Technische Informatik für Ingenieure (TIfI)
WS 2005/2006, Vorlesung 2

Ekkart Kindler

II. Grundlagen der Programmierung

- Überblick
- Syntaxdefinition
- Variablen und Ausdrücke
- Anweisungen
- …

Literale (Konstanten)


```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```


Variablen


```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```

Variable:
Ein „Behälter“ zum „Aufbewahren“ eines Wertes.

Variablendeklarationen


```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```

Variablendeklaration:
Definition eines „Behälters“ durch „Beschriftung“ mit einem Namen und „Wertebereich“.

Operatoren & Ausdrücke


```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```

Ausdruck:
Berechnungsvorschrift, die aktuelle Werte von Variablen und Konstanten gemäß der **Operatoren** miteinander verknüpft.

Zuweisungen

```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```

Zuweisung:
Weist das Ergebnis der Auswertung eines Ausdrucks einer Variablen zu (legt das Ergebnis in den „Behälter“).

Anweisungen & Ablaufsteuerung

```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```

Ablaufsteuerung:
Legt fest in welcher Reihenfolge die **Anweisungen** durchlaufen werden.

Schlüsselwörter

```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```


Überblick

- Variablen
- Ausdrücke
  - Variablen
  - Literale (Konstanten)
  - Operatoren
- Anweisungen
  - Zuweisungen
  - Bedingte Anweisung
  - Schleifen
- …

1. Syntax und Semantik

`( x + 7` `x * y` `n *(n + 1) / 2` `x - 7` `x / 0` `( x - 7 )` `-(x + y)` `x % y` `*(x - y)`

- Welche Wörter (Zeichenreihen) sind korrekt geformte Programme bzw. Ausdrücke?
  → Definition der **Syntax** einer Sprache
- Was „machen“ sie? Was bedeuten sie?
  → Definition der **Semantik** einer Spache

„Syntaxdefinitionen“

- Viele Beispiele
- Natürliche Sprache
- Ausprobieren →Eclipse (später mehr)
- Induktive Definitionen
- Spezielle Notationen
  - Backus-Naur-Form (BNF)
  - Syntaxdiagramme

Spezielle Notationen

University of Paderborn
Software Engineering Group
E. Kindler

- Wir werden hier die **BNF** und **Syntax-diagramme** als spezielle Notationen der Informatik zur Definition der Syntax kennen lernen
- Wir führen diese Notation am Beispiel der Definition der Syntax von Ausdrücken ein

1.1 Backus-Naur-Form (Beispiel)

University of Paderborn
Software Engineering Group
E. Kindler

