Project 2: Linked Stacks Homework projects should be worked on by each individual student. Brainstorming and sketching out problems on paper or on a whiteboard together are permitted, but do not copy code from someone else or allow your code to be copied. Students who commit or aid in plagiarism will receive a 0% on the assignment and be reported.

Information

Topics: Linked structures, stacks, exceptions

Turn in: Turn in all source files - .cpp, .hpp, and/or .h files. **Do not turn in Visual Studio files**.

Starter files: Download from GitHub.

Building and running: If you are using Visual Studio, make sure to run with debugging. (Don't run without debugging!) Using the debugger will help you find errors.
To prevent a program exit, use this before `return 0;`

```
cin.ignore();
cin.get();
```

Tips:

- Always make sure it builds. Only add a few lines of code at a time and build after each small change to ensure your program still builds.
- One feature at a time. Only implement one feature (or one function) at a time. Make sure it builds, runs, and works as intended before moving on to the next feature.
- Search for build errors. Chances are someone else has had the same build error before. Copy the message into a search engine and look for information on *why* it occurs, and *how* to resolve it.
- Use debug tools, such as breakpoints, stack trace, and variable watch.
- Don't implement everything in one go. Don't just try typing out all the code in one go without building, running, and testing. It will be much harder to debug if you've tried to program everything all at once.

Contents

PART 1

Getting started

About

Program purpose

This program loads in a list of courses from the `courses.txt` file. This text file contains a list of classes: their codes and names, and one prerequisite. You won't have to write the file reader; that has been implemented for you.

```
COURSE      MATH111     Fundamentals_of_Mathematics

COURSE      MATH115     Elementary_Algebra
PREREQ      MATH111

COURSE      MATH116     Intermediate_Algebra
PREREQ      MATH115

COURSE      MATH173     Precalculus
PREREQ      MATH116
```

Figure 1.1: Part of the input text file

In the program, the user can view a list of all classes, or view prerequisites for a given class. All classes will be stored in a `LinkedList`, and your Stack will be used to collect lists of prerequisites, until you hit a course that does not have any prerequisites.

You can use the LinkedList as reference while you're implementing the Stack.

```
-------------------------------------------
 | GET PREREQS |
 ---------------

 Enter class code

 >> CS250

Classes to take:
1.  CS134   Programming_Fundamentals
2.  CS200   Concepts_of_Programming_Algorithms_Using_C++
3.  CS235   Object_Oriented_Programming_Using_C++
4.  CS250   Basic_Data_Structures_Using_C++
```

Figure 1.2: The program showing a list of prerequisites

Project files

The project contains the following files. You will be modifying the files marked with *.

```
Project 2 - Stacks/
├── CUTEST/
│   ├── StringUtil.hpp ...  Tester files
│   ├── TesterBase.hpp ...  Tester files
│   └── TesterBase.cpp ...  Tester files
├── DATA_STRUCTURES/
│   ├── Node.hpp ...  The Node used by List and Stack
│   ├── LinkedList.hpp ...  Linked List (already written)
│   └── * LinkedStack.hpp ...  Linked Stack
├── EXCEPTIONS/
│   └── * CourseNotFoundException.hpp ...
│     Custom exception
├── UTILITIES/
│   └── Menu.hpp ...  Menu utility for program
├── Course.hpp ...  The Course struct
├── CourseCatalog.hpp ...  Header for the main program
├── Tester.cpp
├── * CourseCatalog.cpp ...  Implementation for the main
│                            program
└── prereq_finder.hpp ...  Contains main()
```

```
    courses.txt ...  Input text file
```

Path issues

You will need to take notice of where `test_results.html` gets written to, and where `courses.txt` is read from. It will be the same path on your machine, but *where* it should be on your machine depends on the IDE you're using.

When you run the program, it should try to tell you the proper path:

```
* TEST LOG WILL BE WRITTEN OUT TO:
[...]/Projects 2018-01/Project 2 - Stacks/Project2-Stacks
   student
```

Figure 1.3: The program trying to display the program's running directory

Tester

Tests are already written in this project. It uses Rachel's cuTEST framework. The tests will run when you launch the program, and there will be some *basic* output to the console window with an overview of tests being run and what passes/fails. For **more information** on the tests, make sure to open the `test_results.html` file.

