INF2100

Oppgaver uke 40 og 41 2014

For å få trening i å programmere en parser, skal vi aller først gjøre det for et veldig enkelt programmeringsspråk: **E**. Dette språket består av uttrykk med de fire vanligste regneartene +, –, * og /; grammatikken[1] ser du i figur 1 på neste side. Tallkonstanter består av bare ett siffer[2] og variabelnavn er på bare én bokstav. Tre variabler er forhåndsdefinert:[3]

M	1000
C	100
X	10

En implementasjon[4] av en interpret[5] av dette språket ser slik ut:[6]

```
import java.io.*;
import java.util.*;

class E {
    public static void main(String arg[]) {
        Scanner.init();

        Program p = Program.parse();
        if (Scanner.curToken != Token.eofToken)
            Error.error("Syntax error: Illegal "+Scanner.curToken);
        p.printTree();  Log.writeln();

        System.out.println("The value is "+p.eval());
    }
}

abstract class SyntaxUnit {
    abstract long eval();
    abstract void printTree();
}

class Program extends SyntaxUnit {
    Expression e;

    @Override long eval() {
        return 0;  //** Endres
    }
```

[1] Grammatikken til E viser at E har operatorer med ulik prioritet: * og / binder sterkere enn + og – så uttrykket `2+3*5` gir verdien 17.

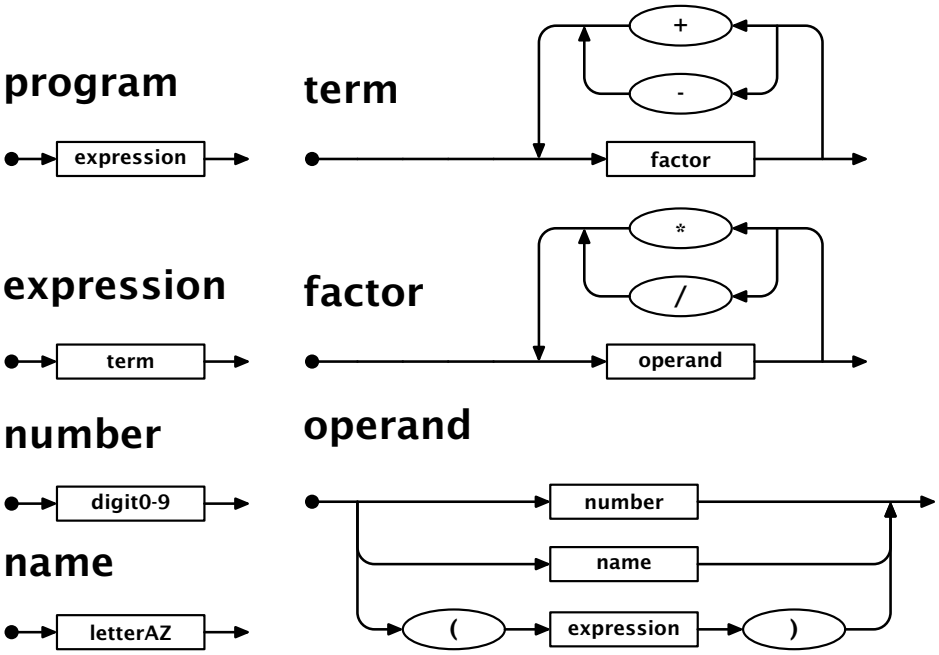
[2] Selv om konstantene bare har ett siffer, kan godt resultatet bli større; se eksemplet.

[3] Det er ikke mulig å definere flere variabler selv (i tilfelle du lurte på det).

[4] I denne implementasjonen av E får vi lov å bruke alt vi ønsker fra Java-biblioteket.

[5] Siden dette er en *interpret* og ikke en vanlig *kompilator,* vil den beregne verdien av uttrykket i stedet for å lage kode som gjør det. Men både en interpret og en kompilator vil først gjøre en syntaksanalyse (en «parsering») av koden.

[6] Denne koden finnes i `~inf2100/kode/E/E.java` som du også finner på nettet ved å klikke på denne linken: `http://inf2100.at.ifi.uio.no/kode/E/E.java`.