<Ausdruck> ::= <Variable> | <Konstante> |
<UnOp> <Ausdruck> |
<Ausdruck> <BinOp> <Ausdruck> |
"(" <Ausdruck> ")"

<Variable> ::= <Buchstabe> { <Buchstabe> | <Ziffer> }

<Buchstabe> ::= "a" | "b" | "c" | ... | "z" |
"A" | "B" | "C" | ... | "Z" | "_"

<Ziffer> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

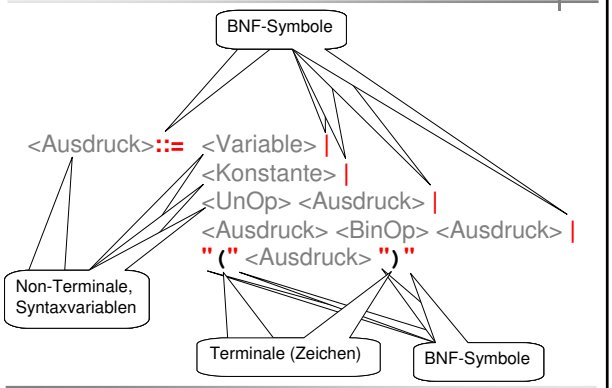
<Konstante> ::= ["+" | "−"] <Ziffer> { <Ziffer> }

<UnOp> :: = " +" | "−"

<BinOp> :: = " +" | "−" | " *" | "/" | "%"

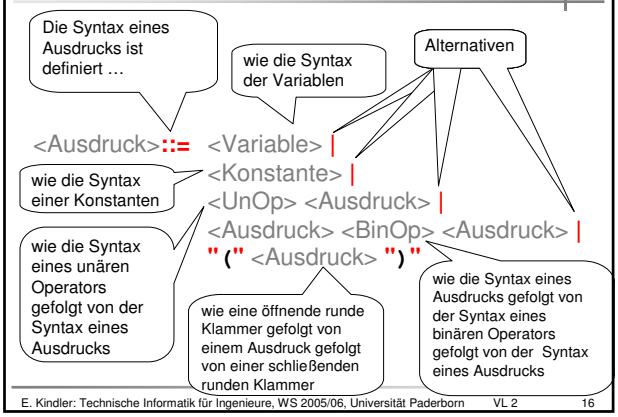
BNF: Notation

University of Paderborn
Software Engineering Group
E. Kindler



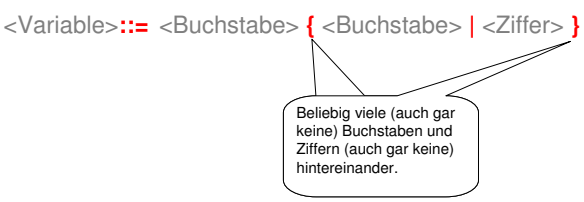
BNF: Bedeutung

University of Paderborn
Software Engineering Group
E. Kindler



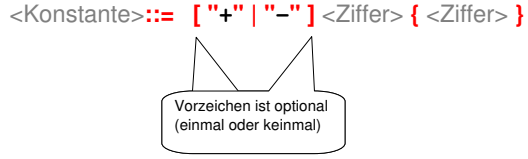
BNF: Notation und Bedeutung

University of Paderborn
Software Engineering Group
E. Kindler



BNF: Notation und Bedeutung

University of Paderborn
Software Engineering Group
E. Kindler

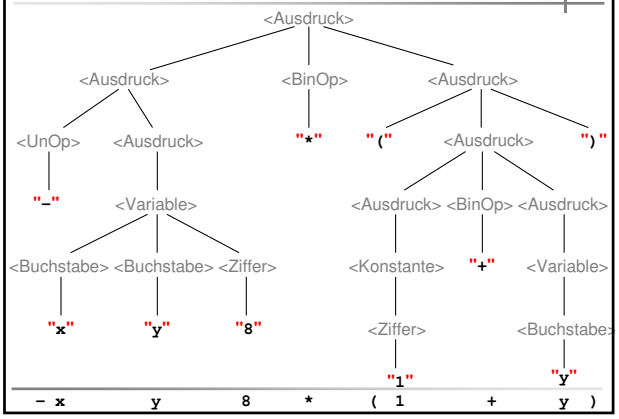


BNF: Notation und Bedeutung

<Buchstabe> ::= "a" | "b" | "c" | … | "z" |
"A" | "B" | "C" | … | "Z" | "_"

Gehört nicht zur BNF; hier müßte man alle Buchstaben wirklich hinschreiben

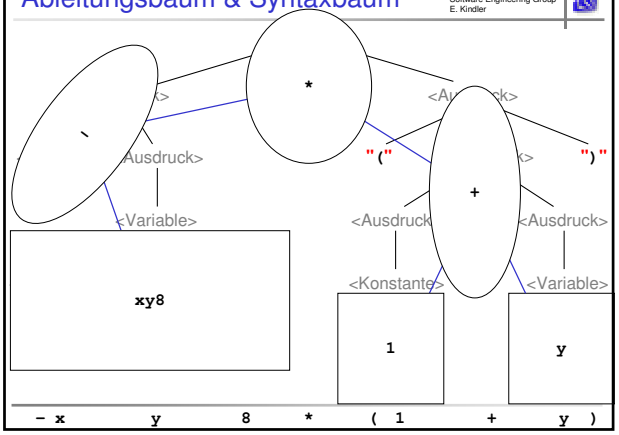
BNF & Ableitungsbäume



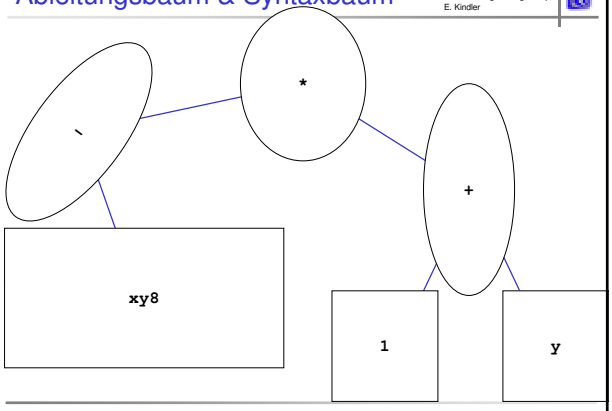
Ableitungsbäume

- entstehen aus einer Syntaxvariablen durch schrittweises Anwenden der BNF-Regeln; dabei sind die Kinder einer Syntaxvariable durch die entsprechende BNF-Regel definiert
- wenn an den Blättern nur Terminalsymbole vorkommen, können wir von links nach rechts ein Wort „einsammeln“; dieses Wort ist dann eine syntaktisch korrekte Zeichenreihe (für das entsprechende syntaktische Konstrukt)
- Alle Wörter, die wir auf diese Weise erzeugen können, sind syntaktisch korrekt bzgl. der BNF-Definition

Ableitungsbaum & Syntaxbaum



Ableitungsbaum & Syntaxbaum



Ableitungsbäume

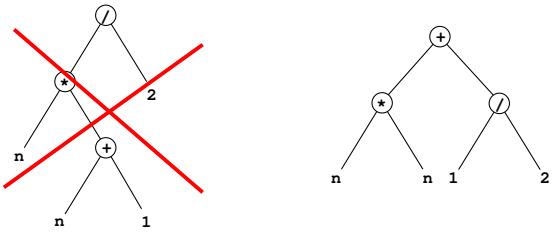
- definieren nicht nur die korrekten Wörter, sondern auch eine Struktur auf diesen Wörtern

Achtung!!
So wie wir das hier gemacht haben, ist die Struktur eines Wortes nicht immer eindeutig definiert. Es kann verschiedene Bäume für dasselbe Wort geben (z.B. für x - 3 * y).

In diesen Fällen wird die Struktur durch Zusatzregel eindeutig festgelegt:
- Punkt vor Strich
- Unär vor Binär
- Von links nach rechts

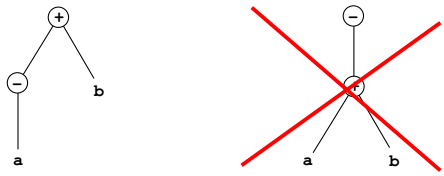
Zusatzregel: Punkt vor Strich


