ADVANCED PROGRAMMING
2015-16

Programming Assignment 1

The assignment involves a programming activity and includes an optional *problem*. The submission must include all required files for compiling and executing the proposed solution. It is allowed to develop the solution into teams, however the proposed solution should be discussed individually.

Introduction: The Tiny Programming Language

Tiny is a functional programming language and programmers will code their programs through expressions. Tiny contains a small set of data types. It has the three primitive types *boolean*, *number*, and *string* and the special valued *undefined*. You can assume standard operators for the primitive data types.

As usual, Tiny has one uniform construct for giving names to values: the **let** definition. The (standard) let syntax is:

let *name* = *expr1* **in** *expr2*

and the semantics is the obvious one.

Functions are values in Tiny, they are introduce as *Lambda* value with **fun**, and one can bind function values to names with **let**. As an example the following definition introduce a function named inc and applies it to the number 0.

let inc = **fun** n -> n + 1 in inc 0

Functions have static (lexical) scoping and are first order values. For simplicity we assume that functions are not recursive.

The set of Tiny expressions includes if-the-else, while, do, break, continue, return. They work the same as in other programming languages.

Tiny records are like the record of OCAML (or struct's of C): they are tuples except that their components (also called fields) are labeled. This means that their type is independent of any ordering. For instance the expression

let myR = {foo = 1; bar = "yikes"} **in** myR.foo

introduces a record with two components. As usual we can use dot notation for accessing record fields.

Differently, from OCAML, Tiny records include *methods* that allow one to program *actions* to be performed on record fields. As an example consider the

following expression:

```
let mm = {firtstName = “Mickey”; lastName = “Mouse”;
          fullName = fun() -> {return this.firstName + " " + this.lastName;}
in mm.fullname()
```

returns the string “Mickey Mouse”.

The problem set (mandatory)

Define the interpreter for the Tiny language in OCAML. It is mandatory to implement the type checking required to ensure that Tiny programs operate correctly with types. It is up to the candidate to fill the gaps in the specification and provide motivations for the choices made.

The problem set (optional)

The optional problem requires modifying the Tiny language with concurrent programming features. References and examples can be found in Chapter 18. Concurrent Programming with Async of the book *Real World OCaml*.

We assume to equip the Tiny language with the **eval()** function. The **eval()** function evaluates Tiny code as an independent thread. For example the expression

```
eval(let x= 5 in let z = 37 in x+z);

eval(let z = 42 in z);
```

spawns two independent threads both returning as result the integer value 42.

Outline the design of the interpreter for the Tiny language extended with **eval().** It is up to the candidate to fill the gaps in the specification and provide motivations for the choices made.

Grading

The breakdown for the programming assignment is as follows:

- Problem set (Mandatory): 60%
- Problem set (Optional): 30%
- Other factors (code quality): 10%

Submission

The AP homework should be submitted by e-mail by **02-11-2015 (midnight)** to gian-luigi.ferrari@unipi.it with the subject prefix **[AP-HW1]**.

The submission must include all required files for compiling and executing the proposed solution.