Interpretation and Compilation / MIEI / FCT UNL
Final Test 11 Dec 2018 (Duration: 90m)

STUDENT NUMBER [_____] NAME [_____]

ONE.
Explain the difference between dynamic typechecking and static typechecking.

TWO.
Consider adding the following expressions to the language of the course

E :=	**...**	% other expressions
|	E**::**E	% list cons
|	**nil**	% empty list
|	**empty** E	% empty list test
|	**head** E	% head of list
|	**tail** E	% tail of list

These expressions allows the language to manipulate homogeneous lists of values. An homogeneous list is a list of values all of the same type. For example, the result of evaluating the expression **let** x = 5+2 **in** (2+3)::x::**nil end** returns the list 5::7::**nil**. In this list, 5 is the head element and 7::**nil** is the list in the tail, e.g., the list of all elements in the list except the head. The operations **head** E and **tail** E evaluate to the head element of the list denoted by E and tail E returns the tail of the list denoted by E. Conventionally, we let **tail nil** evaluate to **nil**.

let l1 = 1::2::3::4::nil **in head** l1 **end** evaluates to 1

let l1 = 1::2::3::4::nil **in tail** l1 **end** evaluates to 2::3::4::**nil**

let l1 = 1::nil **in tail tail** l1 **end** evaluates to **nil**

The operation **empty** E returns **true** if E evaluates to the empty list, and **false** otherwise.

As our language is typed, we introduce a new type **List** T, where T is any type. For example **List int** is the type of lists of whose elements are integers, **List List int** is the type of lists whose elements are lists of integers, and **List bool** is the type of lists of whose elements are booleans. The type **List** admits the following typing rules for the several list related expressions

$$\frac{\text{Delta} \vdash \text{E1:T} \qquad \text{Delta} \vdash \text{E2} : \textbf{List}\ \text{T}}{\text{Delta} \vdash \text{E1} :: \text{E2} : \textbf{List}\ \text{T}} \qquad \frac{\text{Delta} \vdash \text{E} : \textbf{List}\ \text{T}}{\text{Delta} \vdash \textbf{head}\ \text{E} : \text{T}}$$

$$\frac{\text{Delta} \vdash \text{E} : \textbf{List}\ \text{T}}{\text{Delta} \vdash \textbf{tail}\ \text{E} : \textbf{List}\ \text{T}} \qquad \text{Delta} \vdash \textbf{nil} : \text{List T} \qquad \frac{\text{Delta} \vdash \text{E} : \textbf{List}\ \text{T}}{\text{Delta} \vdash \textbf{empty}\ \text{E} : \textbf{bool}}$$

1. Explain, justifying, why these typing rules ensure that, at runtime, all values in a list are of the same type.

2. Explain if each of the following programs is well typed or not, justifying. If the program is well typed, indicate the type of the value returned by the program.

let x = **2+3 in let y=2 in x::y::nil end end**

let x = **0 in let y=2 in x::y::x end end**

let x = **0 in (x<0)::(x>0)::nil end**

THREE.
Now, suppose you need to implement a typechecker for this language with lists.

1. Define a Java class ListType to represent the list type for typechecking purposes (e.g., similar to the class representing the **ref** T type in our language). This class implements the interface **IType** (or similar).

2. Define, as Java classes, the abstract syntax for the expressions E1::E2, **head** E, **tail** E and **empty** E. All sucb classes should implement the interface **ASTNode**, and support a typechecking method with the following signature.

IType typeckeck(Env<IType>) throw TypeCheckError;

Implement the typecheck method for the indicated 4 kinds of expressions.

FOUR.
Consider again the programming language studied in the course till now, and the compilation schemes for the JVM studied.

a) List the JVM instructions that your compiler should generate for expression 2 + (3+2) * 5

Consider the expression **let** x = 1 **in** x + x **end.**

b) Describe and explain, the best as you can, the code generated by your compiler for this expression.