17. Nov

(E)BNF: kurz für "(Extended-)Backus-Naur-Form", ist ein rekursiver Weg Regeln zu definieren, welche bestimmen, wie Elemente eines Alphabets kombiniert werden dürfen

Bsp.

```
Array    = '[' Elements ']'
Elements = Element | Elements ',' Element
Element  = Integer | Array
Integer  = Digit | Integer Digit
Digit    = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
```

- Die Digits bilden hier das sogenannte Alphabet.
- Ein Integer ist dann rekursiv definiert und erzeugt eine aneinanderhängende Kette von Digits.
- Ein Element ist entweder (durch | gekennzeichnet) ein Integer oder ein Array.
- Elements sind dann wieder rekursiv definierte Ketten von Elementen
- Array sind dann Elements mit "[" und "]" versehen

Neben dem "Oder": | kommen mit der EBNF noch weitere Schreibweisen hinzu

{...} = kann ausgelassen oder beliebig oft wiederholt werden.

[...] = Optional (0-Mal oder 1-Mal)

Somit kann unser Beispiel wie folgt umgeschrieben werden:

BNF

```
Array    = '[' Elements ']'
Elements = Element | Element ',' Elements
Element  = Integer | Array
Integer  = Digit | Integer Digit
Digit    = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
```

EBNF

```
Array    = '[' Elements ']'
Elements = Element [',' Elements]
Element  = Integer | Array
Integer  = Digit {Digit}
Digit    = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
```

Wichtige Funktionen zur Parser-Programmierung

consume(in, c): in ist der Stream, c ein Charakter. Ist c der nächste Charakter wird true zurückgegeben und der Charakter consumed. Falls nicht, wird nur false zurückgegeben.

lookahead(in): in ist der Stream. Es wird der nächste **non-whitespace**-Charakter ausgegeben. Falls kein Input vorhanden, wird 0 ausgegeben.

peek(in): in ist der Stream. Es wird der nächste Charakter ausgegeben. Falls kein Input vorhanden, wird 0 ausgegeben.

struct: Ein struct ist ein "selbstgebauter Typ".

Form:

```
struct name {
    // Member-Variabeln
};
```

Ein Beispiel wäre

```
struct student {
    int alter;
    std::string name;
    std::vector<double> noten;
};
```

Initialisierung : `student harry = {19, "harry", {1, 1.1, 1.05}};`

Abrufen : `harry.alter`