CS 187: Programming with Data Structures (Spring 2010)

Lecture 7: Stacks

Rui Wang
Announcements

• My office hour today is canceled.
• Assignment 2 is due at 4pm Thursday this week.
• Assignment 2 should be submitted in SPARK.
• Discussion section tomorrow (about assignment 2)
Announcements

• Assignment 1 grade will be returned to you through email by Thursday.
  – Future assignment grades will be released in SPARK.

• Quizzes will be assigned in SPARK in the future.
• Solutions to the assignments, and the tester program we use for grading will be posted in SPARK.
Summary of Arrays

• Why not use Arrays for everything?
  – Unordered array takes $O(1)$ to insert, but $O(N)$ to search.
  – Ordered array takes $O(\log N)$ to search, but $O(N)$ to insert.
  – Both take $O(N)$ to delete.
Summary of Arrays

• Wouldn’t it be wonderful if we can do all of them in O(1)? Or if not, O(logN)?

• While this is not realistic, we will look at other data structures that provide a variety of different tradeoffs; and learn about when to use each of them.
Today

• Stacks
• Two simple applications of stacks
  – Reversing a word
  – Delimiter matching
Stacks – Heads up

• Programmer’s tool
  – Arrays are typically used as data storage structures in apps such as a database (e.g. personal records, inventories …)
  – In contrast, stacks are often used as programmer’s tool, and are not typically used for data storage.
Stacks – Heads up

• Restricted Access
  – In an array, you can access any element by using its index
  – In contrast, stacks only allow access of elements through a restricted manner. Also, you can only access one element at a time.
Stacks – A Familiar Example

• A can of tennis balls
  – Imagine the entire can represents an array, and each ball is an element.
  – It only allows access to one element at a time: the last element.
  – If you remove the last element, you can then access the next-to-last element.
  – There is no way to directly access the element at the bottom.
Stacks – Another Example

• A dynamic list of tasks you perform everyday:
  – Imagine you start your day by working on task A.
  – At any time you may be interrupted by a co-worker asking you for temporary help on task B.
  – While you work on B, someone may interrupt you again for help on task C.
Stacks – Another Example

• A dynamic list of tasks you perform everyday:
  – Imagine you start your day by working on task A.
  – At any time you may be interrupted by a co-worker asking you for temporary help on task B.
  – While you work on B, someone may interrupt you again for help on task C.
  – When you are done with C, you will resume working on B.
  – Then you go back to work on A.
  – Think about the sequence of tasks you perform.
Stacks – Any other example?
Stacks

- A stack stores an array of elements but with only two simple operations:
  - **Push**: add an element to the top of the stack
  - **Pop**: remove the top element of the stack.

- Pop always removes the last element that’s added to the stack. This is called **LIFO** (Last-In-First-Out).
Stacks

• Note that an element cannot be inserted to the middle of the array.
• The only way you modify the elements is through the push and pop operations.
• This capability turns out to be very useful in many programming situations.
• In a computer, the stack an essential data structure for handling program calls and returns.
Stacks

• Workshop Applet
• 3 simple methods:
  – Push
  – Pop
  – Peek (similar to pop but does not remove)

• Problems
  – What happens if you try to push when the stack is full
  – What happens if you try to pop when the stack is empty?
Stacks

• Code

• Note that the underlying implementation uses an array to store elements.

• However, the interface methods of Stack hide the implementation away, and do not allow you to arbitrarily modify the array: you can only push or pop to modify.

• Again, this is an example of Abstraction.
Stacks

• Note that for ++ (increment) operator:
  
  ```
  stackArray[++top] = j;
  ```

  is equivalent to:

  ```
  ++top;
  stackArray[top] = j;
  ```
• On the other hand:

\[
\text{stackArray[top++]} = j;
\]

is equivalent to:

\[
\text{stackArray[top]} = j;
\text{top++;}
\]

• The same applies to \( -- \) (decrement) operator.
Stacks

- Example:

  Stack s = new Stack(10);
  s.push(80);
  s.push(40);
  s.push(20);
  System.out.println(s.pop());
  System.out.println(s.pop());
  System.out.println(s.pop());
Stacks

• What about this:

Stack s = new Stack(10);
s.push(80);
s.push(40);
s.push(20);
System.out.println(s.pop());
System.out.println(s.peek());
System.out.println(s.pop());
Stacks

• Cost analysis?
Stacks

• Cost analysis
  – Neither push nor pop is dependent on the number of elements. Hence O(1).
Application 1 – Revering a word

• You type in a word, and the program outputs the work with letter in reverse order:
  program -> margorp

• A stack is naturally suitable for this:
  – Push each character one-by-one to a stack
  – Then pop the stack one-by-one

• Code
Application 2 – Delimiter Matching

• You want to make sure if the parentheses in an mathematical expression is balanced:
  \( (w \times (x + y) / z - (p / (r - q))) \)
Application 2 – Delimiter Matching

• You want to make sure if the parentheses in an mathematical expression is balanced:
  \((w \times (x + y) / z - (p / (r - q)))\)

• It may have several different types of delimiters: braces{}, brackets[], parentheses().
  – Each opening on the left delimiter must be matched by a closing (right) delimiter.
  – Left delimiters that occur later should be closed before those occurring earlier.
Application 2 – Delimiter Matching

• Examples:

c[d]    // correct
a{b[c]d}e // correct
a{b(c)d}e // not correct; ] doesn't match 

a[b{c}d]e} // not correct; nothing matches final }

a{b(c)    // not correct; nothing matches opening {
Application 2 – Delimiter Matching

• It’s easy to achieve matching using a stack:
  – Read characters from the string.
  – Whenever you see a left (opening) delimiter, push it to the stack.
  – Whenever you see a right (closing) delimiter, pops the opening delimiter from the stack and match.
  – If they don’t match, report error.
Application 2 – Delimiter Matching

• It’s easy to achieve matching using a stack:
  – Read characters from the string.
  – Whenever you see a left (opening) delimiter, push it to the stack.
  – Whenever you see a right (closing) delimiter, pops the opening delimiter from the stack and match.
  – If they don’t match, report error.
  – What happens if the stack is **empty** when you try to match a closing delimiter?
Application 2 – Delimiter Matching

• It’s easy to achieve matching using a stack:
  – Read characters from the string.
  – Whenever you see a left (opening) delimiter, push it to the stack.
  – Whenever you see a right (closing) delimiter, pops the opening delimiter from the stack and match.
  – If they don’t match, report error.
  – What happens if the stack is empty when you try to match a closing delimiter?
  – What happens if the stack is non-empty after all characters are read?
Application 2 – Delimiter Matching

• Example:
  \[ a \{ b ( c [ d ] e ) f } \]

• Code

• Why does this work?
  – Delimiters that are opened last must be closed first.
  – This conforms exactly with the LIFO property of the stack.