

## Blatt 03 – A3.2: Pretty Printer

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

---

### Aufgabe & Quellen

- **Ziel:** Parse-Tree traversieren und Programm konsistent einrücken (4 Spaces, 1 Statement/Zeile).
- **Quelle (Aufgabe):** *sheet03.md*, Abschnitt A3.2.
- **Quelle (Technik):** *antlr-parsing.md*, „Arbeiten mit dem Visitor-Pattern“, „Kontext-Objekte“, „Starten des Parsers“. *Exakte Foliennummern: Nicht ableitbar.*

### PrettyPrinter.java (Visitor)

**Ablage:** `src/main/java/PrettyPrinter.java` (Starter-Projekt). Erwartet `Mini.g4` aus A3.1.

```
// package ;
```

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

```
public class PrettyPrinter extends MiniBaseVisitor {
    private final StringBuilder out = new StringBuilder();
    private int indent = 0;
```

```
    private void nl() { out.append("\n"); }
    private void ind(){ for (int i = 0; i < indent; i++) out.append(' '); }
```

```
    public String format(ParseTree tree) {
        visit(tree);
        return out.toString();
    }
```

```
    @Override public Void visitStart(MiniParser.StartContext ctx) {
        for (var s : ctx.stmt()) visit(s);
    }
```

```

    return null;
}

@Override public Void visitAssign(MiniParser.AssignContext ctx) {
    ind();
    out.append(ctx.ID().getText()).append(" := ").append(printExpr(ctx.expr()));
    nl();
    return null;
}

@Override public Void visitIfStmt(MiniParser.IfStmtContext ctx) {
    ind(); out.append("if ").append(printExpr(ctx.expr())).append(" do"); nl();
    indent += 4;
    for (var s : ctx.stmt(0)) visit(s);
    indent -= 4;

    if (ctx.getText().contains("else")) {
        ind(); out.append("else do"); nl();
        indent += 4;
        for (int i = 1; i < ctx.stmt().size(); i++) visit(ctx.stmt(i));
        indent -= 4;
    }
    ind(); out.append("end"); nl();
    return null;
}

@Override public Void visitWhileStmt(MiniParser.WhileStmtContext ctx) {
    ind(); out.append("while ").append(printExpr(ctx.expr())).append(" do"); nl();
    indent += 4;
    for (var s : ctx.stmt()) visit(s);
    indent -= 4;
    ind(); out.append("end"); nl();
    return null;
}

private String printExpr(MiniParser.ExprContext e) {
    return switch (e) {
        case MiniParser.MulDivContext m -> printExpr(m.expr(0)) + " " + m.getChild(1).getText();
        case MiniParser.AddSubContext a -> printExpr(a.expr(0)) + " " + a.getChild(1).getText();
        case MiniParser.CmpContext c -> printExpr(c.expr(0)) + " " + c.getChild(1).getText();
        case MiniParser.ParenContext p -> "(" + printExpr(p.expr()) + ")";
        case MiniParser.NameContext n -> n.ID().getText();
        case MiniParser.IntContext n -> n.INT().getText();
    };
}

```

```

        case MiniParser.StrContext s  -> s.STRING().getText();
        default -> throw new IllegalStateException();
    };
}

public static void main(String[] args) throws Exception {
    String src = ""
a    := 0
    if 10 < 1 do
a    := 42    # Zuweisung
else do
    a := 7
end
""";
    CharStream input = CharStreams.fromString(src);
    MiniLexer lexer = new MiniLexer(input);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    MiniParser parser = new MiniParser(tokens);
    ParseTree tree = parser.start();

    System.out.println(new PrettyPrinter().format(tree));
}
}

```

### Erwartete Ausgabe

```

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

```