CS302 - Data Structures *using C++*

Topic: Implementation of the ADT Stack

Kostas Alexis

AUTONOMOUS ROBOTS LAB

Array-based Implementation

```
/** ADT Stack: Array-based implementation.
@file ArrayStack.h */
#ifndef ARRAY_STACK_
#define ARRAY_STACK_
#include "StackInterface.h"

template<class ItemType>
class ArrayStack : public StackInterface<ItemType>
{
     private:
          static const int DEFAULT_CAPACITY = maximum-size-of-stack;
          ItemType items[DEFAULT_CAPACITY]; // Array of stack items
          int top;                          // Index to top of stack
public:
     ArrayStack();                          // Default constructor
     bool isEmpty() const;
     bool push(const ItemType& newEntry);
     bool pop();
     ItemType peek() const;
}; // end ArrayStack

#include "ArrayStack.cpp"
#endif
```

AUTONOMOUS ROBOTS LAB

Array-based Implementation

```cpp
/** @file ArrayStack.cpp */
#include <cassert>
#include "ArrayStack.h"

template<class ItemType>
ArrayStack<ItemType>::ArrayStack() : top(-1)
{
} // end default constructor

// Copy constructor and destructor are supplied by the compiler

template<class ItemType>
bool ArrayStack<ItemType>::isEmpty() const
{
      return top < 0;
} // end isEmpty

template<class ItemType>
bool ArrayStack<ItemType>::push(const ItemType& newEntry)
{
      bool result = false;
      if (top < DEFAULT_CAPACITY - 1) //Does stack have room for newEntry?
      {
            top++
            items[top] = newEntry;
            result = true;
      } // end if

      return result;
} // end push
template<class ItemType>
bool ArrayStack<ItemType>::pop()
{
      bool result = false;
      if (!isEmpty())
      {
            top--;
            result = true;
      } // end if
      return result;
} // end pop

template<class ItemType>
ItemType ArrayStack<ItemType>::peek() const
{
      assert (!isEmpty()); // Enforce precondition during debugging
      // Stack is not empty; return top
      return items[top];
} // end peek
// end of implementation file
```

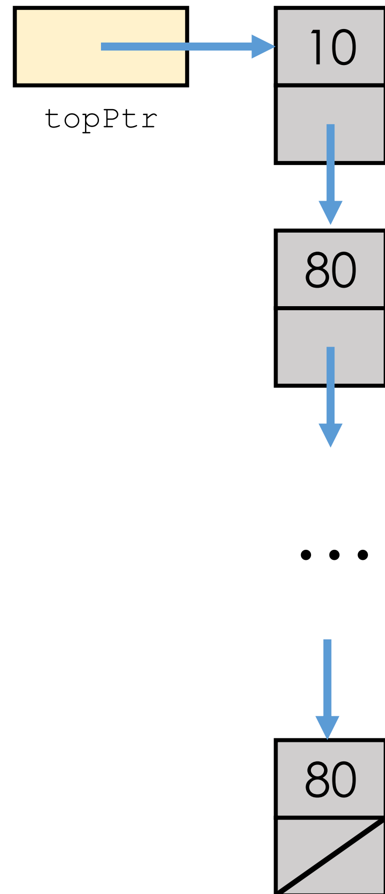
AUTONOMOUS ROBOTS LAB

Array-based Implementation

- Protecting the ADT's interface
  - Implement stack as a class
  - Declaring items and top as private
- Note
  - `push` receives `newEntry` as constant reference argument
  - `push` uses `newEntry` as an alias – no copy made

AUTONOMOUS ROBOTS LAB

Link-based Implementation



Link-based Implementation

```
/** @file LinkedStack.cpp */
#include <cassert>                // For assert
#include “LinkedInterface.h”      // Header file

template<class ItemType>
LinkedStack<ItemType>::LinkedStack() : topPtr(nullptr)
{
} // end default constructor

template<class ItemType>
LinkedStack<ItemType>::LinkedStack(const LinkedStack<ItemType>& aStack)
{
      // Point to nodes in original chain
      Node<ItemType>* origChainPtr = aStack.topPtr;
      if (origChainPtr == nullptr)
            topPtr = nullptr;     // Original stack is empty
      else
      {
            // Copy first node
            topPtr = new Node<ItemType();
            topPtr->setItem(origChainPtr->getItem());
            // Point to first node in new chain
            Node<ItemType>* newChainPtr = topPtr;
            // Advance original-chain pointer
            origChainPtr = origChainPtr->getNext();
```

```
            while (origChainPtr != nullptr)
            {
                  // Get next item from original chain
                  ItemType nextItem = origChainPtr->getItem();
                  // Create a new node containing next item
                  Node<ItemTYpe>* newNodePtr = new
                  Node<ItemType>(nextItem);
                  // Link new node to end of new chain
                  newChainPtr->setNext(newNodePtr);
                  // Advance pointer to new last node
                  newChainPtr = newChainPtr->getNext();
                  // Advance original-chain pointer
                  origChainPtr = origChainPtr->getNext();
            } // end while
      } // end if
} // end copy constructor
```

AUTONOMOUS ROBOTS LAB
N

Link-based Implementation

```
template<class ItemType>
LinkedStack<ItemType>::~LinkedStack()
{
	// Pop until stack is empty
	while (!isEmpty())
		pop;
}

template<class ItemType>
bool LinkedStack<ItemType>::push(const ItemType& newItem)
{
	Node<ItemType>* newNodePtr = new Node<ItemType>(newItem, topPtr);
	topPtr = newNodePtr;
	newNodePtr = nullptr;
	return true;
} // end push

template<class ItemType>
bool LinkedStack<ItemType>::pop()
{
	bool result = false;
	if (!isEmpty())
	{
		// Stack is not empty; delete top
		Node<ItemType>* nodeToDeletePtr = topPtr;
		topPtr = topPtr->getNext();
		// Return deleted node to system
		nodeToDeletePtr->setNext(nullptr);
		delete nodeToDeletePtr;
		nodeToDeletePtr = nullptr;
		result = true;
	} // end if
```

```
	return result;
} // end pop

template<class ItemType>
ItemType LinkedStack<ItemType>::peek() const
{
	assert(!isEmpty()); // Enforce precondition @ debugging

	// Stack is not empty; return top
	return topPtr->getItem();
} // end peek

template<class ItemType>
bool LinkedStack<ItemType>::isEmpty() const
{
	return topPtr == nullptr;
} // end isEmpty
// end of implementation file
```

AUTONOMOUS ROBOTS LAB

Implementations using Exceptions

- Method peek does not expect client to look at top of an empty stack
  - assert statement merely issues error message, and halts execution
- Consider having peek throw an exception
  - Listings follow on next slides

AUTONOMOUS ROBOTS LAB

Implementations using Exceptions

```
/** @file PrecondViolatedExcept.h */
#ifndef PRECOND_VIOLATED_EXCEPT_
#define PRECOND_VIOLATED_EXCEPT_

#include <stdexcept>
#include <string>

class PrecondViolatedExcept: public std::logic_error
{
public:
     PrecondViolatedExcept(const std::string& message =“”);
}; // end PRecondViolatedExcept

#endif
```

Implementations using Exceptions

```
/** @file PrecondViolatedExcept.cpp */
#include “PrecondViolatedExcept.h”

PrecondViolatedExcept::PrecondViolatedExcept(const std::string& message)
: std::logic_error(“Precondition Violated Exception: ” + message)
{
} // end constructor
```

Thank you