



Wintersemester 2006/07

**Einführung in die Informatik für
Naturwissenschaftler und Ingenieure**
(alias **Einführung in die Programmierung**)
(Vorlesung)

Prof. Dr. Günter Rudolph
Fachbereich Informatik
Lehrstuhl für Algorithm Engineering

**Inhalt**

- Erste Programme (mit Ausgabe)
- Exkurs: Grammatiken

**Das erste C++ Programm:**

```
#include <iostream>

int main()
{
    std::cout << "Das ist eine Zeichenkette!" << '\n';
    return 0;
}
```

- `#include <iostream>` bindet Ein-/Ausgabemöglichkeit aus Bibliothek ein
- `int main()` kennzeichnet Hauptprogramm, gibt Datentyp integer zurück
- `std::cout` ist der Ausgabestrom; alles rechts von `<<` wird ausgegeben
- `return 0` gibt den Wert 0 an das Betriebssystem zurück (0: alles OK!)

**Noch ein C++ Programm:**

```
#include <iostream>
#include <climits>

int main()
{
    std::cout << "int:      "
              << INT_MIN << " ... "
              << INT_MAX << std::endl;
    return 0;
}
```

- `#include <climits>` bindet Konstanten für Wertebereiche ein
- `INT_MIN` und `INT_MAX` sind Konstanten aus Bibliothek `climits`
- `std::endl` ist eine Konstante für Beginn einer neuen Zeile



Einfache Datentypen

- Logischer Datentyp `bool`

- Zum Speichern von Wahrheitswerten „wahr“ und „falsch“
- Wertevorrat: `true` und `false`
- Datendefinition: `bool b;`
- Zuweisung: `b = true;`
oder: `int x = 9; b = x > 7;`
- Zum Überprüfen von **Bedingungen**
- Operationen:

Name	C/C++	Beispiel
AND	<code>&&</code>	<code>b && x < 7</code>
OR	<code> </code>	<code>b x > 8</code>
NOT	<code>!</code>	<code>!b</code>



Wahrheitstabeln

A	B	A && B	A	B	A B	A	!A
false	false	false	false	false	false	false	true
false	true	false	false	true	true	false	true
true	false	false	true	false	true	true	false
true	true	true	true	true	true	true	false

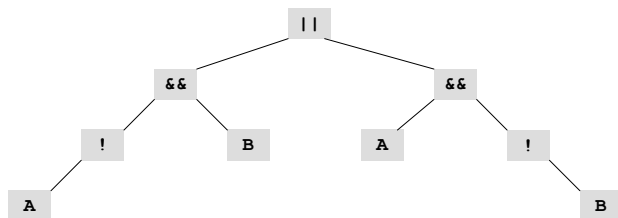
Priorität der Operatoren

1. NOT
2. AND
3. OR

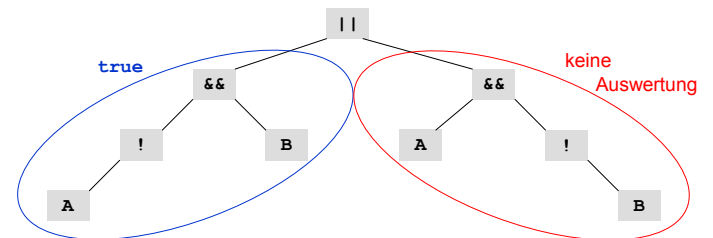


Weitere ableitbare Operationen

A NAND B	<code>!(A && B)</code>
A NOR B	<code>!(A B)</code>
A \Rightarrow B (Implikation)	<code>A !B</code>
A XOR B (Antivalenz)	<code>!A && B A && !B</code>



- Auswertung von links nach rechts
- Abbruch, sobald Ergebnis feststeht:
 - `A && false = false`
 - `A || true = true`
- Beispiel:
`bool A = false, B = true;`



Darstellung von Information

- Boolesche Ausdrücke

- Vergleiche: < kleiner
<= kleiner oder gleich
> größer
>= größer oder gleich
== gleich
!= ungleich

Achtung:

- == testet auf Gleichheit
- = wird bei einer Zuweisung verwendet

Darstellung von Information

Wofür werden boolesche Ausdrücke gebraucht?

- ... um Bedingungen formulieren zu können
- ... um den Kontrollfluss steuern zu können
- ... für Fallunterscheidungen: `if Bedingung wahr then mache etwas;`

```
#include <iostream>

int main()
{
    int a = 10, b = 20;
    if (a < b) std::cout << "kleiner";
    if (a > b) std::cout << "groesser";
    if (a == b) std::cout << "gleich";
    return 0;
}
```

später
mehr

Exkurs: namespace std

Im **Standard-Namensraum** wird **Standardfunktionalität** bereitgestellt:

- z.B. Ausgaben auf den Bildschirm, Eingaben von der Tastatur, ...

