Lecture 14

Recursion
Announcements for Today

Prelim 1

• Tonight at 7:30-9pm
  ▪ A–G (Olin 155)
  ▪ H–K (Olin 165)
  ▪ L–R (Olin 255)
  ▪ S–Z (Upson B17)
• Graded by Noon on Fri
  ▪ Scores will be in CMS
  ▪ In time for drop date
• Make-ups Fri @ 6:30

Other Announcements

• Reading: 5.8 – 5.10
• Assignment 3 now graded
  ▪ Mean 93, Median 100
  ▪ Typical for this assignment
• Survey for A3 still active
• Assignment 4 posted Fri
  ▪ Uses material from today
  ▪ Due two weeks from today
  ▪ Get started immediately!

10/17/12
Recursion
Recursion

- **Recursive Definition:**
  A definition that is defined in terms of itself

- **Recursive Function:**
  A function that calls itself (directly or indirectly)

- **Recursion:** If you get the point, stop; otherwise, see Recursion

- **Infinite Recursion:** See Infinite Recursion
A Mathematical Example: Factorial

• Non-recursive definition:
  \[ n! = n \times (n-1) \times \ldots \times 2 \times 1 \]
  \[ = n (n-1 \times \ldots \times 2 \times 1) \]

• Recursive definition:
  \[ n! = n \times (n-1)! \quad \text{for } n \geq 0 \]
  \[ 0! = 1 \]

What happens if there is no base case?
Factorial as a Recursive Function

```python
def factorial(n):
    """Returns: factorial of n.
    Pre: n ≥ 0 an int""
    if n == 0:
        return 1
    return n*factorial(n-1)
```

- $n! = n \cdot (n-1)!$
- $0! = 1$

What happens if there is no base case?
Example: Fibonacci Sequence

• Sequence of numbers: 1, 1, 2, 3, 5, 8, 13, ...

  \[ a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \]

  ▪ Get the next number by adding previous two
  ▪ What is \( a_8 \)?

A: \( a_8 = 21 \)
B: \( a_8 = 29 \)
C: \( a_8 = 34 \)
D: None of these.
Example: Fibonacci Sequence

- Sequence of numbers: 1, 1, 2, 3, 5, 8, 13, ...

  \[ a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6 \]

  - Get the next number by adding previous two
  - What is \( a_8 \)?

- Recursive definition:
  - \( a_n = a_{n-1} + a_{n-2} \) Recursive Case
  - \( a_0 = 1 \) Base Case
  - \( a_1 = 1 \) (another) Base Case

Why did we need two base cases this time?
Fibonacci as a Recursive Function

```python
def fibonacci(n):
    """Returns: Fibonacci no. $a_n$
    Precondition: $n \geq 0$ an int""
    if n <= 1:
        return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

What happens if we forget the base cases?
Fibonacci as a Recursive Function

```python
def fibonacci(n):
    
    """Returns: Fibonacci no. \( a_n \)
    Precondition: \( n \geq 0 \) an int"""
    if n <= 1:
        return 1
    return fibonacci(n-1) + fibonacci(n-2)

return (fibonacci(n-1) +
        fibonacci(n-2))
```

- Function that calls itself
  - Each call is new frame
  - Frames require memory
  - \( \infty \) calls = \( \infty \) memory
Fibonacci: # of Frames vs. # of Calls

- Fibonacci is very inefficient.
  - \( \text{fib}(n) \) has a stack that is always \( \leq n \)
  - But \( \text{fib}(n) \) makes a lot of redundant calls
Recursion as a Programming Tool

• Later we will see iteration (loops)
• But recursion is often a good alternative
  ▪ Particularly over sequences (lists, strings)
• Some languages only have recursion
  ▪ “Functional languages”; topic of CS 3110

A4: Recursion to draw fractal shapes
String: Two Recursive Examples

```python
def length(s):
    """Returns: # chars in s""
    # {s is empty}
    if s == ":
        return 0
    # { s at least one char }
    return 1 + length(s[1:])

def num_es(s):
    """Returns: # of ‘e’s in s""
    # { s is empty }
    if s == ":
        return 0
    # { s at least one char }
    return 1 if s[0] == 'e' else 0 + num_es(s[1:])
```

Imagine len(s) does not exist

10/17/12
Two Major Issues with Recursion

• **How are recursive calls executed?**
  - We saw this with the Fibonacci example
  - Use the call frame model of execution

• **How do we understand a recursive function (and how do we create one)?**
  - You cannot trace the program flow to understand what a recursive function does – too complicated
  - You need to rely on the function specification
How to Think About Recursive Functions

1. Have a precise function specification.
2. Base case(s):
   - When the parameter values are as small as possible
   - When the answer is determined with little calculation.
3. Recursive case(s):
   - Recursive calls are used.
   - Verify recursive cases with the specification
4. Termination:
   - Arguments of calls must somehow get “smaller”
   - Each recursive call must get closer to a base case
def num_es(s):
    """Returns: # of ‘e’s in s""
    # {s is empty}
    if s == '':
        return 0
    # { s at least one char }
    return ((1 if s[0] == 'e' else 0) + num_es(s[1:]))

- Break problem into parts
  number of e’s in s =
  number of e’s in s[0] + number of e’s in s[1:]

- Solve small part directly
  number of e’s in s =
  (1 if s[0] == 'e' else 0) + number of e’s in s[1:]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>len(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>H</td>
<td>ello World!</td>
</tr>
</tbody>
</table>
Understanding the String Example

• **Step 1:** Have a precise specification

