Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

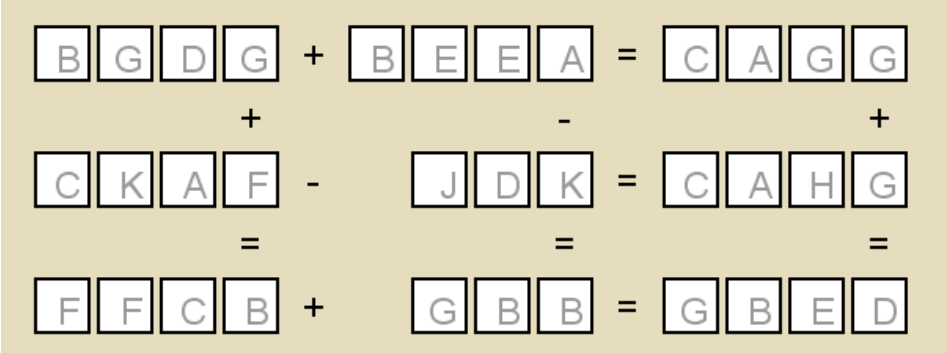
Compiler und Interpreter

– Praktikum, Teil 1 –

Prof. Dr. Michael Neitzke

Entwicklung einer DSL

- Entwickeln Sie eine DSL für Symbolrätsel.



- Es gelten folgende Randbedingungen:
  - Additions- und Subtraktionsaufgaben
  - Neun **beliebig lange** Ziffernblöcke
  - Drei Aufgaben waagerecht und drei senkrecht
  - Eingabe einer Aufgabe muss über einen Texteditor möglich sein
- Beispiel-Aufgaben finden Sie unter
  http://www.janko.at/Raetsel/Symbolrechnen/index.htm

Präsentation der DSL

- Bereiten Sie sich darauf vor, ggf. Ihre DSL im Rahmen des Praktikums zu präsentieren.
- Diese Präsentation würde am Anfang des Praktikums erfolgen. Die entsprechenden Arbeiten müssen also während der Vorbereitung des Praktikums geleistet werden.
- Die Dauer der Präsentation soll 5-10 Minuten betragen.

Definition einer Grammatik

- Entwickeln Sie eine Grammatik für Ihre DSL.
- Setzen Sie die Grammatik in ANTLR um. Verwenden Sie dafür ANTLRWorks. Testen Sie die verschiedenen Darstellungsmöglichkeiten in ANTLRWorks.
- Testen Sie mit verschiedenen Eingaben (Rätseln), auch fehlerhaften.

Definition einer Grammatik für MiniP

- Auf der nächsten Seite finden Sie eine Beschreibung der fiktiven Programmiersprache MiniP, einer Teilmenge von Pascal.
- Entwickeln Sie mit ANTLR eine Grammatik für MiniP und testen Sie die Grammatik anhand einiger Beispiele. Darunter sollen auch komplexe Beispiele sein, so dass die verschiedenen Konzepte von MiniP getestet werden.
- Hinweise:
  - Kommentare werden vom Scanner überprüft, aber es wird kein Kommentar-Token an den Parser geliefert!
  - Whitespace wird ebenfalls ignoriert, d.h. keine Whitespace-Tokens
  - Für den Scanner ist es häufig nützlich, wenn in seinen Token-Definitionen Makros für häufig benötigte Zeichenketten benutzt werden können, z.B. LETTER für Buchstaben oder DIGIT für Ziffern. Diese Makros werden in ANTLR als Fragments bezeichnet. Wenn Sie am Zeilenanfang das Schlüsselwort „fragment" aufführen, können Sie dahinter eine Makro-Definition aufführen.

Die Programmiersprache "MiniP"

- Programm-Struktur
  program [<Deklarationen>] **begin** <Anweisungen> **end**

- Kommentare
  ~~//… (eine Zeile) oder~~
  /* … */ kann über mehrere Zeilen gehen!!!

- Symbole
  Groß-/Kleinschreibung spielt keine Rolle
  White spaces trennen Symbole, spielen sonst keine Rolle

  Symbolklassen
  - **id**entifier — fängt mit einem Buchstaben an. Der eventuell folgende Rest darf Buchstaben, Ziffern oder Unterstrich (_) enthalten.
  - Konstanten: integer,
    real (ohne Exponentialdarstellung),
    string ('Das ist ein string'),
    boolean (**true**, **false**)
  - Trennzeichen / Operatoren: **+** | **-** | ***** | **/** | **,** | **;** | **:=** | **=** | **<>** | **<** | **<=** | **>** | **>=**| **(** | **)**
  - Schlüsselworte: **program** | **begin** | …

- Deklarationen
  Jede Deklaration wird mit einem Semikolon abgeschlossen.
  (**integer** | **real** | **string** | **boolean**) <kommaseparierte Liste von **ids**>

- Anweisungen (mindestens eine):
  Jede Anweisung wird mit einem Semikolon abgeschlossen.
  - Wertzuweisung: id := <arith.Ausdruck> | ~~<booleanConst>~~ | <stringConst> | <Vergleich>
  - arith. Ausdruck enthält: **(**, **)**, **+**, **-**, *****, **/**, Vorzeichen, arithConst, arithVariable
    Vorrang: Klammern vor Punkt- vor Strichrechnung
  - if-Anweisung: **if** … **then** … [**else** …] **fi**
  - Vergleich: (**=** | **<>** | …) zwischen arithmetischen Variablen oder Konstanten
  - while-Anweisung: **while** … **do** … **od**
  - repeat-Anweisung: ~~**repeat** … **until**~~ …
  - read-Anweisung: **read** (<Variable>)
  - print-Anweisung: **print**[**ln**] (<arithVariable> | <stringConst>) **nur println**

  Die I/O-Anweisungen können Sie auch gern komplexer vorsehen.

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences