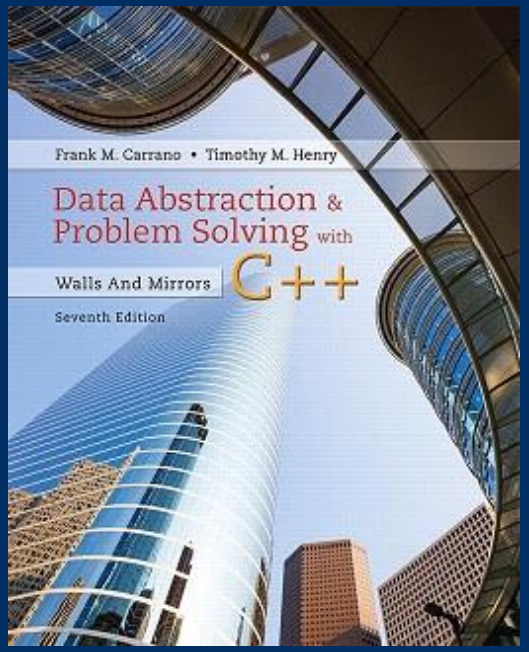
Chapter 7
Implementations of the ADT Stack

CS 302 - Data Structures

M. Abdullah Canbaz



Modified from authors' slides

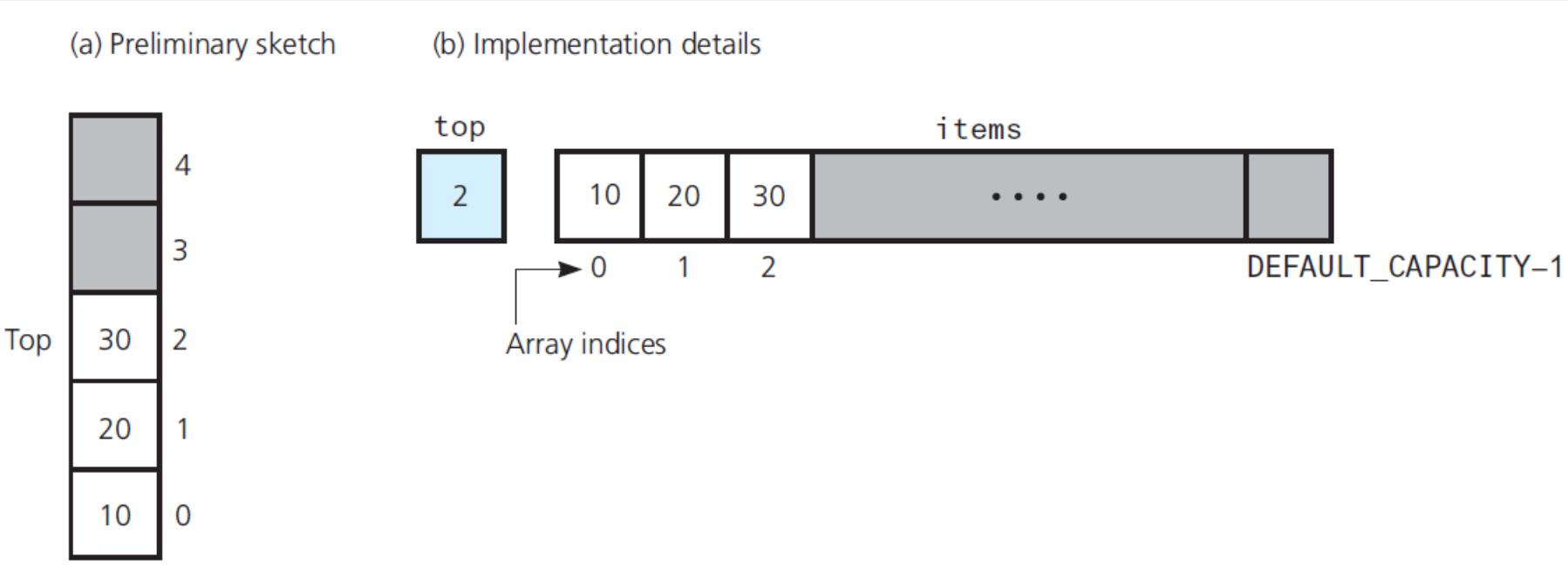
Reminders

- TAs
  - Shehryar Khattak,
    Email: *shehryar [at] nevada {dot} unr {dot} edu*,
    Office Hours: Friday, 11:00 am - 1:00 pm at ARF 116
  - Athanasia Katsila,
    Email: *akatsila [at] nevada {dot} unr {dot} edu*,
    Office Hours: Thursdays, 10:30 am - 12:30 pm at SEM 211

- Quiz 3 due tonight 11:59pm

An Array-Based Implementation

- Using an array to store a stack's entries



An Array-Based Implementation

- The header file for an array-based stack

```
1  /** ADT stack: Array-based implementation.
2   @file ArrayStack.h */
3
4  #ifndef ARRAY_STACK_
5  #define ARRAY_STACK_
6
7  #include "StackInterface.h"
8
9  template<class ItemType>
10 class ArrayStack : public StackInterface<ItemType>
11 {
12 private:
13    static const int DEFAULT_CAPACITY = maximum-size-of-stack;
14    ItemType items[DEFAULT_CAPACITY]; // Array of stack items
15    int      top;                     // Index to top of stack
```

An Array-Based Implementation

- The header file for an array-based stack

```
16  public:
17     ArrayStack();                          // Default constructor
18     bool isEmpty() const;
19     bool push(const ItemType& newEntry);
20     bool pop();
21     ItemType peek() const;
22  }; // end ArrayStack
23
24  #include "ArrayStack.cpp"
25  #endif
```

An Array-Based Implementation

```
1  /** @file ArrayStack.cpp */
2
