

# School of Computer Science and Software Engineering

## CS3322 Programming Languages and Implementation

### Assignment 3

#### Due 5pm Thursday 26th of September

The purpose of this assignment is to demonstrate the implementation of scanners and parsers.

We will implement an evaluator for boolean arithmetic expressions on hexadecimal values, such as “0wxFF AND 0wx9A OR 0wx11”. The evaluator consists of a scanner (generated with ML-Lex) and a parser (implemented manually in ML).

The SML/NJ standard library offers a structure *Word* that implements hexadecimal values (written as a hexadecimal value prefixed with **0wx**) and boolean arithmetic operations such as **andb**, **orb**, **xorb** on these (see Ullman, Sec. 9.4.2 or the SML/NJ documentation). You should use this structure in your implementation.

The evaluator should read one or several boolean expressions from a file and return the value of the last expression. Expressions are separated and terminated by a semicolon. If any expression is prefixed with **PRINT** it should additionally print the value to **TextIO.stdout**.

The expressions contained in the file consist of constants for hexadecimal words (denoted in the same way as in ML), the infix operators **AND**, **OR**, **NOT**, **XOR** and parentheses. They can also contain identifiers for variables. These can be arbitrary words of the form  $[aA\dots zZ]^*$  but not the reserved words for the boolean operators. Variable names are case-sensitive.

All variables must be assigned values before they are used in an expression. An assignment statement takes the form “variable-name := expression ;”, e.g. **Test := 0wx7f ;**

Each variable can only be assigned once. A variable value is only valid in a single input file, i.e. every call to the evaluation function starts with an empty symbol table.

For these variables, you will need to keep a simple symbol table in which the variable values are stored and where subsequent expression evaluations can look up the value.

Example: consider the following input file:

```
Test := 0wx7f ;
PRINT 0wxFF AND 0wx8E7 OR NOT 0wx12 ;
0wx18 OR Test ;
```

The evaluator should return the value **val it = 0wx7f : word** and print the value “0wx7ffffef” to **stdout**.

1. Define a suitable datatype for the tokens to be returned by the lexer. Using ML-Lex implement a suitable scanner for the expression files described above. You can adapt the example lexer that was discussed in Lecture 2. **[3 mark]**
2. Write down an LL(1) grammar for this expression language. **[1 mark]**
3. In SML/NJ implement a recursive descent parser for the expression files described above. You can adapt the recursive descent expression parser discussed in Lecture 3. **[4 marks]**
4. Extend your code for the recursive descent parser with semantic actions and attributes so that it evaluates the expressions in the file. The toplevel-function should be

```
val evaluate = fn : string -> word
```

which takes a filename as an argument, evaluates this file and returns the value of the last expression, as side-effects printing the values of all expressions that have been prefixed with the `PRINT` command. [2 marks]

## Submission Instructions

The above exercises contribute 10% to your total CSE3322 mark.

The assignment is due **5pm Thursday 26th of September**.

Submission will be electronic. You should use `/cs/cc/bin/submit` to submit your assignment. To do so create a directory that contains all the files which you have to submit (see below) and submit this directory as a whole. Login with `ssh` to `ra-clay.cc.monash.edu.au` or `sng.cc.monash.edu.au` and then type `/cs/cc/bin/submit`. `Submit` will ask you for: the course code which is `cse3322`, the assessment code which is `ass3`, your student I.D., and the name of the directory to be submitted which should be `Ass3`. If for some reason you cannot use `submit` then send your answer as an email attachment to `Bernd.Meyer@infotech.monash.edu.au`

You must submit the following items in a directory called `Ass3`:

1. The lexer specification in a file `hex-lex.lex`,
2. The output of ML-LEX for this specification in a file `hex-lex.lex.sml`,
3. The program for the parser in a file `hex-parse.sml`. **Note that when loading this file, the lexer must automatically be loaded.**
4. A Postscript or PDF file called `grammar.ps` or `grammar.pdf` containing the grammar for Question 2. If you are unable to generate such an electronic file you can submit the grammar in handwritten form to the general office.

If you have solved Part 4 of the assignment you can just submit the complete code in the file `hex-parse.sml`. You do not need to submit a separate solution to Part 3 in this case.

Your programs will be marked on correctness, style, efficiency, clarity and documentation.

Assignments handed in after the due date will attract a late penalty of 5% per day unless special consideration applies or there has been prior agreement in writing from the lecturer. No submission will be accepted later than one week after the due date.