Static Code Analysis

Lab. 1 – Predictive top-down parsers in Java

Bartosz Bogacki '2007.

Exercise 1:

Write syntax analyzer in Java for language $a^n b^n$, where n>0. White spaces and new line characters should be ignored. If input stream is correct according to specified rules, the program should print "OK" message. Otherwise "Error" message should appear preceded with the line number where error occurred. For example, for the following input stream:

```
aaa
b
bb
```

the response of the program should be:

```
OK
```

for the input stream like:

```
aaa
ba
bb
```

the output should be:

```
2: Syntax error
```

Exercise 2:

Input stream consists of a number of rows. Each row is built of a number of characters ‘x’ followed by zero or more white spaces. Characters form triangle shape, as a number of ‘x’ characters in row is equal to the row number. This means, that in first row is only one character ‘x’, in second row there are two characters ‘x’ and so on. Write syntax analyzer in Java, that will verify that the input stream is valid according to specified rules. If so, then “OK” message should appear. Otherwise the parser should print “Error!!!” message.

For the following input stream:

```
x
xx
xxx
xxxx
xxxxx
```

The output should be:

```
OK
```

For the input stream like:

```
x
xx

xxxx
xxxxx
```

The output is:

```
Error !!!
```

Exercise 3:

Write syntax analyzer in Java for language $a^n b^n c^n$, where n>=0. White spaces and new line characters should be ignored. If input stream is correct according to specified rules, the program should print “OK” message. Otherwise “Error” message should appear preceded with the line number where error occurred. For example, for the following input stream:

```
aaa
b
bb
ccc
```

the response of the program should be:

```
OK
```

for the input stream like:

```
aaa
ba
bbbcc
```

the output should be:

```
2: Syntax error
```

Exercise 4:

Write syntax analyzer in Java for **WHILE language**.

The WHILE language is a very simple imperative language. It contains procedures and a main program, each of which consists of a sequence of simple statements plus two control structures, namely an **if-then-else** conditional and a **while** loop.

WHILE language only knows variables and constants of type **integer**, except for the **boolean** constants **true** and **false**, so there is no type information for variables. All **aexpressions** are of type **integer** and all **bexpressions** are of type **boolean**.

Constant may be any integer that can be represented with 32bit.

Identifier may contain digits and underscores but must start with a letter.

WHILE language is **case insensitive**. The grammar is like:

[verte]

```
program -> “program” declarations “begin” statements “end”
program -> “program” identifier declarations “begin” statements “end”

declarations -> declarations declaration
declarations -> ε

declaration -> “proc” identifier “(” identifier “)” “begin” statements “end”

block -> statement
block -> “(” statements “)”

statements -> statement
statements -> statements statement

statement -> “skip” “;”
statement -> identifier “:=” aexpression “;”
statement -> “if” bexpression “then” block
statement -> “if” bexpression “then” block “else” block
statement -> “while” bexpression “do” block
statement -> “call” identifier “(” aexpression “)” “;”

aexpression -> identifier
aexpression -> constant
aexpression -> aexpression “+” aexpression
aexpression -> aexpression “-” aexpression
aexpression -> aexpression “*” aexpression
aexpression -> aexpression “/” aexpression
aexpression -> “-” aexpression
aexpression -> “(” aexpression “)”

bexpression -> “true”
bexpression -> “false”
bexpression -> aexpression “<” aexpression
bexpression -> aexpression “<=” aexpression
bexpression -> aexpression “>” aexpression
bexpression -> aexpression “>=” aexpression
bexpression -> aexpression “=” aexpression
bexpression -> aexpression “<>” aexpression
bexpression -> “not” bexpression
bexpression -> “(” bexpression “)”
```

Modify given grammar to be unambiguous, eliminate left-recursion, do left factoring and write scanner and parser in Java.

Good luck! ☺