

# COP4020 Homework 5

R. van Engelen

November 19, 2002

Due date: December 9, 2002

## Part 1

Download the example Calc.java program from:

<http://www.cs.fsu.edu/~engelen/courses/COP4020/Calc.java>

Make sure you can compile this program by typing:

```
javac Calc.java
```

Just in case: when this doesn't work, find a machine (e.g. linprog) with Java installed, copy the program to that machine, and compile.

Now type:

```
java Calc
```

Type an expression to evaluate, such as  $(1+2)*3$ ; followed by [ENTER]. This should output 6.

The program implements the following attribute grammar:

|                                   |  |  |
|-----------------------------------|--|--|
| <code>&lt;expr&gt;</code>         | <code>-&gt; &lt;term&gt; &lt;term_tail&gt;</code>  | <code>term_tail.subtotal := term.value;</code><br><code>expr.value := term_tail.value</code>   |
| <code>&lt;term_tail1&gt;</code>   | <code>-&gt; '+' &lt;term&gt; &lt;term_tail2&gt;</code><br><code>  '-' &lt;term&gt; &lt;term_tail2&gt;</code><br><code>  empty</code>         | <code>term_tail2.subtotal := term_tail1.subtotal+term.value;</code><br><code>term_tail1.value := term_tail2.value</code><br><code>term_tail2.subtotal := term_tail1.subtotal-term.value;</code><br><code>term_tail1.value := term_tail2.value</code><br><code>term_tail1.value := term_tail1.subtotal</code>                         |
| <code>&lt;term&gt;</code>         | <code>-&gt; &lt;factor&gt; &lt;factor_tail&gt;</code>  | <code>factor_tail.subtotal := factor.value;</code><br><code>term.value := factor_tail.value</code>   |
| <code>&lt;factor_tail1&gt;</code> | <code>-&gt; '*' &lt;factor&gt; &lt;factor_tail2&gt;</code><br><code>  '/' &lt;factor&gt; &lt;factor_tail2&gt;</code><br><code>  empty</code> | <code>factor_tail2.subtotal := factor_tail1.subtotal*factor.value;</code><br><code>factor_tail1.value := factor_tail2.value</code><br><code>factor_tail2.subtotal := factor_tail1.subtotal/factor.value;</code><br><code>factor_tail1.value := factor_tail2.value</code><br><code>factor_tail1.value := factor_tail1.subtotal</code> |
| <code>&lt;factor1&gt;</code>      | <code>-&gt; '(' &lt;expr&gt; ')'</code><br><code>  '-' &lt;factor2&gt;</code><br><code>  number</code>                                       | <code>factor1.value := expr.value</code><br><code>factor1.value := -factor2.value</code><br><code>factor1.value := number</code>   |

where the indexing of the nonterminals, e.g. `<factor1>` and `<factor2>` with 1 and 2, is used to accommodate the semantic rules.

Extend the Calc.java program with the power  $\wedge$  operator, such that e.g.  $-3^2$  and  $2^3^4$  can be parsed and calculated. Note that power operator is right associative and has the highest operator precedence, even higher than unary minus, so  $-3^2$  is grouped as  $-(3^2)$  and  $4^3^2$  is grouped as  $4^(3^2)$ . The grammar productions to include are:

|  |  |
|--|--|
| <pre> &lt;factor1&gt;    -&gt; - &lt;factor2&gt;                 &lt;power&gt; '^' &lt;factor2&gt;                 &lt;power&gt; &lt;power1&gt;    -&gt; '(' &lt;expr&gt; ')                 number </pre> | <pre> factor1.value := -factor2.value factor1.value := power.value ^ factor2.value factor1.value := power.value power1.value := expr.value power1.value := number </pre> |
|--|--|

where `<power>` is a new nonterminal.

Make the necessary changes to the grammar and the `Calc.java` code by including the new production rules to enable the parsing of the power operator. Rename the class `'Calc'` to `'Calc1'` and save the changes to file `'Calc1.java'`. Compile, run, and test your program:

```

javac Calc1.java
java Calc1

```

Type an expression to evaluate, such as `-4^3^2`; followed by [ENTER]. This should output `-262144`.