```python
def num_es(s):
    """Returns: # of ‘e’s in s"""
    # {s is empty}
    if s == ":
        return 0
    # { s at least one char }
    return (1 if s[0] == 'e' else 0) + num_es(s[1:])
```

“Write” your return statement using the specification

• **Step 2:** Check the base case
  - When `s` is the empty string, 0 is returned.
  - So the base case is handled correctly.

10/17/12
Recursion
• **Step 3:** Recursive calls make progress toward termination

```python
def num_es(s):
    '''Returns: # of ‘e’s in s'''
    # {s is empty}
    if s == '':
        return 0
    # { s at least one char }
    # return # of ‘e’s in s[0]+# of ‘e’s in s[1:]
    return (1 if s[0] == 'e' else 0) + num_es(s[1:])
```

- **Step 4:** Recursive case is correct
  - Just check the specification
Exercise: Remove Blanks from a String

1. Have a precise specification

   ```python
def deblank(s):
    """Returns: s but with its blanks removed""
```

2. Base Case: the smallest String s is "."

   ```python
   if s == ":
    return s
   ```

3. Other Cases: String s has at least 1 character.

   ```python
   return (s[0] with blanks removed) + (s[1:] with blanks removed)
   ```

("" if s[0] == ' ' else s[0])
What the Recursion Does

deblank

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
</table>
What the Recursion Does

debblank

[Diagram]

[Boxes labeled 'a', 'b', 'c']

10/17/12
What the Recursion Does

deblank

1 2 3

deblank

a b c

a

deblank

b c
What the Recursion Does

deblank

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{cc}
\text{a} & \\
\end{array}
\]

\[
\begin{array}{cc}
\text{b} & \text{c} \\
\end{array}
\]

\[
\begin{array}{cc}
\text{b} & \text{c} \\
\end{array}
\]
What the Recursion Does

deblank

a  b  c

da  b  c

b  c

deblank

b  c

deblank

c
What the Recursion Does

deblank

a b c

deblank

a b c

a

deblank

b c

b

deblank

b c

b c

deblank

c

deblank

c

What the Recursion Does

deblank

a b c
deblank

a b c
deblank

b c
deblank

b c
deblank

c
What the Recursion Does

deblank

10/17/12
What the Recursion Does

deblank

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td></td>
</tr>
</tbody>
</table>

10/17/12

Recursion
What the Recursion Does

deblank

\[\begin{array}{c}
a \\ b \\ c
\end{array}\]

deblank

\[\begin{array}{c}
a \\ b \\ c
\end{array}\]

deblank

\[\begin{array}{c}
a \\ b \\ c
\end{array}\]

deblank

\[\begin{array}{c}
a \\ b \\ c
\end{array}\]
What the Recursion Does

deblank

a  b  c

da  b  c

c

c

✗

✗

Recursion
What the Recursion Does

deblank

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

debblank

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\end{array}
\]

debblank

\[
\begin{array}{cc}
\text{b} & \text{c} \\
\end{array}
\]

debblank

\[
\begin{array}{c}
\text{b} \\
\end{array}
\]

debblank

\[
\begin{array}{c}
\text{c} \\
\end{array}
\]

debblank

\[
\begin{array}{c}
\text{c} \\
\end{array}
\]
What the Recursion Does

deblank

\[
\begin{array}{ccc}
    & a & b & c \\
\end{array}
\]

\[
\begin{array}{ccc}
    a & b & c \\
\end{array}
\]

\[
\begin{array}{ccc}
    b & c \\
\end{array}
\]

\[
\begin{array}{ccc}
    c \\
\end{array}
\]
What the Recursion Does

deblank

10/17/12 Recursion
Exercise: Remove Blanks from a String

```python
def deblank(s):
    """Returns: s with blanks removed"""
    if s == 
        return s

    # s is not empty
    if s[0] is a blank:
        return s[1:] with blanks removed

    # s not empty and s[0] not blank
    return (s[0] +
            s[1:] with blanks removed)
```

- Sometimes easier to break up the recursive case
  - Particularly om small part
  - Write recursive case as a sequence of if-statements

- Write code in *pseudocode*
  - Mixture of English and code
  - Similar to top-down design

- Stuff in **red** looks like the function specification!
  - But on a smaller string
  - Replace with deblank(s[1:])

10/17/12 Recursion 33
Exercise: Remove Blanks from a String

```python
def deblank(s):
    """Returns: s with blanks removed"""
    if s == ":
        return s
    # s is not empty
    if s[0] in string.whitespace:
        return deblank(s[1:]):
    # s not empty and s[0] not blank
    return (s[0] +
            deblank(s[1:]))
```

- Check the four points:
  1. Precise specification?
  2. Base case: correct?
  3. Recursive case: progress toward termination?
  4. Recursive case: correct?

Expression: `x in thelist` returns True if `x` is a member of list `thelist` (and False if it is not)
Next Time: A Lot of Examples