Recursion and Recursive Functions (Part 2)

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Nov 2011

OBJECTIVES

At the end of the laboratory session, the student is expected to

- Define what a recursion is.
- Identify the parts of a recursive function.
- Manually trace and solve recursive functions.

MORE ON RECURSIVE FUNCTIONS

So far, we have encountered some "basic" recursive functions. This time, we will deal with more complicated recursive functions. This is where our basic knowledge about the topic comes into play. Let's start with one of the most popularly used example for recursion. Take a look at the number series below:

```
1  1  2  3  5  8  13  21  34  55 …
```

What do you observe?

The number series above is the Fibonacci series and it can be modeled to solve the nth Fibonacci number using recursive functions. A typical recursive function would look something like this:

```
int Fibo( int k){
      if(k == 0) return 0;
      else if (k == 1) return 1;
      else return Fibo(k-1)+Fibo(k-2);
}
```

The above code works with the assumption that the 0^{th} Fibonacci number is 0 and the 1^{st} Fibonacci number is 1. Try tracing the code and see if you arrive at the given values shown in the series.

Exercise 2: Recursive Functions

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Nov 2011

Instructions: This is a pen and paper exercise. You are required to answer all of the items listed in this paper.

1. Create a recursive function `Power` that computes the power of x raised to y. For simplicity, `x, y >0`.

2. Create a recursive function `GCF` that computes the greatest common factor between a and b. Also assume `a, b > 0`.

3. Trace the Fibonacci using input `k=11`. Show the tracing on how you solved the recurrence.

4. Trace the Ackermann Function below using
   a. `m=0, n=0`
   b. `m=1, n=0`
   c. BONUS `m=5, n=5`

```
int ackermann(int m, int n)
{
        if (m == 0) return n + 1;
        if (n == 0) return ackermann(m - 1, 1);
        return ackermann(m - 1, ackermann(m, n - 1));
}
```