252-0027

Einführung in die Programmierung

Übungen

Eclipse und EBNF

Departement Informatik
ETH Zürich

Wer bin ich

- **Mein Name: Jonas Wetzel**
- **2. Jahr Informatik**
- **programming go brr**
- **Bei Fragen: Email oder über Discord**

Wieso ist EProg ein banger Fach?

- **Übungen machen Spass, man kann actually coden**
- **Wir lernen object-oriented programming, eines der main programming paradigms**
  - inheritance, polymorphism, and encapsulation
- **man kann sehr gut für die Prüfung lernen, 6 ist achievable**

Organisatorisches

- Neue Aufgaben: **Dienstag Abend** (im Normalfall)
- Abgabe der Übungen bis **Dienstag Abend (23:59)** Folgewoche
  - Abgabe immer via Git
  - Lösungen in separatem Projekt auf Git

- **course website: lec.inf.ethz.ch/infk/eprog/2024/**
- **meine website: n.ethz.ch/~jwetzel/ (extra Material, Infos, etc.)**

Eclipse bereits früher installiert?

- Neue Version: **Java 21 (Letztes Jahr Java 17)**
  - Java 21 aus dem Oracle Archiv installieren -> JRE 21 in Eclipse als Standard auswählen
  - Informationen zu Fehlermeldungen und Behebungen direkt auf der Website
    - Sonst Eclipse und Java deinstallieren, JDK 21 und Eclipse neu installieren
  - Sehr wichtig, weil Korrektur der Bonusaufgaben auch Java 21 benutzt

Mehr bei Eclipse

Eclipse Installation

- Instruktion zur Installation findet ihr in Übung 0.
  - https://lec.inf.ethz.ch/infk/eprog/2024/exercises/sheets/uebungsblatt0.pdf

- Let’s go

Repetition: Ableitungen

- Ableitungstabelle
  - Erste Zeile ist Startregel
  - Letzte Zeile ist Zeichenfolge
  - Übergang zwischen zwei Zeilen entspricht Ableitungsschritt

- Ableitungsbaum
  - Wurzel ist Namen der Startregel
  - Blätter sind Zeichen
  - Verbindungen stehen für einen Ableitungsschritt

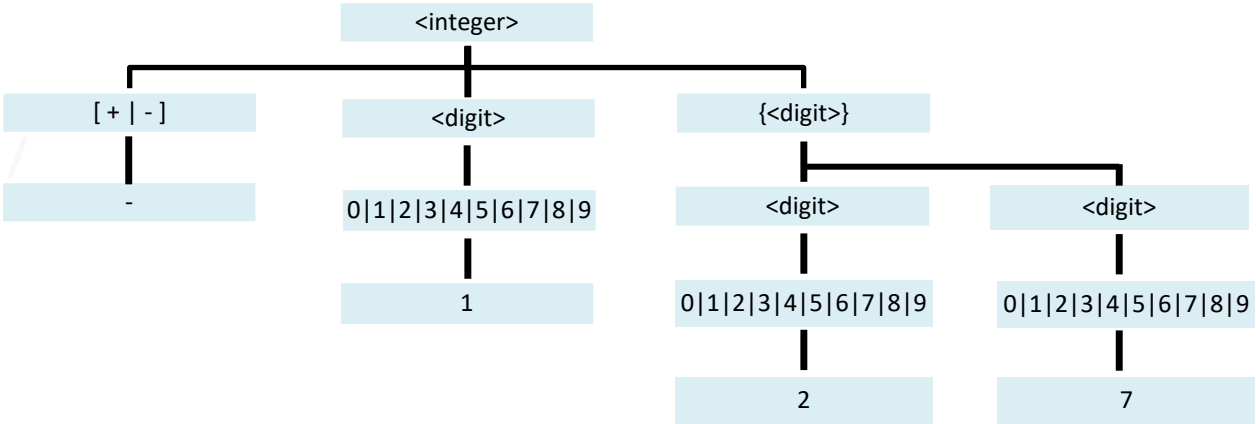
Beispiel: Ableitung von -127 als Baum

<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer>	⟵	[+ | -] <digit> {<digit>}

Beispiel: Ableitung von -127 als Baum

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ⟵ [+ | -] <digit> {<digit>}



Beispiel: Ableitung von -127 als Tabelle

(R1)	<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(R2)	<integer>	⟵	[+ | -] <digit> {<digit>}

Beispiel: Ableitung von -127 als Tabelle

(R1)	<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(R2)	<integer>	⟵	[+ | -] <digit> {<digit>}

<integer>	⟵	[+ | -] <digit> {<digit>}	(R2)
	⟵	+ | - <digit> {<digit>}	Option gewählt
	⟵	- <digit> {<digit>}	- gewählt
	⟵	- <digit> <digit> <digit>	2 mal wiederholt
	⟵	- 1 <digit> <digit>	(R1) und 1 gewählt
	⟵	- 1 2 <digit>	(R1) und 2 gewählt
	⟵	- 1 2 7	(R1) und 7 gewählt

EBNF Notation

- In alten Prüfungen wird oft kursiv verwendet für EBNF Regeln.
- *digit* statt <digit>
- dieses Jahr ist es immer so: <digit>

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

1245

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

00972

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

separator ⇐ _

char ⇐ A | B | C | D | E | F | a | b | c | d | e | f

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [. *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } h

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

00100h

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

1a00

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

1a00h

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

1_000_000

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

001ab.001h

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

209AB

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

4.9901

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

deadface

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

4_000.0

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

00100H

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

001ab.001

EBNF: Legal / Nicht Legal

Gegeben sei die EBNF-Beschreibung von *value*

digit ⇐ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

separator ⇐ `_`

char ⇐ `A` | `B` | `C` | `D` | `E` | `F` | `a` | `b` | `c` | `d` | `e` | `f`

num ⇐ *digit* { [*separator*] *digit* }

int ⇐ *digit* { *digit* }

real ⇐ *digit* { *digit* } [`.` *digit* { *digit* }]

cd ⇐ *char* | *digit*

hexa1 ⇐ *cd* { *cd* }

hexa2 ⇐ *digit* { *digit* } `h`

hexa ⇐ *hexa1* | *hexa2*

value ⇐ *num* | *real* | *int* | *hexa*

0x0ABC

Bonus

- Erstellen Sie eine Beschreibung `<palindrome>`, welche als legale Symbole alle Zahlen zulässt, die von vorne und hinten gleich gelesen werden und die nur die Ziffern von 1 bis 4 verwenden. Beispiele sind 11, 232, 444
- Erstellen Sie eine Beschreibung `<five>`, welche alle Summen von positiven Zahlen zulässt, welche 5 ergeben. Beispiele sind “1 + 4”, “2 + 1 + 1 + 1”, “5”
- Erstellen Sie eine Beschreibung für `<oddEight>`, die alle Zahlen enthält, in denen die Ziffer 8 ungerade oft vorkommt.