3  #include <cassert>              // For assert
4  #include "ArrayStack.h"         // Header file
5
6  template<class ItemType>
7  ArrayStack<ItemType>::ArrayStack() : top(-1)
8  {
9  }  // end default constructor
10
11 // Copy constructor and destructor are supplied by the compiler
12
13 template<class ItemType>
14 bool ArrayStack<ItemType>::isEmpty() const
15 {
16    return top < 0;
17 }  // end isEmpty
18
19 template<class ItemType>
20 bool ArrayStack<ItemType>::push(const ItemType& newEntry)
21 {
```

- The implementation file for an array-based stack

An Array-Based Implementation

```
20  bool ArrayStack<ItemType>::push(const ItemType& newEntry)
21  {
22     bool result = false;
23     if (top < DEFAULT_CAPACITY - 1) // Does stack have room for newEntry?
24     {
25        top++;
26        items[top] = newEntry;
27        result = true;
28     }  // end if
29
30     return result;
31  }  // end push
32  template<class ItemType>
33  bool ArrayStack<ItemType>::pop()
34  {
35     bool result = false;
36     if (!isEmpty())
37     {
38        top--;
39        result = true;
40     }  // end if
```

- The implementation file for an array-based stack

An Array-Based Implementation

```
40      } // end if
41
42      return result;
43   } // end pop
44
45   template<class ItemType>
46   ItemType ArrayStack<ItemType>::peek() const
47   {
48      assert (!isEmpty()); // Enforce precondition during debugging
49
50      // Stack is not empty; return top
51      return items[top];
52   } // end peek
53   // end of implementation file
```

