Autor: Melanie Galip

Sprachkonzepte

Übungsaufgabe 2

Aufgabe 2 a)

```
package formatSpecification;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class FormatSpecification {
    private static final String frmatSpecifier
    /*    Formatspecifikation Syntax:
        %[argument_index$][flags][width][.precision]conversion
        argument_index: arg -> 1$, arg2 -> 2$, ...
         width; Feldgröße; minimale Ausgabebreite (Anzahl Zeichen)
         .precision: '.' gefolgt von Anzahl an Nachkommastellen
         minimale Formatspezifikation ist %conversion; nicht optional
         %conversion: einzelnes Zeichen, das spezifiziert wie ein
         Argument dargestellt wird: d, f, c, s, n ... */
    = "%(\\d+\\$)?([-#+ 0,(\\<]*)?(\\d+)?(\\.\\d+)?([a-zA-Z])";
// regex; zu allgemein; Rest und Datum/Zeit

    private static final Pattern formatToken =
Pattern.compile(formatSpecifier); // regex wird übergeben



    public static void main(String[] args) {

        String a = "xxx %d yyy%n";
        /* Ausgabe:
            %d 4 6
            %n 10 12 */
        // String a = "xxx%1$d yyy"; // Ausgabe: %1$d 3 7
        // String a = "%1$-02.3dyyy"; // Ausgabe: %1$-02.3d 0 9

        // iterieren über den String
        // übergeben den String a, auf den wir matchen wollen
        Matcher matcher = formatToken.matcher(a);

        // hat eine Methode find(), die true zurückgibt,
        // falls im String, den wir matchen wollen,
```

```
        // ein pattern gefunden wurde
         while(matcher.find()) {
             // mit matcher.group() kriegen wir die pattern heraus
             // start() und end() gibt die Indizes der pattern aus
             System.out.println(matcher.group() + " " +
             matcher.start() + " " + matcher.end());
         }

    }

}
```

Aufgabe 2b)

```
lexer grammar TimeFormatLexer;


CLOCK        : HOUR ':' MINUTE;

fragment MINUTE        : [0-5][0-9] ' ' SPEC;

fragment SPEC        : ('am' | 'pm');


fragment HOUR        : '1'[012]|[1-9];
```

Übungsaufgabe 3

Aufgabe 3a)

```
grammar Expr;

/* the grammar name and file name must match */

@header {
    package antlr;
}

// first production rules; lower case convention; parser grammar

// Start Variable
prog: (decl | expr)+ EOF;

decl: ID ':' INT_TYPE '=' NUM;

/* the earlier the prod rule
 * the higher the precedence
```

```
 * ANTLR resolves ambiguities in favor of the
 * alternatives given first.
 */

 /* I want the antlr tool to generate a visit
  * method for every alternative
  * daher # .. define labels besides the prod. grammar */
expr: expr '*' expr
    | expr '+' expr
    | ID
    | NUM;

// then specify Tokens; lexer grammar
ID : [a-z][a-zA-Z0-9_]*;
NUM : '0' | '-'?[1-9][0-9]*;
INT_TYPE : 'INT';
// comments and white spaces should be ignored
COMMENT : '--' ~[\r\n]* -> skip;
WS : [ \t\n]+ -> skip;
```

Syntax, um Parse Tree zu erstellen:

```
grun Expr prog -gui &
```

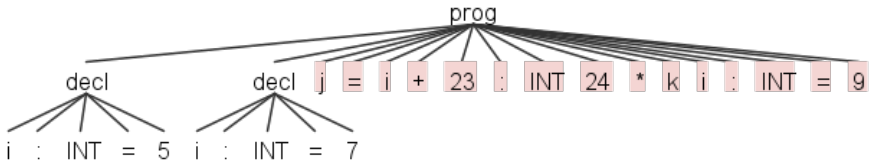
oder, wenn man test files hat:

```
grun nameGrammar StartVar test.txt -gui &
```

Im folgenden möchte anhand einiger Testeingaben den Parse Tree ableiten:

```
grun Expr prog -gui &
i : INT = 5
 i: INT = 7
j = i + 23 : INT
24 * k
 i: INT = 9
```

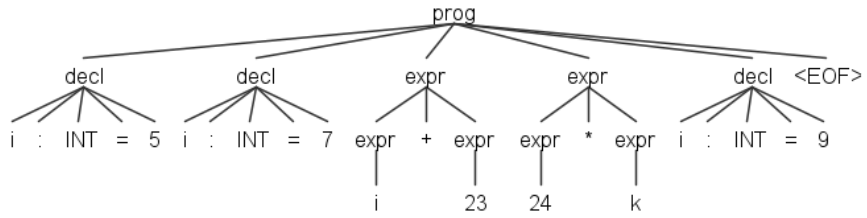
liefert




```
grun Expr prog -gui &
i: INT = 5
 i: INT = 7
```

