CS 301 Homework 1 – due 3:30pm, Wednesday, January 20, 1999

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
	→	id '`=`' exp
	→	`if0` exp `then` exp `else` exp
term	→	ID
	→	NUM
	→	'`(`' exp '`)`'
	→	'{' block '}'
block	→	`var` vars '`;`' exps
	→	exps
vars	→	var
	→	id vars
exps	→	exp
	→	exp '`;`' exps

The keywords are `var`, `if0`, `then`, and `else`. Identifiers (ID) are sequences of alphanumeric characters, beginning with a letter, excluding the keywords. Numbers (NUM) are sequences of 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` $v_1 \ldots v_n$ `;` $e_1 \ldots e_m$ } introduces local variables $v_1, \ldots, v_n$ with initial value 0, whose scope is the expressions $e_1, \ldots, e_m$; 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 $e_1, e_2, \ldots, e_m$ 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 v `=` e assigns the value of e to variable v; its overall value is the new value of v.
- The expression `if0` e `then` e_t `else` e_f 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 and writes out the integer result.

Your interpreter should take a single command-line argument specifying the name of a file containing the program to interpret. If the 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 and Assignment Submission

Much of the work for this assignment has already been done for you. The relevant files are available in `/u/cs301acc/1` and via the course web page. Files `ast.h` and `ast.c` define a data type and constructor functions for representing the abstract syntax of E expressions. File `parse0.c` contains a lexical analyzer and parser for most of E, with the exception of the subtraction operator (`-`) and the `if0` expression; `check0.c` contains a static checker for the same subset of E. File `main0.c` is a driver that reads, parses, and checks expressions conforming to the subset.

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

(a) Extend the code in `parse0.c` and `check0.c` to support the subtraction operator and the `if0` expression.

(b) Write a new function `int interp(Exp e)` that interprets a program in the E language (including subtraction and `if0`) and returns its value. You may assume that `interp` will only be called on programs that have passed the checker, as illustrated in the commented-out portion of the driver `main0.c`.

Hint: your interpreter can be structured as a recursive function quite similar to the checker.

A working solution to the assignment is in `/u/cs301acc/1/interpreter`. Your program should generate the same output as this one for correct programs; error messages may differ in format, though not in substance.

You will probably want to place your interpreter code in a new file `interp.c`. In any case, be sure to include in your submission a `makefile` that builds your interpreter given your source files and leaves the executable program in `interpreter`.