```
n * n + 1 / 2
```



Zusatzregel: Unär vor Binär

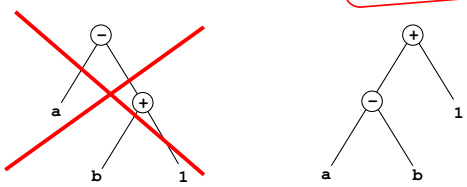

```
- a + b
```



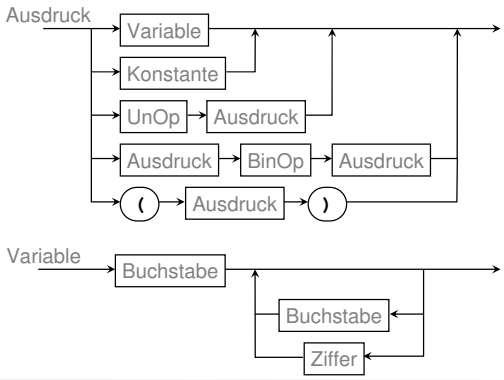
Zusatzregel: Von links nach rechts


```
a - b + 1
```

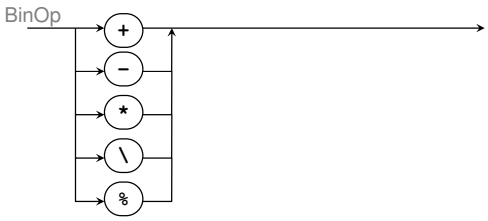
Achtung!
„Vor“ heißt im Baum weiter unten!!



1.2. Syntaxdiagramme



Syntaxdiagramme



…

Syntaxbäume

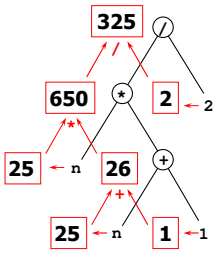
- Die Terminale auf dem Pfad durch ein Syntaxdiagram definieren die syntaktisch korrekten Wörter
- Wenn man auf einen „Kasten“ für ein Non-Terminal stößt, muß man zunächst dieses Diagramm fertig abarbeiten (und alle Terminale einsammeln), bevor man im eigentlichen Diagramm weiter macht

1.3 Semantik

- Die Semantik definiert die Bedeutung eines syntaktisch korrekten Wortes
- Bei Ausdrücken ist das die Auswertung des Ausdruck für eine feste Belegung der Variablen mit Werten

Auswertungsformular

Für `n = 25`: `n * (n + 1) / 2`



Semantik der Operatoren

`+` (binär): Summe der beiden Operanden
`+` (unär): Wert des Operanden

`–` (binär): Differenz der beiden Operanden
`–` (unär): Oparand multipliziert mit -1

`*` (binär): Produkt der beiden Operanden

`/` (binär): Ganzzahlige Division der beiden Operanden

`%` (binär): Rest der ganzahligen Division der beiden Operanden (modulo)

2. Ausdrücke und Zuweisungen

- Variablen
  - Deklaration
  - Initialisierung
- Ausdrücke
- Zuweisungen

2.1 Variablen

- Variablen sind Behälter für einen Wert: „Wert der Variablen“
- Sie haben einen Namen, mit dem man auf den Wert zugreifen kann

Der Wertebereich der Variablen wird durch ihren Typ festgelegt; darüber reden wir aber erst später. Bis dahin ist der Wertebereich aller Variablen die Menge der ganzen Zahlen (`int`).

Variablen

- Variablen müssen **vor** dem ersten Auftreten im Programm deklariert werden
- Der Wert, den die Variable am Anfang hat, wird bei der Deklaration explizit oder implizit festgelegt

Wenn man den Wert nicht explizit festlegt, ist der initiale Wert einer Variablen 0.

Variablendeklaration

