Blatt 03 – A3.3: AST (Transformation & Printer)

Modul: Compilerbau — Gruppe: Adrian Kramkowski, Abdelhadi Fares, Yousef Al Sahli, Abdelraoof Sahli

---

Aufgabe & Quellen

- **Ziel:** Parse-Tree ➜ kompakter AST; danach AST formatiert ausgeben.
- **Quelle (Aufgabe):** *sheet03.md*, A3.3.
- **Quelle (Technik):** *antlr-parsing.md*, „Vom Parse-Tree zum AST“, „Arbeiten mit Pattern Matching/Visitor“, „Kontext-Objekte“. *Exakte Foliennummern: Nicht ableitbar.*

AST-Modell (minimale, ausgaberelevante Knoten)

```
package ast;

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

public final class Nodes {
  // --- Statements ---
  public interface Stmt {}

  public static final class Assign implements Stmt {
    public final String id;
    public final Expr value;
    public Assign(String id, Expr value) { this.id = id; this.value = value; }
  }

  public static final class If implements Stmt {
    public final Expr cond;
    public final List thenBranch;
    public final List elseBranch; // evtl. leer
    public If(Expr cond, List thenBranch, List elseBranch) {
      this.cond = cond; this.thenBranch = thenBranch; this.elseBranch = elseBranch;
    }
  }
```

```
  public static final class While implements Stmt {
    public final Expr cond;
    public final List body;
    public While(Expr cond, List body) { this.cond = cond; this.body = body; }
  }

  // --- Expressions ---
  public interface Expr {}

  public enum Op { MUL, DIV, ADD, SUB, EQ, NE, LT, GT }

  public static final class Bin implements Expr {
    public final Op op; public final Expr lhs; public final Expr rhs;
    public Bin(Op op, Expr lhs, Expr rhs) { this.op = op; this.lhs = lhs; this.rhs = rhs; }
  }

  public static final class Name implements Expr {
    public final String id;
    public Name(String id) { this.id = id; }
  }

  public static final class IntLit implements Expr {
    public final String lexeme; // Rohtext, keine Semantik nötig
    public IntLit(String lexeme) { this.lexeme = lexeme; }
  }

  public static final class StrLit implements Expr {
    public final String lexeme;
    public StrLit(String lexeme) { this.lexeme = lexeme; }
  }
}
```

Parse-Tree ➜ AST (Builder)

```
import ast.Nodes.*;
import ast.Nodes;
import java.util.ArrayList;
```

```
import java.util.List;

public final class AstBuilder extends MiniBaseVisitor {

  public List build(MiniParser.StartContext root) {
    @SuppressWarnings("unchecked")
    List prog = (List) visitStart(root);
    return prog;
  }

  @Override public Object visitStart(MiniParser.StartContext ctx) {
    List list = new ArrayList<>();
    for (MiniParser.StmtContext s : ctx.stmt()) {
      list.add((Nodes.Stmt) visit(s));
    }
    return list;
  }

  @Override public Object visitAssign(MiniParser.AssignContext ctx) {
    return new Nodes.Assign(ctx.ID().getText(), toExpr(ctx.expr()));
  }

  @Override public Object visitIfStmt(MiniParser.IfStmtContext ctx) {
    List thenList = new ArrayList<>();
    for (MiniParser.StmtContext s : ctx.stmt(0)) thenList.add((Nodes.Stmt) visit(s));

    List elseList = new ArrayList<>();
    if (ctx.getText().contains("else")) {
      for (int i = 1; i < ctx.stmt().size(); i++) elseList.add((Nodes.Stmt) visit(ctx.stmt(i)));
    }
    return new Nodes.If(toExpr(ctx.expr()), thenList, elseList);
  }

  @Override public Object visitWhileStmt(MiniParser.WhileStmtContext ctx) {
    List body = new ArrayList<>();
    for (MiniParser.StmtContext s : ctx.stmt()) body.add((Nodes.Stmt) visit(s));
    return new Nodes.While(toExpr(ctx.expr()), body);
  }

  // --- Expressions ---
  private Nodes.Expr toExpr(MiniParser.ExprContext e) {
    if (e instanceof MiniParser.MulDivContext) {
      MiniParser.MulDivContext m = (MiniParser.MulDivContext) e;
```

```
      return new Nodes.Bin(op(m.getChild(1).getText()), toExpr(m.expr(0)), toExpr(m.e
    } else if (e instanceof MiniParser.AddSubContext) {
      MiniParser.AddSubContext a = (MiniParser.AddSubContext) e;
      return new Nodes.Bin(op(a.getChild(1).getText()), toExpr(a.expr(0)), toExpr(a.exp
    } else if (e instanceof MiniParser.CmpContext) {
      MiniParser.CmpContext c = (MiniParser.CmpContext) e;
      return new Nodes.Bin(op(c.getChild(1).getText()), toExpr(c.expr(0)), toExpr(c.exp
    } else if (e instanceof MiniParser.ParenContext) {
      MiniParser.ParenContext p = (MiniParser.ParenContext) e;
      // Klammern im AST nicht nötig: Struktur ist bereits im Bin-Baum kodiert
      return toExpr(p.expr());
    } else if (e instanceof MiniParser.NameContext) {
      MiniParser.NameContext n = (MiniParser.NameContext) e;
      return new Nodes.Name(n.ID().getText());
    } else if (e instanceof MiniParser.IntContext) {
      MiniParser.IntContext n = (MiniParser.IntContext) e;
      return new Nodes.IntLit(n.INT().getText());
    } else if (e instanceof MiniParser.StrContext) {
      MiniParser.StrContext s = (MiniParser.StrContext) e;
      return new Nodes.StrLit(s.STRING().getText());
    }
    throw new IllegalStateException("unhandled expr node: " + e.getClass());
  }

  private static Nodes.Op op(String s) {
    if ("*".equals(s)) return Nodes.Op.MUL;
    if ("/".equals(s)) return Nodes.Op.DIV;
    if ("+".equals(s)) return Nodes.Op.ADD;
    if ("-".equals(s)) return Nodes.Op.SUB;
    if ("==".equals(s)) return Nodes.Op.EQ;
    if ("!=".equals(s)) return Nodes.Op.NE;
    if ("<".equals(s)) return Nodes.Op.LT;
    if (">".equals(s)) return Nodes.Op.GT;
    throw new IllegalArgumentException("op: " + s);
  }
}
```

