CS 105

Introduction to Scientific Computing

Topic #18 – Recursion

Matt Burlick
Stevens Institute of Technology

ASSIGNMENT 12

- Given 2 numbers, n and m, compute their product, n*m using the *Russian Peasant Method* defined as

$$product(n, m) = \begin{cases} product(\frac{n}{2}, 2m), & \text{if } n \text{ is even} \\ product(\frac{n-1}{2}, 2m) + m, & \text{if } n \text{ is odd} \\ m, & \text{if } n = 1 \end{cases}$$

- How can we easily write a function called *product* that does this?

NECESSARY SKILLS

- How to write recursive functions

TOPICS

1. Recursive Programming
2. Pros and Cons of recursion

READING

- None

WHAT IS RECURSION?

- Many problems can be solve by defining a solution in terms of prior solutions:
  - Fibonacci Number: $f_n = f_{n-1} + f_{n-2}$
  - Factorial: $x!=x*(x-1)!$
- The process of solving a problem by calling on prior solutions is called *recursion*.

RECURSIVE PROBLEMS

- Recursive Problems have 2 parts:
  - Initial conditions
    - The non-recursive part
    - Fibonacci: $f_0 = 0$, $f_1 = 1$
    - Factorial: 1!=1
  - The recursive relation
    - The formulae that involves prior solutions
    - Fibonacci: $f_n = f_{n-1} + f_{n-2}$
    - Factorial: $x! = x*(x-1)!$

RECURSION IN PROGRAMMING

- Recursive programming involves a function calling itself
  - The input parameters will be updated to specify the earlier problem(s)
- The function must also specify the initial/base cases

RECURSION EXAMPLE

```
function f = fibonacci(n)
        if(n==0)
                f=0;
        elseif(n==1)
                f=1;
        else
                f=fibonacci(n-1)+fibonacci(n-2);
        end
end
```

RECURSION EXAMPLE

```
function x = factorial(n)
        if(n==1)
                x = 1;
        else
                x=n*factorial(n-1);
        end
end
```

RECURSION TRACING

- fibonacci(3)
  - f=fibonacci(2)+fibonacci(1)
    =(fibonacci(1)+fibonnaci(0)) + fibonacci(1)
    = 1 + 0 + 1
    = 2
- factorial(4)
  - x = 4*factorial(3)
    = 4*(3*factorial(2))
    = 4*(3*(2*factorial(1)))
    = 4*(3*(2*1))
    = 4*(3*2)
    = 4*6
    = 24

PROS AND CONS OF RECURSION

- Pros
  - Model a lot of equations naturally.
  - Some algorithms are easy to code this way
- Cons
  - May involve many calls to the function and having to store all sorts of temporary data

ITERATIVE (LOOPY!) FACTORIAL

- How can we do factorial without recursion?
  - Loops!
- function x = factorialLoop(n)

```
        x=1;
        for i=n:-1:1
                x = x*i;
        end
end
```