```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```


Variablendeklaration

<VarDecl>::= <Type> <Identifier> ["=" <Expression>]
{ "," <Identifier> ["=" <Expression>] }

<Identifier>::= <Letter> { <Letter> | <Digit> }

<Letter>::= "a" | "b" | "c" | ... | "z" |
"A" | "B" | "C" | ... | "Z" | "_"

<Digit>::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<Type>::= "int"

Diese Definition wird später noch erweitert!

2.2 Ausdrücke

<Expression>::= <Identifier> | <Literal> |
<UnOp> <Expression> |
<Expression> <BinOp> <Expression> |
"(" <Expression> ")"

<Literal>::= ["+" | "-"] <Digit> { <Digit> }

<UnOp> :: = "+" | "-"

<BinOp>:: = "+" | "-" | "*" | "/" | "%"

Auch diese Definition wird später noch erweitert!

Variablendeklaration

```
public class Fakultaet {

  public static void main (String[] args) {

    Out.print("Geben Sie eine Zahl ein: ");
    int n = In.readInt();

    int erg = 1;

    while (n > 0) {
      erg = erg * n;
      n = n - 1;
    }

    Out.println("Das Ergebnis ist " + erg + ".");
  }

}
```

Das ist gemäß unserer Definition nicht zulässig: es ist (gemäß unserer Def.) kein Ausdruck. Das ist ein Methodenaufruf; das lernen wir aber erst viel später kennen.

2.3. Zuweisungen

```
erg = erg * n;
```

- Der Ausdruck wird ausgewertet und dann das Ergebnis als neuer Wert für die Variable übernommen
- Die Variable, der ein Wert zugewiesen wird, darf im Ausdruck vorkommen; bei der Auswertung wird der aktuelle Wert benutzt; erst danach erhält die Variable den neuen Wert
- Eine Zuweisung hat nichts mit einer mathematischen Gleichung gemein

Um das zu verdeutlichen, hat man früher für die Zuweisung andere Symbole benutzt: := oder ←

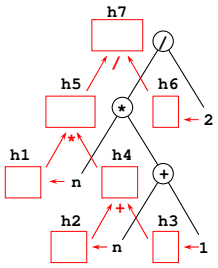
BNF: Zuweisung

<Assignment>::= <Identifier> "=" <Expression>

Randbedingung: Eine Variable muß vor ihrem ersten Auftreten (in einem Ausdruck oder einer Zuweisung) deklariert werden.

„Semantik“


```
erg = n * (n + 1) / 2;
```




```
int h1 = n;
int h2 = n;
int h3 = 1;
int h4 = h2 + h3;
int h5 = h1 * h4;
int h6 = 2;
int h7 = h5 / h6;

erg = h7;
```


2.5. Seiteneffekte


```
erg = erg * n--;
```

- Der Wert der Variablen wird nach dem Auftreten in der Auswertung um eins vermindert (mit **n++** um eins erhöht)
- Entsprechend wird mit **--n** und **++n** der Wert der Variablen vor dem Auswerten um eins vermindert bzw. erhöht.

BNF: Erweiterung

<Expression> ::= <Identifier> | <Literal> |
<UnOp> <Expression> |
<Expression> <BinOp> <Expression> |
"(" <Expression> ")" |
<Identifier> "++" | <Identifier> "--" |
"++" <Identifier> | "--" <Identifier>

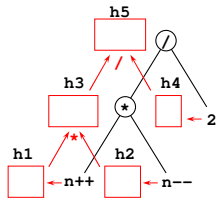
<Literal> ::= ["+" | "-"] <Digit> { <Digit> }

<UnOp> :: = " +" | "-"

<BinOp> :: = " +" | "-" | " *" | "/" | "%"

Beispiel


```
erg = n++ * n-- / 2;
```




```
int h1 = n;
n = n + 1;
int h2 = n;
n =  n - 1;
int h3 = h1 * h2;
int h4 = 2;
int h5 = h3 / h4;

erg = h5;
```

Das ist aber meist sehr schlechter Stil: **Seiteneffekte!**