Programmierung

Grundlagen

PG

Tina Wegener, Ralph Steyer

2. Ausgabe, 1. Aktualisierung, April 2014

HERDT

6 Grundlegende Sprachelemente

In diesem Kapitel erfahren Sie

- was Syntax und Semantik bedeuten
- was Datentypen, Variablen und Konstanten sind
- welche Operatoren es gibt
- was unter Ausdrücken zu verstehen ist

6.1 Syntax und Semantik

Programmiersprachen sind, wie auch natürliche Sprachen, nach definierten Regeln aufgebaut. Diese Regeln legen fest, welche Zeichen verwendet werden dürfen, wie die Zeichen angeordnet sein müssen und welche Bedeutung bestimmte Zeichenfolgen haben.

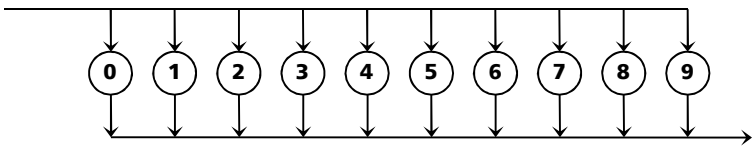
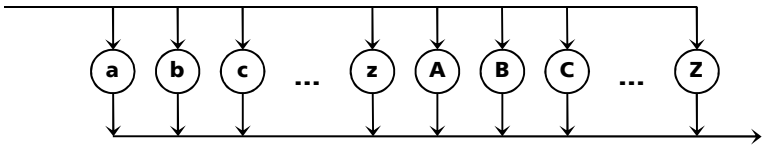
- Die **Syntax** einer Sprache bestimmt den Aufbau der Sätze. Auf Programmiersprachen bezogen legt die Syntax z. B. fest, wie Anweisungen aufgebaut sind.
- Die **Semantik** erklärt die Bedeutung der Sätze. Auf Programmiersprachen bezogen wird durch die Semantik z. B. beschrieben, was eine Anweisung bedeutet.

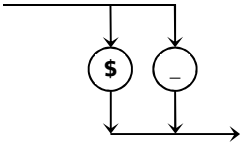
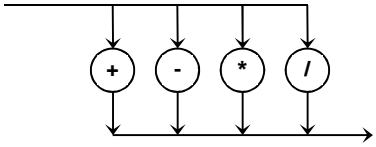
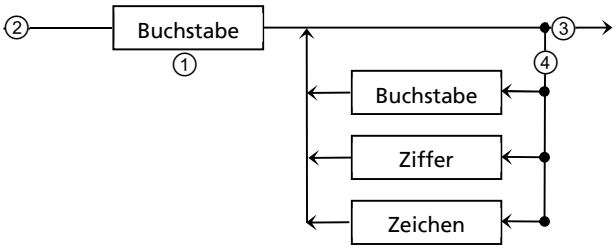
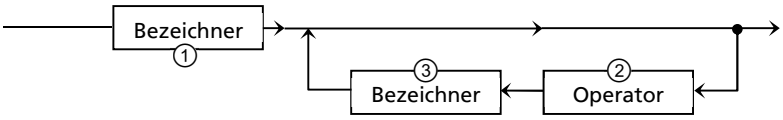
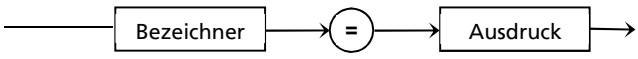
Die Syntax einer Sprache kann mithilfe von Syntaxdiagrammen oder der Backus-Naur-Form dargestellt werden. In Syntaxdiagrammen wird die Grammatik einer Sprache grafisch dargestellt und ist dadurch leichter lesbar. Die Backus-Naur-Form (BNF) verwendet eine textuelle Darstellung der Syntax und ist dadurch mit jedem Texteditor zu erfassen. In Sprachbeschreibungen wird häufig die erweiterte Backus-Naur-Form (EBNF) verwendet, die im Vergleich zur BNF mehr Möglichkeiten bietet.