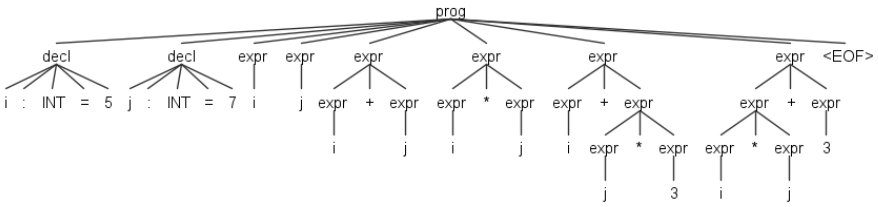
```
i + 23
24 * k
   i: INT = 9
```

liefert




```
grun Expr prog -gui &
i: INT = 5
j: INT = 7
i
j
i + j
i * j
i + j * 3
i * j + 3
```

liefert



Aufgabe 3b) + Übungsaufgabe 4

```
grammar Expr;

/* the grammar name and file name must match */

@header {
    package antlr;
}
```

```
// first production rules; lower case convention; parser grammar

// Start Variable
prog: (decl | expr)+ EOF        # Program;

decl: ID ':' INT_TYPE '=' NUM    # Declaration;

/* the earlier the prod rule
 * the higher the precedence
 * ANTLR resolves ambiguities in favor of the
 * alternatives given first.
 */

 /* I want the antlr tool to generate a visit method
  * for every alternative
  * daher # .. define labels besides the prod. grammar */
expr: expr '*' expr                # Multiplication
    | expr '+' expr                # Addition
    | ID                         # Variable
    | NUM                          # Number;

// then specify Tokens; lexer grammar
ID : [a-z][a-zA-Z0-9_]*;
NUM : '0' | '-'?[1-9][0-9]*;
INT_TYPE : 'INT';
// comments and white spaces should be ignored
COMMENT : '--' ~[\r\n]* -> skip;
WS : [ \t\n]+ -> skip;
```

```
package app;

import java.io.IOException;

import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTree;

import antlr.ExprLexer;
import antlr.ExprParser;
import expression.AntlrToProgram;
import expression.ExpressionProcessor;
import expression.MySyntaxErrorListener;
import expression.Program;
```

```
public class ExpressionApp {

    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.print("Usage: file name\n");
        } else {
            // filename als arg
            String filename = args[0];
            // parser starten und filename als param übergeben
            ExprParser parser = getParser(filename);
            /*
             * method prog() in the ExprParser ist the starting point
             * for the compilation.
             * So as soon as you invoke the particular method
             * its gonna start parsing
             */

            /*
             * tell ANTLR to build a parse tree parse from
             * the start symbol 'prog'
             */
            ParseTree antlrAST = parser.prog();

            if (MySyntaxErrorListener.hasError) {
                // let the syntax error be reported
            } else {

                /*
                 * create a tree from our particular
                 * expression package/ model structure in the
                 * expression package create a visitor
                 * for converting the parse tree into
                 * Program/Expression object
                 */

                // the top Level is the AntlrToProgram
                AntlrToProgram progVisitor = new AntlrToProgram();
                Program prog = progVisitor.visit(antlrAST);

                /*
                 * adding listener to the parser keep track
                 * of syntax error that occured during
                 * the parsing process see class MySyntaxErrorListener
                 */

                if (progVisitor.semanticErrors.isEmpty()) {
```

```
                /*
                 * if there is no error,
                 * printing out the evaluation result
                 */

                // übergeben eine Liste von Expressions
                ExpressionProcessor ep =
                new ExpressionProcessor(prog.expressions);
                /* iterieren über die Liste.. und evaluieren,
                    wenn möglich */
                for (String evaluation : ep.getEvaluationsResults())
                {
                    System.out.println(evaluation);
                }

            } else { // there are errors; print them out
                for (String err : progVisitor.semanticErrors) {
                    System.out.println(err);
                }
            }

        }

    }
}

/*
 * Here the types of parser and lexer are
   specific to the grammar name Expr.g4.
 */
private static ExprParser getParser(String fileName) {
    ExprParser parser = null;

    try {
        CharStream input = CharStreams.fromFileName(fileName);
        /* Lexer enthält den input,
        der die einzelnen tokens beinhaltet */
        ExprLexer lexer = new ExprLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        parser = new ExprParser(tokens);

        // syntax error handling
        parser.removeErrorListeners();
        parser.addErrorListener(new MySyntaxErrorListener());
    } catch (IOException e) {
        // TODO Auto-generated catch block
```

```
            e.printStackTrace();
        }

        return parser;
    }

}
```

```
package expression;

public abstract class Expression {

}
```

```
package expression;

import java.util.ArrayList;
import java.util.List;

public class Program {
    public List<Expression> expressions;

    public Program() {
        this.expressions = new ArrayList<>();
    }

    public void addExpression(Expression e) {
        expressions.add(e);
    }
}
```

```
package expression;

public class Variable extends Expression {
    String id;

    public Variable(String id) {
        this.id = id;
    }

    @Override
