Aufgabe A4.6

Wir haben uns angesehen, welche Open-Source-Projekte handgeschriebene recursive descent Parser verwenden oder später darauf umgestiegen sind. Dabei ist uns aufgefallen, dass viele Projekte diesen Ansatz wählen, weil er gut kontrollierbar ist und sich leicht anpassen lässt.

Ein wichtiges Beispiel ist **CPython**. Dort wurde früher ein automatisch erzeugter Parser genutzt. Später ist das Projekt auf einen selbst geschriebenen Parser umgestiegen, um die Grammatik einfacher erweitern zu können und um mehr Kontrolle über das Parsing-Verhalten zu haben. Auch **Go**, **Oberon** und ältere **Pascal**-Dialekte setzen auf handgeschriebene Parser, weil diese Lösung einfach aufgebaut ist und gut zu verstehen bleibt. Ähnliches findet man in vielen kleinen Sprachen und Lehrprojekten.

Unsere Diskussion zu ANTLR

In unserer Gruppe haben wir die Vor- und Nachteile von ANTLR gesammelt und diskutiert.

Vorteile:

- ANTLR nimmt viel Arbeit ab, sobald eine Grammatik definiert ist.
- Das Testen von Beispielen ist schnell möglich, da ANTLR direkt Parsebäume anzeigen kann.
- Durch Listener und Visitoren lässt sich die Weiterverarbeitung gut strukturieren.
- Für größere Projekte ist die klare Trennung von Grammatik und Code oft hilfreich.

Nachteile:

- Die Einrichtung kann am Anfang mühsam sein, besonders bei Plugins und Build-Tools.
- Die erzeugten Klassennamen und die Groß-/Kleinschreibung wirken manchmal ungewohnt.
- Für kleine Sprachen wirkt ANTLR oft zu groß, während ein handgeschriebener Parser viel direkter ist.

Aus unserer Sicht eignet sich ANTLR gut für Sprachen, die größer werden oder bei denen mehrere Personen an der Grammatik arbeiten. Handgeschriebene Parser sind angenehmer, wenn man möglichst viel Kontrolle möchte oder eine einfache Sprache baut, bei der der Aufwand gering bleiben soll.