Darstellung der Syntaxdiagramme

Die Syntaxdiagramme einer Sprache sind sehr umfangreich, da alle Konstrukte einer Sprache beschrieben werden. Das folgende Beispiel zeigt, wie ein numerischer Ausdruck einer Sprache dargestellt werden kann.

Beispiel: Syntaxdiagramme

Syntaxbegriff	Syntaxdiagramm und Bedeutung
<Ziffer>	 Eine Ziffer kann ein Wert von 0 bis 9 sein.
<Buchstabe>	 Ein Buchstabe kann jeder kleine und große Buchstabe des Alphabets sein.

Syntaxbegriff	Syntaxdiagramm und Bedeutung
<Zeichen>	 Ein Zeichen kann entweder $ oder _ sein.
<Operator>	 Ein Operator ist eines der Zeichen +, -, * oder /.
<Bezeichner>	 Ein Bezeichner muss mit einem Buchstaben ① beginnen. Anschließend können weitere Bestandteile folgen oder nichts. Dies wird durch die Pfeile und die Linien veranschaulicht. Wenn Sie die Linie vom Startpunkt ② aus verfolgen, kommen Sie auf jeden Fall zum Buchstaben ①. Vom Buchstaben aus gelangen Sie an einen Knotenpunkt ③. Verfolgen Sie die Linie geradeaus weiter, erreichen Sie das Ende (der Bezeichner besteht in diesem Fall aus genau einem Buchstaben). Verfolgen Sie aber die vertikale Linie ④, gelangen Sie entweder zu einem weiteren Buchstaben, einer Ziffer oder einem Zeichen. Danach kehren Sie wieder auf die ursprüngliche Linie und somit zu dem Knotenpunkt zurück. Nun können Sie den Weg beenden oder wieder den vertikalen Weg gehen.
<Ausdruck>	 Ein Ausdruck kann ein Bezeichner ① sein oder ein Bezeichner ①, gefolgt von einem Operator ② und einem Bezeichner ③. Ein Ausdruck kann aber auch mehr als zwei (beliebig viele) Bezeichner, die jeweils durch Operatoren miteinander verknüpft sind, beinhalten.
<Zuweisung>	 Eine Zuweisung besteht aus einem Bezeichner, dem Zuweisungsoperator = und einem Ausdruck.

Das Beispiel bezieht sich auf keine spezielle Sprache, sondern soll das Darstellungsprinzip verdeutlichen.

Darstellung der EBNF

Symbol	Beschreibung
`<begriff>`	Syntaxbegriff
`::=`	"ist definiert als" oder linke Seite "kann ersetzt werden durch" rechte Seite
{ }	Der in Klammern eingeschlossene Teil kann beliebig oft wiederholt werden oder entfallen.
`[ ]`	Der in eckigen Klammern eingeschlossene Teil ist optional (kann auch entfallen).
`<a>|<b>`	Alternative (logisches Oder) bedeutet: <a> *oder* <b> wird verwendet.
`"x"`	Zeichen oder Wörter in Anführungszeichen sind Bestandteile der Sprache und müssen genauso geschrieben werden.

Beispiel: EBNF

Die oben aufgeführten Syntaxdiagramme wurden hier in die EBNF umgesetzt.

```
<ziffer>       ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
<buchstabe>    ::= "a"|"b"|"c"|...|"z"|"A"|"B"|"C"|...|"Z"
<zeichen>      ::= "$"|"_"
<operator>     ::= "+"|"-"|"*"|"/"
<bezeichner>   ::= <buchstabe>{<buchstabe>|<ziffer>|<zeichen>}
<ausdruck>     ::= {<bezeichner> <operator>} <bezeichner>
<zuweisung>    ::= <bezeichner> "=" <ausdruck>
```

Die Beschreibung einer Sprache in der EBNF muss immer bis in die kleinste syntaktische Einheit (Zeichen der Sprache) auflösbar sein. Bevor ein Ausdruck, wie

```
<alternative> ::= "if (" <logischer ausdruck> ") " <anweisung> ["else"  <anweisung>]
```

angegeben werden kann, müssen zuvor die Syntaxbegriffe `<logischer ausdruck>` und `<anweisung>` erklärt sein.