Test set	Test	Prerequisite functions	Pass/fail	Expected output	Actual output	Comments
Test_Stack	Create an empty Stack, make sure Size() is 0.	• Size() • Push() • Top() • Pop()	**passed**	A. Size = 0	A. Size = 0	
Test_Stack	Create a Stack. Push 5. Size() should be 1. Top() should be 5.	• Size() • Push() • Top() • Pop()	**FAILED**	A. Size = 1 B. Top = 5	A. Size = 0 B. Top = -100	• Caught exception while getting Top()! • Bad Size value. Make sure you're setting the size in the constructor, and have written the Size() function, and are incrementing the size during Push().

Figure 1.4: The test output.

The test output will contain information for each test, including:

- The test name and description of the test
- List of prerequisite functions

- Whether it passed or failed
- Expected and actual values
- Suggestions on what to check to fix

PART 2

Lab specifications

Creating the CourseNotFound exception

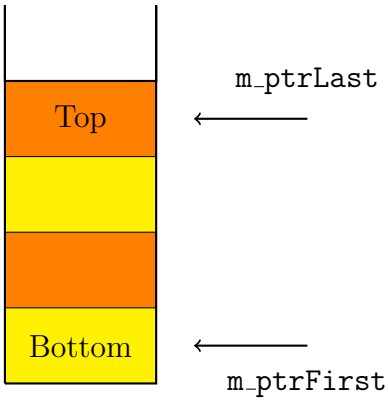
Before you implement the Stack, you should have the CourseNotFound custom exception written. If the user tries to access the Top() item from the Stack, but the Stack is empty, then this exception will be thrown.

In EXCEPTIONS/CourseNotFoundException.hpp, you'll create a class named `CourseNotFound`, which inherits from the `runtime_error` exception. All you need here is a constructor that takes in a string parameter for the error, and then call the `runtime_error`'s constructor with that information.

```
1 class CourseNotFound : public runtime_error
2 {
3     public:
4     CourseNotFound( const string& error )
5         : runtime_error( error )
6     {
7         // Nothing to do here
8     }
9 };
```

Figure 2.1: Code for the CourseNotFound exception

Writing the LinkedStack



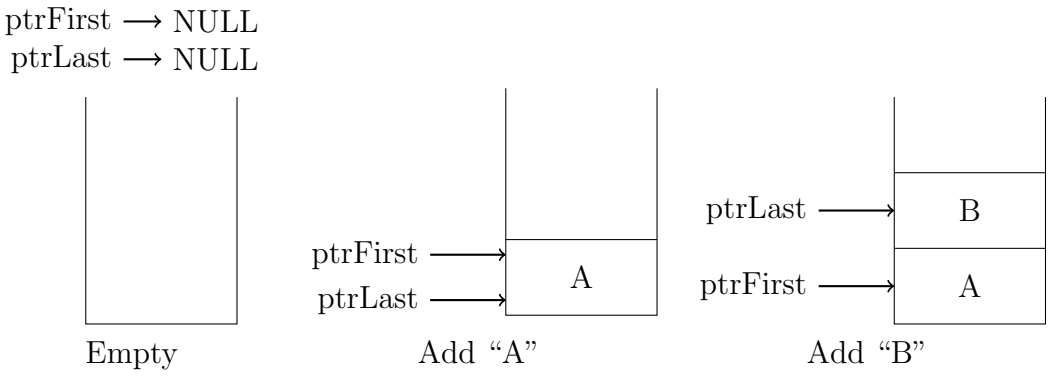
Within the `DATA_STRUCTURES/Stack.hpp` file, you will be implementing a LinkedStack. This means, like a LinkedList, it will use Node pointers, instead of being implemented with an array.

You can use the `LinkedList` either as reference, or as a parent for the LinkedStack as you're implementing it.

Stack functionality

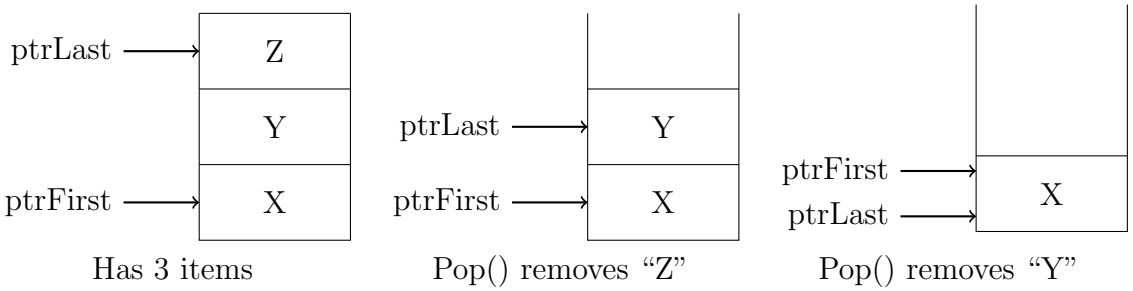
Push

As you add items to the stack, it fills from bottom-up. Push will not throw any exceptions.



