Interpretation and Compilation / MIEI / FCT UNL
Midterm Test 6 NOV 2018

STUDENT NUMBER [_____] NAME [_____]

ONE. Consider the following expression language defined by the abstract syntax

E :=	**stringliteral**	% string constant
|	**booleanliteral**	% boolean constant
|	E**@**E	% string concatenation
|	**empty** E	% empty string test
|	**head** E	% head of string
|	**tail** E	% tail of string

This is a tiny toy language to manipulate strings.

A String literal is an expression such as "", **"hello"**, **"icl"**, **"foo123"** (a sequence of characters between quotes).
A boolean is either **true** or **false**.
The operation **empty** E returns **true** if E evaluates to the empty string, and **false** otherwise.
If E evaluates to a non-empty string, the operation **head** E returns the one-character length string with the first character in such string (e.g. **head "hello"** evaluates to the string "h"). If E evaluates to the empty string then **head** E produces an error.
If E evaluates to a non-empty string, the operation **tail** E returns the string obtained by removing the first character on it (e.g. **tail "hello"** evaluates to the string **"ello"**). If E evaluates to the empty string then **tail** E produces an error.
E1**@**E2 evaluates to the result of concatenating the strings that result from evaluating E1 and E2 (e.g**., tail "hello" @ head "hello"** evaluates to "elloh").
Operations E**@**E, **head** E, **empty** E and **tail** E are only defined on string values, if something else is given, that will produce an error!

0. Explain the diference between concrete syntax and abstract syntax.

1. Define, as best as you can, the token **stringliteral** in javacc syntax as a regular expression.

2. The values of the language are strings and booleans. Define them using a Java class hierarchy implementing interface **IValue**.

3. Define, as a Java class hierarchy the abstract syntax of the language. All classes should implement the interface **ASTNode**, and support an evaluation method with the following signature.

```
IValue eval() throw RuntimeError;
```

TWO. Consider the programming language studied in the course till now. This is an imperative-functional language. Consider the following program.

```
0.   let  f =
1.     fun x, b ->
2.         let
3.           x = new x
4.           s = new b
5.         in
6.            while !x>0 do
7.              s := !s + !x ; x := !x – 1
8.            end;
9.            !s
10.         end
11.    end
12.  in
13.       f(4,0)+f(6,1)
14.  end
```

Answer to the following questions about declarations and scope.

0. For each identifier declaration in the program (either as a fun parameter or as a let definition) indicate its scope (stating what is the number of the line where the scope starts and the line where the scope ends).

1. Draw as a picture the state of the interpreter environment when **x := !x – 1** is evaluated for the second time (assume right-to-left evaluation of arithmetic expressions).

THREE. Consider again the programming language studied in the course till now. Explain ALL you would need to do to extend it with the following new construct representing iteration. The abstract syntax is as follows.

from id = E1 **to** E2 **do** E3 **end**

The intended semantics is as follows. First E1 and E2 are evaluated, and should produce integer values v1 and v2. Then, the body E3 should be evaluated for every integer between v1 and v2 (inclusively). For instance, the code

let S = **new** 0 **in from** x = -2 **to** 2 **do** S := !S + x; !S **end end**

will evaluate to 0. Notice that in this construct the identifier id is declared locally with scope the body E3. In each iteration id is bound to a different integer value. In case some type error occurs, a runtime error should be signaled.

Define the **ASTFrom** class representing the abstract syntax of the from construct, and the evaluation method in it, using the signature

IValue eval(Environment) throw RuntimeError