CSCU9Y4 Spring 2018

Programming Language Paradigms

Tutorial S18/8

Supplementary material 8, plus the OCaml examples. There are many online OCaml tutorials. Bansal Chapter 9 is only marginally helpful here.

Reading for this tutorial: Chapter 9 of Comparative Programming Languages, Chapter 14 of Watt, and Chapter 15 of Sebesta. Google "functional programming", the terms in question 1, and "type inference".

1. The following concepts are quite central to imperative languages, but rather different in declarative languages. Define these terms and explain how they differ in the two paradigms:
   (i) assignment
   (ii) aliasing
   (iii) side effects
   In particular, consider the significance of the structure of the (von Neumann) computer for each term (and why that's not important for functional and logic programming languages).
   Lastly, define this concept which is more commonly used with respect to declarative languages:
   (iv) referential transparency
   Can we have referential transparency in an imperative language?

2. Here is a function definition in OCaml.

```
let rec mystery = function
   | (0, t) -> 0
   | (n, h::t) -> h + mystery ((n - 1), t);;
```

   Evaluate the following expression by hand, showing every step of the evaluation process:

```
mystery (3, [9;8;7;6;5;4;3;2;1])
```

   What does the function `mystery` do? (That is, sum up in English the relationship between inputs and outputs of this function.)

3. Rather than force the programmer to declare all types, functional languages often use type inference to ensure expressions are correctly typed.

   (a) What does this mean? Does the language have to be statically or dynamically typed? Can it be strongly or weakly typed?

   (b) Give the type of the function `mystery` in question 2 above.

University of Stirling
Department of Computing Science and Mathematics
RGC/CES

4. Consider the function `take`:

```
let rec take (n:int) (lst: 'a list) = match n with
  | 0 -> []
  | n -> match lst with
         | [] -> []
         | h::t -> h:: take (n-1) t;;
```

   Explain what `take` does, the type of `take` and the idea of *polymorphism*.

5. Write OCaml definitions for the following:

   (a) A function `tripleit` which takes an int list and returns the int list with all the elements multiplied by three.
   (b) A function `skipvalue` which takes an int n and a list (of anything) and returns the list with all occurrences of n removed. (You might do this with an int list first to understand the problem.)
   (c) How is pattern matching used in these functions?

6. Define `tripleit` using the *higher order function* `map`. What is a higher order function?