Figur 1: Grammatikken til språket E

```
    static Program parse() {
        return null;  //** Endres
    }

    @Override void printTree() {
        //** Endres
    }
}

class Expression extends SyntaxUnit {
    Term t = new Term();

    @Override long eval() {
        return t.eval();
    }

    static Expression parse() {
        Expression e = new Expression();

        Log.enterParser("<expression>");
        e.t = Term.parse();
        Log.leaveParser("</expression>");
        return e;
    }

    @Override void printTree() {
        t.printTree();
    }
}

class Term extends SyntaxUnit {
    List<Factor> factors = new ArrayList<Factor>();
    List<Token> opers = new ArrayList<Token>();

    @Override long eval() {
        return 0;  //** Endres
```

```
    }

    static Term parse() {
        return null;  //** Endres
    }

    @Override void printTree() {
        //** Endres
    }
}

class Factor extends SyntaxUnit {
    List<Operand> operands = new ArrayList<Operand>();
    List<Token> opers = new ArrayList<Token>();

    @Override long eval() {
        return 0;  //** Endres
    }

    static Factor parse() {
        return null;  //** Endres
    }

    @Override void printTree() {
        //** Endres
    }
}

abstract class Operand extends SyntaxUnit {
    static Operand parse() {
        return null;  //** Endres
    }
}

class OperandName extends Operand {
    char id;

    @Override long eval() {
        return 0;  //** Endres
    }

    static OperandName parse() {
        return null;  //** Endres
    }

    @Override void printTree() {
        //** Endres
    }
}

class OperandNumber extends Operand {
    long n;

    @Override long eval() {
        return 0;  //** Endres
    }

    static OperandNumber parse() {
        return null;  //** Endres
    }
```

```
    @Override void printTree() {
        //** Endres
    }
}

class OperandExpr extends Operand {
    Expression e;

    @Override long eval() {
        return 0;  //** Endres
    }

    static OperandExpr parse() {
        return null;  //** Endres
    }

    @Override void printTree() {
        //** Endres
    }
}

enum Token { nameToken, numberToken, plusToken, minusToken, mulToken,
        divToken, leftParToken, rightParToken, eofToken }

class Scanner {
    public static Token curToken;
    public static char curName;
    public static int curNumber;

    private static LineNumberReader f;

    public static void init() {
        f = new LineNumberReader(new InputStreamReader(System.in));
        readNext();
    }

    public static void readNext() {
        curToken = null;
        while (curToken == null) {
            int c = '?';
            try {
                c = f.read(); // Read one character
            } catch (IOException e) {
                Error.error("Read error!");
            }

            if (c < 0) {
                curToken = Token.eofToken;
            } else if (c == '+') {
                curToken = Token.plusToken;
            } else if (c == '-') {
                curToken = Token.minusToken;
            } else if (c == '*') {
                curToken = Token.mulToken;
            } else if (c == '/') {
                curToken = Token.divToken;
            } else if (c == '(') {
                curToken = Token.leftParToken;
            } else if (c == ')') {
                curToken = Token.rightParToken;
            } else if ('A'<=c && c<='Z' || 'a'<=c && c<='z') {
```

```
                curToken = Token.nameToken;  curName = (char)c;
            } else if (Character.isDigit(c)) {
                curToken = Token.numberToken;  curNumber = c-'0';
            } else if (Character.isWhitespace(c)) {
                // Ignore space
            } else {
                Error.error("Illegal character: '"+(char)c+"'!");
            }
        }
        // For testing:
        // System.out.println("Scanner: Read a "+curToken);
    }
}

class Error {
    static void error(String message) {
        System.err.println("ERROR: "+message);
        System.exit(1);
    }
}

class Log {
    public static boolean doLogParser = false, doLogTree = false;
    private static int parseLevel = 0;

    public static void enterParser(String symbol) {
        if (! doLogParser) return;
        for (int i = 1;  i <= parseLevel;  ++i)
            System.out.print("  ");
        System.out.println(symbol);
        ++parseLevel;
    }

    public static void leaveParser(String symbol) {
        if (! doLogParser) return;
        --parseLevel;
        for (int i = 1;  i <= parseLevel;  ++i)
            System.out.print("  ");
        System.out.println(symbol);
    }

    public static void write(String s) {
        if (! doLogTree) return;
        System.out.print(s+" ");
    }

    public static void writeln() {
        if (! doLogTree) return;
        System.out.println();
    }
}
```

Eksempel

Om vi sender inn programmet

```
M+M+X+1
```

skal svaret av evalueringen bli

```
> java E <e1.e
The value is 2011
```

Oppgave 1

Tegn parseringstreet til programmet i eksemplet over og til eksemplet i oppgave 4.

Oppgave 2

Fyll ut de tomme utgavene av parse og eval.

Tilleggsspørsmål 2a

Trenger vi klassen SyntaxUnit? Hva skjer om vi fjerner den (og alle referanser til den)?

Oppgave 3

Legg inn kall på enterParser og leaveParser i hver parse-metode slik at vi kan følge kallene.

Eksempel

Om vi sender inn dette uttrykket

```
5 + 2*C - 2*(M-1)/3
```

bør resultatet bli slik:

```
<program>
  <expression>
    <term>
      <factor>
        <operand> (a number)
        </operand>
      </factor>
      <factor>
        <operand> (a number)
        </operand>
        <operand> (a name)
        </operand>
      </factor>
      <factor>
        <operand> (a number)
        </operand>
        <operand> (an inner expression)
          <expression>
            <term>
              <factor>
                <operand> (a name)
                </operand>
              </factor>
              <factor>
                <operand> (a number)
                </operand>
              </factor>
            </term>
          </expression>
        </operand>
        <operand> (a number)
```

```
        </operand>
      </factor>
    </term>
  </expression>
</program>
```

Oppgave 4

Legg inn en metode printTree i hver klasse slik at den interne formen av uttrykket blir skrevet ut.

Eksempel

Om vi sender inn eksemplet fra oppgave 1, skal resultatet bli

```
M + M + X + 1
```