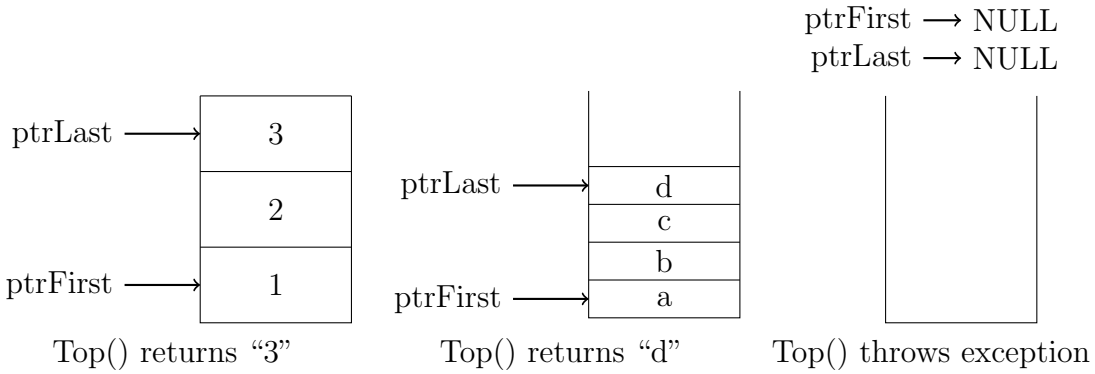
Pop

Pop removes the top-most item from the stack. Pop will not throw any exceptions. If Pop is called on an empty Stack, nothing occurs.



Top

Top returns the value of the top-most item of the stack. Top will throw a `CourseNotFound` exception if the stack is empty when Top() is called.



Testing

As you write the functionality for the stack, make sure to run the tests to validate your work. Once all tests pass, you should be able to move on to implementing the main program's functionality.

Implementing the prereq functionality

The program's main functionality is held in the `CourseCatalog` files. Make sure to mark functions that don't throw exceptions as `noexcept`.

void ViewCourses()

This function will not throw any exceptions.

Use the `LinkedList`'s `Size()` and `operator[]` functions to access each course in the list in a for-loop. Display each course's code, name, and prerequisite.

```
-----------------------------------------------
 | VIEW COURSES |
 ----------------

#    CODE      TITLE
       PREREQS
-----------------------------------------------
0    MATH111   Fundamentals_of_Mathematics

1    MATH115   Elementary_Algebra
       MATH111

2    MATH116   Intermediate_Algebra
       MATH115

3    MATH173   Precalculus
       MATH116
```

Figure 2.2: A partial view of the class output

Course FindCourse(const string& code)

This function takes in a course `code`, such as "CS250", "MATH171", etc. It will search through the `m_courses` LinkedList, and if found, it will return that course.

Otherwise, it will throw the `CourseNotFound` exception.

void ViewPrereqs()

This function will not throw any exceptions.

First, you need to get the course code from the user.
Use the `FindCourse` function to get that course.

```
current = FindCourse( courseCode );
```

Make sure to wrap it in a try/catch! If the `CourseNotFound` exception is caught, then display an error message and leave the `ViewPrereqs` function.

```
-------------------------------------------
 | GET PREREQS |
 ---------------

 Enter class code

 >> CS123

Error! Unable to find course CS123!
```

Figure 2.3: Unable to find a course by its code

After the course has been found, create a Stack of Courses

```
LinkedStack<Course> prereqs;
```

and push the current course that you've found.

Then, create a while loop. We will keep re-using the `current` Course object, storing each prerequisite, until we run out. So, for the loop, continue looping *while* `current.prereq != ""`.

Within the loop, use `FindCourse` again to find the current course's prerequisite. Surround this in a try/catch!
If the course isn't found, then **break** out of the while loop.
Otherwise, if the course is found, then push it onto the prereqs stack.

The loop will stop once we've run out of prerequisites. At this point, the stack should be full of classes, with the class the user selected at the **bottom**, and each prereq, in order, until we get to the top.

Finally, we're going to display all the classes to the user. Create another while loop. This one will keep looping *while* the `prereqs` stack is *not empty*.

Within the loop, get the top course from the stack. Display its code and name. Then, pop it off the stack.