Hinweis zur Schreibweise der Syntax im Buch

- ✔ Schlüsselwörter werden fett hervorgehoben.
- ✔ Optionale Angaben stehen in eckigen Klammern [ ].
- ✔ Drei Punkte (...) kennzeichnen, dass weitere Angaben folgen können.
- ✔ Sofern eckige Klammern oder drei Punkte (...) als Bestandteil des Quelltextes erforderlich sind, wird darauf in der Erläuterung explizit hingewiesen.

6.2 Grundlegende Elemente einer Sprache

Alle Sprachen werden durch ihre eigene Syntax und Semantik beschrieben. Sprachen können sich z. B. hinsichtlich des Aufbaus von Bezeichnern, der zulässigen Datentypen (vgl. Abschnitt 6.3) und der verfügbaren Operationen (vgl. Abschnitt 6.6) unterscheiden. Einige Elemente oder Mechanismen sind Bestandteil vieler Sprachen und sich sehr ähnlich. Diese werden im weiteren Verlauf des Buchs exemplarisch erklärt.

Bezeichner

Bezeichner (engl. identifier) geben Dingen in Programmen, z. B. Variablen (vgl. Abschnitt 6.5), einen Namen, über den sie im Programm angesprochen werden können. Die Anzahl der Zeichen und welche Zeichen zulässig sind, ist in den Sprachen verschieden. Einige Sprachen unterscheiden auch zwischen Groß- und Kleinschreibung (**case-sensitive**).

In Programmen können Sie als Bezeichner beispielsweise sowohl englische als auch deutsche Namen wählen. Innerhalb eines Programms sollte aber eine einheitliche Schreibweise gewählt werden. Englische Namen haben den Vorteil, dass sie oft kürzer sind, keine Umlaute enthalten, zu den reservierten Wörtern passen und, falls Programme international eingesetzt werden sollen, besser verständlich sind.

Bezeichner in Java

Für den Aufbau eines Bezeichners in Java gelten folgende Regeln:

- ✔ Bezeichner müssen mit 'a' ... 'z', 'A' ... 'Z', '_' oder '$' beginnen und können dann beliebig fortgesetzt werden.
- ✔ Bezeichner beinhalten keine Leerzeichen und keine Sonderzeichen.
- ✔ Java unterscheidet zwischen Groß- und Kleinschreibung. `Name`, `NAME` und `name` sind drei unterschiedliche Bezeichner für Elemente in Java.
- ✔ Java verwendet den Bezeichner in seiner kompletten Länge (alle Stellen sind signifikant).

In Java können auch Buchstaben aus anderen Alphabeten genutzt werden, da Java den Unicode-Zeichencode verwendet. Dies ist allerdings nicht zu empfehlen, da z. B. Probleme mit Editoren und Dateisystemen auftreten können. Empfehlenswert ist es, die Zeichen 'a' ... 'z', 'A' ... 'Z', '_' und '0' ... '9' zu verwenden.

Reservierte Wörter

In jeder Sprache ist eine Anzahl von Wörtern als Schlüsselwörter definiert. Diese reservierten Wörter haben in der Programmiersprache eine spezielle Bedeutung.

Alle reservierten Wörter in Java

`abstract`	`const` ②	`final`	`int`	`public`	`throw`
`assert`	`continue`	`finally`	`interface`	`return`	`throws`
`boolean`	`default`	`float`	`long`	`short`	`transient`
`break`	`do`	`for`	`native`	`static`	`true` ①
`byte`	`double`	`goto` ③	`new`	`strictfp`	`try`
`case`	`else`	`if`	`null` ①	`super`	`void`
`catch`	`enum`	`implements`	`package`	`switch`	`volatile`
`char`	`extends`	`import`	`private`	`synchronized`	`while`
`class`	`false` ①	`instanceof`	`protected`	`this`	

- ✔ Die drei mit ① gekennzeichneten reservierten Wörter (`false`, `null`, `true`) stellen konstante vordefinierte Werte dar und werden als **Literale** bezeichnet.
- ✔ Alle anderen hier aufgeführten reservierten Wörter sind sogenannte **Schlüsselwörter**.
- ✔ Alle reservierten Wörter dürfen **nicht** als Bezeichner verwendet werden.
- ✔ `const` ② und `goto` ③ sind ebenfalls reservierte Schlüsselwörter, die allerdings in Java nicht verwendet werden. Trotzdem können keine Variablen mit diesen Namen deklariert werden.