    public String toString() {
        return id;
    }
}
```

```
package expression;
```

```
public class VariableDeclaration extends Expression {

    // int i = 5;
    // type id = value
    public String id;
    public String type;
    public int value;

    public VariableDeclaration(String id, String type, int value) {
        this.id = id;
        this.type = type;
        this.value = value;
    }
}
```

```
package expression;

public class Number extends Expression {
    int num;

    public Number(int num) {
        this.num = num;
    }

    @Override
    public String toString() {
        return new Integer(num).toString();
    }
}
```

```
package expression;

public class Multiplication extends Expression {
    Expression left;
    Expression right;

    public Multiplication(Expression left, Expression right) {
        this.left = left;
        this.right = right;
    }

    @Override
    public String toString() {
        return left.toString() + " * " + right.toString();
    }

}
```

```java
package expression;

public class Addition extends Expression {
    Expression left;
    Expression right;

    public Addition(Expression left, Expression right) {
        this.left = left;
        this.right = right;
    }

    @Override
    public String toString() {
        return left.toString() + " + " + right.toString();
    }

}
```

```java
package expression;

import java.util.Collections;
import java.util.List;

import org.antlr.v4.runtime.BaseErrorListener;
import org.antlr.v4.runtime.Parser;
import org.antlr.v4.runtime.RecognitionException;
import org.antlr.v4.runtime.Recognizer;
import org.antlr.v4.runtime.Token;

public class MySyntaxErrorListener extends BaseErrorListener {
    public static boolean hasError = false;

    @Override
    public void syntaxError(Recognizer<?, ?> recognizer,
                            Object offendingSymbol,
                            int line, int charPositionInLine,
            String msg, RecognitionException e) {
        hasError = true;

        List<String> stack =
        ((Parser) recognizer).getRuleInvocationStack();
        Collections.reverse(stack);
        System.err.println("Syntax Error!");
        System.err.println("Token " + "\"" +
        ((Token) offendingSymbol).getText() + "\"" +
            " (line " + line + ", column "
```

```
                + (charPositionInLine + 1) + ")" + ": " + msg);
        System.err.println("Rule Stack: " + stack);

    }

}
```

```
package expression;

import java.util.ArrayList;
import java.util.List;

import antlr.ExprBaseVisitor;
import antlr.ExprParser.ProgramContext;

public class AntlrToProgram extends ExprBaseVisitor<Program> {

    // to be accessed by the main application program
    public List<String> semanticErrors;

    public Program visitProgram(ProgramContext ctx) {
        Program prog = new Program();

        semanticErrors = new ArrayList<>();
        /* a helper visitor for transforming each subtree
        into an Expression object*/
        AntlrToExpression exprVisitor =
        new AntlrToExpression(semanticErrors);
        for (int i = 0; i < ctx.getChildCount(); i++) {
            if (i == ctx.getChildCount() - 1) {
                /* last child of the start symbol is EOF */
                /* do not visit this child and attempt
                   to convert it to an Expression object */
            } else {
                prog.addExpression(exprVisitor.visit(ctx.getChild(i)));
            }
        }

        return prog;

    }

}
```

```
package expression;

import java.util.ArrayList;
```

```java
import java.util.List;

import org.antlr.v4.runtime.Token;

import antlr.ExprBaseVisitor;
import antlr.ExprParser.AdditionContext;
import antlr.ExprParser.DeclarationContext;
import antlr.ExprParser.MultiplicationContext;
import antlr.ExprParser.NumberContext;
import antlr.ExprParser.VariableContext;

public class AntlrToExpression extends ExprBaseVisitor<Expression> {

    /*
     * Given that all visit_* methods are called in a top-down fashion,
     * we can be sure that the order in which we add declared variables
     * in the 'vars' is identical to how they are declared in the input
     * program. The way nodes are visites corresponds to the the
     * order of lines in input program (test dateien mit input..)
     */
    private List<String> vars;
    // stores all the variables declared in the program so far
    private List<String> semanticErrors;
    /* 1. duplicate declaration 2. reference to undeclared variable
     * note that semantic errors are different from syntax errors.
     * syntax errors are not grammatically correct
     * semantic errors ... meaning */

    public AntlrToExpression(List<String> semanticErrors) {
        vars = new ArrayList<>();
        this.semanticErrors = semanticErrors;
    }

    /*
     * schaut man sich die grammar an, sieht man,
     * dass die parser regel decl 5
     * childs erzeugt (ID, ':', INT_TYPE, '=' und NUM).
     * Indexieren ab 0, daher 0 bis
     * 4 im parse tree sowie Liste...
     */
    @Override
    public Expression visitDeclaration(DeclarationContext ctx) {
        /* important where the mistake is located
