Slides + code available at
www.github.com/pkorth/eecs280_recitation

EECS 280

Discussion 02: Jan 21, 2015

MichiganEngineering

Agenda

- Logistics
- Brief review of lecture material
  - Recursion
  - Tail recursion
- Demo
- Lab 02: Recursion

Slides + code available at
www.github.com/pkorth/eecs280_recitation

Logistics

- Lab 02
  - Required pre-lab and post-lab
  - Due Friday, 1/23

- Project 2
  - Recursion, trees, lists
  - Due Monday, 2/2

Recursion

A function is said to be **recursive** if it calls itself.

The problem can be defined **in terms of itself**.

This can help simplify some problems by dividing a seemingly large, complicated problem into **smaller subparts** that add together to make the whole.

Recursion

There are two parts to any recursive function:

Base case
The problem can't go any smaller. We know the answer immediately.

Recursive case
Perform a computation for this step, then combine this result with the result of the next smaller step.

Recursion

```
unsigned int factorial(unsigned int n) {
  if (n == 0) {
    return 1;

  } else {
    return n * factorial(n - 1);
  }
}
```

Factorial

0! = 1

1! = 1

n! = n x (n - 1)!

4! = 4 x 3 x 2 x 1 = 24

Tail Recursion

Using recursion can call the function a lot of times, meaning a lot of stack frames, meaning a lot of memory.

Tail recursion remediates this by eliminating the current stack frame so that we use at most one stack frame.

Tail Recursion

How? Get rid of the “other” part of the recursive call.

From recursive to tail recursive:

1. Pass the “current” result into the function call itself, leaving only the recursive function call in the return.
2. Usually need a “helper” function for this argument.

Tail Recursion

```cpp
unsigned int factorial(unsigned int n) {
  if(n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

```
factorial(5)

5 * factorial(4)
      4 * factorial(3)
            3 * factorial(2)
                  2 * factorial(1)
                        1 * factorial(0)
                              1
```

Tail Recursion

```
unsigned int factorial(unsigned int n) {
  return fact_help(n, 1);
}

unsigned int fact_help(unsigned int n,
                       unsigned int so_far) {
  if(n == 0) {
    return so_far;
  } else {
    so_far *= n;
    n -= 1;
    return fact_help(n, so_far);
  }
}
```

factorial(5)

fact_help(5, 1)
fact_help(4, 5)
fact_help(3, 20)
fact_help(2, 60)
fact_help(1, 120)
fact_help(0, 120)

Tail Recursion

Is it tail recursive?

- return 5 * foo(1, 2);
- return bar(7, foo(1, 2 + 3));
- return baz(1, 2) + bar(3, 4);

Tail Recursion

Is it tail recursive?

● return 5 * foo(1, 2);	Nope, 5 is left over
● return bar(7, foo(1, 2 + 3));	Yep!
● return baz(1, 2) + bar(3, 4);	No, bar() is left over

Slides + code available at
www.github.com/pkorth/eecs280_recitation

Lab 02

Goal

Practice writing iteration vs recursion vs tail recursion

Tasks

1. Complete pre-lab survey
2. Write “hailstone” as iterative and tail recursive
3. Write “countDigits” as iterative, recursive, tail recursive
4. Complete post-lab survey

MichiganEngineering