# Einführung in die Informatik I (autip)

Dr. Stefan Lewandowski

Fakultät 5: Informatik, Elektrotechnik und Informationstechnik
Abteilung Formale Konzepte
Universität Stuttgart

31. Oktober 2007

Inhalte heute:

- Wdh.: Erweiterte Backus-Naur-Form, Syntaxdiagramme
- Struktur eines Ada95 Programms
- Syntax von Deklarationen und Bezeichnern in Ada 95
- Einfache Datenstrukturen: integer (mit Attributen)
- Unterbereiche natural und positive
- Ein-/Ausgabe mit Get/Put
- Funktionen

## Sprachen zur Beschreibung der Syntax von Sprachen

- (E)BNF - (Erweiterte) Backus-Naur-Form
  - wird z.B. auch im Ada Reference Manual verwendet
- Syntaxdiagramme
  - sind etwas anschaulicher, aber in der Praxis auch etwas umständlicher zu handhaben

Hier sollen zunächst nur die wenigen Bausteine der EBNF und der Syntaxdiagramme vorgestellt werden, so dass wir damit die Syntax von Ada 95 formal beschreiben zu können.

Grenzen der Möglichkeiten $\rightarrow$ später im Semester.

Für den Moment reicht zu wissen: EBNF und Syntaxdiagramme sind gleich mächtig.

## Erweiterte Backus-Naur-Form (EBNF)

- Formale Beschreibungssprache für Zeichenketten
- Terminale (direkt aufgeschriebene Zeichenketten): z.B. “Text“, “42“, . . . – in Anführungszeichen (z.T. auch ohne)
- Nichtterminale (Variablen): z.B. <if-Anweisung>, <Ausdruck>, <Ziffer>, . . . – in spitze Klammern
- „wird definiert durch“: „::=“
- Alternativen: z.B. <Ziffer> ::= “0“|“1“|“2“|“3“|“4“|“5“|“6“|“7“|“8“|“9“ – Trennung durch senkrechten Strich
- Optionale Elemente: z.B. [ “else“ <Anweisungs-Folge> ], . . . – in eckige Klammern, 0 oder 1 Mal
- Iterationen: z.B. { “elsif“ <boolescher-Ausdruck> “then“ <Anweisungs-Folge> } – geschweifte Klamm., 0, 1, 2, ... Mal

Es gibt in der Literatur diverse Variationen davon (z.B. Markierung des Endes einer Regel durch einen Punkt, (..)* statt {..}, . . . )

## Syntaxdiagramme

- Formale Beschreibungssprache für Zeichenketten
- Terminale (direkt aufgeschriebene Zeichenketten): z.B. Text, 42, ...– in Kreise (oder Ellipsen)
- Nichtterminale (Variablen): z.B. if-Anweisung, Ausdruck, Ziffer, ...– in Rechtecke
- „wird definiert durch“: Name steht über dem Diagramm
- Kreise und Rechtecke werden durch Pfeile verbunden
- Alternativen, optionale Elemente, Iterationen: durch Aufsplittung der Pfeile/Verbindungen

Die durch ein Syntaxdiagramm definierten Zeichenfolgen ergeben sich durch alle möglichen Reihenfolgen vom Anfang zum Ende des Syntaxdiagramms zu gelangen.

- Sinn von formaler Syntax: struktureller Aufbau der Sprache wird verdeutlicht
- Programmaufbau als EBNF (→ Tafel)
- identifier am Ende identisch
- Kommentare überall erlaubt `--` bis Zeilenende
- Ada 95 unterscheidet bei reservierten Wörtern und Bezeichnern keine Groß- und Kleinschreibung. Dies ließe sich ebenfalls in EBNF ausdrücken, aber `"procedure"` ist schlicht kürzer und besser lesbar als
  `("p"|"P")("r"|"R")("o"|"O")...("e"|"E")`
- Syntax für Deklarationen (→ Tafel)
- Wozu Deklaration als Konstante? → Les- und Wartbarkeit: Ohne ein entsprechendes Ada-Konstrukt wäre es nur ein Versprechen des Programmierers, den Inhalt dieser Variablen nie zu verändern – durch das Schlüsselwort `constant` wird es aber durch den Ausführer garantiert, dass dieser Wert im Programm gleich bleibt.
- Syntax für Bezeichner (→ nächste Folie)

## Bezeichner

Zum Benennen der Dinge, mit denen wir arbeiten (Variablen, Prozeduren und noch auch einiges andere), dürfen wir uns in Ada Namen (Bezeichner, englisch: identifier) ausdenken, die nur wenigen Einschränkungen unterliegen:

- Jeder Name muss mit einem Buchstaben anfangen
- Als weitere Zeichen sind erlaubt: Buchstaben, Ziffern und der Unterstrich '_', aber keine sonstigen Zeichen, auch keine Leerzeichen
- Es dürfen nicht mehrere Unterstriche hintereinander vorkommen, das letzte Zeichen des Namens darf kein Unterstrich sein.

Bezeichner dürfen beliebig lang sein; Groß- und Kleinbuchstaben werden nicht unterschieden („Muh“ und „muH“ sind ein und derselbe Name).

Einige erlaubte Bezeichner: „c“, „c1“, „ein_langer_bezeichner“

und ein paar Wörter, die keine zulässigen Bezeichner sind:

„1c“, „keine leerzeichen“, „unerlaubtes_zeichen!“

- Einige Bezeichner sind für Ada-Sprachelemente reserviert, etwa „procedure“.
- Andere, wie etwa „Get“ stehen für Dinge, die in der Ada-Sprachbibliothek bereits definiert sind. Die Regeln, in wieweit man diese Namen für eigene Sachen nutzen darf, sind recht kompliziert; normalerweise wird man das auch nicht machen wollen (Ausnahmen folgen ggf. im Laufe des Semesters).
- Auswahl geeigneter Bezeichner $\rightsquigarrow$ guter Programmierstil: Aussagekräftige, aber nicht zu lange Namen

übliche Konventionen:

- Schlüsselwörter werden stets klein geschrieben
- Variablen, Konstanten, Prozeduren und Funktionen beginnen mit einem Großbuchstaben, Rest klein – Ausnahmen z.B. „Erg_in_Prozent“

Zahlen $\mathbb{Z} := \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ im Rechner:

Datentypen `integer` (entspricht $\mathbb{Z}$),
`natural` (entspricht $\mathbb{N} \cup \{0\}$) und
`positive` (entspricht $\mathbb{N}$)

Attribute `integer'first` und `integer'last`

Programm zur Berechnung der Fakultät

- Programm zur Berechnung der Fakultät
- unschön:
  - bisher keine Zuweisung der Art `x:=fak(n)` möglich
  - Ausgabe der Zahlen unschön
  - Standardeinstellungen sind für mehrzeilige Ausgaben gut geeignet (evtl. etwas breit), für einzelne Zahlen aber unnötig viele Leerzeichen
  - eingeschränkter Zahlenbereich, unpraktische Fehlerbehandlung