         * for example test1.txt duplicate declaration in line 2 column 2
         * record for particular context ctx in params whats the line
         * and whats the column number
```

```
     * ID() is a method generated to correspond to the token ID
     * to the source grammar */

    // äquivalent to: ctx.getChild(0).getSymbol()
    Token idToken = ctx.ID().getSymbol();
    int line = idToken.getLine();
    int column = idToken.getCharPositionInLine() + 1;

    String id = ctx.getChild(0).getText();
    // maintaining the vars list for semantic errors reporting
    if (vars.contains(id)) {
        semanticErrors.add("Error: variable " + id +
        " already declared (" + line + ", " + column + ")");
    } else {
        vars.add(id);
    }
    String type = ctx.getChild(2).getText();
    int value = Integer.parseInt(ctx.NUM().getText());
    return new VariableDeclaration(id, type, value);
}

@Override
public Expression visitMultiplication(MultiplicationContext ctx) {

    // recursively visit the left subtree of the
    // current Multiplication
    Expression left = visit(ctx.getChild(0)); // node
    Expression right = visit(ctx.getChild(2));
    return new Multiplication(left, right);
}

@Override
public Expression visitAddition(AdditionContext ctx) {
    Expression left = visit(ctx.getChild(0));
    Expression right = visit(ctx.getChild(2));
    return new Addition(left, right);
}

/*
 * wenn ich versuche, eine Variable zu besuchen,
 * woher soll ich wissen, ob diese
 * Variable wirklich existiert, also schon deklariert wurde?
 * zum Bsp.
 * i: INT = 5
 * i: INT = 7
 * i = 23
```

```
     * 24 * k
     * i: INT = 9
     *
     * k wurde vorher zum Bsp. nicht deklariert i wurde sogar zwei mal
     * deklariert.
     * zwei Arten von Fehlern: keine grammatikalischen Fehler da die
     * Satzstruktur in Ordnung ist Prod.regeln eingehalten,
     * aber wir haben hier semantic errors..
     * daher oben weitere Datenstruktur: Liste, die semantische Fehler
     * speichert,
     * erstellt
     */
    @Override
    public Expression visitVariable(VariableContext ctx) {
        // entweder über token name ID oder über getchild(index: 0)
        Token idToken = ctx.ID().getSymbol();
        int line = idToken.getLine();
        int column = idToken.getCharPositionInLine() + 1;

        String id = ctx.getChild(0).getText();
        if (!vars.contains(id)) {
            semanticErrors.add("Error: variable " + id +
            " not declared (" + line + ", " + column + ")");
        }
        return new Variable(id);
    }

    @Override
    public Expression visitNumber(NumberContext ctx) {
        String numText = ctx.getChild(0).getText();
        int num = Integer.parseInt(numText);
        return new Number(num);
    }

}
```

```
package expression;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/*
 * this class will be used for our main program
 * of the compiler */
```

```
public class ExpressionProcessor {
    List<Expression> list;
    // symbol table for storing values of variable
    public Map<String, Integer> values;

    public ExpressionProcessor(List<Expression> list) {
        this.list = list;
        values = new HashMap<>();
    }

    // Bsp. i + j * 3; wollen das Ergebnis als String in einer Liste
    // speichern
    public List<String> getEvaluationsResults() {
        List<String> evaluations = new ArrayList<>();

        // dafür müssen wir über die Liste von Expressions iterieren
        for (Expression e : list)
            if (e instanceof VariableDeclaration) {
                VariableDeclaration decl = (VariableDeclaration) e;
                values.put(decl.id, decl.value);
            } else {
            // e instanceof Number, Variable, Addition, Multiplication
                // all these can be evaluated
                // input is the source expression, ex. i + j * 3
                String input = e.toString();
                int result = getEvalResult(e);
                evaluations.add(input + " is " + result);

            }

        return evaluations;
    }

    private int getEvalResult(Expression e) {
        int result = 0;

        if (e instanceof Number) {
            Number number = (Number) e;
            result = number.num;
        } else if (e instanceof Variable) {
            Variable var = (Variable) e;

            /*
             * Returns the value to which the specified key is mapped,
             * or null if this map
```

```
             * contains no mapping for the key.
             */

            // bekommen also den int wert für diese var
            result = values.get(var.id);

        } else if (e instanceof Addition) {
            Addition add = (Addition) e;
            // wird rekursiv aufgerufen
            int left = getEvalResult(add.left);
            int right = getEvalResult(add.right);
            result = left + right;
        } else { // e instance of Multiplication
            Multiplication mult = (Multiplication) e;
            int left = getEvalResult(mult.left);
            int right = getEvalResult(mult.right);
            result = left * right;

        }

        return result;

    }

}
```

Übungsaufgabe 5

Aufgabe 5a)

```
import java.nio.file.Files;
import java.nio.file.Paths;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public final class Procedural {
    private Procedural() { }

    private static final int MIN_LENGTH = 20;

    public static void main(String[] args) throws IOException {
        var input = Files.newBufferedReader(Paths.get("sample1.txt"));
        var lines = new LinkedList<String>();
```

```
    long start = System.nanoTime();

    readLines(input, lines);
    removeEmptyLines(lines);
    removeShortLines(lines); // alle kürzer als MIN_LENGTH
    int n = totalLineLengths(lines);

    long stop = System.nanoTime();

    System.out.printf("result = %d (%d microsec)%n",
    n, (stop - start) / 1000);
}

private static void readLines(BufferedReader input,
LinkedList<String> lines) throws IOException {
    String line = input.readLine();
    while (line != null) {
        lines.add(line);
        line = input.readLine();
    }
 }

private static int totalLineLengths(LinkedList<String> lines) {
    return lines.size();
}

private static void removeShortLines(LinkedList<String> lines) {
    for(var line : lines) {
        if (line.toString().length() < MIN_LENGTH) {
            lines.remove(line);
        }
    }

}

private static void removeEmptyLines(LinkedList<String> lines) {
    ArrayList<String> arr = new ArrayList<>();
    for(var line : lines) {
        if(line.isBlank()) {
            arr.add(line);
        }
    }

    lines.removeAll(arr);
```

```
    }


    // TODO: Klassenmethoden readLines, removeEmptyLines,
    // removeShortLines, totalLineLengths
}
```

Die **prozedurale Programmierung** lässt sich der **imperativen Programmierparadigma** zuordnen. Die prozedurale Programmierung bezeichnet die Programmierung einer sequentiellen Abfolge von Befehlen. Dabei werden wiederverwendbare Teile/Codeabschnitte in Funktionen ausgelagert. Die Funktionen werden als Prozeduren bezeichnet.

Das Java-Programm beinhaltet Anweisungen, die einen Seiteneffekt haben.

Anweisungen sind Befehle, die einen Seiteneffekt haben:

1. print - Ausgabe
2. input - Eingabe
3. Variablen - Verarbeiten
4. Bedingungen und Schleifen (if, for, while, ...)
5. Prozeduren (Funktionen) - immer wieder genutzten Code in eigenständige Bereiche auslagern, um sie einfacher wieder aufrufen zu können. Bei diesem Auslagern in Prozeduren geht es also um das Wiederverwenden von Code.

Aufgabe 5b)

3 Möglichkeiten:

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.LinkedList;

public class Functional {
    private Functional() { }

    public static void main(String[] args) throws IOException {
        var lines = new LinkedList<String>();
        long start = System.nanoTime();

            Files.readAllLines(Paths.get("sample1.txt"))
                .stream()
                .filter(line -> !line.isEmpty())
                .filter(line -> line.length() >= 20)
```

```
                .forEach(lines::add);

                long stop = System.nanoTime();

                System.out.printf("result = %d (%d microsec)%n",
                lines.size(), (stop - start) / 1000);

    }

}
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.LinkedList;
import java.util.stream.Stream;

public class Functional2 {

    private Functional2() { }

    public static void main(String[] args) throws IOException {
        var lines = new LinkedList<String>();

        try (BufferedReader reader =
        new BufferedReader(new FileReader("sample1.txt"))) {
            long start = System.nanoTime();

                // lines() gibt ein Stream auf die einzelnen
                // Zeilen zurück
                Stream<String> stream = reader.lines();
                stream
                .filter(line -> !line.isEmpty())
                .filter(line -> line.length() >= 20)
                .forEach(lines::add);

                long stop = System.nanoTime();

                System.out.printf("result = %d (%d microsec)%n",
                lines.size(), (stop - start) / 1000);
        }

