CMSC208 Grammars, Languages, and Automata Blaheta

Homework 9

Due: 8 April 2025

v20240329-1530

Problem 9.1 — theoretical

The grammars we've seen for a simple arithmetic expression have been as follows:

$$\begin{array}{rcl} E & \rightarrow & \text{number} \\ E & \rightarrow & E + E \\ E & \rightarrow & E * E \\ E & \rightarrow & (E) \end{array}$$

and

$$\begin{array}{rcl} E & \rightarrow & M \\ E & \rightarrow & E + M \\ M & \rightarrow & N \\ M & \rightarrow & M * N \\ N & \rightarrow & \text{number} \\ N & \rightarrow & (E) \end{array}$$

Show by drawing concrete parse trees how they perform differently on the expression

$$1 * 2 + 3 * 4 * (5 + 6)$$

Explain why the second grammar is better (including what "better" would mean in this context).

For the next two problems, consider the following tiny grammar:

$$\begin{array}{lcl}
P & \rightarrow & \texttt{program}\ S\ \texttt{\$} \\
S & \rightarrow & \texttt{(+}\ L\ \texttt{+)} \\
 & \rightarrow & \texttt{iflt}\ E\ E\ \texttt{then}\ S \\
 & \rightarrow & \texttt{iflt}\ E\ E\ \texttt{then}\ S\ \texttt{else}\ S \\
 & \rightarrow & \texttt{print}\ E \\
L & \rightarrow & \epsilon \\
 & \rightarrow & S\ L \\
E & \rightarrow & \mathit{num}
\end{array}$$

Note that S is suggestive of "statement", L of "list", and E of "expression", although in this language the only "expressions" are number literals.

Problem 9.2 — theoretical

The language P has two features of interest: a nestable branching construct, and a recursive definition that includes the empty string (written as ϵ) as one of its expansions. Give at least three strings in the language P, each diagrammed with a valid parse tree (recommendation: turn the paper sideways, write the entire string horizontally at the bottom so the tree can have the root at the top). The strings you choose should include:

- A simple one that has a single if-then-else where each branch has a single print statement
- An illustration of the use of the L production to make a block of multiple statements (but the overall string should still be an element of P)
- An illustration of the nesting conditional structure that has an ambiguous parse; show one of the parses and describe in words how the other would be different

Problem 9.3 — practical

Using the code from class as a model, write a set of C$^\sharp$ classes that can store the information from any parsed program of P, and on the call of a particular method will generate a valid C++ program to execute it. Suggested: use classes named `ProgramNode`, `StmtNode`, `BlockStmtNode`, `CondStmtNode`, and `PrintStmtNode`, and possibly others if you find them useful (but think about the relationship between those node types).

The semantics of the language should be mostly straightforward: a statement block between `(+` and `+)` should execute all the statements in the list; `iflt` is short for "if less than" and performs that comparison on the two expressions (numbers) that are given to it; and `print` should send the provided expression (number) to the console. You can assume *num* is an integer.

As with the class example, your program should have at least *something* in place to demonstrate that it works (though not necessarily a comprehensive test suite), but doesn't actually have to do the work of parsing the language—your examples can be entered manually with calls to `new CondStmtNode` and so forth.

Hand in the files containing the C$^\sharp$ code (and preferably also a readme) using the handin script:

```
handin cmsc208 hwk9 myfile.cs otherfile.cs README.txt
```

If you want to put the parse trees in electronic form too I'll accept them that way, but I think they'll be mostly easier to do on paper. (Please write neatly!)

Collaboration policy: **For Problems 9.1–9.2:** group work! If you work with other people on this homework, you can just hand in one copy and put all your names on top. There will be a revision cycle for this. **For Problem 9.3:** collaborative. You each have to hand in your own version of the assignment, but you can talk to other people about the problems. Mention in a comment or readme who you worked with. (Still no copying, though.)