CS159

Nathan Sprague

March 19, 2018

Recursive Methods / Recursive Programming

Activation Records

Every method call results in an activation record which contains:

- Local variables and their values.
- The location (in the caller) of the call.

Tracing Recursive Methods...

Recursion is Not Always the Best Approach

```
1  int factorial(int n)
2  {
3      int value;
4
5      if   (n == 0)
6          value = 1;
7      else
8          value = n * factorial(n - 1);
9
10     return value;
11 }
```

VS

```
1
2  int factorial(int n)
3  {
4      int value = 1;
5
6      for (int i=2; i <= n; i++)
7      {
8          value *= i;
9      }
10
11     return value;
12 }
```

Recursion is Not Always the Best Approach

```
1  static final int [] FACTORIALS =
2      {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880,
3       3628800, 39916800, 479001600 };
4
5  int factorial(int n) {
6      return FACTORIALS[n];
7  }
```

http://stackoverflow.com/questions/36559371/efficiently-calculate-factorial-in-32-bit-machine

Recursive Problem Solving

Recursion is often a good idea when a problem can be solved by breaking it into one or more smaller problems of the same form. The process is:

- Figure out how to solve the easy case, i.e. the base case.
- Figure out how to move the hard case toward the easy case.

Recursion Pseudocode

Nearly every recursive method ends up looking like the following:

```
1
2  recursiveMethod(input)
3  {
4        if (input represents a base case)
5        {
6             handle the base case directly.
7        }
8        else
9        {
10            call recursiveMethod one or more times
11            passing it only part of the input.
12       }
13 }
```

There may be more than one base case.

Recursion Pseudocode Variations

Sometimes there is nothing to do for the base case. We just want to stop:

```
1  recursiveMethod(input)
2  {
3       if (input is NOT the base case)
4       {
5            call recursiveMethod one or more times
6            passing it only part of the input.
7       }
8
9       // No else statement.  Nothing to do for the base case.
10 }
```

The Coin Problem

Problem: determine the minimum number of coins needed to make change for a given amount. E.g. we have pennies, nickels and dimes, and we need to 7 cents worth of change. How many coins will it take? Pseudocode:

- If the change amount is equal to a coin denomination.
  - Return 1; // Base case! only one coin is needed.
- Otherwise consider every possible way of breaking the change amount in two. Each split results in two smaller versions of the change problem. Solve these recursively, add the results, and return the lowest sum.
- E.g. if the amount is 7 cents we can try:
  - 1 and 6 $\rightarrow 1 + 2 = 3$
  - 2 and 5 $\rightarrow 2 + 1 = 3$
  - 3 and 4 $\rightarrow 3 + 4 = 7$