EECS268:Lab4

From ITTC

Navigation

Home

Information

Syllabus
Schedule

Classwork

Labs
Homework
Submitting Work

Contents

Due time

This lab is due before your next lab session begins

Required Files

- Makefile
- main.cpp
- StackInterface.h
- Stack.h/hpp
- Node.h/hpp (similar to prior implementation but templated)
- PreconditionViolationException.h/cpp

Provided Files

- Test.h (http://people.eecs.ku.edu/~jgibbons/eecs268/2016spring/lab04/Test.h) and Test.cpp (http://people.eecs.ku.edu/~jgibbons/eecs268/2016spring/lab04/Test.cpp)
- Test_Stack.h (http://people.eecs.ku.edu/~jgibbons/eecs268/2016spring/lab04/Test_Stack.h) and Test_Stack.cpp (http://people.eecs.ku.edu/~jgibbons/eecs268/2016spring/lab04/Test_Stack.cpp)

Overview

This lab is a primer for creating an interface as well as continuing to utilize inheritance. Inheritance is a way to make an extension to, or a more specialized version of, a class. An interface, in C++, is a class that has all pure virtual methods, except for the destructor. In this lab you will have one interface, StackInterface, which will contain the pure virtual methods - again, minus the destructor - that form a obligation to the inheriting class to implement.

Some important notes about interfaces:

- Any method that accepts a base class as a parameter, can be passed any of its descendants.
- You **cannot** create an instance of an Interface nor any abstract class (a class with one or more pure virtual methods); it has no implementation so you cannot create an instance of it; you can only declare and pass around instances of subclasses
- A class that inherits from an interface or abstract class **must** implement all pure virtual methods. Otherwise this subclass will be considered abstract as well.
- We define a virtual destructor with an empty body in the interface to ensure that the destructor for all classes derived from the interface will be called when appropriate
- Since no instance of the interface itself will ever be created, we do not define any constructor for interfaces

```
class TestInterface
{
    public:
    virtual ~TestInterface() {}; //virtual destructor with empty definition. It's not

    virtual someFunction() = 0; //pure virtual function, must be implemented by subcla
};
```

Requirements

StackInterface

Note You do not have a StackInterface.hpp. This file acts as a contract that any subclass must fulfill. The descriptions are to be used in your documentation for this class and the Stack class.

- virtual ~StackInterface() {};
- virtual bool isEmpty() const = 0;
  - returns true if the stack is empty, false otherwise
- virtual void push(const T value) = 0;
  - Entry added to top of stack
- virtual void pop() throw(PreconditionViolationException) = 0;
  - assumes the stack is not empty

        - top of the stack is removed
        - throws PreconditionViolationException when a pop is attempted on an empty stack. Does not return a value in this case.

- virtual T peek() const throw(PreconditionViolationException) = 0;
    - assumes the stack is not empty
    - returns the value at the top of the stack
    - throws a PreconditionViolationException is attempted on an empty stack. Does not return a value in this case.

- virtual int size() const = 0;
    - returns the size of the stack

- virtual std::vector<T> toVector const = 0;
    - Used for testing, not a standard Stack method
    - Example: If you push 5, push 10, and push 15 note that the the vector will have 15 as it's first value. In other words, the front of the vector represents the top of the stack

Stack

- Implements all methods from the StackInterface class.
- You do not have to keep your methods virtual in this class, but it would not do any harm
- Put the precondition, postcondition, return, and throws comments in this class also
- Use your Node class from the linked list we made as the building blocks of the stack, but you'll need to make a templated version

- ~Stack()
    - Delete all elements in the stack

Any method that throws an exception needs to pass a meaningful message to into the constructor of the exception object.

- Use the following messages
    - "Pop attempted on an empty stack"
    - "Peek attempted on an empty stack"

PreconditionViolationException

Ding Dong Who ordered the absurdly long class name? Here you go!

This class inherits from std::runtime_error. This will be an amazing brief class definition, but will prove useful. Since we're inheriting from runtime_error which implemented the std::exception class, we can catch basic exceptions:

```
try
{
  something that could throw a PreconditionViolationException
}
catch(std::exception& e)
{
   std::cout << e.what(); //print what happened
}
```

- PreconditionViolationException(const char* message)
  - calls the constructor of std::runtime_error, passing it the message

main.cpp

Create a program that does the following:

- Obtain an integer from the user
- Print the reverse of that number
- Tell the user if the number is palindrome
- Repeat until the user wants to quit

Example:

```
Input a number: 12345
12345 backwards is 54321
12345 is not a palindrome

Do you want to quit (y/n)?: n

Input a number: -32123
-32123 backwards is -32123
-32123 is a palindrome

Do you want to quit (y/n)?: y

Thanks! Exiting...
```

Rubric

- 60pts Tests
  - See provided test class
- 25pts Palindrome
  - 10pts Detects positive palindromes and zero as a palindrome
  - 10pts Detects negative palindromes
  - 5pts User interface is stable against bad input
- 10pts Memory Management:
  - There should be no memory leaks and a equal number of allocations and deallocations. Use valgrind ./YourProgramName to verify
  - Any memory leaks or unequal number of allocations to deallocations (barring the bug mentioned in previous labs) will result in a loss of all 10 points
- 5pts Documentation
  - Comments: Pre conditions, Post conditions, Return descriptions in header files. Not require for hpp/cpp files
  - Formatting: Rational use of white space and indentation. Header and implementation files easily readable

Emailing Your Submission

Once you have created the tarball with your submission files, email it to your TA. The email subject line **must** look like "[EECS 268] *SubmissionName*":

> [EECS 268] Lab 0#

Note that the subject should be *exactly* like the line above. Do not leave out any of the spaces, or the bracket characters ("[" and "]"). In the body of your email, include your name and student ID.