The while loop should end once the stack is empty.

```
-------------------------------------------
 | GET PREREQS |
 ---------------

 Enter class code

 >> MATH254

Classes to take:
1.  MATH111 Fundamentals_of_Mathematics
2.  MATH115 Elementary_Algebra
3.  MATH116 Intermediate_Algebra
4.  MATH173 Precalculus
5.  MATH241 Calculus_I
6.  MATH242 Calculus_II
7.  MATH243 Calculus_III
8.  MATH254 Differential_Equations
```

Figure 2.4: Getting prereqs for MATH254

PART 3

Example output

```
-------------------------------------------
 | Running tests... |
 --------------------

* TEST LOG WILL BE WRITTEN OUT TO:
/home/rayechell/TEACHING/cs250/cs-250-private-files/Projects
   2018-01/Project 2 - Stacks/Project2-Stacks SOLUTION

Running testset 1 out of 1:     Test_Stack()
 * Create an empty Stack, make sure Size() is 0.
                ...        PASS
 * Create a Stack. Push 5. Size() should be 1. Top() should
   be 5.         ...        FAIL
 * Create a Stack. Push 'A' & 'B'. Size() should be 2. Check
   order.       ...        FAIL
 * Create a Stack. Push 3 items and Pop. Check Size(). Check
   Top().       ...        FAIL



 NOTE: CHECK "test_result.html" FOR FULL DETAILS.

-------------------------------------------
 | LOADING COURSES |
 -------------------

 * 17 courses loaded
```

Figure 3.1: Program starts and runs tests

```
-------------------------------------------
 | MAIN MENU |
 -------------

 1. View all courses
 2. Get course prerequisites
 3. Exit

 >> 2
```

Figure 3.2: Main menu

```
-------------------------------------------
 | GET PREREQS |
 ---------------

 Enter class code

 >> CS250

Classes to take:
1.  CS134   Programming_Fundamentals
2.  CS200   Concepts_of_Programming_Algorithms_Using_C++
3.  CS235   Object_Oriented_Programming_Using_C++
4.  CS250   Basic_Data_Structures_Using_C++
```

Figure 3.3: Getting prereqs for CS250

```
-------------------------------------------
 | GET PREREQS |
 ---------------

 Enter class code

 >> CS123

Error! Unable to find course CS123!
```

Figure 3.4: Unable to find a course by its code

```
-------------------------------------------
 | VIEW COURSES |
 ----------------

#    CODE      TITLE
        PREREQS
-------------------------------------------
0    MATH111   Fundamentals_of_Mathematics
1    MATH115   Elementary_Algebra
        MATH111
2    MATH116   Intermediate_Algebra
        MATH115
3    MATH173   Precalculus
        MATH116
4    MATH241   Calculus_I
        MATH173
5    MATH242   Calculus_II
        MATH241
6    MATH243   Calculus_III
        MATH242
7    MATH254   Differential_Equations
        MATH243
8    CS134     Programming_Fundamentals
9    CS200     Concepts_of_Programming_Algorithms_Using_C++
        CS134
10   CS210     Discrete_Structures_I
        MATH171
11   CS211     Discrete_Structures_II
        CS210
12   CS235     Object_Oriented_Programming_Using_C++
        CS200
13   CS250     Basic_Data_Structures_Using_C++
        CS235
14   FL165     Elementary_Chinese_I
15   FL166     Elementary_Chinese_II
        FL165
16   FL192     Intermediate_Chinese_I
        FL166


 Press ENTER to continue...
```

Figure 3.5: Viewing all courses

PART 4

Grading breakdown

Breakdown	
Score	
Item	**Task weight**
Using exceptions, try/catch	10.00%
Tests passing	10.00%
Stack constructor	5.00%
Stack Push	10.00%
Stack Pop	10.00%
Stack Top	10.00%
Stack Size	5.00%
CourseNotFound exception	5.00%
CourseCatalog FindCourse	10.00%
CourseCatalog ViewCourses	10.00%
CourseCatalog ViewPrereqs	15.00%

Score totals	100.00%

Penalties	
Item	**Max penalty**
Syntax errors (doesn't build)	-50.00%
Logic errors	-10.00%
Run-time errors	-10.00%
Memory errors (leaks, bad memory access)	-10.00%
Ugly code (bad indentation, no whitespacing)	-5.00%
Ugly UI (no whitespacing, no prompts, hard to use)	-5.00%
Not citing code from other sources	-100.00%
Not all tests run (Tests crash)	-10.00%
Penalty totals	