## Part 2

Extend the attribute grammar of `Calc1.java` with a new attribute for all nonterminals: the `sym` inherited attribute is a symbol table with identifier-value bindings that defines the bindings of identifiers in the scope (context) of a `let` expression. A `let`-expression is an expression with a scope for a variable. For example, `let x=2 in x^2+x` is a `let`-expression with a local variable `x` with a scope that is limited to the expression `x^2+x`. The `let`-expression can be used anywhere within an expression. For example, `(let x=2 in x^2+x)+5` evaluates to 11.

Add a new production rule for the `let` expression to the grammar and add semantic rules to handle the symbol table:

|   |   |
|---|---|
| <pre> &lt;expr1&gt;    -&gt; 'let' identifier '=' &lt;expr2&gt; 'in' &lt;expr3&gt;                 &lt;term&gt; &lt;term_tail&gt; ... &lt;power1&gt;  -&gt; '(' &lt;expr&gt; ')'                 identifier                 number </pre> | <pre> expr2.sym := expr1.sym; expr3.sym := expr1.sym.put(identifier=expr2.value); expr1.value := expr3.value term.sym := expr1.sym; term_tail.sym := expr1.sym; term_tail.subtotal := term.value expr1.value := term_tail.value  expr.sym := power1.sym; power1.value := expr.value power1.value := power1.sym.get(identifier) factor1.value := number </pre> |
|---|---|

The semantic rule `expr2.sym := expr1.sym` copies the symbol table of the context in which `expr1` is evaluated to the context of `expr2`. The value of `expr2` is assigned to the identifier by adding a new entry to the symbol table which is passed to the context of the third expression, which is performed by the operation `expr3.sym := expr1.sym.put(identifier=expr2.value)`. The value of `expr2` is also the value of the `let`-expression, so `expr1.value := expr3.value`.

Extend the semantic rules of all other productions to include assignments for the `sym` attribute of all non-terminals. This is necessary to pass the symbol table to all parts of the parse tree of an expression so an identifier that occurs somewhere down the tree can be evaluated (table lookup).

To implement a symbol table with identifier-value bindings, you can use the Java `java.util.Hashtable` class as follows:

```

import java.util.*;
...
public class Calc
{ ...
    public static void main(String argv[]) throws IOException

```

```

{ ...
  Hashtable sym = new Hashtable();
  ...
  int value = expr(sym);
  ...
  private static int expr(Hashtable sym) throws IOException
  { if (token == tokens.TT_WORD && tokens.sval.equals("let"))
    { getToken(); // advance to identifier
      String id = tokens.sval;
      getToken(); // advance to '='
      getToken(); // advance to <expr>
      int value = expr(sym);
      Hashtable symnew = sym.clone();
      symnew.put(id, new Integer(value));
      getToken(); // skip over 'in'
      return expr(symnew);
    }
    else
    { int subtotal = term(sym);
      return term_tail(subtotal, sym);
    }
  }
  private static int power(Hashtable sym) throws IOException
  { ...
    else if (token == tokens.TT_WORD)
    { String id = tokens.sval;
      getToken();
      return ((Integer)sym.get(id)).intValue();
    }
  }
  ...
}

```

The put method puts a key and value in the hashtable, where the value must be a class instance so an `Integer` instance is created. The get method returns the value of a key. The `intValue` method of `Integer` class returns an `int`.

Make the necessary changes to the grammar and the `Calc1.java` code by renaming the class 'Calc1' to 'Calc2' first. Save the changes to file 'Calc2.java'. Compile, run, and test your program. The value of `(let a=2 in 3*a)+1;` should be 7. The value of `(let a=1 in (let a=a+2 in 3*a)+a);` should be 10.

### Part 3

Add Java exception handling to your `Calc2.java` code by defining a new exception `SyntaxError`. An exception should be raised when an illegal character is found, a closing `)` is not found, and a `=` or `in` are not used in the `let` expression. The exceptions should propagate the error to the main program which prints the diagnostics of the error. For example, `1+(2*3; should output closing )' is missing`. You must handle these errors using Java exceptions and the message should be printed by a Java exception handler in a `catch` clause. Submit this final version of the `Calc2.java` code.

Electronic submission is **required**. Send your sources (with sufficient explanation on how to compile and run the programs) to [engelen@cs.fsu.edu](mailto:engelen@cs.fsu.edu)