    }

}
```

```
import java.io.IOException;
```

```java
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.LinkedList;
import java.util.stream.Collectors;

public class Functional3 {
    private Functional3() { }

    public static void main(String[] args) throws IOException {
        long start = System.nanoTime();


            Files.readAllLines(Paths.get("sample1.txt"))
                .stream()
                .filter(line -> !line.isEmpty())
                .filter(line -> line.length() >= 20)
                .collect(Collectors.toCollection(LinkedList::new))
                .forEach(System.out::println);

                long stop = System.nanoTime();

                System.out.printf("(%d microsec)%n",
                (stop - start) / 1000);

    }

}
```

Aufgabe 5c)

Das funtkionale Programm hat eine viel größere Laufzeit.

Hier ein Beispielausgabe beider Programme:

Prozedural liefert:

result = 9 (2291 microsec)

und Functional liefert:

result = 9 (24862 microsec)

Übungsaufgabe 6

Aufgabe 6a)

Seite 25: Matching von Listen

- X=john, Y=likes, Z=fish

- X=cat, Y=[]
- X=mary, Y=likes, Z=[wine]
- X=the, Y=hare, Z=[[is, here]]
- die ersten Elemente der Liste 1 und 2 werden gematcht. Beides sind keine Variablen, aber sie sind gleich daher Ausgabe: T = [norfolk]
- die ersten Elemente der Liste 1 und 2 werden gematcht. Beides sind keine Variablen und unterschiedlich, daher fail
- P= white, Q=[horse]

Seite 26: Berechnung der Fakultät in Prolog

da es keinen Rückgabewert gibt, kann nicht die fak Fkt. nur mit einem param aufgerufen werden, also fak(5) liefert keinen Rückgabewert. Es müssen zwei params übergeben werden: fak(5,X). X=120

Lösung:

fak(0,1).

fak(N,X) :- N>0, M is N-1, fak(M,Y), X is N*Y.

Erklärung: fak(N) ist N * fak(N-1). Daher N-1 einer Variablen M zuweisen.

Dann kann fak(N-1), also Fak(M) berechnet werden. Das Ergebnis von fak(N) ist N * Fak(M) und Fak(M) hat als Ergebnis Y, daher insgesamt N * Y und dieses wird mit dem is Op. der Variablen X zugewiesen.

Seite 28: Rekursion bei Listen - append-Anfragen

- append(X , Y, [1,2,3,4]).

  Hier gibt es zwei Variablen. Möchte man genau eine Lösung, muss mindestens die Länge einer Liste bekannt sein. Wäre X bekannt, würde man so lange den Head abspalten, bis sich eine leere Liste für X ergibt, und man den 1. Fakt anwenden kann, wodurch die rechte Liste Y zugewiesen wird.

  Wäre Y bekannt, könnte man solange den Head aus der rechten Liste abspalten, bis Y und die rechte Liste gleich sind. Durch Rückverfolgen der Rekursion, kann man X ermitteln, indem die abgespaltenen Heads aus der rechten Liste X zugewiesen werden.

  Trotzdem findet Prolog alle möglichen Lösungen:

  X = [],

  Y = [1,2,3,4];

X = [1],

Y = [2,3,4];

X = [1,2],

Y = [3,4];

X = [1,2,3],

Y = [4];

X = [1,2,3,4],

Y = [];

false.

Wenn append(X, Y, [1,2,3,4]) mit dem ersten Fakt gematcht wird, wird Y die Liste [1,2,3,4] zugewiesen, also mit der rechten Liste gleichgesetzt. Dementsprechend wird X auf [] unifiziert. Daraus ergibt sich: X = [], Y = [1,2,3,4];

Sucht man nach weiteren Alternativen wendet Prolog den zweiten append Fakt an, was zu einem rekursiven Aufruf führt. Dies könnte folgendermaßen aussehen:

?- append(X , Y, [1,2,3,4]).

→ 2. Fakt anwenden

append(X, Y, [1|2,3,4]) :- append(X, Y, [2,3,4]).

→ 1. Fakt anwenden

append(X, [2,3,4], [2,3,4]) liefert nach Rückverfolgung der Rekursion für X = [1], Y = [2,3,4];

sucht man nach weiteren Alternativen, wird die Rekursion öfter angewendet wodurch sie die rechte Liste verkleinert und die X größer wird...

- append(X, [1,2,3,4], Y).

  Hier gibt es keine Begrenzung für die Rekursion. Die Bregrenzung ist eigentlich auf der linken und auf der rechten Seite, da dort immer ein Element abgespalten werden kann und append rekursiv aufgerufen werden kann. Da auf der linken und rechten Seite eine Variable steht, kann nie eine Grenze gefunden werden, um zu wissen, wo die Rekursion terminiert. Man kann aber den ersten Fakt, der keine Rekursion enthält, anwenden. Das heißt, Y wird mit der mittleren Liste gleichgesetzt und X ist dementsprecht die leere Liste:

  X = [],

  Y = [1,2,3,4]

Aufgabe 6b)

Lösung:

```
sum([],0).

sum([Head|Tail], TotalSum):-
sum(Tail, Sum1),
TotalSum is Head+Sum1.
```

query

```
?- list_sum([1,2,3,4], Sum).
```

answer

```
Sum = 10
```

Aufgabe 6c)

```
zug(konstanz, 08.40, offenburg, 10.59).
    zug(konstanz, 08.40, karlsruhe, 11.49).
    zug(konstanz, 08.53, singen, 09.26).
    zug(singen, 09.37, stuttgart, 11.32).
    zug(offenburg, 11.29, mannheim, 12.24).
    zug(karlsruhe, 12.08, mainz, 13.47).
    zug(stuttgart, 11.51, mannheim, 12.28).
    zug(mannheim, 12.39, mainz, 13.18).


verbindung(Start, Ab, Ziel, [Ziel]) :-
    zug(Start, Abfahrt, Ziel, _), Abfahrt >= Ab.
verbindung(Start, Ab, Ziel, [H|T]) :-
    zug(Start, Abfahrt, H, An), Abfahrt >= Ab,
    verbindung(H, An, Ziel, T).
```

Ausgabe:

```
verbindung(konstanz, 8.00, mainz, Reiseplan).
Breakpoint 6520 in 1-st clause of verbindung/4 at line 17
        Call: zug(konstanz, _8040, mainz, _8172)
        Fail: zug(konstanz, _8040, mainz, _8172)
        Redo: verbindung(konstanz, 8.0, mainz, _7610)
        Call: zug(konstanz, _8046, _8042, _8048)
        Exit: zug(konstanz, 8.4, offenburg, 10.59)
        Call: 8.4>=8.0
        Exit: 8.4>=8.0
        Call: verbindung(offenburg, 10.59, mainz, _8044)
         Call: zug(offenburg, _8068, mainz, _8202)
         Fail: zug(offenburg, _8068, mainz, _8202)
         Redo: verbindung(offenburg, 10.59, mainz, _8044)
         Call: zug(offenburg, _8068, _8064, _8070)
         Exit: zug(offenburg, 11.29, mannheim, 12.24)
         Call: 11.29>=10.59
         Exit: 11.29>=10.59
         Call: verbindung(mannheim, 12.24, mainz, _8066)
          Call: zug(mannheim, _8090, mainz, _8226)
          Exit: zug(mannheim, 12.39, mainz, 13.18)
          Call: 12.39>=12.24
          Exit: 12.39>=12.24
         Exit: verbindung(mannheim, 12.24, mainz, [mainz])
        Exit: verbindung(offenburg, 10.59, mainz, [mannheim, mainz])
        Exit: verbindung(konstanz, 8.0, mainz, [offenburg, mannheim, mainz])
Reiseplan = [offenburg, mannheim, mainz]
Reiseplan = [karlsruhe, mainz]
Reiseplan = [singen, stuttgart, mannheim, mainz]
false
?- verbindung(konstanz, 8.00, mainz, Reiseplan).
```

Übungsaufgabe 7

```
package stg;

import org.stringtemplate.v4.STGroupFile;
import org.stringtemplate.v4.ST;

public final class App {
    private App() {
    }

    public static void main(String[] args) throws Exception {
```

```
        Class<?> cl1 = Class.forName("java.lang.String");
        Class<?> cl2 = Class.forName("java.util.Iterator");
        Class<?> cl3 = Class.forName("java.time.Month");

        Class<?> arr[] = { cl1, cl2, cl3 };

        ST templ = new STGroupFile("app.stg").getInstanceOf("root");
        templ.add("n", arr);
        String htmlpage = templ.render();
        System.out.println(htmlpage);
    }

}
```

```
delimiters "$", "$"

root(n) ::= <<
<!DOCTYPE html>
<html lang="de">
<head>
<style type="text/css">
th, td { border-bottom: thin solid; padding: 4px; text-align: left; }
td { font-family: monospace }
</style>
</head>
<body>
<b>Sprachkonzepte, Aufgabe 7</b>
$n:classes(); separator="\n"$
</body>
</html>
>>

classes(c) ::= <<
<h2>$c$</h2>
<table>
$if(c.interface)$
<tr><th>Methods</th></tr>
<tr><td>$c.methods:method(); separator="<br>\n"$</td></tr>
$else$
<tr><th>Interface</th><th>Methods</th></tr>
$c.interfaces:interfaces(); separator="\n"$
$endif$
</table>
>>

interfaces(i) ::= <<
```

```
<tr>
<td valign=top>$i.name$</td>
<td>$i.methods:method(); separator="<br>\n"$</td>
</tr>
>>

method(m) ::= <<
$m.returnType.name$ $m.name$($m.parameterTypes:methodParameterTypes();
separator=", "$)
>>

methodParameterTypes(p) ::= <<
$p$
>>
```

Ausgabe:

