driver only validates the functionality of your data structure (the first 3 parts of the grading sheet).

Remember to exclude any main function when running your data structure against the test driver. Your main function will overshadow the test driver's main and the tests will not run.

Implementation Notes

The Modulus Operator (%)

You may find it useful, when counting around the circle, to keep your counting position within the range 0 <= |c| < n. Since negative counting numbers are permissible, it is important to know how the % operator handles negative numbers.

In pure mathematics, the result of a modulus operation on a negative number is always positive (for example -7 mod 10 = 3). In many programming languages, including C and C++, the % operator has a slightly different behavior. While it always gives a number modularly congruent to the pure mathematical model, it will often preserve the sign of the number (example "-7 % 10 == -7" results to true, but "-7 % 10 == 3" is false). Generally speaking, the definition of the / and the % operators for C++ satisfies this equality:

$$a == (a / b) * b + a \% b$$

A concrete example:

$$(-28 / 5) * 5 + -28 \% 5$$

$$(-5) * 5 + (-3)$$

$$-25 – 3$$

$$-28$$

Notice, because of truncated the division the remainder needed to be negative so that when the product was added back to its remainder, it resulted in the original number.

Compiling Your Program and Running Valgrind

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 NameOfProgramToCreate
```

"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
```

Make sure that you recompile your code (with the g++ command) every time that you make a change. To run you program 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
```

and there are no invalid reads/writes, conditional jumps based on uninitialized values, etc. 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 printed out the grading sheet (not necessary to print entire packet), filled out the student column, and signed the bottom.
2. You have attached a UML diagram of your implementation to the grading sheet or specified that an electronic one is included with your code.
3. You have set up your factory class properly.
4. 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, go get a TA to watch you submit your ".zip" to Learning Suite and verify that the TA initials that we have received your grading sheet.

Grading Sheet

Student 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 invalid file names provided 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, c: counting number, 1 <= |c| <= n. Rejects any invalid input and reprompts the user.
__/15pts	__/15pts	7. Successfully simulates the Josephus Problem 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 Signature

TA Signature