CS 291/242: Object-Oriented Design and Programming

Programming Assignment 3
Due Monday, September 29th, 2003

Unbounded Stack

A limitation of your bounded Stack implementation of the ADT Stack is that stacks cannot grow beyond their initial size. Your next assignment will be to write an “unbounded” stack using dynamic memory and a linked list. Note that this change only affects the stack representation, but does not affect the stack interface. Your task is to reimplement the methods that operate upon objects of class `Stack`:

```
/* -*- C++ -*- */
#include <stdlib.h>

// Forward declaration (use the "Cheshire Cat" approach
// for information hiding).
template <class T> class Node;

template <class T>
class Stack
  // = TITLE
  //     Implement a generic LIFO abstract data type.
  //
  // = DESCRIPTION
  //     This implementation of a Stack uses a linked list.
{
public:

  typedef T TYPE;
  // C++ trait.

  // = Initialization, assignment, and termination methods.

  Stack (size_t size_hint);
  // Initialize a new stack so that it is empty.  The
  // <size_hint> can be ignored for now.

  Stack (const Stack<T> &s);
  // The copy constructor (performs initialization).

  void operator= (const Stack<T> &s);
  // Assignment operator (performs assignment).

 ~Stack (void);
  // Perform actions needed when stack goes out of scope.

  // = Classic Stack operations.

  void push (const T &new_item);
  // Place a new item on top of the stack. Does not check if the
  // stack is full.

  void pop (T &item);
  // Remove and return the top stack item. Does not check if stack
  // is full.

  void top (T &item) const;
  // Return top stack item without removing it. Does not check if
  // stack is empty.

  // = Check boundary conditions for Stack operations.

  int is_empty (void) const;
  // Returns 1 if the stack is empty, otherwise returns 0.

  int is_full (void) const;
  // Returns 1 if the stack is full, otherwise returns 0.

  int operator == (const Stack<T> &s) const;
  // Checks for Stack equality.

  int operator != (const Stack<T> &s) const;
  // Checks for Stack inequality.

private:
  Node<T> *head_;
  // Head of the linked list of <Nodes>.
};
```

Note that `push()`, `pop()`, and `top()` do *not* explicitly check whether the stack is empty or full. It is therefore necessary to call `is_empty()` or `is_full()` before adding, removing, or viewing a stack element. It is trickier than you think to get this right!

The following `Node` class defines operations on a node in the linked list. Since your solution uses the "Cheshire Cat" approach to information hiding, you can actually put this implementation into your Stack.C file. Therefore, clients of this class won't have access to the implementation at all!

```
template <class T>
class Node
{
  // = TITLE
  //     Implement a generic <Node>.
  //
  // = DESCRIPTION
  //     <Node>s can be chained together via their <next_> field.
public:
  Node (const T &val, Node<T> *next = 0);
  // Create and initialize a node to a certain value, assigning
  // <next> to <next_>.

  Node (void);
  // Just create the node, no initializing.

  Node (const Node<T> &from);
  // The copy constructor (performs initialization).

  Node<T> *next (void) const;
  // Get the pointer to the next node.
  void next (Node<T> *new_next);
  // Set the pointer to the next node.

  void value (T &a) const;
  // Get the current value at this node.

  void value (const T &a);
  // Set the current value of this node.

private:
  T value_;
  // Value at this node.

  Node<T> *next_;
  // Pointer to the next node.
};
```

For this assignment, please use the same test driver as before, and just modify the `Stack.h` and `Stack.cpp` implementation to use dynamically allocated memory and a linked list. You can get the "shells" for the program from `www.cs.wustl.edu/~schmidt/cs291/assignment4`. The Makefile, `main.cpp`, and `Stack.h` files are written for you. All you need to do is edit the `Stack.cpp` and `Stack.i` files to add the methods that implement the linked list version of the Stack ADT.

Extra Capabilities for Grad Students

If you are signed up for this course as CS 291 please add the following extensions to the original assignment:

1. *Exception handling* – Add exception handling to the `Stack` class to decouple error handling from normal processing. In particular, throw the `Overflow` exception if an attempt is made to `push()` on a full stack and the `Underflow` exception occurs if an attempt is made to `pop()` from an empty stack. Moreover, make sure that the stack implementation behaves gracefully if memory is exhausted.