Exercise 4: Complexity Analysis

To determine the complexity of a program, metrics can be calculated from its source code, which allow an estimate of how difficult a program is to understand and maintain. There are various complexity metrics, but let us define one here that calculates the complexity of MicroJava methods from the complexity of their statements. To do this, we define the basic complexity of each statement type as follows:

- assignment 1
- method call 2
- ++ or -- 1
- if statement 1 + 2 * (complexity of nested statements)
- while statement 1 + 2 * (complexity of nested statements)
- break statement 5
- return statement 1
- read statement 1
- print statement 1

The complexity of a method is the sum of the complexities of its statements. Since nesting increases complexity, the complexity of nested statements is multiplied by a factor of 2. For example, the statement

```
if (x > 0) val = 1; else val = 2;
```

has a total complexity of 5: each assignment has a basic complexity of 1, which is multiplied by a factor of 2, resulting in a total of 4. The if statement itself has a basic complexity of 1, which is added to the total, giving 5. If the if statement were also nested, its total complexity would again be multiplied by a factor of 2, and so on.

Use Coco/R to implement a tool that calculates and prints the complexity of all methods in a MicroJava program. Use the MicroJava grammar from Appendix A as the parser specification and turn it into an attributed grammar such that the complexity of methods is calculated from the basic complexities of their statements. You can ignore function calls in expressions.

Solution

As terminal symbols, we have ident, number and charConst, which are declared in the scanner specification. In addition to blanks, line endings and tabs should be ignored.

We use the MicroJava grammar from Appendix A as the parser specification. Most productions do not require special processing. We only need to process method declarations, statements and statement blocks.

```
COMPILER Metrics

//------------------ global semantic declarations -----------------
   static int factor = 2; // increased complexity factor for nested statements

//---------------------- scanner specification -----------------------
CHARACTERS
   letter      = 'A'..'Z' + 'a'..'z'.
   digit       = '0'..'9'.
   noQuote = ANY - '\''.

TOKENS
   ident        = letter {letter | digit | '_'}.
   number     = digit {digit}.
   charConst = '\'' (noQuote | '\\' ('r' | 'n' | 't')) '\''.

COMMENTS FROM "//" TO "\r\n"
IGNORE '\r' + '\n' + '\t'
```

```
//---------------------- parser specification ------------------------
PRODUCTIONS

Metrics      = "program" ident {ConstDecl | VarDecl | ClassDecl} "{" {MethodDecl} "}".
ConstDecl = "final" Type ident "=" (number | charConst) ";".
VarDecl     = Type ident {"," ident } ";".
ClassDecl  = "class" ident "{" {VarDecl} "}".

MethodDecl                       (. int complexity; .)
=  (Type | "void") ident         (. String name = t.val; .)
   "(" [FormPars] ")"
   {VarDecl}
   Block <out complexity>        (. System.out.println(name + ": " + complexity); .) .

Block <out int complexity>       (. int c; .)
=  "{"                           (. complexity = 0; .)
   { Statement <out c>           (. complexity += c; .)
   }
   "}".

Statement <out int complexity>(. int c; .)
=                                (. complexity = 1; .) // default
   (  Designator  ( "=" Expr
                  | ActPars      (. complexity = 2; .)
                  | "++"
                  | "--"
                  ) ";"
   |  "if" "(" Condition ")"
      Statement <out c>          (. complexity += c * factor; .)
      [  "else"
         Statement <out c>       (. complexity += c * factor; .)
      ]
   |  "while" "(" Condition ")"
      Statement <out c>          (. complexity += c * factor; .)
   |  "break" ";"                (. complexity = 5; .)
   |  "return" [Expr] ";"
   |  "read" "(" Designator ")" ";"
   |  "print" "(" Expr ["," number] ")" ";"
   |  Block <out complexity>
   |  ";"
   ).

FormPars  =  Type ident  {"," Type ident}.
Type        =  ident ["[" "]"].
ActPars    =  "(" [ Expr {"," Expr}] ")".
Condition  =  CondTerm {"||" CondTerm}.
CondTerm =  CondFact {"&&" CondFact}.
CondFact  =  Expr RelOp Expr.
RelOp       =  "==" | "!=" | ">" | ">=" | "<" | "<=".
Expr         =  ["-"] Term {AddOp Term}.
Term         =  Factor {MulOp Factor}.
Factor      =  Designator [ActPars]
             |  number
             |  charConst
             |  "new" ident ["[" Expr "]"]
             |  "(" Expr ")".
Designator=  ident {"." ident | "[" Expr "]"}.
AddOp      =  "+" | "-".
MulOp       =  "*" | "/" | "%".

END Metrics.
```

In MethodDecl, we get the complexity of the method's block as the output attribute of Block and print it together with the name of the method.

In Statement, we calculate the complexity of a single statement, which is 1 for most statement types, so we set complexity to 1 by default and only change it for the statements that have a complexity other than 1. if and while statements have nested statements, whose complexity we get via the output attribute c, which we multiply by a factor of 2 and add it to the complexity of the if or while statement. A statement can also be a block that returns the complexity of the statements in it via an output attribute.

This is all we have to do! We still need a main program that gets the name of the input file as a command line argument, creates a scanner and a parser and starts the syntax analysis:

```
class Metrics {
   public static void main (String[] arg) {
      Scanner scanner = new Scanner(arg[0]);
      Parser parser = new Parser(scanner);
      parser.Parse();
      System.out.println(parser.errors.count + " errors detected");
   }
}
```

We process the compiler specification with Coco/R:

```
java -jar Coco.jar Metrics.atg
```

thus generating a scanner and a parser. Then we compile all Java files

```
javac Scanner.java Parser.java Metrics.java
```

and invoke our complexity analyzer for a specific MicroJava sample program (e.g. Eratos.mj):

```
java Metrics Eratos.mj
```

If Eratos.mj contains the following MicroJava program

```
program Eratos
   char[] sieve;
   int max;                  // maximum prime to be found
   int npp;                  // numbers per page
{
   void put (int x)
   {  if (npp == 10) {print(chr(13)); print(chr(10)); npp = 0;}
      print(x, 5);
      npp = npp + 1;
   }

   void found (int x)
      int i;
   {  put(x);
      i = x; while (i <= max) {sieve[i] = 'o'; i = i + x;}
   }

   void main()
      int i, ready;
   {  read(max); npp = 0; sieve = new char[max+1];
      i = 0; while (i <= max) {sieve[i] = 'x'; i = i + 1;}
      i = 2;
      while (i <= max) {
         found(i);
         ready = 0;
         while (ready == 0) {
            if (i > max) ready = 1;
            else if (sieve[i] == 'x') ready = 1;
            else i = i + 1;
         }
      }
   }
}
```

the invocation of Metrics results in the following output:

```
put: 9
found: 8
main: 71
```