CS 187: Programming with Data Structures (Spring 2010)

Lecture 8: Stacks Applications

Rui Wang
Announcements

• Assignment 2 is due tomorrow (Friday) at 4pm
• TA office hours today are extended to 5pm.
  – Please go to office hours if you have assignment questions, or post them on the discussion group.
• In addition, one TA (Hannah) will be at EDLAB tomorrow (Friday) 2-3pm.
Announcements

• Quiz 2 has been assigned on SPARK. Please complete it by the discussion section next week.

• Quiz 1 paper-based grades can be picked up during TA office hours today.
Assignment 2

• Written section: Big-O Notation
• Read slides please.
• Some additional examples:
  • $T(N) = N^3 + N^2 + N + 1000$
  • $T(N) = 4N + 10^{10}$
  • $T(N) = 2^{\log_2 N} + N \log_2 N + 1500$
Assignment 2

• Program analysis.
• Two steps:
  1) write down how many times a loop statement is executed \(\rightarrow\) This is the cost function.
  2) find the Big-O notation.
Assignment 2

• Programming part: number guessing.
  1. Think of a 4-digit number
  2. The program makes a guess
  3. You tell it how many digits match (not which ones)
  4. The program continues to guess until the correct number is found.

• Example.
Assignment 2

• Two important points:
  1. Use elimination to reduce the pool of guesses
  2. How to compute the number of matches between two numbers?
Assignment 2

• How do I compare the digits?
• There are two ways.
  1. Use integer division/modulo
     What does \( x \% 10 \) give you?
     How about \( (x / 10) \% 10 \)?
     and \( (x / 100) \% 10 \)?
Assignment 2

• How do I compare the digits?
• There are two ways.

2. Convert the integer to string

```java
String s = Integer.toString(x);
s.charAt(0);
s.charAt(1);
...
```
Assignment 2

- Be careful when removing elements from ArrayList.
- Instead of removing elements, you can create a new ArrayList and copy valid guesses.
Summary of Stacks

- A stack support two simple operations: push and pop.
- The last element push to stack is popped first.
- LIFO (last-in-first-out).
- Stacks are very useful as programming tools.
Today

• More applications of the stack
  – Evaluating postfix expressions
  – Program stacks
  – Simulating nested loops
App 1 – Postfix Notation

• Task: evaluate arithmetic expressions.
• Familiar arithmetic expressions:
  2+3
  2*(3+4)
  ...

• The **operators** are placed between two **operands**. This is called **infix** notation.
App 1 – Postfix Notation

• For computers to parse the expressions, it’s more convenient to represent expressions in postfix notation.

• Here operators are placed after operands.

  2 3 +
  A B /

• Also known as Reverse Polish Notation (RPN)
App 1 – Postfix Notation

- More complex examples:

<table>
<thead>
<tr>
<th>Infix</th>
<th>Postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>A+B–C</td>
<td>AB+C–</td>
</tr>
<tr>
<td>A*B/C</td>
<td>AB*C/</td>
</tr>
<tr>
<td>A+B*C</td>
<td>ABC*+</td>
</tr>
<tr>
<td>A*B+C</td>
<td>AB*C+</td>
</tr>
<tr>
<td>A*(B+C)</td>
<td>ABC+*</td>
</tr>
<tr>
<td>A<em>B+C</em>D</td>
<td>AB<em>CD</em>+</td>
</tr>
<tr>
<td>(A+B)*(C–D)</td>
<td>AB+CD–*</td>
</tr>
<tr>
<td>((A+B)*C)–D</td>
<td>AB+C*D–</td>
</tr>
<tr>
<td>A+B*(C–D/(E+F))</td>
<td>ABCDEF+/-*+</td>
</tr>
</tbody>
</table>
App 1 – Postfix Notation

• Playing with RPN calculator is fun!
• We can convert any infix expression to a postfix expression.
  – But we will skip this part as it’s a bit involved.
• Instead, we will look at how to evaluate a postfix expression using a simple program based on the stack data structure.
App 1 – Postfix Notation

• An example:
  \[3 \ 4 \ 5 \ + \ * \ 6 \ 1 \ 2 \ + \ / \ -\]

• This is equivalent to the infix expression:
  \[3 \times (4+5) - 6 / (1+2)\]
App 1 – Postfix Notation

• An example:

$$3 \ 4 \ 5 \ + \ * \ 6 \ 1 \ 2 \ + \ / \ -$$

How do we evaluate this postfix expression?
App 1 – Postfix Notation

• Now we need to write a program to achieve the evaluation.

• Note that whenever you encounter an operator, you apply it to the last two operands you’ve seen.

• This suggests the use of a stack to store the operands.
App 1 – Postfix Notation

- Now we need to write a program to achieve the evaluation.

<table>
<thead>
<tr>
<th>Item Read from Postfix Expression</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operand</td>
<td>Push it onto the stack.</td>
</tr>
<tr>
<td>Operator</td>
<td>Pop the top two operands from the stack and apply the operator to them. Push the result.</td>
</tr>
</tbody>
</table>

- Let see how this algorithm works on:

\[ 3 \ 4 \ 5 \ + \ * \ 6 \ 1 \ 2 \ + \ / \ - \]
App 1 – Postfix Notation

• Code
• What happens if the stack is empty while you try to pop an element?
App 2 – Program Stacks

• Modern computer architecture typically uses stacks to handle function calls and returns.
• Function calls are executed in last-in-first-out fashion. Similar to the everyday task example.
• Stack memory
• Local variables in a method are allocated in stack memory
• These are called a stack frame.
App 2 – Program Stacks

• Example:

```java
method1() {
    int x = 5;
    int y = 6;
    ...
    method2(x);
    ...
}
```

```java
method2(int i) {
    int x = i;
    int y = i+1;
    ...
    method3(x, y);
    ...
}
```
App 3 – Simulate a Nested Loop

for (int i=0; i<n; i++) {
    ...
    for (int j=0; j<n; j++) {
        ...
        for (int k=0; k<n; k++) {
            ...
        }
    }
}
App 3 – Simulate a Nested Loop

• This is often the simplest (brute force) way of solving many search/optimization problems.
• One example is the n-queen problem.
App 3 – Simulate a Nested Loop

• If you know \( n \) ahead of time (say, \( n=8 \)), you can probably write an \( n \)-nested loop to solve the problem.

• What if \( n \) is a parameter?

• In assignment 3, you will be using the stack data structure to solve the \( n \)-queen problem for any arbitrary \( n \).