TDT4165 PROGRAMMING LANGUAGES

Exercise 3

Higher-Order Programming

Fall 2019

Preliminaries

This exercise is about higher-order programming and tail recursion. There are four basic operations that underlie the techniques of higher-order programming: procedural abstraction, genericity, instantiation and embedding.

Relevant reading: Chapters 1-3 in CTMCP, especially 2.5 and 3.6.

Compiling code must be in the form:

```
functor import Application System define
    % Your code here
    {Application.exit 0}
end
```

Compile with

```
ozc -c your-filename.oz
ozengine your-filename.ozf
```

If you do not call `{Application.exit 0}` (or your program hangs) then you can kill the it using the ctrl+c key combination.

You can deliver one file with all the code and theory answers as comments in that file.

Evaluation

You will get a score from 0 to 100 on this exercise. This score will later be converted into a final assignment score. Information on how this conversion will be done will be published on BlackBoard. Points for each task and sub-task is stated in the exercise. For the programming tasks you will be awarded full score for a correct implementation. Points may be deducted from this score as a percentage of the available points for the given task:

- Code does not run without modifications, but is otherwise correct. (20 % deduction)
- The implementation is not declarative. (100 % deduction)
- Modules other than the System and Application modules are imported and used. (100 % deduction)
- The implementation is correct but is overly complex, long or redundant. (20 % deduction)
- Unreasonable indentation of code. (20 % deduction)
- Functions, procedures and identifiers have names that are not meaningful. (20 % deduction)

Procedural abstraction

- The ability to convert any statement into a procedure value

Task 1 (20 p)

(a) Implement `proc {QuadraticEquation A B C ?RealSol ?X1 ?X2}`. (10 p)
`X1` and `X2` binds to the real solution(s) to the quadratic equation. `RealSol` binds to true if there exists a real solution, false otherwise. See the appendix for more information about the quadratic equation. You may ignore complex solutions beyond setting RealSol to false.
What are the values of `X1`, `X2` and `RealSol` respectively when `A = 2, B = 1` and `C = -1`? (1 p)
What are the values of `X1`, `X2` and `RealSol` respectively when `A = 2, B = 1` and `C = 2`? (1 p)

(b) Why are procedural abstractions useful? Give at least two reasons. (4 p)

(c) What is the difference between a procedure and a function? (4 p)

Genericity

- The ability to pass procedure values as arguments to a procedure call.

Task 2 (5 p)

In Assignment 2 you implemented `fun {Length List}`, which goes through the list recursively and returns the number of elements in the list.

Implement `fun {Sum List}`, which returns the sum of the values of the list elements.

Task 3 (35 p)

Your implementations of Length and Sum are probably very similar. Wouldn't it be nice with a more general version?

Length and Sum both reduce a list to a single value by using some combining operation on its elements. A fold is an abstraction that captures this pattern.
Write a function `fun {RightFold List Op U}` which goes through a list recursively and, through the use of a combining operation Op, accumulates and returns a result. U is the neutral element for the operation. The function should be right-associative, meaning it should perform the operations from right to left. E.g. if the operation is addition, and the list is [1 2 3 4], the function should calculate (1 + (2 + (3 + 4))) and return 10. If the operation is length, the function should calculate (1 + (1 + (1 + 1))) and return 4.
Implement Length and Sum using RightFold and test your solution using `{Length [1 2 3 4]}` and `{Sum [1 2 3 4]}`.

Hint: The Op parameter can be provided as an anonymous function: `fun {$ X Y} functionbody end`

(a) Implement `fun {RightFold List Op U}`. (15 p)

(b) Explain each line of code in `RightFold` in your own words. (5 p)

(c) Implement `fun {Length List}` and `fun {Sum List}` using `RightFold`. (10 p)

(d) For the Sum and Length operations, would left fold (a left-associative fold) and right fold give different results? What about subtraction? (3 p)

(e) What is a good value for U when using RightFold to implement the product of list elements? (2 p)

Instantiation

- The ability to return procedure values as results from a procedure call

Task 4 (10 p)

We have already seen that functions can be used as input to other functions. Functions can also return other functions. Implement `fun {Quadratic A B C}` that returns a function which can be used to calculate f(x) of a quadratic polynomial:

$$f(x) = Ax^2 + Bx + C$$

`{System.show {{Quadratic 3 2 1} 2}}` should display 17.

Embedding

- The ability to put procedure values in data structures

Task 5 (15 p)

Functions can be embedded in data structures. One use for this is to implement lazy (delayed) evaluation. The idea here is to build a data structure on demand, instead of building it all at once. This makes it possible to implement infinite lists.

(a) Implement `fun {LazyNumberGenerator StartValue}` that generates an infinite list of incrementing integers on demand, using higher-order programming. Do not use the built in lazy keyword or take advantage of threads or dataflow variables to implement the laziness. (10 p)
`{LazyNumberGenerator 0}.1` should return 0.
`{{LazyNumberGenerator 0}.2}.1` should return 1.
`{{{{{{LazyNumberGenerator 0}.2}.2}.2}.2}.2}.1` should return 5.

(b) Give a high-level description of your solution and point out any limitations you find relevant. (5 p)

Tail Recursion

In this assignment you have used recursion to implement most of your functions. Tail recursion is a special case of recursion.

Task 6 (15 p)

(a) Is your Sum function from Task 2 tail recursive? If yes, explain why. If not, implement a tail recursive version and explain how your changes made it so. (10 p)

(b) What is the benefit of tail recursion in Oz? (2 p)

(c) Do all programming languages that allow recursion benefit from tail recursion? Why/why not? (3 p)

Appendix

The Quadratic Equation

The quadratic equation is any equation on the form

$$ax^2 + bx + c = 0$$

where x is an unknown number and a, b and c are known constants. The solutions are any values of x that satisfy the equation. The number of solutions is determined by the value of the discriminant:

$$d = b^2 - 4ac$$

If d is positive there are two real solutions. If zero, there is exactly one solution (or two identical solutions if you will). If negative, there are no real solutions. The real solutions are the results of quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4bc}}{2a}$$