CSC207.01 2014S, Class 10: Debugging

Overview

- Preliminaries.
  - Admin.
  - Forthcoming work.
  - Questions on the HW.
  - Additional Questions.
- A bit about debugging.
- Lab.
- Reflection (maybe).

Preliminaries

Admin

- Missing: PS, AS14, SR (not SamR), SM
- How have the writeups been? It strikes me that they are useful, but I'd like to hear your opinions.
  - A huge pain, but definitely helpful. [x2]
  - Less than an hour in most cases, but "it's another thing"
- Note: Tonight's 8pm Rosenfield talk is cancelled due to weather.
- Extra credit:
  - Wednesday Extra, February 5 at 4:30 in 3821, AppDev
  - Convo Feb. 5 at noon in JRC 101: Pussy Riot and Putin.
  - CS Table Friday: NP Completeness
  - More?

Upcoming Work

- Reading for Wednesday: Exceptions in Java.
- No writeup today!
- Homework 3 due tomorrow night.

Questions on HW3

How should we cite help?

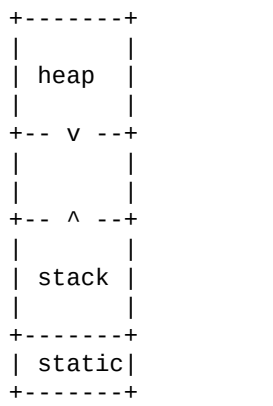
> At the top or bottom of your assignment: "My awesome mentor, Earnest, and my only slightly clueless colleague Sam, helped a lot with debugging on problem 3." "I also took info from http://www.stackoverflow.com/answers2silly232523412431/how-do-i-concatenate-strings-in-java.html"

Other Questions

What do you mean by "the stack"?

The area of memory used to keep track of local variables and other information for procedure calls. Most of the time, we think about your program using three kinds of memory: static memory (needed for the whole run of the program), the heap (used for things you allocated/free), and the stack (used for storing info on procedure calls).

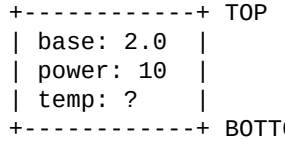
Note that I'm doing the following on the whiteboard, but I've recorded something similar on the EBoard so that there's a record.



For example, consider our recursive `expt` procedure.

```
public static double expt(double base, int power)
{
  if (power == 0)
    {
      return 1.0;
    } // if power == 0
  else if ((power % 2) == 0)
    {
      double temp = expt(base, power/2);
      return temp*temp;
    } // if the power is even
  else
    {
      double temp = expt(base, power-1);
      return base * temp;
    } // if the power is odd
} // expt(double, int)
```

Suppose we ask for 2.0^10 by calling `expt(2.0,10)`



That’s the even case, so we do a recursive call. We need to keep the current base and power, so we add another "stack frame" to the stack.

```
+------------+ TOP
| base: 2.0  |
| power: 5   |
| temp: ?    |
+------------+
| base: 2.0  |
| power: 10  |
| temp: ?    | // next line: return temp*temp
+------------+ BOTTOM
```

That’s the odd case. And it requires another recursive call.

```
+------------+ TOP
| base: 2.0  |
| power: 4   |
| temp: ?    |
+------------+
| base: 2.0  |
| power: 5   |
| temp: ?    | // next line: return base*temp
+------------+
| base: 2.0  |
| power: 10  |
| temp: ?    | // next line: return temp*temp
+------------+ BOTTOM
```

Doing the next few cases, you’ll see that we end up with

```
+------------+ TOP
| base: 2.0  |
| power: 0   |
| temp: ?    |
+------------+
| base: 2.0  |
| power: 1   |
| temp: ?    | // next line: return base*temp
+------------+
| base: 2.0  |
| power: 2   |
| temp: ?    | // next line: return temp*temp
+------------+
| base: 2.0  |
| power: 4   |
| temp: ?    | // next line: return temp*temp
+------------+
| base: 2.0  |
| power: 5   |
| temp: ?    | // next line: return base*temp
+------------+
| base: 2.0  |
| power: 10  |
| temp: ?    | // next line: return temp*temp
+------------+ BOTTOM
```

We then return from the topmost call.

```
+------------+ TOP
| base: 2.0  |
| power: 1   |
| temp: 1.0  | // next line: return base*temp
+------------+
| base: 2.0  |
| power: 2   |
| temp: ?    | // next line: return temp*temp
+------------+
| base: 2.0  |
| power: 4   |
| temp: ?    | // next line: return temp*temp
+------------+
| base: 2.0  |
| power: 5   |
| temp: ?    | // next line: return base*temp
+------------+
| base: 2.0  |
| power: 10  |
| temp: ?    | // next line: return temp*temp
+------------+ BOTTOM
```

And the next

```
+------------+ TOP
| base: 2.0  |
| power: 2   |
| temp: 2.0  | // next line: return temp*temp
+------------+
| base: 2.0  |
| power: 4   |
| temp: ?    | // next line: return temp*temp
+------------+
| base: 2.0  |
| power: 5   |
| temp: ?    | // next line: return base*temp
+------------+
| base: 2.0  |
| power: 10  |
| temp: ?    | // next line: return temp*temp
+------------+ BOTTOM
```

And the next

```
+------------+ TOP
| base: 2.0  |
| power: 4   |
| temp: 4.0  | // next line: return temp*temp
+------------+
| base: 2.0  |
| power: 5   |
| temp: ?    | // next line: return base*temp
+------------+
```

```
| base: 2.0  |
| power: 10  |
| temp: ?    | // next line: return temp*temp
+------------+ BOTTOM
```

And the next

```
+------------+ TOP
| base: 2.0  |
| power: 5   |
| temp: 16.0 | // next line: return base*temp
+------------+
| base: 2.0  |
| power: 10  |
| temp: ?    | // next line: return temp*temp
+------------+ BOTTOM
```

And the next

```
+------------+ TOP
| base: 2.0  |
| power: 10  |
| temp: 32.0 | // next line: return temp*temp
+------------+ BOTTOM
```

And we return 32.0.

A bit about debugging

- Debugging with printf has its uses, but should not be your primary mechanism.
- gdb is your friend when you program in C.
- Eclipse debugger is our focus for today.

Lab

Can you explain the difference between "step over", "step into", and "step out of"?

I can try. Consder the `expt` function above. Suppose we're at the following line.

```
double temp = expt(base, power/2);
```

We might want to say "Just do the call; I don't care what happens."
That's "step over".

We might want to say "Hmmm ... I think I need to look within the call." That's "step into".

We might say "Okay, I'm happy with the current procedure call. Let's return to the caller." That's "step out".

Reflection (maybe)