Schlüsselwörter werden in der Darstellung von Quelltextauszügen in diesem Buch **fett** hervorgehoben.

Kommentare

Zur Dokumentation eines Quelltextes verwenden Sie Kommentare. Diese sollen Ihnen und anderen Mitarbeitern an einem Projekt helfen, den Quelltext auch nach längerer Zeit noch zu verstehen. Kommentare sollten beispielsweise verwendet werden, um

- ✔ den Zweck von Variablen zu beschreiben,
- ✔ die Verwendung einer Methode (bzw. Funktion oder Prozedur) zu erläutern,
- ✔ einen nicht selbsterklärenden Algorithmus zu beschreiben.

Durch Kommentare wird die Größe des ausführbaren Programms nicht erhöht. Kommentare werden vom Compiler nicht berücksichtigt.

Die verschiedenen Programmiersprachen lassen unterschiedliche Arten von Kommentaren zu. Kommentare werden durch spezielle für die Programmiersprache festgelegte Zeichen eingeleitet bzw. eingeschlossen.

Beispiel: Kommentartypen in Java

Einzeilige Kommentare	`//`	Kommentar bis zum Ende der Zeile Alle Zeichen der Zeile hinter // werden vom Compiler überlesen.
(Mehrzeiliger) Kommentarblock	`/*` `*/`	Kommentar über mehrere Zeilen Ab der Zeichenkombination /* werden alle Zeichen im Quelltext überlesen, bis die Zeichenkombination */ auftritt.

6.3 Standarddatentypen (elementare Datentypen)

In einem Programm verarbeiten Sie Daten, die sich in ihrer Art unterscheiden:

- ✔ Zahlen - numerisch
- ✔ Zeichen - alphanumerisch
- ✔ boolesche Daten - logisch

Ein **Datentyp** ist eine Menge darstellbarer Werte. Viele Programmiersprachen besitzen vordefinierte elementare Datentypen, auch primitive Datentypen genannt. Diese Datentypen unterscheiden sich in der Art der Daten und in dem zulässigen Wertebereich.

Numerische Datentypen

Numerische Datentypen lassen sich, wie in der Mathematik, in ganze Zahlen (Integer) und reelle Zahlen (Real) unterteilen. Dabei sind die Wertebereiche und die Genauigkeit der Zahlen durch die Festlegung des Speicherbereichs begrenzt. Numerische Datentypen werden z. B. für Berechnungen eingesetzt.

Integer-Datentypen

Integer-Datentypen sind **Ganzzahlen** und besitzen keine Nachkommastellen. Eine Programmiersprache bietet meist mehrere Integer-Datentypen an, die sich durch die Bezeichnung und die Größe des Wertebereichs unterscheiden. Die Größe des Wertebereichs ist von der Größe des Speicherplatzes abhängig, der für den Datentyp vorgesehen ist, z. B. 2, 4 oder 8 Byte.

Der kleinste bzw. größte darstellbare Wert eines **vorzeichenlosen Integer-Datentyps** kann wie folgt berechnet werden. Für `x` wird dabei die zulässige Speichergröße in Bit eingesetzt: $0 \ldots 2^x - 1$

Für **Integer-Datentypen mit Vorzeichen** kann folgende Formel verwendet werden: $-2^{x-1} \ldots 2^{x-1}-1$
Ein Bit wird dabei für die Speicherung des Vorzeichens verwendet.

Beispiel: Integer-Datentypen in Java

Datentyp	Wertebereich	Speichergröße
`byte`	`-128 ... 127`	1 Byte
`short`	`-32768 ... 32767`	2 Byte
`int`	`-2.147.483.648 ... 2.147.483.647`	4 Byte
`long`	`-9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807`	8 Byte

- ✔ Integer-Datentypen werden im Computer immer genau dargestellt.
- ✔ Es treten keine Rundungsfehler bei der Darstellung der Zahlen auf.