AST-Pretty-Printer

```
import ast.Nodes.*;
import ast.Nodes;
import java.util.List;

public final class AstPrinter {
  private final StringBuilder out = new StringBuilder();
  private int indent = 0;

  private void nl(){ out.append('\n'); }
  private void ind(){ for (int i=0;i program){
    for (Nodes.Stmt s : program) printStmt(s);
    return out.toString();
  }

  private void printStmt(Nodes.Stmt s){
    if (s instanceof Nodes.Assign) {
      Nodes.Assign a = (Nodes.Assign) s;
      ind(); out.append(a.id).append(" := ").append(printExpr(a.value)); nl();
    } else if (s instanceof Nodes.While) {
      Nodes.While w = (Nodes.While) s;
      ind(); out.append("while ").append(printExpr(w.cond)).append(" do"); nl();
      indent += 4; for (Nodes.Stmt st : w.body) printStmt(st); indent -= 4;
      ind(); out.append("end"); nl();
    } else if (s instanceof Nodes.If) {
      Nodes.If i = (Nodes.If) s;
      ind(); out.append("if ").append(printExpr(i.cond)).append(" do"); nl();
      indent += 4; for (Nodes.Stmt st : i.thenBranch) printStmt(st); indent -= 4;
      if (!i.elseBranch.isEmpty()){
        ind(); out.append("else do"); nl();
        indent += 4; for (Nodes.Stmt st : i.elseBranch) printStmt(st); indent -= 4;
      }
      ind(); out.append("end"); nl();
    } else {
      throw new IllegalStateException("unknown stmt");
    }
  }

  private String printExpr(Nodes.Expr e){
    if (e instanceof Nodes.Bin) {
```

```
      Nodes.Bin b = (Nodes.Bin) e;
      return printExpr(b.lhs) + " " + sym(b.op) + " " + printExpr(b.rhs);
    } else if (e instanceof Nodes.Name) {
      return ((Nodes.Name)e).id;
    } else if (e instanceof Nodes.IntLit) {
      return ((Nodes.IntLit)e).lexeme;
    } else if (e instanceof Nodes.StrLit) {
      return ((Nodes.StrLit)e).lexeme;
    }
    throw new IllegalStateException("unknown expr");
  }

  private static String sym(Nodes.Op op){
    switch (op){
      case MUL: return "*"; case DIV: return "/"; case ADD: return "+"; case SUB
      case EQ: return "=="; case NE: return "!="; case LT: return "<"; case GT: r
      default: throw new IllegalArgumentException();
    }
  }
}
```

Demo (Parse ➜ AST ➜ Ausgabe)

```
import ast.Nodes;
import java.util.List;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;

public class RunAstDemo {
```

```
  public static void main(String[] args) throws Exception {
    String src = ""
      + "a     := 0\n"
      + "    if    10 < 1 do\n"
      + "a    :=     42\n"
      + "else do\n"
      + "        a :=      7\n"
      + "  end\n";

    CharStream input = CharStreams.fromString(src);
    MiniLexer lexer = new MiniLexer(input);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    MiniParser parser = new MiniParser(tokens);

    MiniParser.StartContext tree = parser.start();

    AstBuilder builder = new AstBuilder();
    List program = builder.build(tree);

    AstPrinter printer = new AstPrinter();
    System.out.println(printer.print(program));
  }
}
```

Erwartete Ausgabe (wie in A3.2 gefordert):

```
a := 0
if 10 < 1 do
    a := 42
else do
    a := 7
end
```