Informatik I – Übung 8

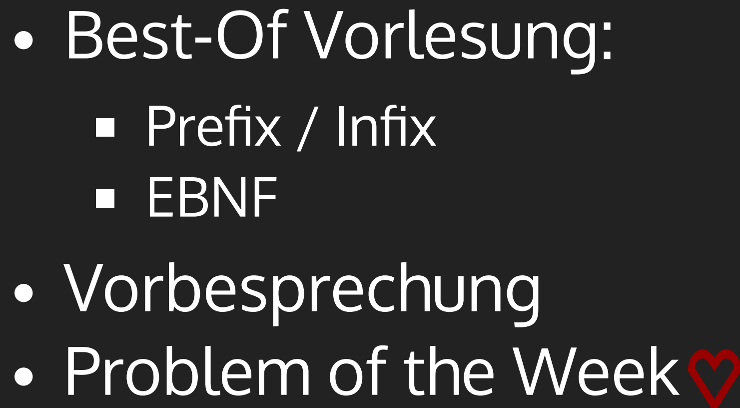
Pascal Schärli

pascscha@student.ethz.ch

12.04.2019

ETH zürich

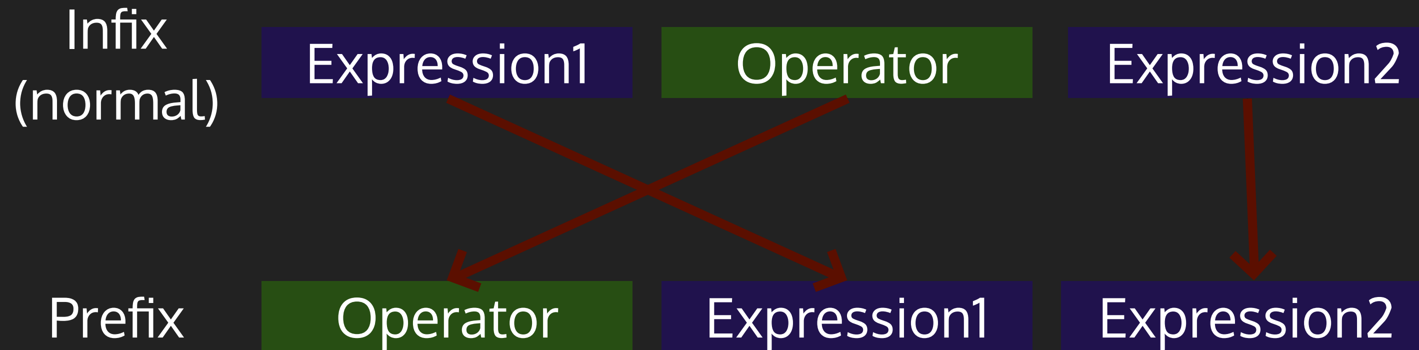
Was gibts heute?



- Best-Of Vorlesung:
  - Prefix / Infix
  - EBNF
- Vorbesprechung
- Problem of the Week ♡

Best of Vorlesung

Prefix Notation

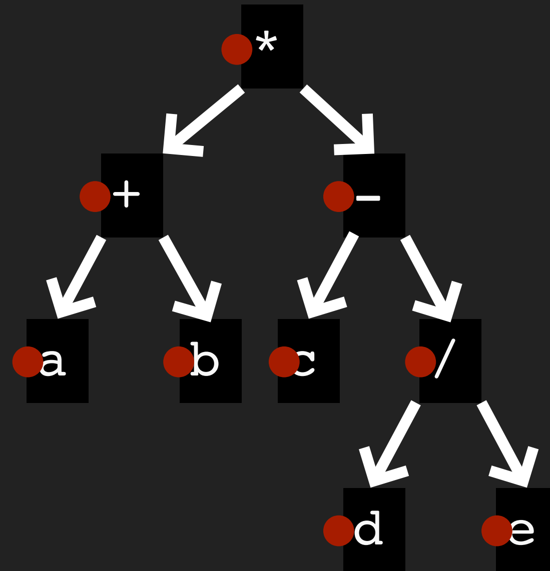


Infix	Prefix
`a + b`	`+ a b`
`a * b - c`	`- * a b c`

Prefix Notation

Gegen Uhrzeigersinn um den Baum Gehen und wenn man links von einer Node ist aufschreiben.

```
(a + b) * (c - d / e)
```




```
* + a b - c / d e
```

Infix vs Prefix Notation

Infix		Prefix
`a * b`		`* a b`
`e / f`		`/ e f`
`(a * b) / (c * d)`		`/ * a b * c d`
`a * b + c * d`		`+ * a b * c d`
`a + b * c + d`		`+ + a * b c d`
`(a + b) * (c - d / e)`		`* + a b - c / d e`

BNF

- Die **B**ackus-**N**aur-**F**orm (BNF) ist ein Rekursiver Weg um Elemente aus einem "Alphabet" mit gegebenen Regeln zusammenzusetzen.

