Compilers Design: LR parser

Spring 2007

Programming Assignment 3

Due Date: June 20, 2007

You are going to write a LR parser (you should write an SLR(1) or LR(1) parser, not another parser in this assignment) for a toy programming language whose grammar is defined below. The grammar of this toy programming language:

program → compoundstmt
stmt → ifstmt | whilestmt | assgstmt | compoundstmt
compoundstmt → **{** stmts **}**
stmts → stmt stmts | ε
ifstmt → **if (** boolexpr **) then** stmt **else** stmt
whilestmt → **while (** boolexpr **)** stmt
assgstmt → **ID =** arithexpr **;**
boolexp → arithexpr boolop arithexpr
boolop → **<** | **>** | **<=** | **>=** | **==**
arithexpr → multexpr arithexprprime
arithexprprime → **+** multexpr arithexprprime | **-** multexpr arithexprprime | ε
multexpr → simpleexpr multexprprime
multexprprime → ***** simpleexpr multexprprime | **/** simpleexpr multexprprime | ε
simpleexpr → **ID** | **NUM** | **(** arithexpr **)**

In this grammar, *program* is the start symbol. You may assume that each token is separated with at least one white space character (for simple reading). **ID** and **NUM** are also tokens (you do not have use actual identifier or numbers, just use these names).

Before you create the parser, you should create the SLR(1) (or LR(1)) parsing tables of the grammar above (by hand). Your parser should read a program, and it should print the right most derivation of that program if it is a correct program. If the program is incorrect (contains syntax errors), it should print syntax error messages (together with line numbers). Your program should be able to recover after a syntax error, and continue to parsing.

Test your LR parser with correct and incorrect programs. The output of a test with a correct program should be the right most derivation of that program; and the output of a test with an incorrect program should be a sequence of error messages (together with line numbers). For example, the output of your parser for the following correct program

```
{ ID = NUM ; }
```

can be the following right most derivation:

program => compundstmt => { stmts } => { stmt stmts } => {stmt } => { assgstmt } => { ID = arithexpr ; }
=> { ID = multexpr arithexprprime ; } => { ID = multexpr ; } => { ID = simleexpr multexprprime ; }
=> { ID = simpleexpr ; } => { ID = NUM ; }

You should hand-in:

- Your source program
- One or two page document which includes the SLR(1) or LR(1) parsing tables of the grammar and explains how does your parser handle syntax errors (hard copy - typewritten)
- Some test runs of your program with correct and incorrect programs
- Above all email to xmju@sei.ecnu.edu.cn

DO IT YOURSELF – CHEATING WILL BE PUNISHED