IN2030

Løsningsforslag

Uke 38 2023

Oppgave 1–3

De ferdige klassene for å skanne og parsere minispråket E. (Denne koden finnes også i ~inf2100/e/e2/.)

E

```
class E {
    public static void main(String arg[]) {
        Scanner s = new Scanner(arg[0]);
        Expression e = Expression.parse(s);
        e.prettyPrint();  System.out.println();
    }
}
```

ESyntax

```
abstract class ESyntax {
    abstract void prettyPrint();

    static int parseLevel = 0;

    static void enterParser(String nonterm) {
        for (int i = 1;  i <= parseLevel;  ++i)
            System.out.print("  ");
        System.out.println("<"+nonterm+">");
        parseLevel++;
    }

    static void leaveParser(String nonterm) {
        parseLevel--;
        for (int i = 1;  i <= parseLevel;  ++i)
            System.out.print("  ");
        System.out.println("</"+nonterm+">");
    }
}
```

Expression

```
abstract class Expression extends ESyntax {
    static Expression parse(Scanner s) {
        enterParser("expression");

        Expression e = Term.parse(s);

        leaveParser("expression");
        return e;
    }
}
```

Term

```
import java.util.ArrayList;
```

```
class Term extends Expression {
    ArrayList<Factor> operands = new ArrayList<>();
    ArrayList<Token> oprs = new ArrayList<>();

    static Term parse(Scanner s) {
        enterParser("term");

        Term t = new Term();
        t.operands.add(Factor.parse(s));
        while (s.curToken().kind == TokenKind.plusToken ||
               s.curToken().kind == TokenKind.minusToken) {
            t.oprs.add(s.curToken());
            s.readNextToken();
            t.operands.add(Factor.parse(s));
        }

        leaveParser("term");
        return t;
    }

    @Override
    void prettyPrint() {
        operands.get(0).prettyPrint();
        for (int i = 1;  i < operands.size();  i++) {
            System.out.print(" " + oprs.get(i-1).kind + " ");
            operands.get(i).prettyPrint();
        }
    }
}
```

Factor

```
import java.util.ArrayList;

class Factor extends ESyntax {
    ArrayList<Atom> operands = new ArrayList<>();
    ArrayList<Token> oprs = new ArrayList<>();

    static Factor parse(Scanner s) {
        enterParser("factor");

        Factor f = new Factor();
        f.operands.add(Atom.parse(s));
        while (s.curToken().kind == TokenKind.astToken ||
               s.curToken().kind == TokenKind.slashToken) {
            f.oprs.add(s.curToken());
            s.readNextToken();
            f.operands.add(Atom.parse(s));
        }

        leaveParser("factor");
        return f;
    }

    @Override
    void prettyPrint() {
        operands.get(0).prettyPrint();
        for (int i = 1;  i < operands.size();  i++) {
            System.out.print(" " + oprs.get(i-1).kind + " ");
            operands.get(i).prettyPrint();
        }
    }
}
```

Atom

```
abstract class Atom extends ESyntax {
    static Atom parse(Scanner s) {
        enterParser("atom");

        Atom a;
        if (s.curToken().kind == TokenKind.leftParToken)
```

```
            a = InnerExpr.parse(s);
        else
            a = Number.parse(s);

        leaveParser("atom");
        return a;
    }
}
```

InnerExpr

```
class InnerExpr extends Atom {
    Expression expr;

    static InnerExpr parse(Scanner s) {
        enterParser("inner expr");

        InnerExpr ie = new InnerExpr();
        s.readNextToken();  // Skip past '('.
        ie.expr = Expression.parse(s);
        s.readNextToken();  // Skip past ')'.

        leaveParser("inner expr");
        return ie;
    }

    @Override
    void prettyPrint() {
        System.out.print("(");  expr.prettyPrint();
        System.out.print(")");
    }
}
```

Number

```
class Number extends Atom {
    int val;

    static Number parse(Scanner s) {
        enterParser("number");

        Number n = new Number();
        n.val = s.curToken().numVal;
        s.readNextToken();

        leaveParser("number");
        return n;
    }

    @Override
    void prettyPrint() {
        System.out.print(val);
    }
}
```

Scanner

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

class Scanner {
    private LineNumberReader sourceFile = null;
    private ArrayList<Token> curLineTokens = new ArrayList<>();

    public Scanner(String fileName) {
        try {
            sourceFile = new LineNumberReader(
                            new InputStreamReader(
                                new FileInputStream(fileName),
                                "UTF-8"));
        } catch (IOException e) { }
        readNextLine();
    }

    public Token curToken() {
        return curLineTokens.get(0);
    }

    public void readNextToken() {
        if (! curLineTokens.isEmpty())
            curLineTokens.remove(0);
    }

    private void readNextLine() {
        String line = null;
        try {
            line = sourceFile.readLine();
        } catch (IOException e) { }

        // Find all the tokens:
        int pos = 0;
        while (pos < line.length()) {
            char c = line.charAt(pos++);

            if (Character.isWhitespace(c)) {
                // Ignore spaces.
            } else if (isDigit(c)) {
                Token t = new Token(TokenKind.integerToken);
                t.integerLit = c - '0';
                curLineTokens.add(t);
            } else if (c == '*') {
                curLineTokens.add(new Token(TokenKind.astToken));
            } else if (c == '(') {
                curLineTokens.add(new Token(TokenKind.leftParToken));
            } else if (c == '-') {
                curLineTokens.add(new Token(TokenKind.minusToken));
            } else if (c == '+') {
                curLineTokens.add(new Token(TokenKind.plusToken));
            } else if (c == ')') {
                curLineTokens.add(new Token(TokenKind.rightParToken));
            } else if (c == '/') {
                curLineTokens.add(new Token(TokenKind.slashToken));
            }
        }
        curLineTokens.add(new Token(TokenKind.eofToken));
    }

    private boolean isDigit(char c) {
        return '0'<=c && c<='9';
    }
}
```

Token

```
class Token {
    TokenKind kind;
    long integerLit;

    Token(TokenKind k) {
        kind = k;
    }

    String showInfo() {
        String t = kind + " token";
        if (kind == TokenKind.integerToken)
            t += ": " + integerLit;
        return t;
    }
}
```

TokenKind

```
enum TokenKind {
    integerToken("integer literal"),

    astToken("*"),
    leftParToken("("),
    minusToken("-"),
    plusToken("+"),
    rightParToken(")"),
    slashToken("/"),

    eofToken("E-o-f");

    private String image;

    TokenKind(String im) {
        image = im;
    }

    public String toString() {
        return image;
    }
}
```