- Die BNF ist eine Sammlung von Substitutionsregeln, mit welchen alle erlaubten Sätze erstellt werden können.

BNF – Beispiel

Alphabet: {A, a, _}

Regeln:

- "A" kann nur als das Erste Symbol oder direkt nach "_" kommen.
- "_" kann nicht als das Erste oder Letzte Symbol Kommen
- "_" kommt nur einzeln Vor.

```
1 seq = term | term "_" seq
2 term = "A" | "A" lowerterm | lowerterm
3 lowerterm = "a" | "a" lowerterm
```

Beispiel

Aaa_aa

Aaaa

aaaa

BNF – Beispiel

```
1 seq = term | term "_" seq
2 term = "A" | "A" lowerterm | lowerterm
3 lowerterm = "a" | "a" lowerterm
```

Kann man "Aaa_A" mit dieser BNF ausdrücken?

```
term "_" seq
"A" lowerterm "_" seq
"Aa" lowerterm "_" seq
"Aaa_" seq
"Aaa_" term
"Aaa_A"
```

EBNF

- Die **E**xtended-**B**ackus-**N**aur-**F**orm hat zusätzliche Syntax
  -> kompaktere Notationen
- {...} -> Inhalt kann beliebig viel (inkl. 0) Mal wiederholt werden.
- [...] -> Inhalt wird entweder 0 oder 1 Mal wiederholt

BNF

```
1 seq = term | term "_" seq
2 term = "A" | "A" lowerterm | lowerterm
3 lowerterm = "a" | "a" lowerterm
```

EBNF

```
1 seq = term [ "_" seq ]
2 term = "A" { "a" } | "a" { "a" }
```

EBNF

EBNF

```
1 seq = term [ "_" seq ]
2 term = "A" { "a" } | "a" { "a" }
```

A

Aaaa

aaaA

A_A

a

Aa_Aa

_

Aa___a

EBNF in C++

```
1 seq = term [ "_" seq ]
2 term = "A" { "a" } | "a" { "a" }
```

Für Jede Regel machen wir eine Funktion vom Typ bool, welche prüft ob die Regel erfüllt ist oder nicht.

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

// PRE:  Valid stream is.
// POST: Returns true if stream contains term anc
//        extracts it, otherwise false.
bool term(std::istream &is);
```

EBNF in C++

Wir erstellen eine lookahead Funktion, welche schaut, was das nächste Zeichen ist, ohne dies vom String zu entfernen.

```
// PRE:  Valid stream is.
// POST: Leading whitespace characters are extracted from is, and the
//       first non-whitespace character is returned (0 if there is none).
char lookahead (std::istream& is)
{
    is >> std::ws;           // skip whitespaces
    if (is.eof())
        return 0;            // end of stream
    else
       return is.peek();     // next character in is
}
```

EBNF in C++

Wir erstellen eine consume Funktion, welche einen Charakter vom Stream entfernt.

```
// PRE:  Valid stream is.
// POST: Reads char from stream and returns true if that char matches
//       the expected char, otherwise false is returned.
bool consume(std::istream& is, char expected)
{
    char actual;
    is >> actual;
    return actual == expected;
}
```

EBNF in C++

Die Sequenz Funktion prüft zuerst, ob wir einen Term haben. Danach prüft es ob noch ein optionales '_' kommt, falls ja folgt eine weitere Sequenz.

```
// PRE:  Valid stream is.
// POST: Returns true if stream contains seq and extracts it, otherwise fals
bool seq(std::istream& is){
    // seq = term [ "_" seq ]

    bool valid = term(is); // Every seq has to start with a term
    if(!valid){            // If it does not start with a term, return false
        return false;
    }

    if(lookahead(is) == '_'){ // check if we have another seq
        consume(is, '_');      // "consume" the "_" from the instream
        return seq(is);        // Check if the rest forms a valid seq
    }
    else{                  // If there is no following seq
        return true;
    }
}
```

EBNF in C++

Ein Term liest zuerst ein Buchstabe ein, dieser Kann 'A' oder 'a' sein.
Danach liest dieser alle darauf folgenden 'a' ein.

