Kompilation von Code

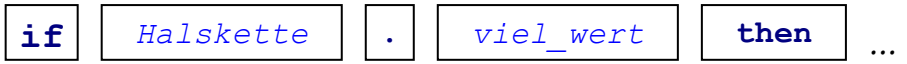
Der Text einer Eiffelklasse ***DIEB***, wie zum Beispiel

```
class DIEB inherits KRIMINELLER
feature
        stehlen is
                -- stiehlt Daten von anderen
        do
                if Halskette.viel_wert then
                        Halskette.nehmen
                end
        end
end
```

nennt man *Quelltext* (*sourcecode*). Der Quelltext ist die Quelle eines zukünftigen Programmes. Der Quelltext selbst ist noch kein Programm im eigentlichen Sinn, weil der Computer damit nicht direkt etwas anfangen kann. Vergleichbar mit einer Geschichte auf japanisch, welche du wahrscheinlich auch nicht verstehst, die aber trotzdem eine Geschichte ist.

Damit der Computer den Sinn des Quelltext versteht und es laufen lassen kann, muss er zuerst in die Maschinensprache übersetzt werden. Das Programm, das den Quelltext in Maschinensprache übersetzt, heisst Compiler.

Der Compiler selbst ist wieder in verschiedene Teile aufgegliedert. So wie der Mensch liest er den Quelltext Zeichen für Zeichen von links nach rechts, beginnend mit der ersten Zeile und zuletzt mit der untersten Zeile. Er gruppiert dabei die Worte zu kleinen *Klötzen* (*tokens*). Ein Teil der Klasse ***DIEB*** sähe folgendermassen aus:



In einem zweiten Schritt versucht er zu prüfen, ob das Sinn ergibt. Das heisst, er überprüft die Grammatik. Wie hält der Compiler die Grammatik fest? Dafür haben John Backus und Peter Naur vor vielen Jahren eine Sprache für Grammatiken entwickelt – eine Metasprache namens *Backus-Naur-Form*.

John Backus

John Backus wurde am 3. Dezember 1924 in Philadelphia, geboren und ist ein Protagonist in der Welt der Informatik. Er hat zum Beispiel die erste Hochsprache FORTRAN erfunden. Dabei ist ihm aufgefallen, dass man zum Beschreiben der Syntax einer Sprache eine standardisierte Sprache benötigt – die BNF.

Peter Naur

Peter Naur wurde 1928 in Dänemark geboren. Er war obwohl seiner Leistungen (hat Algol 60 mitentworfen) nicht sehr bekannt – deshalb hies die BNF anfänglich auch nur Backus-Normal-Form, bis Donald Knuth vorschluf, das N für Naur zu verwenden.

Was ist Backus-Naur-Form?

In der Informatik benötigt man oft eine syntaktische Beschreibung, zum Beispiel für eine Programmiersprache wie Eiffel. Die Backus-Naur-Form

ist eine syntaktische Beschreibungssprache, welche eine standardisierte Grammatik hat.

Wie sieht eine BNF aus?

Syntaxregeln

Die BNF setzt sich aus mindestens einer oder auch mehreren Syntaxregeln zusammen. Eine Syntaxregel besteht aus einem Namen (links), einem Gleichheitszeichen und einer Reihe von Symbolen. Eine Syntaxregel wird immer mit einem Punkt abgeschlossen:

```
Buch = Titelseite Inhaltsverzeichnis Text.
```

Oft spricht man auch lediglich von Regel statt Syntaxregel.

Symbole und Terminalsymbole

Symbole sind "Verweise" auf Syntaxregeln, deren Name gleich lautet wie das Symbol. Beispielsweise das Symbol `Titelseite` im vorherigen Beispiel verweist auf eine Regel mit Namen `Titelseite`.
Terminalsymbole sind abschliessende Symbole, die nicht mehr auf weitere Regeln verweisen. Dank ihnen kommt man mit der BNF auch irgendwann mal zu einem Ende. Terminalsymbole werden immer in Anführungs- und Schlusszeichen geschrieben, zum Beispiel:

```
EineBestimmteZeichenkette = "Irgendein Satz kann hier
stehen".
```

Konkatenation

Symbole können einfach aneinandergereiht, indem sie mit einem Abstand getrennt werden.

```
A = B C.
```

aber nicht: `A = BC.` , weil sonst `BC` auch wieder ein Symbol wäre.

Alternative

Wenn man mehrere Symbole zur Auswahl stellen möchte, dann kann man dies mit dem senkrechten Strich Zeichen tun. Bei einer solchen oder-Auswahl kann immer nur ein und genau ein Symbol ausgewählt werden. Zum Beispiel kann ein Buchstabe folgendermassen definiert werden:

```
Vokal = "A" | "E" | "I" | "O" | "U".
```

Option

Wenn ein Symbol nur fakultativ ist, dann schreibt man das Symbol in einer eckigen Klammer. Zum Beispiel:

```
Paar = "Hans liebt Susanne" [" sehr!"].
```

Repetition

Soll ein Symbol null-, ein-, oder unendlich viel mal vorkommen, dann schreibt man das Symbol in einer geschweiften Klammer gefolgt von einem Stern. Zum Beispiel:

```
HeutigeTermine = { Zeit Termin ", " }*.
```

Soll ein Symbol ein- bis unendlich viel mal vorkommen, dann schreibt man das Symbol in einer geschweiften Klammer gefolgt von einem Plus-Zeichen. Zum Beispiel:

```
Namenliste = { Vorname ";" Nachname ZEILENWECHSEL }+.
```

Gruppierung

Wenn Symbole gruppiert werden sollen, so dass klar wird, welcher Operator (Alternative, Option etc.) welche Symbole mit einbezieht, dann verwendet man die normalen runden Klammern:

```
A = B | C | D.
```

ist unterschiedlich zu

```
A = ( B | C ) | D.
```

Rekursion

Oft möchte man Dinge verschachteln. Zum Beispiel in der Mathematik können Additionen und Multiplikationen verschachtelt werden, wobei sie in ihrer Form immer noch gültig bleiben:

```
Rechnung = Term [PlusOperator Term].
Term = Faktor [MalOperator Faktor].
Faktor = (Zahl {Zahl}) | "(" Rechnung ")".
Zahl = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"
|
"8" | "9".
PlusOperator = "+".
MalOperator = "*".
```

Das heisst also, dass beim fetten Rechnung erneut die obere Regel Rechnung angewendet wird, was zur Folge hat, dass eine verschachtelte Rechnung generiert werden kann, zum Beispiel:

```
1 + 5 * ( 4 + 3 )
```

wobei 4 + 3 erneut auf die Syntaxregel Rechnung verweist. Ein Ersetzungsdiagramm dazu sieht dann wie folgt aus:

Referenzen

BNF-Notation – http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html

John Backus – http://en.wikipedia.org/wiki/John_Backus

Peter Naur – http://en.wikipedia.org/wiki/Peter_Naur