lecture11: Stacks

Sean Massung

Largely based on slides by Cinda Heeren
CS 225 UIUC

26th June, 2013
Announcements

- mp3.1 extra credit due Friday night (6/28)
- lab_quacks due Saturday night (6/29)
The Stack ADT

A Stack is a First-In, Last-Out (FILO) or Last-In, First Out (LIFO) structure. An analogy is a “stack” of papers or cards.

ADT functions include:

1. push
2. pop
3. peek
4. size
5. empty
What use is a Stack?

- Memory allocation
  - What you’ve been using for stack memory allocations
  - Confusingly, heap allocations do not use the heap data structure!
- Doing traversals
  - mp4, lab_graphs, possibly mp7
- Parsing
  - CS 421 (compilers), CS 498jhm (natural language processing)
- Undo histories
- Converting a number to binary (how?)
- Balancing parentheses (how?)
Printing a number in binary

```cpp
void printBinary(int n)
{
    Stack<bool> s;
    while(n > 0)
    {
        s.push(n % 2);
        n /= 2;
    }
    while(!s.empty())
    {
        cout << s.pop();
        cout << endl;
    }
}
```
Balancing parentheses with a stack

In order to prepare you for theory classes, this problem is left as an exercise to the student.
What choices do we have?

```cpp
/** @file stack.h */

template <class T>
class Stack
{
    public:
        Stack(); // + big 3
        void push(const T & elem);
        T pop();
        T peek() const;
        size_t size() const;
        bool empty() const;

    private:
        // what should go in here?

};
```
Linked list or array?

```cpp
/** @file stack.h */

template <class T>
class Stack
{

public:
    Stack(); // + big 3
    void push(const T & elem);
    T pop();
    T peek() const;
    size_t size() const;
    bool empty() const;

private:
    size_t _size;
    size_t _capacity;
    T* _data;
};
```

```cpp
/** @file stack.h */

template <class T>
class Stack
{

public:
    Stack(); // + big 3
    void push(const T & elem);
    T pop();
    T peek() const;
    size_t size() const;
    bool empty() const;

private:
    // Node class...
    Node* _head;
    size_t _size;
};
```
Wait, we made these already...

```cpp
/** @file stack.h */
#include "vector.h"

template <class T>
class Stack
{
    public:
        Stack(); // + big 3
        void push(const T & elem);
        T pop();
        T peek() const;
        size_t size() const;
        bool empty() const;

    private:
        Vector<T> _array;
};

/** @file stack.h */
#include "list.h"

template <class T>
class Stack
{
    public:
        Stack(); // + big 3
        void push(const T & elem);
        T pop();
        T peek() const;
        size_t size() const;
        bool empty() const;

    private:
        List<T> _list;
};
```
The array implementation

```cpp
/** @file stack.h */
#include "vector.h"

template <class T>
class Stack
{
    public:
        Stack(); // + big 3
        void push(const T & elem);
        T pop();
        T peek() const;
        size_t size() const;
        bool empty() const;
    
    private:
        Vector<T> _array;
};
```

```cpp
/** @file stack.cpp */

template <class T>
Stack<T>::Stack(): _array(Vector<T>());

template <class T>
void Stack<T>::push(const T & elem)
{
    _array.push_back(elem);
}

template <class T>
T Stack<T>::pop()
{
    return _array.pop_back();
}
```
Decisions for the linked list implementation

/** @file stack.h */
#include "list.h"

template <class T>
class Stack
{
    public:
        Stack(); // + big 3
        void push(const T & elem);
        T pop();
        T peek() const;
        size_t size() const;
        bool empty() const;
    private:
        List<T> _list;
};

/** @file stack.cpp */

template <class T>
Stack<T>::Stack(): _list(List<T>()){}

template <class T>
void Stack<T>::push(const T & elem)
{
    _list.insertFront(elem);
    // or...?
    _list.insertBack(elem);
}

template <class T>
T Stack<T>::pop()
{
    return _list.removeFront();
    // or...?
    return _list.removeBack();
}
Linked list intricacies

- When implementing a Stack with a linked list, does it matter where the “top” of the Stack is (head or tail)?
- Does it matter if the linked list is a SLL or a DLL?
- We are concerned with an efficient running time!
Another running time showdown!

<table>
<thead>
<tr>
<th>Function</th>
<th>Array</th>
<th>SLL (tail is top)</th>
<th>DLL (tail is top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>push</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>peek</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another running time showdown!
Which implementation do you think is better *empirically*? That is, in practice on actual hardware. We always need to consider real-life performance as well as theoretical performance!

By the way, you should explore the running times on your own when the top of the Stack is the head of the SLL and DLL. What changes?