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10, b = 20;
    if (a < b) std::cout << "kleiner";
    if (a > b) std::cout << "groesser";
    if (a == b) std::cout << "gleich";
    return 0;
}
```

falls Compiler einen Bezeichner nicht findet, dann Erweiterung mit `std`.

Beispiel:

Bezeichner → ???
`std::Bezeichner` ☺

⇒ führt zu kleineren Programmtexten

Darstellung von Information

Anmerkung:

- In Programmiersprache C und vor 1993 auch in C++ existierte kein boolescher Datentyp!
- Stattdessen: Simulation mit Datentyp `int`
- Konvention: Wert ungleich Null bedeutet `true` sonst `false`
- Beispiele:
 - `int x = 8;`
`if (x) x = 0;`
 - `char c = 'y';`
`if (c) c = '\n';`
- Das ist auch jetzt noch möglich!
⇒ Empfehlung: Besser den booleschen Datentyp verwenden!

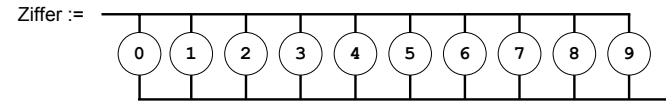


Woher weiß man, was man in C++ schreiben darf und was nicht?

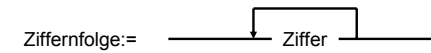
- Natürliche Sprache festgelegt durch
 - Alfabeth
 - Orthografie
 - Wortbedeutungen
 - Grammatik
- Aktueller C++ Standard: ISO/IEC 14882:2002
- Es wurde u.a. eine formale Grammatik für C++ festgelegt (für alle verbindlich).



Grafische Darstellung



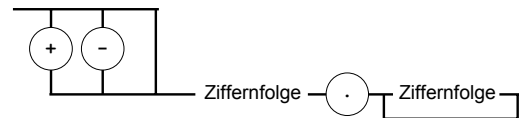
Ohne Pfeile: „von links nach rechts, von oben nach unten“



Ganzzahl mit Vorzeichen :=



Festkommazahlen :=



Grafische vs. textuelle Darstellung von Grammatiken

- Grafische Darstellung anschaulich aber Platz raubend
- Textuelle Darstellung kompakter und automatisch zu verarbeiten

Ziel

- Beschreibung von syntaktisch korrekten C++ Programmen

Konkreter

- Sie sollen lernen, formale Grammatiken zu lesen und zu verstehen,
 - um sie in dieser Veranstaltung für ihre Zwecke nutzen zu können,
 - um einen fundamentalen Formalismus in der Informatik kennen zu lernen,
 - um andere Programmiersprachen leichter erlernen zu können.

Definition

Eine kontextfreie Grammatik $G = (N, T, S, P)$ besteht aus

- einer endlichen Menge von Nichtterminalen N ,
- einer endlichen Menge von Terminalen T ,
- einem Startsymbol $S \in N$,
- einer endlichen Menge von Produktionsregeln der Form $u \rightarrow v$, wobei
 - $u \in N$ und
 - v eine endliche Sequenz von Elementen von N und T ist, sowie
- der Randbedingung $N \cap T = \emptyset$.

Beispiel

$T = \{ +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$

$N = \{ Z, A, D \}$

$S = \{ Z \}$

$Z \rightarrow +A$

$Z \rightarrow -A$

$Z \rightarrow A$

$A \rightarrow D$

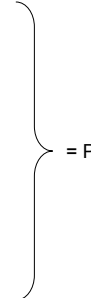
$A \rightarrow AD$

$D \rightarrow 0$

$D \rightarrow 1$

...

$D \rightarrow 9$



Kompaktere Notation:

$Z \rightarrow +A \mid -A \mid A$

$A \rightarrow D \mid AD$

$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Beispiel

$T = \{ +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$

$N = \{ Z, A, D \}$

$S = \{ Z \}$

$Z \rightarrow +A \mid -A \mid A$

$A \rightarrow D \mid AD$

$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

- Nichtterminale sind Platzhalter.
- Man kann dort eine Produktionsregel anwenden.
- Der Ersetzungsprozess endet, wenn alle Nichtterminale durch Terminale ersetzt worden sind.

Beispiel

$T = \{ +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$

$N = \{ Z, A, D \}$

$S = \{ Z \}$

$Z \rightarrow +A \mid -A \mid A$

$A \rightarrow D \mid AD$

$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Können wir mit dieser Grammatik +911 erzeugen?

Start mit $Z \rightarrow +A$, wende Produktionsregel $A \rightarrow AD$ auf A an, ergibt $Z \rightarrow +AD$

Wende $A \rightarrow AD$ auf A an, ergibt $Z \rightarrow +ADD$

Wende $A \rightarrow D$ auf A an, ergibt $Z \rightarrow +DDD$,

Wende $D \rightarrow 9$ auf das erste D , $D \rightarrow 1$ auf die übrigen D an, ergibt $Z \rightarrow +911$.



Notation der Grammatik im C++ Buch von Bjarne Stroustrup

- **Nichtterminale:** Wörter in *kursiver* Schrift
- **Terminale:** Zeichen in nicht *proportionaler* Schrift
- Alternativen wie
 - $D \rightarrow 0|1|2|3|4|5|6|7|8|9$ sind dargestellt via
 - D : eins von
0 1 2 3 4 5 6 7 8 9
- Optionale (Nicht-)Terminale durch tiefgestelltes *opt*
 - $sign_{opt}$



Beispiel: Bezeichner

- *identifizier*:
 - nondigit*
 - identifizier nondigit*
 - identifizier digit*
- *nondigit*: eins von
universal-character-name
_ a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- *digit*: eins von
0 1 2 3 4 5 6 7 8 9
- *universal-character-name*:
\u *hex-quad*
\U *hex-quad hex-quad*
- *hex-quad*:
hex hex hex hex
- *hex*: eins von
digit
a b c d e f
A B C D E F