```
<!DOCTYPE html>
<html lang="de">
<head>
<style type="text/css">
th, td { border-bottom: thin solid; padding: 4px; text-align: left; }
td { font-family: monospace }
</style>
</head>
<body>
<b>Sprachkonzepte, Aufgabe 7</b>
<h2>class java.lang.String</h2>
<table>
<tr><th>Interface</th><th>Methods</th></tr>
<tr>
<td valign=top>java.io.Serializable</td>
<td></td>
</tr>
<tr>
<td valign=top>java.lang.Comparable</td>
<td>int compareTo(class java.lang.Object)</td>
</tr>
<tr>
<td valign=top>java.lang.CharSequence</td>
<td>int length()<br>
java.lang.String toString()<br>
int compare(interface java.lang.CharSequence,
interface java.lang.CharSequence)<br>
char charAt(int)<br>
boolean isEmpty()<br>
java.util.stream.IntStream codePoints()<br>
```

```
java.lang.CharSequence subSequence(int, int)<br>
java.util.stream.IntStream chars()</td>
</tr>
<tr>
<td valign=top>java.lang.constant.Constable</td>
<td>java.util.Optional describeConstable()</td>
</tr>
<tr>
<td valign=top>java.lang.constant.ConstantDesc</td>
<td>java.lang.Object
resolveConstantDesc(class java.lang.invoke.MethodHandles$Lookup)</td>
</tr>
</table>
<h2>interface java.util.Iterator</h2>
<table>
<tr><th>Methods</th></tr>
<tr><td>void remove()<br>
void forEachRemaining(interface java.util.function.Consumer)<br>
boolean hasNext()<br>
java.lang.Object next()</td></tr>
</table>
<h2>class java.time.Month</h2>
<table>
<tr><th>Interface</th><th>Methods</th></tr>
<tr>
<td valign=top>java.time.temporal.TemporalAccessor</td>
<td>int get(interface java.time.temporal.TemporalField)<br>
long getLong(interface java.time.temporal.TemporalField)<br>
java.lang.Object query(interface java.time.temporal.TemporalQuery)<br>
java.time.temporal.ValueRange
range(interface java.time.temporal.TemporalField)<br>
boolean isSupported(interface java.time.temporal.TemporalField)</td>
</tr>
<tr>
<td valign=top>java.time.temporal.TemporalAdjuster</td>
<td>java.time.temporal.Temporal
adjustInto(interface java.time.temporal.Temporal)</td>
</tr>
</table>
</body>
</html>
```

Übungsaufgabe 8

colab.research.google.com

Aktienkurse abfragen mit Python

- Daten abfragen (mit Ticker Symbol)
  - jede Aktie ist mit einem Symbol/einer Nummer an einer Börse registriert
  - MSF.DE Microsoft Deutschland
  - Infos abfragen mit yfincace.Ticker
  - print gibt ein yfinance Ticker object zurück
  - .info gibt alle Daten zu dieser Aktie zurück
  - historische Daten einsehen - Preisverlauf der Aktie → microsoft.historypy

    ```
    microsoft.history(period='1d', interval='1m')
    ```

→ liefert Daten des letzten Handelstag und innerhalb dieses einen Tages sollen die Daten in einem 1-Minuten-Zyklus ausgegeben werden

```
#%pip install yfinance
import yfinance, json

symbol = 'MSF.DE'
microsoft = yfinance.Ticker(symbol)
daten = microsoft.info
#print(json.dumps(daten, indent=4))
#print(f'{daten["regularMarketPrice"]} {daten["currency"]}')
#print(microsoft.history(period='1d', interval='1m')['Close'][-1])
```

Welche typischen Eigenschaften werden hier ausgenutzt?

- automatisiertes Ausführen kleiner Codeabschnitte
- deklarationsfreie Syntax: symbol, microsoft, daten
  - implizite Deklaration von Variablen sowie dynamische Typisierung

    → es wird nicht explizit angegeben, welchen Datentyp die Variablen symbol, microsoft und daten haben sollen
- Quelltext kann zeilenweise interpretiert werden und muss vorher nicht übersetzt werden
- zusammengesetze Datentypen können deklarationsfrei und flexibel genutzt werden