CS 235 Midterm

Version 0.1

Instructor: R. P. Burton

May 25-27, 2016 (Wednesday 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](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 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 Spring 2016 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, 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. on Friday, 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 Data Structure**

You will implement a Circular Double-linked List for this midterm. This list should consist of nodes that contain strings, which will represent the names of people in the list. 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.

This exam is accompanied by an interface, CircleDLLInterface.h, which describes the required functionality of the Circular Double-linked List. You must implement your solution to the midterm by using a Circular Double-linked List of your own creation. Your data structure will be graded entirely by test driver, which will be made available to you and will be similar to other test drivers used in the course.

In addition to the Circular Double-linked List, you will write a program that uses your list to perform a few additional operations. 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 that you are implementing (further information can be found in the interface file)
3. Instructions on how to navigate the terminal, compile, and run your program with 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 allows the user to perform the following operations:

1. Import a file containing names to add to the list
2. Display the current list of names with indices
3. Prepend a name manually to the list
4. Append a name manually to the list
5. Append several names simultaneously to the list
6. Remove a node at a given index
7. Randomly shuffle the list
8. Clear the list
9. Quit

After an operation is complete, except for quit, return to the menu. To select an operation, prompt the user to enter a number 1-9. Do not ask the user to input letters or words such as “import” or “display” to select operations from the main menu.

**Operation 1: Import the file**

This operation should import a new list of names into the list. If the list is not empty, clear the list and then import the names. The files will consist of a series of names with exactly one name on each line. We will include the file extension in the filename so you do not need to append any file extension to the filename that we give you. You may assume that the contents of the files are all properly formatted, but you must check to see that a file was successfully opened in the event that an invalid filename is given. You may assume that the files do not contain duplicate names. For example, the contents of a file named “NoEvolution.txt” might be:

Farfetch’d

Kangaskhan

Pinsir

Tauros

Lapras

Ditto

Aerodactyl

Articuno

Zapdos

Moltres

Mewtwo

Mew

All names will consist of a single word and there will be only one name on each line. Names should be imported into the list in the order that they appear in the file. For example, in the above list “Farfetch’d” would be located at index “0”, Kangaskhan would be located at index 1, etc...

**Operation 2: Display the names with indices**

This operation should display each member of the list and the associated index. While you will not be graded on specific formatting, please make all output readable and organized. If the list is empty, inform the user and return to the menu. An example of printing the list after importing “NoEvolution.txt” would be:

0 - Farfetch’d

1 - Kangaskhan

2 - Pinsir

3 - Tauros

...

**Operation 3 & 4: Prepend and Append names to the List**

These operations will read a name from the command line for fine tuning of the list. If prepend was selected, add the name to the front of the list. If append was selected add the name to the end of the list. If the name already exists in the list, do not add it again; instead, inform the user and return to the menu.

**Operation 5: Append several names manually**

This operation should allow the user to input several names at once to be imported into the list. When this operation is selected, the user should be prompted to enter in several names. The user will enter all the names simultaneously and on the same line, for example the user might enter:

Ryan Kyle Jason Adam Timnah Andrew Jacob Robert Matt

On a single line, and the names should be appended onto the list in the order that they were entered. For this operation you may want to consider using the getline() and get() functions, which are documented on cplusplus.com.

**Operation 6: Remove by index**

This option reads 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 of who was removed.

**Operation 7: Randomly Shuffle the List**

This option will randomly shuffle the names in the list. 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. Partial credit will be given for algorithms that shuffle the list only somewhat randomly.

**Operation 8: Clear the List**

This operation should clear the list so that size() returns 0. After this operation the list should be empty.

**Operation 9: Quit**

Exit the program with a traditional parting phrase, ASCII art is awesome but not required.

**Implementation Notes**

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**

And there are no invalid reads/writes, conditional jumps based on uninitialized values, etc. then you are doing fine. Note that there is a documented bug in the current version of g++ that may cause a leak of 72,704 bytes in one block, which you are not responsible for.

**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 (it is not necessary to print the 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 properly set up your factory class.
4. You have compressed your code, interface files, factory, and main function all into 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 they have received your grading sheet.

**Grading Sheet**

Student Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ TA Initials\_\_\_\_

Days Late\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **Student:** | **TA:** | **The Circular Double-linked List:** |
| \_\_/20pts | \_\_/20pts | 1. Insert functions properly add to the list (6 points each, 20 points for completion). |
| \_\_/20pts | \_\_/20pts | 2. Remove functions properly remove from the list (6 points each, 20 points for completion). |
| \_\_/20pts | \_\_/20pts | 3. Accessor functions (at 12 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. |
| \_\_/10pts | \_\_/10pts | 2. Reads an input file and successfully creates the List. Rejects invalid file names provided by the user and returns to the menu. |
| \_\_/5pts | \_\_/5pts | 3. Prints the contents of the List with indices beginning at 0. |
| \_\_/5pts | \_\_/5pts | 4. Properly appends and prepends names to the List. Rejects any invalid input and returns to the menu. |
| \_\_/10pts | \_\_/10pts | 5. Allows the user to input several names on the same line. Correctly imports the names to the List. |
| \_\_/10pts | \_\_/10pts | 6. Correctly removes from the List by index and correctly clears the List for operations 6 and 8. |
| \_\_/15pts | \_\_/15pts | 7. Randomly shuffles the order of the names in the List. |
|  |  | **Other:** |
| \_\_/10pts | \_\_/10pts | 1. Appropriate UML diagram attached. |
| \_\_/15pts | \_\_/15pts | 2. Your code successfully passes Valgrind and does not crash. (No partial credit). |
| \_\_/5pts | \_\_/5pts | 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 |