# Tutorial 9 (Week of 5Nov)

**Question 1**. Language Concepts (20 marks)

(a) If a function body has an **`if`** statement with a missing **`else`** clause, then an exception is raised by Oz if its condition is **`false`**. Explain why this behavior is correct. However, this situation does not occur for procedures. Explain why not. (7 marks)

(b) One can claim that both the **`if`** and the **`case`** statements are of equal expressive power. Elaborate on the truth or falsity of this claim. (8 marks)

(c) Given the following procedure: (5 marks)

```
proc {Test X}
    case X of
      f(a Y c) then {Browse 'case 1'}
      else {Browse 'case 2'}
    end
 end
```

Predict what would happen when you execute the following codes:

(i) `declare X Y {Test f(X b Y)}`
(ii) `declare X Y {Test f(a Y d)}`
(iii) `declare X Y {Test f(a Y c)}`
(iv) `declare X Y {Test f(X Y d)}`
(v) `declare X Y {Test f(X Y c)}`

**Question 2**. Lambda Calculus (25 marks)

(a) Consider the following lambda expressions. <u>Circle</u> the *free variables* in these expressions. (7 marks)

- $(\lambda\ x\ .\ y)$
- $(\lambda\ x\ .\ x)$
- $(\lambda\ x.\ (\lambda\ y.\ y))\ x$
- $(\lambda\ x.\ (\lambda\ y.\ x))\ x$
- $(\lambda\ x.\ (\lambda\ y.\ x))\ y$
- $\lambda\ z.\ ((\lambda\ x.\ z)\ (\lambda\ x.\ z))$
- $(\lambda\ z.\ (\lambda\ x.\ z))\ (\lambda\ x.z)$

(b) Consider the following lambda expressions. Count the number of *redexes* (reducible subexpressions) in each of these lambda terms. (5 marks)

- $(\lambda$ x. x) $(\lambda$ x. x)
- $(\lambda$ x. $(\lambda$ x. x) x) $(\lambda$ x. x)
- $(\lambda$ x. x x) $(\lambda$ x. x x)
- $(\lambda$ x. y) $((\lambda$ x. x x)$(\lambda$ x. x x))
- $(\lambda$ x. x $(\lambda$ x. x))

(c) Perform beta reductions using call-by-value (*leftmost-innermost*) strategy for the following lambda expressions. If the reduction is non-terminating, suggest an alternative reduction that terminates for the given code, if any. (7 marks)

- $(\lambda$ x. x) $(\lambda$ x. x)
- $(\lambda$ x. $(\lambda$ x.x) x) $(\lambda$ x. x)
- $(\lambda$ x. x x) $(\lambda$ x. x x) $((\lambda$ x. x) $(\lambda$ x. x))
- $(\lambda$ x. y) $((\lambda$ x. x x)$(\lambda$ x. x x))
- $(\lambda$ x. x $(\lambda$ x. x))

(d) Given a lambda term **T**. How would you show that this term is a *fix-point* operator? Comment *briefly* on the significance of fix-point operators. (6 marks)

**Question 3**. Stack ADT (20 marks)

Consider a stack ADT that is non-declarative whose operations may have side-effects. An example operation is given below :

```
Push :: Stack<X>, X --> ()
// takes a stack and an element which is pushed
// to the top of the stack
```

When executed, this procedure will modify its stack by pushing a new element on the top of the stack.

(a) Provide more stack operations that would allow you to construct, modify and query the stack ADT. Give only the polymorphic type interface *without* implementation details. (8 marks)

(b) Show how you would implement this non-declarative stack ADT by showing how each of its operations may be implemented in Oz. (**Hint** : You may need to use mutable data structure, such as Cell, Array or Dictionary.) (12 marks)

**Question 4**. Concurrency (15 marks)

The following is a naive attempt to write a concurrent `Filter` function:

```
fun {Filter L F}
 case L of
   X|Xs then if thread {F X} end
             then X|{Filter Xs F}
             else {Filter Xs F} end
   else nil
 end
end
```

(a) Comment *briefly* on the effectiveness of this attempt. (6 marks)

(b) Suggest how you may provide an alternative `Filter` operation with better concurrency. Outline the key steps that you need to make. Please provide a narrative of your solution, but do not provide any program code at all. (**Hint** : You may make use of non-declarative message-passing concurrency scheme.) (9 marks)