

CS 321 Homework 1 – due 4:30pm, Monday, October 18, 2010

(© Andrew Tolmach 2010. All rights reserved.)

Interpreting the E Language

The E language is a small expression-based language described by the following grammar.

program → *exp*

exp → *term*
→ *term* '+' *term*
→ *term* '-' *term*
→ *set ID* '=' *exp*
→ if0 *exp* then *exp* else *exp*

term → ID
→ NUM
→ '(' *exp* ')'
→ '{' *block* '}'

block → var *vars* ';' *exprs*
→ *exprs*

vars → ID
→ ID *vars*

exprs → *exp*
→ *exp* ';' *exprs*

The keywords are var, set, if0, then, and else. Identifiers (ID) are sequences of alphabetic characters and digits, starting with an alphabetic character, and excluding the keywords. Numbers (NUM) are sequences of decimal digits.

A program is just an expression, which evaluates to a single integer result.

The semantics of expressions, terms, and blocks can be informally described as follows.

- The term consisting of the identifier *v* evaluates to the current value of variable *v*.
- A numeric term *n* evaluates to the integer represented by the digits of *n*.
- The block term { var *v₁* ... *v_n* ; *e₁* ... *e_m* } introduces local variables *v₁, ..., v_n* with initial value 0, whose scope is the expressions *e₁, ..., e_m*; within these expressions, any outer declarations of the *v_i* are hidden. All variables must be declared in this manner before being referenced or assigned to. The block is evaluated by evaluating each of the *e₁, e₂, ..., e_m* in turn; its overall value is *e_m*.
- The term (*e*) has the value of expression *e*.

- Expression $t_1 + t_2$ (resp. $t_1 - t_2$) is evaluated by evaluating first t_1 and then t_2 and then adding (resp. subtracting) the resulting values.
- The expression `set v = e` assigns the value of e to variable v ; its overall value is the new value of v .
- The expression `if0 e then et else ef` is evaluated as follows: first e is evaluated; if the value is 0 then e_t is evaluated and its value serves as the value of the overall expression; otherwise, e_f is evaluated and its value serves as the value of the overall expression.

Your task in this assignment is to produce a static *checker* and *interpreter* for E programs. The checker makes sure – without actually evaluating the program — that all variables are declared (in some enclosing block) before they are used. The interpreter evaluates the program according to the semantic rules given above, and writes out the integer result.

Your program should contain a class `Main` with method

```
public static void main(String argv[ ])
```

in the usual way. The program should expect a single command-line argument specifying the name of a file containing the E program to interpret. If the E program parses and checks correctly, the interpreted result should be written to standard output. Any parsing or checking errors should be written to standard error.

Don't worry about the possibility of integer overflow during evaluation.

Implementation

Much of the work for this assignment has already been done for you. The relevant file, available on the course web page, is `hw1.java`. It defines a family of classes `Exp`, `VarExp`, `ConstExp`, etc. for representing the abstract syntax of most E expressions, with the exception of the subtraction operator (`-`) and the `if0` expression. Each expression class defines a method `boolean check(Env env)` that checks the expression in the context of a particular environment `env` of defined variables. The `Prog` class is used to hold the `Exp` that represents the entire program; it defines a method `boolean check()` that checks the program using the expression checkers. The file also defines a lexical analyzer and parser for the same subset of E. Finally, it defines a class `Main` with static method `main`, which is a driver that reads, parses, and checks expressions conforming to the subset. (Note: It is not a very good Java practice to put multiple class definitions in a single file like this, but it is convenient for small programs like this one.)

To use this file, download it to an empty directory, and then type

```
javac hw1.java
```

to compile it (producing many `.class` files), and then type

```
java Main foo.e
```

to execute it on an E source file `foo.e`.

You are strongly encouraged, though not required, to use this file as the basis of your solution. If you do use them, all you need to do is:

(a) Write new subclasses of `Exp` to represent the subtraction operator and the `if0` expression, write `boolean check(Env)` methods for these new classes, and extend the parser to recognize these new forms and call the appropriate new constructor functions.

(b) Add to `Exp` a new method `int interp()` that computes and returns the integer value of an expression representing a whole program. Your method should handle subtraction and `if0` as well as all the existing AST forms.

You may assume that `interp` will only be called on expressions that have passed the checker, as illustrated in the commented-out portion of the driver `Main.main`.

Hints: Your `interp` code can be structured in a way very similar to the checking method, i.e., by defining a method

```
int interp(ValueEnv env);
```

for `Exp` and each of its subclasses, which computes the value of the expression in an environment `env` that maps all the variables in scope to their current integer values. You can write `ValueEnv` as a subclass of `Env` or as an independent class. You'll also want a method

```
int interp();
```

in `Prog` that evaluates the expression representing the whole program in an empty `ValueEnv`.

A working solution to the assignment is on the web page in file `sol1.jar`. To run this program on an E source file `foo.e`, download it and execute

```
java -classpath sol1.jar Main foo.e
```

Your program should generate the same output as this one for correct programs; error messages may differ in format, though not in substance.

Assignment Submission

Place your solution in a single file called `sol1.java` and mail it to an address to be announced soon. You must mail this file as a plain text *attachment*; the contents of the message itself don't matter. We should then be able to compile your program by creating a fresh directory, saving your attachment, and typing

```
javac sol1.java
```

To run your program on input `foo.e`, we should be able to type

```
java Main foo.e
```

Note that we will be using automated mechanisms to read, compile, and test your programs, so adherence to this naming and mailing policy is important! You may lose points if you fail to submit your program in the correct way.