# Stacks

February 14, 2020

# Overview

- What are stacks?
- Where are stacks used?
- Stack behavior
- Integer stack example
- Stack class (Objects using composition)

# What are Stacks?



- A LIFO[1] structure.
- A stack is a pile of *things*:
  - A stack of dishes in a cafeteria/restaurant.
  - A stack of papers in an office.

[1] Last In, First Out

# Where are stacks used?



- A stack of activation records are maintained when a program is executed. [An activation record contains the data needed for each execution of a function.]
- Compilers frequently use a stack of symbols when parsing source code.
- Simulate recursion.
- Graphics — Transformation matrices.

# Where are stacks used?



- Calculators (RPN)
- Programming languages
  - Argument passing
  - PostScript
  - Forth
  - JVM (Java Virtual Machine)
  - more...[2]

[2] *Stack-oriented programming language*
`https://en.wikipedia.org/wiki/Stack-oriented_programming_language`

# Stack Behavior—Review



Standard operations:

- A new item can be added to the top of a stack, `s`, using the method `s.push( x )`.
- The item at the top of a stack, `s`, can be removed using the method `s.pop()`.
- The item at the top of a stack, `s`, can be viewed using the method `s.peek()`.

# Testing

"Testing leads to failure, and failure leads to understanding."
— Burt Rutan[3]

[3] https://twitter.com/codewisdom/status/1175026071194021890?s=1

# Test Code—Header



Be careful when naming files...

```
/*  testStack.cpp

    Stack test program

    Bruce M. Bolden
    June 16, 1998
 */

#include <iostream>

using namespace std;

#include "stack.h"
```

# Test Code: Push() testing



```
int main()
{
   Stack iStack;

   cout << "Pushing integers onto iStack" << endl;

   for( int i = 0 ; i < 5 ; i++ ) {
      iStack.Push(i);       // push items onto the stack
      cout << i << ' ';
   }
   cout << endl;

   cout << "\nContents of iStack" << endl;
   iStack.Print();          // output the stack contents
```

# Test Code: Pop() testing



```
   cout << endl << "Popping integers from iStack" << endl;

   while( !iStack.IsEmpty() )
      cout << iStack.Pop() << ' ';

   cout << endl;
   iStack.Print();           // output the stack contents

   if( iStack.IsEmpty() )
       cout << "\nThe stack is empty" << endl;
   else
       cout << "\nThe stack is not empty" << endl;

   return 0;
}
```

# Output from Sample Program



```
Pushing integers onto iStack
0 1 2 3 4

Contents of iStack
4
3
2
1
0

Popping integers from iStack
4 3 2 1 0

The stack is empty
```

# Best Code

The best line of code is the one you don't have to write.

— Mark Fenoglio[4]

[4] Instructor at The Big Nerd Ranch, `http://wilddogcow.tumblr.com/page/5`

# Stack Class: Overview



We will implement a stack class using an existing list class. This technique is called *composition*, since the stack is composed of a list.

**Benefit:** Little code needs to be written.

# Stack Class—Interface



```
/* stack.h  Stack class interface  */

#ifndef STACK_H
#define STACK_H  /*  file guards  */

#include "link.h"

class Stack {
public:
   Stack();
   ~Stack();

   void Push(int n);      // push item onto stack
   int Pop();             // remove item from stack
   int IsEmpty();         // is the stack empty?
   void Print();          // print the stack

private:
   LinkedList topPtr;     // pointer to list
};
#endif
```

# Stack Class—Definition



```
/*  stack.cpp --- Definition of Stack class member functions.  */

#include <iostream>
#include <assert.h>

#include "stack.h"


Stack::Stack()
{
}

Stack::~Stack()
{
   // delete topPtr;
   while( !IsEmpty() ) {
       int  n = topPtr.FirstNode();
       topPtr.DeleteNode( n );
   }
}
```

# Stack Class—Push() and Pop()



```
/*  Push and Pop
 */

void Stack::Push(int n)
{
   topPtr.AddNode( n );
}


int Stack::Pop()
{
   assert(!IsEmpty());

   int  n = topPtr.FirstNode();
   topPtr.DeleteNode( n );
   return n;
}
```

# Stack Class—IsEmpty() and Print()



```
int Stack::IsEmpty()
{
    int n = topPtr.Size();
    return (n == 0);
}


void Stack::Print()
{
    topPtr.Print();
}
```

# Linked List Class—Interface



```
/*  link.h --- interface for a linked list of integers class.  */

#ifndef LINK_H
#define LINK_H

#include <bool.h>
#include <iostream>

class LinkedList
{
private:

  struct node {
     int info;
     node * next;
  };
  typedef node * nodeptr;

  nodeptr head;

  int count;
```

# Linked List Class—public Interface



```
public:
        // Constructor
   LinkedList()
   {
      head = NULL;
      count = 0;
   }

       // Destructor
   ~LinkedList()
   {
      nodeptr p = head, n;

      while( p != NULL )
      {
         n = p;
         p = p->next;
         delete n;
      }
   }
```

# Linked List Class—public Interface



```
    // Add a node onto the front of the linked list.
   void AddNode(int x);

    // Delete the first node found with the value x, if one exists.
   void DeleteNode(int x);

    // Return the first node found in the list
   int FirstNode();

    // Output the values in the nodes, one integer per line.
   void Print();

    // Return true if there is a node with the value x.
   bool IsInList(int x);

    // Return a count of the number of nodes in the list.
   int Size();
};

#endif
```

# Linked List Class Definition—Header



```
/*  link.cpp

    Class for a sorted linked list of integers.

    Bruce M. Bolden                 September 19, 2005
 */

#include <bool.h>
#include <iostream>

#include "link.h"
```

# Linked List Class Definition—AddNode()



```
   //  Add an item to the FRONT of the list
void LinkedList::AddNode( int x )
{
    nodeptr n;

        //  allocate new node
    n = new node;
    n->info = x;
    count++;

    if( head == NULL ) {
        head = n;
        n->next = NULL;
    }
    else {
        nodeptr tmp = head;
        n->next = tmp;
        head = n;
    }
}
```

# Linked List Class Definition—DeleteNode()



```
void LinkedList::DeleteNode( int x )
{
    nodeptr prev, curr = head;

    while( curr != NULL && x > curr->info )
    {
        prev = curr;
        curr = curr->next;
    }

    if( x == curr->info )
    {
        if( curr == head )
            head = head->next;
        else
            prev->next = curr->next;

        delete curr;
        count--;
    }
}
```

# Linked List Class Definition—`FirstNode()`



```
int LinkedList::FirstNode()
{
    return head->info;
}
```

# Linked List Class Definition—Print()



```
void LinkedList::Print()
{
    nodeptr p = head;

    while( p != NULL )
    {
        cout << p->info << endl;
        p = p->next;
    }

}
```

# Linked List Class Definition—IsInList() and Size()



```
bool LinkedList::IsInList(int x)
{
    nodeptr p = head;

    while( p != NULL && x > p->info )
        p = p->next;

    return (x == p->info);
}


int LinkedList::Size()
{
    return count;
}
```