Project 2
Implementation of a Recursive Descent Parser
Due Friday, May 15, 2020

1. Problem:

In this assignment you are required to use the tool ANTLR to generate a recursive descent parser for the small language Cactus. The context-free grammar for Cactus consists of the following productions:

program → **main** '(' ')' '{' *declarations statements* '}'

declarations → **int identifier** ';' *declarations*

declarations → ε

statements → *statement statements*

statements → ε

statement → **identifier** '=' *arith_expression* ';'

statement → **if** '(' *bool_expression* ')' '{' *statements* '}' **else** '{' *statements* '}' **fi**

statement → **if** '(' *bool_expression* ')' '{' *statements* '}' **fi**

statement → **while** '(' *bool_expression* ')' '{' *statements* '}'

statement → **read identifier** ';'

statement → **write** *arith_expression* ';'

statement → **return** ';'

bool_expression → *bool_expression* '||' *bool_term*

bool_expression →*bool_term*

bool_term → *bool_term* '&&' *bool_factor*

bool_term → *bool_factor*

bool_factor → '!' *bool_factor*

bool_factor → *rel_expression*

rel_expression → *arith_expression relation_op arith_expression*

relation_op → '==' | '!=' | '>' | '>=' | '<' | '<='

arith_expression → *arith_expression* '+' *arith_term*

| *arith_expression* '-' *arith_term*

| *arith_term*

arith_term → *arith_term* '*' *arith_factor*

| *arith_term* '/' *arith_factor*

| *arith_term* '%' *arith_factor*

| *arith_factor*

arith_factor → '-' *arith_factor*

| *primary_expression*

primary_expression → **integer_constant**
primary_expression → **identifier**
primary_expression → '(' *arith_expression* ') '

You could follow the following steps:

1. Edit a grammar Cactus.g that contains a parser rule for each of the productions in the above context-free grammar. Because the above context-free grammar is not an LL(1) grammar, you need to perform the left recursion elimination transformation and the left factoring transformation to transform it into an LL(1) grammar.

```
// The grammar for Cactus language
grammar Cactus;

// Parser rules
program : MAIN LP RP LB declarations statements RB
   ;
   …

// lexer rules
ELSE : 'else'
FI : 'fi'
    …
ID : …
CONST : …
ADD : '+'
    …
WHITESPACE : …
COMMENT : …
```

2. Use the ANTLR tool to generate the scanner and parser java code.

```
$antlr4 Cactus.g4
```

3. Compile the generated java code.

```
$javac Cactus*.java
```

4. Use the ANTLR tool to execute the scanner and parser.

```
$grun Cactus program -tree
```

If the input is as follows:

A sample Cactus program is given as follows:

```
/* A program to sum 1 to n */
main()
{
        int n;
        int s;
        int i;

        read n;
        if ( n < 1 ) {
              write -1;
              return;
        } else {
              s = 0;
        } fi
        i = 1;
        while ( i <= n) {
              s = s + i;
              i = i + 1;
        }
        write s;
        return;
}
```

The output should be

```
(program main ( ) { (declarations int n ; (declarations int s ; (declarations int i ;
declarations))) (statements (statement read n ;) (statements (statement if
( (bool_expression (bool_term (bool_factor (rel_expression (arith_expression
(arith_term (arith_factor (primary_expression n)) arith_term1)
arith_expression1) (relation_op <) (arith_expression (arith_term (arith_factor
(primary_expression 1)) arith_term1) arith_expression1))) bool_term1)
bool_expression1) ) { (statements (statement write (arith_expression
(arith_term (arith_factor - (arith_factor (primary_expression 1))) arith_term1)
arith_expression1) ;) (statements (statement return ;) statements)) }
(else_statement else { (statements (statement s = (arith_expression (arith_term
(arith_factor (primary_expression 0)) arith_term1) arith_expression1) ;)
statements) } fi)) (statements (statement i = (arith_expression (arith_term
(arith_factor (primary_expression 1)) arith_term1) arith_expression1) ;)
(statements (statement while ( (bool_expression (bool_term (bool_factor
(rel_expression (arith_expression (arith_term (arith_factor (primary_expression
i)) arith_term1) arith_expression1) (relation_op <=) (arith_expression
(arith_term (arith_factor (primary_expression n)) arith_term1)
arith_expression1))) bool_term1) bool_expression1) ) { (statements (statement s
= (arith_expression (arith_term (arith_factor (primary_expression s))
arith_term1) (arith_expression1 + (arith_term (arith_factor (primary_expression
i)) arith_term1) arith_expression1)) ;) (statements (statement i =
(arith_expression (arith_term (arith_factor (primary_expression i)) arith_term1)
(arith_expression1 + (arith_term (arith_factor (primary_expression 1))
arith_term1) arith_expression1)) ;) statements)) }) (statements (statement write
(arith_expression (arith_term (arith_factor (primary_expression s)) arith_term1)
arith_expression1) ;) (statements (statement return ;) statements)))))) })
```

2. Handing in your program

To turn in the assignment, upload a compressed file containing Cactus.g4, Cactus.tokens, Cactus*.java, and Cactus*.class to eCourse2 site.

3. Grading

The grading is based on the correctness of your program. The correctness will be tested by a set of test cases designed by the instructor and teaching assistants.