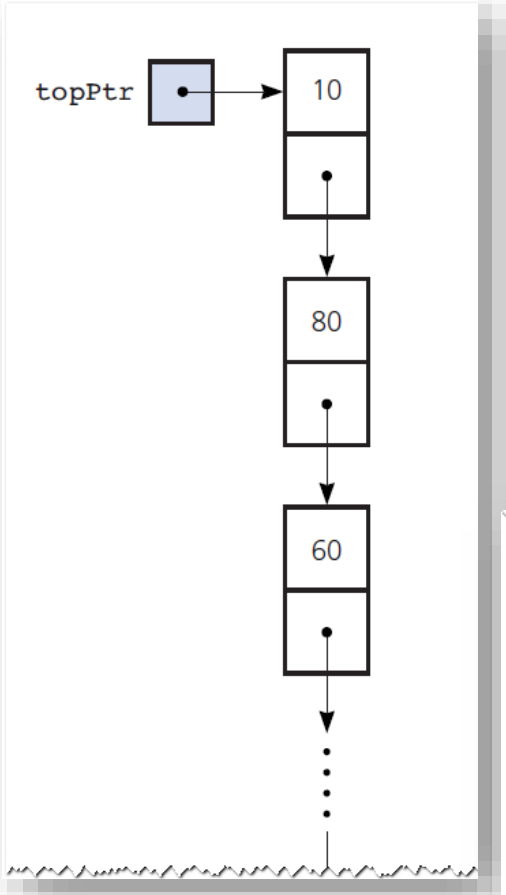
- The implementation file for an array-based stack

An Array-Based Implementation

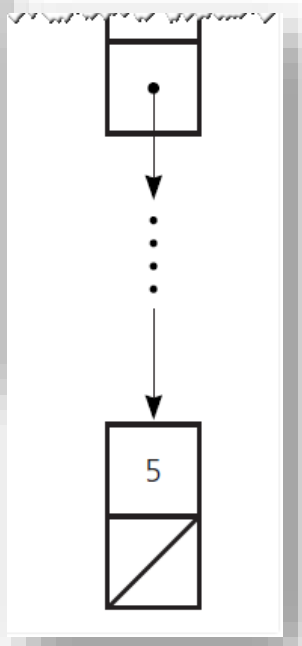
- Protecting the ADT's walls
  - Implement stack as a class
  - Declaring items and top as private

- Note
  - push receives newEntry as constant reference argument
  - push uses newEntry as an alias … no copy made

A Link-Based implementation



- A link-based implementation of a stack



A Link-Based implementation

- The header file for the class LinkedStack

```
1  /** ADT stack: Link-based implementation.
2   @file LinkedStack.h */
3
4  #ifndef LINKED_STACK_
5  #define LINKED_STACK_
6
7  #include "StackInterface.h"
8  #include "Node.h"
9
10 template<class ItemType>
11 class LinkedStack : public StackInterface<ItemType>
12 {
13 private:
14    Node<ItemType>* topPtr; // Pointer to first node in the chain;
15                            // this node contains the stack's top
16
```

A Link-Based implementation

- The header file for the class LinkedStack

```
15                                  // this node contains the stack's top
16
17  public:
18  // Constructors and destructor:
19     LinkedStack();                                    // Default constructor
20     LinkedStack(const LinkedStack<ItemType>& aStack); // Copy constructor
21     virtual ~LinkedStack();                           // Destructor
22
23  // Stack operations:
24     bool isEmpty() const;
25     bool push(const ItemType& newItem);
26     bool pop();
27     ItemType peek() const;
28  }; // end LinkedStack
29
30  #include "LinkedStack.cpp"
31  #endif
```

A Link-Based implementation

- The implementation file for the class LinkedStack

```
1  /** @file LinkedStack.cpp */
2  #include <cassert>              // For assert
3  #include "LinkedStack.h"        // Header file
4
5  template<class ItemType>
6  LinkedStack<ItemType>::LinkedStack() : topPtr(nullptr)
7  {
8  } // end default constructor
9
10 template<class ItemType>
11 LinkedStack<ItemType>::LinkedStack(const LinkedStack<ItemType>& aStack)
12 {
13    // Point to nodes in original chain
14    Node<ItemType>* origChainPtr = aStack.topPtr;
```

A Link-Based implementation

- The implementation file for the class LinkedStack

```
15      if (origChainPtr == nullptr)
16         topPtr = nullptr;              // Original stack is empty
17      else
18      {
19         // Copy first node
20         topPtr = new Node<ItemType>();
21         topPtr->setItem(origChainPtr->getItem());
22
23         // Point to first node in new chain
24         Node<ItemType>* newChainPtr = topPtr;
25
26         // Advance original-chain pointer
27         origChainPtr = origChainPtr->getNext();
28
29         // Copy remaining nodes
30         while (origChainPtr != nullptr)
31         {
32            // Get next item from original chain
33            ItemType nextItem = origChainPtr->getItem();
34
```

