Chapter 5

ADTs Stack and Queue

Outline

• Stack
  – Array-based Implementation
  – Linked Implementation
• Queue
  – Array-based Implementation
  – Linked Implementation
• Comparison
Stacks of Coins and Bills

Stacks of Boxes and Books

TOP OF THE STACK

TOP OF THE STACK
Stacks

• Logical level
  • What do these composite objects all have in common?

Stacks

• Stack
  • An abstract data type in which elements are added and removed from only one end.
  • A “last in, first out” (LIFO) structure.
Stacks

• Logical level
• What operations would be appropriate for a stack?

Stacks

• Transformers
  – Push
  – Pop
• Observers
  – IsEmpty
  – IsFull
  – Top

What about an iterator?
Stacks

• **Application Level**
• *For what type of problems would stacks be useful?*
• **LIFO**: A stack is great for reversing data.
  – Operating system function calls
  – Finding palindromes
  – Expression evaluation & syntax parsing

Stack Applications

• Operating system function calls

```c
void DrawSquare(int x, int y, int edge)
{
    DrawLine(x, y, edge, HORIZONTAL);
    DrawLine(x, y, edge, VERTICAL);
    DrawLine(x+edge, y, edge, VERTICAL);
    DrawLine(x, y+edge, edge, HORIZONTAL);
}
int main()
{
    DrawSquare(1,2,3);
    return 0;
}
```
Stack Applications

• Finding palindromes (Lab!)

Stack Applications

• Help Converting Decimal to Binary (pseudocode)
  1) Read (number)
  2) Loop (number > 0)
     1) digit = number modulo 2
     2) print (digit)
     3) number = number / 2

  // from Data Structures by Gilbert and Forouzan

• Problem: The binary numbers are printed backwards.
  – 19 becomes 11001 instead of 10011

• Solution: push each binary number onto the stack and pop the digit out of the stack and print it at the end.
Stacks

class StackType
{
public:
    StackType();
    ~StackType();
    bool IsEmpty() const;
    bool IsFull() const;
    void Push(ItemType item);
    void Pop();
    ItemType Top() const;

• Implementation Level
• Array-based Implementation
• Linked Structure
Array-Based Implementation

private:
    int top;
    ItemType items[MAX_ITEMS];
};

stack        [0] [1] [2] .... [M.]
.items
.top

Array-Based Implementation

Give a series of operations that could produce this situation
Array-Based Implementation

To what do we initialize data member top? 
0 or -1

When push, do we increment or store first?
0: store and increment
-1: increment and store

Which is better?

- Think about member function Top()
  StackType::StackType()
  { top = -1; }

Array-Based Implementation

• Before we code, we must consider error conditions
• Stack overflow
  – The condition that results from trying to push an element on to a full stack
• Stack underflow
  – The condition that results from trying to pop an empty stack
Array-Based Implementation

//pre: Stack has been initialized.
//post: function value = (stack is empty)
bool StackType::IsEmpty() const
{
    return (top == -1);
}

//pre: stack has been initialized.
//post: function value = (stack is full)
bool StackType::IsFull() const
{
    return (top == MAX_ITEMS);
}

What does const mean?

Array-Based Implementation

//pre: stack has been initialized and is not full
//post: newItem is at the top of the stack.
bool StackType::Push(ItemType newItem)
{
    top++;
    items[top] = newItem;
}
Array-Based Implementation

//pre: stack has been initialized.
//post: if stack is full, throw an exception;
//Else newItem is at the top of the stack.
bool StackType::Push(ItemType newItem)
{
    if (IsFull())
        throw FullStack();
    top++;
    items[top] = newItem;
}

What is FullStack()?

Array-Based Implementation

//pre: stack has been initialized and is not empty.
//post: top element has been removed from stack.
void StackType::Pop()
{
    top--;
}
//pre: stack has been initialized and is not empty.
//post: A copy of the top element is returned.
ItemType StackType::Top() const
{
    return (items[top]);
}
Array-Based Implementation

```cpp
// pre: stack has been initialized.
// post: if stack is empty, throw an exception; else top element
// has been removed from stack.
void StackType::Pop()
{
    if (IsEmpty()) throw EmptyStack();
    top--;
}
```

```cpp
// pre: stack has been initialized.
// post: if stack is empty, throw an exception; else a copy of the
// top element is returned.
ItemType StackType::Top() const
{
    if (IsEmpty()) throw EmptyStack();
    return (items[top]);
}
```

What is `EmptyStack`?

Array-Based Implementation

*Which functions have to be changed if we dynamically allocate the array `items`?*

*Do we need to add any new functions?*
Test Plan

• Clear-box strategy to check each operation.
  – Push() & Pop() while it is empty or full.

Linked Implementation

• The logical level (public part of the class declaration) stays the same;

```cpp
class StackType
{
public:
    StackType();
    ~StackType();
    bool IsEmpty() const;
    bool IsFull() const;
    void Push(ItemType item);
    void Pop();
    ItemType Top() const;
};
```
Linked Implementation

The implementation level (private part of the class declaration) changes

```cpp
private:
    NodeType* topPtr;
};
```

Can we “borrow” code from `UnsortedType` for `Push` and `Pop`?

```cpp
//pre: the stack is not full
//post: the new item is added on top of the stack
void StackType::Push(ItemType newItem) {
    NodeType* location;
    location = new NodeType;
    location->info = newItem;
    location->next = topPtr;
    topPtr = location;
}
```
Linked Implementation

// pre: the stack is not empty
// post: the top item is removed from the top of the stack
void StackType::Pop()
{
    NodeType* tempPtr;
    tempPtr = topPtr;
    topPtr = topPtr->next;
    delete tempPtr;
}

Does this work for stacks of one item? More than one item?

Linked Implementation

More than one item
Linked Implementation

What about the constructor, destructor, and observer functions?

We can borrow all but **Top()** from class **UnsortedType List**

```cpp
//pre: the stack is not empty
//post: the item on the top of the stack is //returned.
ItemType StackType::Top()
{
    return topPtr->info;
}
```
Other Member Functions

- Constructor
- Destructor
  - Free all the node spaces.
- isEmpty
- isFull

Array vs. Linked Structure

- A serious drawback of array-based implementation: the size of a stack must be determined when a stack object is declared.
  - Size is not enough or
  - Space is wasted.
Big-O Comparison

<table>
<thead>
<tr>
<th>Time</th>
<th>Array-Based Implementation</th>
<th>Linked Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class constructor</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Class destructor</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>IsFull()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>IsEmpty()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Push()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Pop()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Reference

- Reproduced by permission of Jones and Bartlett Publishers International.