```
// PRE:  Valid stream is.
// POST: Returns true if stream contains term and extracts it,
//       otherwise false.
bool term(std::istream &is){
    // term = "A" { "a" } | "a" { "a" }
    char first = lookahead(is);
    if(first == 'A' | first == 'a'){
        consume(is, first);
        while(lookahead(is) == 'a'){
            consume(is, 'a');
        }
        return true;
    }
    else{
        return false;
    }
}
```

EBNF in C++

A a a _ A

```
1  bool seq(std::istream& is){
2      bool valid = term(is);
3      if(!valid){
4          return false;
5      }
6      if(lookahead(is) == '_'){
7          consume(is, '_');
8          return seq(is);
9      }
10     else{
11         return true;
12     }
13 }
```

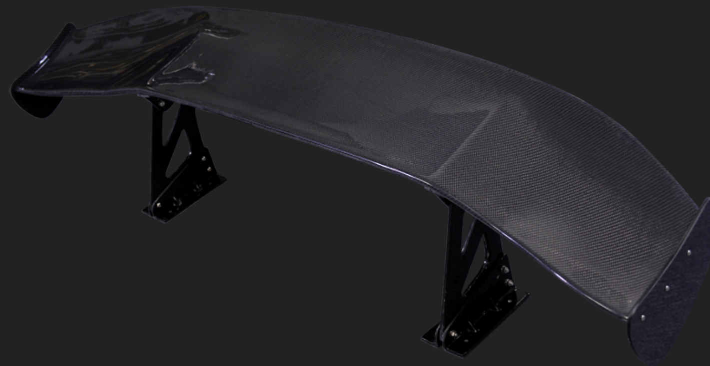
```
1  bool term(std::istream &is){
2      char first = lookahead(is);
3      if(first == 'A' | first == 'a')
4          consume(is, first);
5          while(lookahead(is) == 'a')
6              consume(is, 'a');
7          }
8          return true;
9      }
10     else{
11         return false;
12     }
13 }
```

EBNF in C++

Die Main-Funktion liest eine Zeile ein und übergibt diese dann der Prüfenden Funktion. Danach müssen wir mit lookahead noch prüfen, ob wir alles gelesen haben

```
int main()
{
    std::cout << "please enter a sequence:\n";

    std::string sequence;
    std::cin >> sequence;

    std::stringstream sequence_in(sequence);

    if (seq(sequence_in) && lookahead(sequence_in) == 0) {
        std::cout << "valid\n";
    } else {
        std::cout << "invalid\n";
    }

    return 0;
}
```

Vorbesprechung



1: Trains

```
train        = open | compositions
open         = loco cars
loco         = "*" | "*" loco
cars         = "-" | "-" cars
compositions = composition { composition }
composition  = "<" open loco ">"
```

Schreibt ein Programm, welches bestimmen kann ob ein eingelesener Input dieser EBNF entspricht.

Tipp:

Analog Zu unserem Beispiel

1: Trains

```
train        = open | compositions
open         = loco cars
loco         = "*" | "*" loco
cars         = "_" | "_" cars
compositions = composition { composition }
composition  = "<" open loco ">"
```

```
<*_*><***_____******>

<*_*>   <***_____******>

*_  *   ***_____  ******

*  _    ***  _____
```

2: Money $$$

Wie Viele Münz/Noten-Kombinationen gibt es für X Rappen?

Beispiel 20 Rappen - **4** Lösungen:

Schreibt ein *Rekursives* Programm, welches die Anzahl Möglichen Kombinationen für beliebige X Rappen berechnet.

2: Money $$$

Die Main-Funktion ist bereits geschrieben, alles was Ihr schreiben müsst ist ***eine*** Funktion partitions im File *partitions.cpp*

