Saturday, February 15, 2014

CRASH COURSE: Recursion

Recursion is a fundamental aspect of computer science. It must be learned and grasped by all who hope to pursue the deeper aspects of the field.

For all of those who are still a little foggy and confused about recursion, I've put together the most basic "HOW-TO" guidelines I could think of. I've included several basic examples of "natural recursion" and even offer some insight on the different kinds you'll all learn later on.

**** NATURAL RECURSION ****

What is known as "natural recursion" is the first kind of recursion taught in this course. This and another kind of recursion will be taught before the course's Midterm Exam, so both are important if you wish to do well in the earlier parts of the course.

In Scheme, Natural recursive procedures are typically written in the following form:

```
( define procedure
        ( lambda ( x )
                (  BASE CASE
                        ( BASE CASE OUTPUT )
                                ( REACTION
                                        ( RECURSION ( CHANGE IN x ) ) ) ) ) )
```

The most important parts of a recursive procedure are THE BASE CASE, its output and THE FINAL LINE as denoted above.

** BASE CASE **

The base case can be thought of as the STOPPING POINT and the STARTING POINT. It takes a condition that should PREVENT exception errors and infinite loops. The condition or conditions that it carries should effectively halt the continuation of its procedure and immediately cause it to evaluate whenever the right time has come. Think of it as the most "basic" output the procedure should return. Its return greatly depends on what the procedure is meant to do. For example:

Writing a procedure that takes an argument n and calculates 2^n recursively:

the base case should output 1

Writing a procedure that adds 1 to every element in a list:

the base case should output '() the empty list

** REACTION **

The base should also be able to interact with the procedure's REACTION. The REACTION should be able to evaluate with the output of the BASE CASE with its given OPERATION (i.e. operations with numbers, operations with list, etc. etc.). The return type the procedure outputs is what dictates the operation coupled with the REACTION step. For example:

Outputting a list: REACTION should have an operation applicable to lists
Outputting a decimal/integer: REACTION should have an operation applicable with numbers

These operations should ALSO be able to react with the OUTPUT of the BASE CASE, so that when the condition of the base case is satisfied, the REACTION will be able to operate with the BASE CASE output.

** THE LAST LINE **

Lastly, the final line of the typical natural recursive procedure comprises of RECURSION and CHANGE.

The recursive part is self-explanatory. It simply calls the name of the procedure again whenever the base case condition is not satisfied, signaling the program to run AGAIN within itself. This loop will continue UNTIL it is stopped by the conditions detailed in the program's BASE CASE.

Each iteration of the program should retain the SAME AMOUNT of arguments as detailed in the program's argument initialization (i.e. in the lambda). However, AT LEAST one argument should experience a CHANGE. Most likely, this argument is whatever is defined in the program's BASE CASE. For example:

This program outputs the sum of all integers in a list

```
( define add-me
        ( lambda (ls)
                (if (null? ls)
                        0
                        (+ (car ls)
                                (add-me (cdr ls))))))
```

Notice that in the final line, the variable type remains the same (i.e. a list), but the argument has also experienced a change (i.e. (cdr ls)). With this change, the program should eventually be able to satisfy the condition of the base case (if (null? ls)) in order to prevent an infinite loop.

Once the condition is satisfied, every single REACTION will be evaluated with the FINAL output of the RECURSION line, which SHOULD BE the output of the BASE CASE.

**** FUTURE RECURSION ****

In the later parts of the course, you will learn several different types of recursion, each with its own different aspects and nuisances. However, a lot of the same concepts will remain the same, especially the ones detailed above. At first, Recursion is a difficult concept to grasp, but once you get it, it becomes a very powerful programming tool.

All the best,

Carl Factora