

CS 558 Homework 8 – due 5pm, Tuesday, March 11, 2008

Homework must be submitted by mail to `apt@cs.pdx.edu`, with the subject line “CS558 HW8”. All submitted files (two for this assignment) must be sent as plain-text *attachments* to the mail message (the contents of the message itself will be ignored). It is *your* responsibility to submit the homework in the proper format.

All programs mentioned can be downloaded from the course web page.

1. Recall again the `Env` ADT used in previous assignments ADT.

```
structure Env :>
  sig
    type 'a env
    exception NotFound of string
    val empty : 'a env
    val extend : 'a env -> (string * 'a) -> 'a env
    val lookup : 'a env -> string -> 'a
  end =
struct
  ...
end
```

Your task (similar to the last assignment) is to write a new structure implementation (replacing the `...`) that implements the same functionality as the existing `Env`, but uses the following representation for the abstract type:

```
type 'a env = string -> 'a
```

Do *not* alter the existing `Env` interface and don't accidentally alter the behavior of the operators. In particular, remember that if an environment is extended twice with the same identifier, the more recent extension “hides” the previous one.

Put your new structure definition (only) into a file `sol8_1.sml` and submit it.

2. Consider the following object-oriented language, which we'll call "E8"

```
prog := '(' { class } ')' exp
class := '(' cname cname { member } ')
member := '(' 'field' var exp ')'
        | '(' 'method' mname '(' { var } ') exp ')'
exp := var
     | int
     | '(' ':=' var exp ')'
     | '(' 'while' exp exp ')'
     | '(' 'if' exp exp exp ')'
     | '(' 'write' exp ')'
     | '(' 'block' { exp } ')
     | '(' 'local' '(' { var exp } ') exp ')'
     | '(' '@' exp mname { exp } ')
     | '(' 'new' cname ')'
     | '(' '+' exp exp ')'
     | '(' '-' exp exp ')'
     | '(' '*' exp exp ')'
     | '(' '/' exp exp ')'
     | '(' '<=' exp exp ')'
var := letter { letter | digit }
mname := letter { letter | digit }
cname := letter { letter | digit }
```

As usual, comments may be included by enclosing them between '{' and '}' characters, and they may be nested.

E8 is similar to the language E4, but with object-oriented features added. Instead of top-level function and global definitions, there are top-level class definitions. Class definitions consist of a class name, a superclass name, and a list of members. A member is either a field or a method. A field has an identifier and an initializing expression. A method has a name, zero or more parameter names, and a body expression. Expressions are much as before, except that the format of applications has changed to specify an object expression and a method name from that object to be applied, and there is a **new** constructor expression to create objects of a specified class.

The semantics of E8 are similar to those of other class-based object-oriented languages such as Java or Smalltalk. Classes are arranged in a hierarchy, with an (empty) built-in class **Object** at the root. Objects are created (via **new**) as members of a named class. A newly created object has all the members of that class, plus the members of all its ancestors.

The initial value of each field is specified by an expression in the field definition. Fields can contain integers or other objects. Fields of an object are only accessible for reading and writing within functions defined within the class of that object or its subclasses. All function applications are "virtual" (in C++ terminology); i.e., when a function is applied, its name is searched for first in the receiving class, then in its superclass, and so on. Within any

function body, the identifier “**this**” is always pre-defined to refer to the receiving object of the function call.

A simple example program is in `example.e8`. As usual, further details of the semantics may be revealed by experimenting with the (Java) interpreter (`hw8_2.java`) and reading its source code!

Your task is to modify the interpreter by adding a new expression form

```
exp := ...
      | '(' 'instanceof' cname exp ')'
```

The expression `(instanceof c e)` should return the integer 1 if expression `e` evaluates to an object which was created by `(new c')` where `c'` is `c` or a subclass of `c`; otherwise it should return 0. Using `instanceof` as the test expression of an `if` gives a way to branch according to the runtime class of an object. (This is essentially the same as the `instanceof` operator in Java.)

As usual, parsing support for this expression form is already in `hw8_2.java`.

Put your solution in a file `sol8_2.java`.