

Exercise sheet 3 for 26 February 2003

2003-02-19

Do exercises 3.1, 3.2, 3.3, 3.4, and 3.5. Hand in solutions to all except exercise 3.3 and 3.4.

Exercise 3.1 Write out the rightmost derivation of this string from the expression grammar presented in the lecture, corresponding to `expr/Exprpar.grm`. Take note of the sequence of grammar rules (1–9) used.

```
let z = (17) in z + 2 * 3 end EOF
```

Exercise 3.2 Draw the above derivation as a tree.

Exercise 3.3 Generate the lexer and parser for expressions, compile the abstract syntax `expr/Absyn.sml` and the lexer and parser, and load everything into the interactive system using

```
mosml parse.sml
```

Try the parser on several example expressions, both well-formed and non-well-formed ones, such as these:

```
parse "1 + 2 * 3";
parse "1 - 2 - 3";
parse "1 + -2";
parse "x++";
parse "1 + 1.2";
parse "1 + ";
parse "let z = (17) in z + 2 * 3 end";
parse "let z = 17) in z + 2 * 3 end";
parse "let in = (17) in z + 2 * 3 end";
parse "1 + let x = 5 in let y = 7 + x in y + y end + x end";
```

Exercise 3.4 Using the expression parser from `expr/parse.sml` and the expression-to-stack-machine compiler `scomp` and associated datatypes from `sem2.sml`, define a function `pcomp : string -> sinstr list` that parses a string as an expression and compiles it to stack machine code.

Exercise 3.5 Extend the expression language abstract syntax and the lexer and parser specifications with conditional expressions. The abstract syntax should be `If (e1, e2, e3)`; so you need to modify file `expr/Absyn.sml` as well as `expr/Exprlex.lex` and `expr/Exprpar.grm`. The concrete syntax may be the keyword-laden SML-style:

```
if e1 then e2 else e3
```

or the more light-weight C/C++/Java/C#-style:

```
e1 ? e2 : e3
```

Exercise 3.6 File `expr/javacc/Exprparlex.jj` contains a parser specification that can be used as input to the JavaCC parser generator. Follow the instructions in the source file to generate and compile a parser for the simple expression language. Run it on some well-formed and some ill-formed examples.

The JavaCC parser generator may be downloaded from http://www.webgain.com/products/java_cc/

Exercise 3.7 Determine the steps taken by the parser generated from `expr/Exprpar.grm` during the parsing of this string:

```
let z = (17) in z + 2 * 3 end EOF
```

For each step, show the remaining input, the parse stack, and the action (shift, reduce, or goto) performed. You will need a printout of `Exprpar.output` to do this exercise. Sanity check: the sequence of reduce action rule numbers in the parse should be the exact reverse of that found in the derivation in Exercise 3.1.

Exercise 3.8 Files in the subdirectory `usql/` contain abstract syntax abstract syntax (file `Absyn.sml`), an informal grammar (file `grammar.txt`), a lexer specification (`Sqllex.lex`) and a parser specification (`Sqlpar.grm`) for micro-SQL, small subset of the SQL database query language.

Extend micro-SQL to cover a larger class of SQL SELECT statements. Look at the examples below and decide your level of ambition. You should not need to modify file `parse.sml`. Don't forget to write some examples in concrete syntax to show that your parser can parse them.

For instance, to permit an optional WHERE clause, you may add one more component to the `Select` constructor:

```
datatype stmt =
  Select of expr list          (* fields are expressions *)
         * string list        (* FROM ... *)
         * expr option        (* optional WHERE clause *)
```

so that `SELECT ... FROM ... WHERE ...` gives `Select(..., ..., SOME ...)`, and `SELECT ... FROM ...` gives `Select(..., ..., NONE)`.

The argument to WHERE is just an expression (which is likely to involve a comparison), as in these examples:

```
SELECT name, zip FROM Person WHERE income > 200000

SELECT name, income FROM Person WHERE zip = 2300

SELECT zip, AVG(income) FROM Person GROUP BY zip

SELECT name, town FROM Person, Zip WHERE Person.zip = Zip.zip
```

In a similar way you may add optional GROUP BY and ORDER BY clauses. The arguments to these are lists of column names, as in this example:

```
SELECT town, profession, AVG(income) FROM Person, Zip
WHERE Person.zip = Zip.zip
GROUP BY town, profession
ORDER BY town, profession
```