A Link-Based implementation

- The implementation file for the class LinkedStack

```
34
35          // Create a new node containing the next item
36          Node<ItemType>* newNodePtr = new Node<ItemType>(nextItem);
37
38          // Link new node to end of new chain
39          newChainPtr->setNext(newNodePtr);
40
41          // Advance pointer to new last node
42          newChainPtr = newChainPtr->getNext();
43
44          // Advance original-chain pointer
45          origChainPtr = origChainPtr->getNext();
46       }  // end while
47       newChainPtr->setNext(nullptr); // Flag end of chain
48    }  // end if
49 }  // end copy constructor
50
51 template<class ItemType>
```

A Link-Based implementation

- The implementation file for the class LinkedStack

```
50
51  template<class ItemType>
52  LinkedStack<ItemType>::~LinkedStack()
53  {
54     // Pop until stack is empty
55     while (!isEmpty())
56        pop();
57  }  // end destructor
58
59  template<class ItemType>
60  bool LinkedStack<ItemType>::push(const ItemType& newItem)
61  {
62     Node<ItemType>* newNodePtr = new Node<ItemType>(newItem, topPtr);
63     topPtr = newNodePtr;
64     newNodePtr = nullptr;
65     return true;
66  }  // end push
67
```

A Link-Based implementation

- The implementation file for the class LinkedStack

```
68  template<class ItemType>
69  bool LinkedStack<ItemType>::pop()
70  {
71     bool result = false;
72     if (!isEmpty())
73     {
74        // Stack is not empty; delete top
75        Node<ItemType>* nodeToDeletePtr = topPtr;
76        topPtr = topPtr->getNext();
77
78        // Return deleted node to system
79        nodeToDeletePtr->setNext(nullptr);
80        delete nodeToDeletePtr;
81        nodeToDeletePtr = nullptr;
82
83        result = true;
84     }  // end if
85
```

A Link-Based implementation

- The implementation file for the class LinkedStack

```
86      return result;
87   }  // end pop
88
89   template<class ItemType>
90   ItemType LinkedStack<ItemType>::peek() const
91   {
92      assert(!isEmpty()); // Enforce precondition during debugging
93
94      // Stack is not empty; return top
95      return topPtr->getItem();
96   }  // end peek
97
98   template<class ItemType>
99   bool LinkedStack<ItemType>::isEmpty() const
100  {
101     return topPtr == nullptr;
102  }  // end isEmpty
103  // end of implementation file
```

Implementations That Use 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

Implementations That Use Exceptions

- The header file for the class PrecondViolatedExcep

```
1  /** @file PrecondViolatedExcept.h */
2  #ifndef PRECOND_VIOLATED_EXCEPT_
3  #define PRECOND_VIOLATED_EXCEPT_
4
5  #include <stdexcept>
6  #include <string>
7
8  class PrecondViolatedExcept: public std::logic_error
9  {
10 public:
11    PrecondViolatedExcept(const std::string& message = "");
12 }; // end PrecondViolatedExcept
13
14 #endif
```

Implementations That Use Exceptions

```
1  /** @file PrecondViolatedExcept.cpp */
2  #include "PrecondViolatedExcept.h"
3
4  PrecondViolatedExcept::PrecondViolatedExcept(const std::string& message)
5      : std::logic_error("Precondition Violated Exception: " + message)
6  {
7  }  // end constructor
```

- Implementation file for the class PrecondViolatedExcep

Out of the Box

Remove Duplicates from Sorted Array{easy}

https://leetcode.com/problems/remove-duplicates-from-sorted-array/description/

Multiply Strings {medium}

https://leetcode.com/problems/multiply-strings/description/

Word Search {medium}

https://leetcode.com/problems/word-search/description/