INSTITUTO SUPERIOR TÉCNICO

Instituto Superior Técnico

Programação Avançada

Segundo Teste – 20/6/2016

Número: _____

Nome: _____

Write your number on every page. Your answers should not be longer than the available space, and they can be in Portuguese or English. **In case of doubt, make reasonable assumptions, and explain them in your answer.** You can use the other side of the page for drafts. The exam has **4** pages and the duration is **1 hour**. The grade for each question is written in parenthesis. Good luck.

1. (3.0) Consider the following micro-evaluator:

```
(define (eval expr)
  (cond ((number? expr)
         expr)
        (else
         (apply
          (cond ((eq? (car expr) 'add) +)
                ((eq? (car expr) 'subtract) -)
                ((eq? (car expr) 'multiply) *)
                (else (error 'eval "Unknown operator")))
          (list-of-values (cdr expr))))))

(define (list-of-values exprs)
  (if (null? exprs)
      (list)
      (cons (eval (car exprs))
            (list-of-values (cdr exprs)))))
```

This micro-evaluator can evaluate simple arithmetic expressions such as:

```
> (eval '(add 1 (multiply 2 3) (subtract 5 2)))
10
```

(a) (1.0) How many calls to the function `eval` are needed to evaluate the previous expression?

(b) (1.0) Make the necessary changes to force the evaluator to use a **right-to-left** evaluation order for the arguments of the operators. This means that, in the previous expression, the sub-expression `(subtract 5 2)` must be evaluated before the sub-expression `(multiply 2 3)` which must be evaluated before the sub-expression `1`.

(c) (1.0) Consider the implementation, in the previous evaluator, of a language extension identical to the *arithmetic if* `aif`. This is a control operator that exists in some popular programming languages (e.g., Fortran) that requires four expressions as arguments. The operators evaluates the first expression and, depending on its value, it evaluates just one the remaining expressions: if the value is negative, it evaluates the second expression, if the value is zero, it evaluates the third expression, otherwise, it evaluates the fourth expression.

```
> (eval '(add 1
              (aif (subtract (multiply 2 3) (add 3 4))
                   (add 4 5)
                   6
                   (subtract 7 2))))
10
```

Redefine the evaluator to implement the operator `aif`.

2. (1.0) The `gensym` function exists for a very long time in almost all Lisp dialects. What does it do? What is its purpose? Explain.

3. (2.0) Several programming languages, such as, Haskell, O'Caml, F#, and Clojure, support *lazy evaluation*.

   (a) (1.0) What is *lazy evaluation*? Explain.

   (b) (1.0) Explain a possible implementation for *lazy evaluation* in the metacircular evaluator presented in the course.

4. (1.0) What is the meaning of the acronym *REPL*?

5. (1.0) *Aspect-oriented Programming* was designed to deal with the problems caused by *cross-cutting concerns*, namely *code tangling* and *code scattering*. Explain these terms.

6. (2.0) AspectJ is an extension of the Java language for aspect-oriented programming. AspectJ introduces the concepts of *join point*, *pointcut*, *advice*, *inter-type declaration* and *aspect*. Explain these concepts.