## Was ist BNF?

BNF ist eine Schreibweise, die eine Syntax definiert. Mit ihr kann man schauen, ob gewisse Ausdrücke nach bestimmten Regeln gültig oder ungültig sind. Hier ist ein einfaches Beispiel:

```
snake = head tail | tail head
head = "o"
tail = "-" | "-" tail
```

Es werden drei Ausdrücke definiert.

- Ein Kopf (head) wird dargestellt durch `"o"`.
- In der Definition des Schwanzes (tail) wird diese Definition selber rekursiv erwähnt. Also nach der Regel muss der Schwanz mindestens ein Bindestrich `"-"` sein, darf aber auch aus mehreren bestehen: `"----"`.
- Eine Schlange (snake) wurde nun definiert als: Kopf Schwanz ODER Schwanz Kopf. Mögliche Beispiele dafür sind: `"o-"`, `"o----"`, `"----o"` oder `"-o"`.
  Nach dieser Definition sind folgende Ausdrücke keine korrekten Schlangen: `"----"` (Kopf fehlt), `"o"` (Schwanz fehlt), `"--o--"` (Kopf nicht am Ende), `"----oo"` (zu viele Köpfe),...

## Implementierung

Zwei Funktionen wurden aus dem Programm calculator.cpp, das in der Vorlesung erklärt wurde, übernommen.

- `char lookahead (std::istream& is)` retouniert den nächsten Charakter im istream ohne diesen einzulesen. Es wird 0 retouniert, falls es keinen weiteren Charakter gibt.
- `bool consume (std::istream& is, char c)` liest das nächste Element im istream ein, falls es dem Charakter c entspricht. Falls das gelingt wird true retouniert, andernfalls false.

Nun zu der Implementierung von Kopf, Schwanz und Schlange:

- Beim Kopf wird lediglich geschaut, ob der eingezogene Charakter einem 'o' entspricht.

```
bool head (std::istream& is){
    return consume(is, 'o');
}
```

- Beim Schwanz wird zuerst ein Bindestrich erwartet. Danach wird geschaut, ob das nächste Element auch ein Bindestrich ist, in dem Fall ruft sich die Funktion rekursiv nochmals auf.

```
bool tail (std::istream& is){
    bool valid = consume(is, '-');
    if (lookahead(is) == '-')
        valid = valid && tail(is);
    return valid;
}
```

- Bei der Schlange gibt es zwei Fälle: Entweder ist der Kopf links, dann muss die Reihenfolge Kopf - Schwanz sein. Sonst schaut die Schlange nach rechts und es muss Schwanz - Kopf sein.

```
bool snake(std::istream& is){
    if (lookahead(is) == 'o')
        return head(is) && tail(is);
    else
        return tail(is) && head(is);
}
```

In der Funktion `test` wird getestet, ob der eingegebene Input eine Schlange ist und überprüft, dass der Stringstream nach der Schlange aufhört (`lookahead(input)== 0`).

Adrian Langenbach, 2018

## Ganzes Programm

Als letztes ist hier nun das ganze Programm zusammengesetzt:

```
// snake.cpp

// Syntax in BNF:
// ---------------
// snake = head tail | tail head
// head = "o"
// tail = "-" | "-" tail

#include<iostream>
#include<sstream>

// POST: leading whitespace characters are extracted
//       from is, and the first non-whitespace character
//       is returned (0 if there is no such character)
char lookahead (std::istream& is)
{
    is >> std::ws;
    if (is.eof())
        return 0;            // end of stream
    else
        return is.peek();    // next character in is
}

// POST: if next chararcer in is is ch, consume c and return
//       true, otherwise return false
bool consume  (std::istream& is, char c)
{
    if (lookahead (is) == c) {
    is >> c;
        return true;
  } else
        return false;
}

// PRE:  Valid stream is.
// POST: Returns true if stream contains a head and extracts it,
//       otherwise false.
bool head (std::istream& is){
// head = "o"

    return consume(is, 'o');
}


// PRE:  Valid stream is.
// POST: Returns true if stream contains a tail and extracts it,
//       otherwise false.
bool tail (std::istream& is){
// tail = "-" | "-" tail

    bool valid = consume(is, '-');
    if (lookahead(is) == '-')
        valid = valid && tail(is);
    return valid;
}

// PRE:  Valid stream is.
// POST: Returns true if stream contains a snake and extracts it,
//       otherwise false.
bool snake(std::istream& is){
```

```
// snake = head tail | tail head
    if (lookahead(is) == 'o')
        return head(is) && tail(is);
    else
        return tail(is) && head(is);
}

// PRE:  Valid stream input.
// POST: outputs "valid" if whole stream contains a snake, else output is
   "invalid".
void test(std::istream& input){
    if (snake(input) && lookahead(input) == 0)
        std::cout << "valid\n";
    else
        std::cout << "invalid\n";
}


int main()
{
    std::stringstream input ("o------"); //valid
    test(input);

    std::stringstream input2 ("oo------"); //invalid
    test(input2);

    std::stringstream input3 ("------"); //invalid
    test(input3);

    std::stringstream input4 ("-o"); //valid
    test(input4);

    std::stringstream input5 ("--o----"); //invalid
    test(input5);

    std::stringstream input6 ("------o"); //valid
    test(input6);

    return 0;
}
```