INSTRUCTIONS FOR THE FINAL ONLINE ICL EVALUATION

Dear all,

16 January 2021 (14:00), you will start the final online evaluation session.

A zoom session will be opened at 14:00.

https://videoconf-colibri.zoom.us/j/81109451903?pwd=cWtYNDgwdVhNM2hmTXgwV29NZWhZdz09

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

The web link is here (clickable)

https://docs.google.com/forms/d/e/1FAIpQLSdabzdMfMUVBhsUTLOMgF9u3CDZbCKtMtRao7War7ws10Z7tw/viewform

The form only contains the fields for you to write down the answers, not the questions. There are three questions (some with several items)

The questions will be available in a PDF file uploaded to the CLIP before the test starts.

To access the form you will need to authenticate with
your official student FCT NOVA email, which will be recorded in the form.

---@campus.fct.unl.pt

The evaluation will take place 14:15-16:00.

You may edit and re-submit your form several times until the end of the test.

During the evaluation, I will be answering questions on the zoom chat.

Thanks, all the best!

Luis Caires

Interpretation and Compilation / MIEI / FCT UNL
ONLINE Final Test 2020 – 16 Jan 2021

The language we have studied in the course allows the programmer to define imperative cells using the expression **new** E. For instance, the program:

```
def y=new 1 in
  y := !y + 1;
  !y
end
```

evaluates to 2. It first allocates a reference cell initialized with the value 1, and then increments it, and returns its value.
This final test is about extending the language with arrays of integer mutable reference cells. For example, the program

```
def a = new[10]   %% allocate an array with 10 reference cells and bind it to a
    i = new 0
in
  while !i < 10 do
    a[!i] := !i * !i;   %% assign !i * !i to the reference at a[!i]
    i:= !i + 1
  end;
  println !a[9]     %% print the value stored in the reference at a[9]
end
```

will print out 81.

The grammar must be extended with two additional forms of expressions, each one with two component expressions: **array allocation** and **array indexing**.

```
E := ... |  new [E1]          % array allocation
         |  E1 [E2]           % array indexing
```

Array allocation **new [E_1]** returns a reference to a newly created array, where E_1 is an integer valued expression, giving the number of array elements. Each array element is a mutable cell holding an integer value. The array elements are indexed between positions 0 and E_1–1, and are initially set to zero. If E_1 gives a negative value, there must be an error**.**

Array indexing **E_1 [E_2]** returns a reference to the cell at position E_2 of the array given by reference E_1. If **E_2** is outside the index range, there is an error.

ONE (6 Points). Parsing and abstract syntax.

1) Present the Java classes **ASTNewArray** and **ASTIndex** for representing the abstract syntax of the array allocation and array indexing expressions (just present the class declarations, their fields and constructors).

2) Describe the changes / additions you need to do on your javacc parser to deal with array allocation and array indexing expressions and create the appropriate AST nodes.

TWO (11 Points). We now write the basic interpreter pieces for the language with arrays. To represent array values, you need to define a new kind of value **VArray**.

1) Write the Java code for the **VArray** class, implementing interface **IValue**. Notice that each element of an array value is an integer reference cell value (you may use the Java class you used in your project to represent reference cell values).

2) Write down the **eval** method for classes **ASTNewArray** and **ASTIndex.** Implement dynamic type checking in the method.

THREE (3 Points). You now write the key compiler pieces for the language with arrays. To generate the code for array allocation and array indexing you will need to use the following JVM instructions:

anewarray *classname*
anewarray expects an integer value N on the top of the stack . It leaves on top of stack a reference to a JVM array for N objects of class *classname*.

aastore
aastore expects on the top of of the stack an array reference **a**, an index value **i**, and an element value **v**. It stores **v** into position **i** of array **a**.

aaload
aaload expects on the top of the stack an array reference **a** and an index value **i**. It leaves on top of stack the reference at position **i** of array **a**.

(more info at https://docs.oracle.com/javase/specs/jvms/se11/html/jvms-6.html#jvms-6.5.anewarray)

For instance, the compilation scheme for array indexing $\mathbf{E_1\ [E_2]}$ would be

$[[\ E_1\ ]]$
$[[\ E_2\ ]]$
aaload

1) Explain the compilation scheme for **new $[E_1]$.**
2) Write the **compile** method for classes **ASTNewArray** and **ASTIndex.**

NOTE: In this question don't worry with typechecking, assume that the AST is well typed.