Ryan Draper

CS 326

March 7, 2013

Reading 7

1. A) I felt that the way that they described parameters in 9.2.3 was confusing in that I understand how subprograms work with parameters as a way to access non local variables. I feel that they were trying to over emphasis how limited you are without parameter passing but I’m either tired or felt they were a bit fluffy and long winded with their explanation. B) I didn’t truly understand the difference between methods and subprogams as I went through the chapter I had trouble deciphering how they interact with classes and objects differently. I felt like this is just syntactical sugar as I read through the chapter in how they are called or the lack of their capabilities but for the most part they truly are equal.

2. A) I didn’t understand that C required non-keyword parameters because it was positionally associated though this does make sense that functions need headers and prototypes in order to be accessed as needs to be in the order it is accessed. B) I thought in Haskell functions could not be

3. **Passed by value:** The value of parameter is passed into act as a formal parameter for local value within that function or method.

**Pass by Result:** A copy of the value is returned to the subprogram or caller so that it can use the value but not at the address.

**Pass by Value Result:** a copy that is the same as *pass by value* which is going into the function or sub-program and out of the function or subprogram as *pass by result*. Pass by copy

**Pass by Reference:** A inout mode for parameter passing that calls some pointer like a memory address that acts as a path to the parameter and can be acted upon by the function or sub-program

**Pass by Name:** Uses a access method that is a inout mode for a parameter that semantically binds two parameters names to one instance in all the occurrences of the formal parameter

4. Briefly summarize the major differences between ML, Haskell, and F#.

**ML vs Haskell:** ML cannot overload functions where in Haskell you can. Non-strict semantics meaning Haskell doesn’t need fun to declare a function, also since the value, Haskell has no side effects for expressions or statements since variables are immutable and ML has mutable arrays.

**F#:** This is language with a .Net platform so it supports more features than the first two with a full featured IDE and extensive library. It has discriminated unions and is not strongly typed. Also has mutable arrays unlike the other two. More than one parameter in a function is allowed and can be curried because of this support.