INF3110/4110—Mandatory Exercise 1

To be delivered 05.10.2007

Grammar

Consider the very simple language (VSLAN) defined by the grammar below. There are no declarations and there are only three variables (i, j, k), all of type `int`. `<number>` represents numbers of type `int`.

```
<prog>     ::=  {<stmt> ;}+
<stmt>     ::=  <assign> | <ifthen> | <print>
<ifthen>   ::=  if <boolexp> then <stmt>
<assign>   ::=  <var> := <exp>
<var>      ::=  i | j | k
<boolexp>  ::=  <exp> = <exp>
<exp>      ::=  <number> | <var> | <exp> + <exp>
<print>    ::=  print(<exp>)
```

This is a sample program in the language:

```
i := 1;
j := 2;
k := i + j;
if k = 3 then print(k);
```

In this exercise we will only consider the abstract syntax, so the concrete syntax of the language is not significant.

The task

You are going to write an interpreter for VSLAN in Standard ML. In order to do this, you must:

1. represent a data structure corresponding to a given program. To represent a program as a value in ML, you will typically need to declare datatypes for each of the non-terminals in the grammar.

2. write an `interpret` function. You interpret a program by calling a function `interpret` which takes a program and a state as input and gives a state as output:

   ```
   interpret : prog * state -> state
   ```

You will also need to write other functions which are called from the `interpret` function.

The program state

Since there are only three variables and no procedure calls, the program state should be quite simple to implement. You need to declare a datatype `state` which can hold three named values. [Hint: Two different ways of doing this is to use a record type or to use a list of pairs, but other solutions are also possible.]

In any case, you must also write three functions on the state: `getVal` which retrieves a variable value from the state, `putVal` which updates the state with a new value for a given variable, and to get nice output from the program you must write a function `stateToStr` from a state to a string:

```
getVal : var * state -> int
putVal : var * int * state -> state
stateToStr : state -> string
```

Test run

To test your implementation you should evaluate the following (see the `oblig1_2007.sml` file):

```
(print "(*** Program start ***)\n";
print ("Initial state: \t"^(stateToStr(init)));
print ("Final state: \t"^(stateToStr(interpret(sampleprog:prog,init))));
print "(*** Program end ***)\n");
```

where `init` is a value holding the initial program state and `sampleprogram` is a value holding the ML representation of the sample program. Thus the result of issuing the command:

```
... $ sml oblig1_2007.sml
```

should be something like this:

```
...
[output from sml echoing your declarations]
...
(*** Program start ***)
Initial state:  {i=0, j=0, k=0}
3
Final state:    {i=1, j=2, k=3}
(*** Program end ***)
val it = () : unit
```

Hints

- To convert an int to a string, use the built in function

  ```
  Int.toString : int -> string
  ```

  so to print an integer on a separate line you could for example define and use this function:

  ```
  fun printIntNL(i) = print(Int.toString(i)^"\n") ;
  ```

- In ML you evaluate a series of commands by evaluating the expression $(E_1; E_2; \ldots; E_n)$. When evaluating this expression, the expressions E_1 to E_n are evaluated from left to right. The result is the value of E_n, the values of the other expressions are discarded. This construct has been used in the test run code given above.

- A general note: If you get strange and inexplicable errors during development of your solution it might help to quit and restart the sml environment, since sometimes values you have defined earlier and later changed might interfere with later definitions.

Requirements and deliveries

Your solution may deviate from the sketched solution with regards to the types and functions you define, but we require that you should be able to run the given sample program with the test code. You are also not allowed to use reference types (reference cells). Email the file `oblig1_2007.sml` containing your implementation to your group teacher.