Erweiterte Backus-Naur Form (EBNF)

- EBNF ist eine formale Syntaxbeschreibungssprache.
- Sie wurde erfunden von J.Backus (FORTRAN Entwickler) und P.Naur (Algol Entwickler).
- Wesentliches Merkmal ist die wechselseitig rekursive Definition der syntaktischen Kategorien einer Programmiersprache.

Bausteine der erweiterten Backus-Naur Form (EBNF)

Nicht-terminale Symbole

Nicht-terminale Symbole

bezeichnen die zu definierenden syntaktischen Kategorien.

Terminale Symbole

bezeichnen Symbole der Sprache, die wörtlich zu übernehmen sind. Darstellung: in ' und ' eingeschlossen.

BNF Symbole:

‘ = ' : Definitionszeichen

‘ | ' : Alternative

‘Φ' : Die leere Zeichenfolge

‘ .' : Das Ende einer Regel

' ' ' : Kennzeichnungen von Sprachsymbolen

Zusätzliche Symbole in EBNF:

'( )' , '[ ]' , '{ }' , '< >' : Klammern

Die Beschreibung einer Programmiersprache in EBNF Notation besteht aus:

- ➔einer Startregel,
- ➔sonstigen Regeln
- ➔terminalen und nicht-terminalen Symbolen

- Die terminalen Symbole sind nichts anderes als die Basiselemente der jeweiligen Sprache.
- Nicht-terminale Symbole beschreiben abgeleitete Konstrukte der jeweiligen Programmiersprache und müssen in Regeln erklärt werden.

EBNF Regeln:

Die Startregel definiert was ein Programm ist.
Regeln bestehen aus linken und rechten Seiten:

Regel:

linke Seite = rechte Seite .

Auf einer linken Seite kommt jeweils
genau ein nicht- terminales Symbol vor,
das durch die rechte Seite erklärt wird.

EBNF - rechte Seiten

Rechte Seiten enthalten:

- terminale Symbole, geschrieben in Anführungszeichen
- die leere Zeichenfolge Φ
- Alternativen gruppiert durch das Zeichen: |
- Geklammerte Ausdrücke (nicht in BNF)

EBNF - Klammern

Wir verwenden vier Arten von Klammern:

( ) Runde Klammern dienen lediglich der Gruppierung.

[ ] Eckige Klammern stehen für einen optionalen Inhalt, der Null oder einmal vorkommt.

{ } Geschweifte Klammern stehen für eine beliebige Wiederholung des Inhalts: 0-mal, 1-mal, 2-mal, ...

< > Spitze Klammern stehen für eine beliebige mindestens aber einmalige Wiederholung des Inhalts.

Die spitzen Klammern sind nicht unbedingt nötig ...

Alternative Konventionen für Klammern

( ) Runde Klammern dienen lediglich der Gruppierung.

[ ] Eckige Klammern stehen für einen optionalen Inhalt:
Wiederholung des Inhalts: 0-mal oder 1-mal

( )? Wie eckige Klammern.

( )* Wiederholung des Inhalts: beliebig oft - auch 0-mal.

( )+ Wiederholung des Inhalts: beliebig oft - aber mindestens 1-mal.

Wird gerne als Eingabe für Programme verwendet...

Syntax in EBNF: Ein einfaches Beispiel

```
Ausdruck = Einfacher-Ausdruck [ RelOp Einfacher-Ausdruck ] .

Einfacher-Ausdruck = [ AddOp ] Term { AddOp Term } .

Term = [ 'NOT' ] Faktor { MulOp Faktor } .

Faktor = Variable | '(' Ausdruck ')' .

Variable = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .

AddOp = '+' | '-' .

MulOp = '*' | '/' | 'DIV' | 'MOD' .

RelOp = '=' | '<>' | '<' | '<=' | '>' | '>='.
```

Syntax in BNF: Das selbe Beispiel nochmal!

```
Ausdruck = Einfacher-Ausdruck |
           Einfacher-Ausdruck RelOp Einfacher-Ausdruck .

Einfacher-Ausdruck = Term | AddOp Term |
                     Einfacher-Ausdruck AddOp Term .

Term = Faktor | 'NOT' Faktor | Term MulOp Faktor .

Faktor = Variable | '(' Ausdruck ')' .

Variable = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .

AddOp = '+' | '-' .

MulOp = '*' | '/' | 'DIV' | 'MOD' .

RelOp = '=' | '<> ' | '<' | '<=' | '>' | '>='.
```

Ableitungen

- Nimmt man die erste Regel als Startregel, dann kann man offensichtlich folgende Ausdrücke aus unserer Beispiel-Grammatik ableiten:

```
a+b
(a*b) <> (c DIV f)
```

- Nicht ableiten kann man:

```
a++b
(a*m) <> (c DIV f)
```