Üblicherweise werden Sie für ganzzahlige Werte den Datentyp `int` verwenden, denn er bietet für die meisten Anwendungsfälle einen ausreichenden Wertebereich.

Gleitkomma-Datentypen

Für **Fließkommazahlen** (Dezimalzahlen) werden Gleitkomma-Datentypen mit Vorzeichen verwendet. Der Computer kann jedoch nicht jede Zahl genau darstellen. Dies führt auch bei einfachen Rechnungen zu Rundungsfehlern. Je nach verwendetem Typ lassen sich nur Zahlen mit einer bestimmten Genauigkeit abbilden. Für eine höhere Genauigkeit wird aber auch mehr Speicherplatz benötigt. Die meisten Programmiersprachen besitzen zwei Gleitkomma-Datentypen: Single Precision (einfache Genauigkeit) und Double Precision (doppelte Genauigkeit). Einige Sprachen bieten zusätzlich eine Extended Precision (höhere Genauigkeit).

Für die Speicherung einer Gleitkommazahl einfacher Genauigkeit stehen 32 Bit zur Verfügung.

- ✔ Ein Bit wird für das Vorzeichen (VZ) der Zahl (0 für positiv, 1 für negativ) verwendet.
- ✔ 8 Bit beschreiben den Exponenten.
- ✔ Für die Stellen der Zahl (Mantisse) bleiben dann noch 23 Bit übrig. Damit können 7 Dezimalstellen abgelegt werden.

Gleitkommazahlen mit doppelter Genauigkeit werden standardmäßig in 64 Bit gespeichert.

Die Bezeichnung des Datentyps ist in den jeweiligen Programmiersprachen unterschiedlich.

Beispiel: Gleitkomma-Datentypen in Java

Datentyp	Genauigkeit	Speichergröße
`float`	7 Stellen	4 Byte
`double`	15 Stellen	8 Byte

Üblicherweise werden Sie für Dezimalzahlen den Datentyp `double` verwenden, denn die Computer verfügen zumeist über ausreichenden Speicherplatz, und beim Datentyp `float` machen sich Rundungsfehler deutlich bemerkbar.

Zeichen-Datentyp

Zeichen-Datentypen können beliebige Zeichen des ASCII-Zeichensatzes enthalten. Sie sind nicht auf Zahlen und Buchstaben begrenzt und können auch Sonderzeichen !"§$%&/ speichern. Einzelne Zeichen werden bei der Wertzuweisung durch einfache Hochkommata eingeschlossen. Es ist von der Sprache abhängig, ob Zeichen intern als ASCII- oder Unicode-Zeichen dargestellt werden.

Für Ausgaben (akustisch, auf Bildschirm oder Drucker) werden häufig Steuerzeichen (nicht druckbare Zeichen, auch Escape-Sequenzen genannt) benötigt. Dazu wird auch der Zeichen-Datentyp verwendet.

Escape-Sequenz	Beschreibung
`\a`	Klingelzeichen
`\n`	Führt einen Zeilenvorschub und Wagenrücklauf durch, d. h. springt an den Anfang der nächsten Zeile (NL - new line, LF - line feed)
`\t`	Fügt einen horizontalen Tabulator ein (HT - horizontal tabulator)
`\\`, `\'`, `\?`, `\"`	Stellt das zweite Zeichen dar (\, ', ?, ")

Beispiel: Zeichen-Datentyp in Java

Datentyp	Wertebereich	Speichergröße
`char`	Alle Unicode-Zeichen	2 Byte

Logischer Datentyp

George Boole entwickelte im 19. Jahrhundert eine nach ihm benannte Algebra, die boolesche Algebra. Der Grundgedanke ist, dass es nur die beiden Wahrheitswerte wahr (true) und falsch (false) gibt. Eine solche Variable, die nur diese beiden Werte annehmen kann, ist vom logischen Datentyp `boolean`.

