INSTRUCTIONS FOR THE ONLINE FINAL ICL EVALUATION

Dear all,

The final test 24 Jan 2022 will be online.

By 17:00 we will start the final evaluation session, to end by 18:30.

A zoom session will be opened at 16:45 (15m before) at the usual zoom link for ICL lectures.

After that, I will open a (google) form that you will need to fill and submit.

The URL for the form will be sent to you in the zoom chatbox.

The form will only contain blank fields for you to write down our answers, the questions will be available in separate PDF file, uploaded to the CLIP before the test starts.

Note that your required answer depends on the parity of your student number.

https://docs.google.com/forms/d/e/1FAIpQLSe1k514O_7UJv-u0ucd_56WGPY2Fo4vaJTqsfuAZegyBzPymg/viewform

To access the form you will need to authenticate with your official student FCT NOVA email (---@campus.fct.unl.pt), which will be recorded in the form.

You may edit and re-submit your form as many times as you wish until the end of the test.

During the evaluation session, I will answer questions on the zoom chat.

You may use and consult any materials you want, **but you should avoid that**: it is not needed, and you would better use well your available time.

Thanks, all the best,

Luis Caires

Interpretation and Compilation / MIEI / FCT NOVA

ONLINE Final Test 2021

The C language offers post-increment and pre-increment operations, noted E++ and ++E respectively, where E is **any expression** denoting a memory cell.

As you know, in our ICL language, memory cells are represented by references, so it would make sense to add these post- and pre-increment operations to it.

The effect of evaluating E++ is to return the contents of the reference cell denoted by E, and increment the contents by 1 (the value returned is the content of the cell **before** incrementing).

The effect of evaluating ++E is to increment by 1 the contents of the memory cell denoted by E and then return the new value (the value returned is the content of the cell **after** incrementing).

Notice that both ++E and E++ do an implicit dereference of E (that is, both ++E and E++ return an integer).

For example, consider the programs

```
def i = ref 0 in
  while (i++ < 100) do
    println !i
  end
end;;
```

```
def i = ref 0 in
  while (++i < 100) do
    println !i
  end
end;;
```

the **red program** (left) prints all integers from 1 to 100, the **blue program** (right) prints all integers from 1 to 99.

In both cases E++ and ++E, the expression E **must evaluate** to a **reference cell holding an integer**, otherwise the expressions E++ and ++E have a type error.

ONE.

Define the Java AST class to represent the E++ (if your student number is odd) or ++E (if your student number is even) expression, including its interpreter eval method, which should have the signature

```
        IValue eval(Env<IValue> e) throws Exception
```

Your eval method must implement some dynamic type checking, issuing a runtime type exception if needed.

Note: Here IValue is the Java interface type that denotes language values in the interpreter and Env<IValue> is the Java class for the interpreter environment.

TWO. Consider now the implementation of a compiler for the language with E++ or ++E.

Considering the programs above , write down the JVM code that should be generated for the red program above on the left (if your student number is odd) or for the blue program above on the right (if your student number is even).

Use Jasmin notation for the JVM assembly code.

THREE.

A compiler covering the expression forms E++ or ++E must implement a typechecker, in order to generate correct code.

Write down the typecheck method for E++ (if your student number is odd) or for ++E (if your student number is even).

The signature should be something like:

```
IType typecheck(Env<IType> e) throws Exception
```

Your implementation should issue a static type checking exception if needed.

FOUR.

Write down the compile method for E++ (if your student number is odd) or for ++E (if your student number is even).

The signature should be something like:

```
void compile(CodeBlock c, Env<Coord> e)
```

NOTES:

You may consult whatever material you want.

GOOD WORK!