```
// PRE: 0 <= end <= v.size(), and
//      0 < denominations[0] < denominations[1] < .. denominators[end-1]
//      describes a (potentially empty)
//      sequence of denominations
// POST: return value is the number of ways to partition amount
//       using denominations from denominations[0], ..., denominations[end-1]
unsigned int partitions (unsigned int amount,
                         const std::vector<unsigned int> & denominations,
                         unsigned int end) {
    // Please write your code here ..
}
```

Denominations → Geldstücke

Anzahl Partitionen:

Es gibt 1 Weg kein Geld zu haben
Initiere Anzahl Partitionen mit 0
Wiederhole für alle denominations von 0 bis end:

1. Prüfe ob die denomination überhaupt Platz hat
2. Falls ja, Berechne die Anzahl Partitionen bei welchen alle Geldstücke höchstens so viel Wert sind wie die aktuelle denomination. (rekrursion)
3. Addiere den Wert von 2. zu deiner Anzahl Partitionen.

3: Prefix To Infix

Schreibt ein Programm, welches Prefix zu Infix umwandeln kann.

Input:

```
* + a b - c / d e
```

Output:

```
(a + b) * (c - d / e)
```

3: Prefix To Infix

```
// PRE: valid stream is with operator (+-*/) as first char
//      precedence_operator is the precedence of the operator
//      precedence_left is precedence of left sub-expression
//      precedence_right is precedence of right sub-expression
// POST: Read operator followed by two consecutive prefix expressions e_1 and
//       e_2 from is.
//       Print e_1 operator_name e_2 in infix notation with parenthesis around
//       if precedence_operator < precedence_left
void print_operator(std::istream& is, char operator_name,
                    int precedence_operator, int precedence_left,
                    int precedence_right){
    // TODO
}

// PRE: valid stream is,
//       int precedence describes the precedence of the expression to the left
// POST: Converts prefix expression e from input stream and prints it in infix
//       notation with according parenthesis
void convert(std::istream& is, int precedence_left){
    // TODO
}

int main () {
    convert(std::cin, 0);
    return 0;
}
```

3: Prefix To Infix

```
void print_operator(std::istream& is, char operator_name,
                    int precedence_operator, int precedence_left, int precedence_right)

void convert(std::istream& is, int precedence_left)
```

Precedences:	Operator	Right
+	0	0
-	0	1
/	1	1
*	1	2

```
convert(is, 0)
 └ print_operator(is, "-", 0, 0, 1)
    ├ convert(is, 0)
    └ convert(is, 1)
       └ print_operator(is, "+", 0, 1, 0)
          ├ convert(is, 1)
          └ convert(is, 0)
```

- a + b c

a - (b + c)

precedence_left > precedence_operator

Program of the Week

Viel Spass!

INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):
    IF LENGTH(LIST) < 2:
        RETURN LIST
    PIVOT = INT(LENGTH(LIST) / 2)
    A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
    B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
    // UMMMMM
    RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):
    // AN OPTIMIZED BOGOSORT
    // RUNS IN O(N LOG N)
    FOR N FROM 1 TO LOG(LENGTH(LIST)):
        SHUFFLE(LIST):
        IF ISSORTED(LIST):
            RETURN LIST
    RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBINTERVIEWQUICKSORT(LIST):
    OK SO YOU CHOOSE A PIVOT
    THEN DIVIDE THE LIST IN HALF
    FOR EACH HALF:
        CHECK TO SEE IF IT'S SORTED
            NO, WAIT, IT DOESN'T MATTER
        COMPARE EACH ELEMENT TO THE PIVOT
            THE BIGGER ONES GO IN A NEW LIST
            THE EQUAL ONES GO INTO, UH
            THE SECOND LIST FROM BEFORE
        HANG ON, LET ME NAME THE LISTS
            THIS IS LIST A
            THE NEW ONE IS LIST B
        PUT THE BIG ONES INTO LIST B
        NOW TAKE THE SECOND LIST
            CALL IT LIST, UH, A2
        WHICH ONE WAS THE PIVOT IN?
        SCRATCH ALL THAT
        IT JUST RECURSIVELY CALLS ITSELF
        UNTIL BOTH LISTS ARE EMPTY
            RIGHT?
        NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
    IF ISSORTED(LIST):
        RETURN LIST:
    IF ISSORTED(LIST): //THIS CAN'T BE HAPPENING
        RETURN LIST
    IF ISSORTED(LIST): // COME ON COME ON
        RETURN LIST
    // OH JEEZ
    // I'M GONNA BE IN SO MUCH TROUBLE
    LIST = [ ]
    SYSTEM("SHUTDOWN -H +5")
    SYSTEM("RM -RF ./")
    SYSTEM("RM -RF ~/*")
    SYSTEM("RM -RF /")
    SYSTEM("RD /S /Q C:\*") //PORTABILITY
    RETURN [1, 2, 3, 4, 5]
```