Auch wenn ein Bit im Prinzip für die Speicherung eines logischen Wertes ausreicht, wird meist ein Byte für dessen Speicherung benötigt, da ein Bit keine adressierbare Einheit im Rechner ist. Ein Byte ist die kleinste adressierbare (direkt ansprechbare) Einheit im Rechner.

- ✔ Einige Sprachen bieten zusätzlich einen Datentyp `Bit String`, der eine Sammlung von logischen Werten darstellt, die durch die einzelnen Bit-Werte repräsentiert werden.
- ✔ Nicht in allen Sprachen gibt es einen speziellen logischen Datentyp. Er wird dort meist mithilfe von ganzzahligen Werten verwaltet. Dabei steht 0 für falsch und jeder andere Wert für wahr.

Beispiel: Logischer Datentyp in Java

Datentyp	Wertebereich	Speichergröße
`boolean`	`true, false`	1 Byte

Rangfolge der Operatoren

Oft ist es notwendig, mehr als zwei Operanden miteinander zu vergleichen, z. B.:
`(amount1 > 500) || (amount2 < 801) && !(amount3 == 400)`

Die Abarbeitungsreihenfolge für diese Anweisung kann in verschiedenen Programmiersprachen unterschiedliche Ergebnisse liefern. Das liegt daran, dass der Ausdruck nach bestimmten Vorrangregeln für die Operatoren abgearbeitet wird. Einige Sprachen arbeiten die Ausdrücke prinzipiell von links nach rechts ab.

Wollen Sie sichergehen, dass die Ausdrücke in der richtigen Reihenfolge abgearbeitet werden, setzen Sie Klammern. Wie in der Mathematik werden die Bedingungen in Klammern **immer** zuerst ausgewertet. Auch die Lesbarkeit Ihres Programms wird durch das Setzen von Klammern erhöht.

6.8 Übungen

Übung 1: Arbeiten mit grundlegenden Sprachelementen

Übungsdatei: -- **Ergebnisdatei: *uebung06.pdf***

① Was verstehen Sie unter einer Variablen, und welche Gültigkeitsbereiche gibt es?

② Welche Zahlenart kann eine mit `int` (integer) deklarierte Variable in Java darstellen?

③ Welche Zuweisungen sind nicht korrekt?

Initialisierungen	Zuweisungen
`int i = 0;`	`i = 256;`
`int j = 0;`	`j = -42314;`
`short k = 0;`	`k = j;`
`char b = 'a';`	`b = "Hallo";`

④ Zwei Variablen, `var1` und `var2`, vom Datentyp `char` (character) sind gegeben. Die Werte der beiden Variablen sollen vertauscht werden. Beschreiben Sie den Algorithmus im Pseudocode.

⑤ Stellen Sie die arithmetischen, logischen und Vergleichsoperatoren in einem Syntaxdiagramm und in der erweiterten BNF dar. Fassen Sie dabei unäre und binäre Operatoren in je einer Darstellung zusammen. Verwenden Sie jeweils eines der möglichen Zeichen pro Operator. Erzeugen Sie das Syntaxdiagramm und die erweiterte BNF für einen Ausdruck. Verwenden Sie dabei unäre und binäre Operatoren.

⑥ Werten Sie folgende Bedingungen aus:

- ✔ `(value1 > 500) || (value2 < 800) && !(value3 == 400)`
- ✔ `(value1 > 500) || ((value2 < 800) && !(value3 == 400))`

Die Variablen sollen folgende Werte enthalten:
`value1 = 501; value2 = 799; value3 = 400`
In den Bedingungen wird angenommen, dass die Abarbeitung von links nach rechts erfolgt.

Übung 2: Zwei Variablen vertauschen

Übungsdatei: -- **Ergebnisdatei: *ChangeValues.java***

① Setzen Sie den Pseudocode zum Vertauschen der zwei Variablen, `var1` und `var2`, vom Datentyp `char` (character) in Java um.

Mit dem Befehl `System.out.println (var1);` können Sie den Wert einer Variablen ausgeben. Um verschiedene Werte zusammen mit Text auszugeben, verwenden Sie den Befehl wie folgt: `System.out.println("Wert der ersten Variablen" + var1 + "hier kann noch mehr Ausgabetext stehen." + var2);`