Exercise Week 10

GianAndrea Müller
mailto:muellegi@student.ethz

May 9, 2018

Time Schedule

- 20' Backus-Naur-Form mit Übung
- 10' Datenstrukturen
- 5' Funktionsüberladung
- 10' Operatorüberladung
- 5' Konstante Referenzen

Learning Objectives

- Verständnis der EBNF
- Kenntnis von Datenstrukturen und Funktionsüberladungen

Backus-Naur-Form

BNF

Die BNF ist eine formale Metasprache, die benutzt wird, um kontextfreie Grammatiken darzustellen.

Metasprache

Eine Metasprache ist eine "Sprache über Sprache".

Kontextfreie Grammatik

Eine kontextfreie Grammatik besteht aus Regeln die unabhängig vom Kontext angewandt werden können.

EBNF Metasprache Kontextfreie Grammatik

Backus-Naur-Form

Kurz und simpel

Die Backus-Naur-Form ist eine Sprache die mit einfacher Syntax beschreibt, welche Sätze mit den Wörtern einer Sprache gebildet werden dürfen.

Aufbau

- Alphabet = Terminalsymbole
- Satzbau = Produktionsregeln = Nichtterminalsymbol

```
1 ZifferAusserNull = "1" | "2" | "3" | "4" |
      "5" | "6" | "7" | "8" | "9" ;
2 Ziffer = "0" | ZifferAusserNull ;
```

Erweiterte Backus-Naur-Form: Beispiel

```
1 ZifAussNull = "1" | "2" | "3" | "4" | "5"
    | "6" | "7" | "8" | "9" ;
2 Zif = "0" | ZifAussNull ;
3
4 Zwoelf = "1", "2" ;
5 Dreihundertzwoelf = "3", Zwoelf ;
6
7 NatZahl = ZifAussNull, { Zif } ;
8 GanzeZahl = "0" | [ "-" ], NatZahl ;
```

BNF: Aufgabe 10_1

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

Welcher Satz ist korrekt?

A	☐	aaA	☐
a	☐	A_A	☐
_	☐	Aa_Aa	☐
Aaaa	☐		

Weitere Fragen

Wie viele terminale und nichtterminale Symbole sind in dieser Form enthalten?

BNF: Lösung 10_1

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

Welcher Satz ist korrekt?

A	☑	aaA	☒
a	☑	A_A	☑
_	☒	Aa_Aa	☑
Aaaa	☑		

Weitere Fragen

Es sind 3 terminale Symbole ("a", "A", "_") und drei nichtterminale Symbole ("seq", "term", "lowerterm") enthalten

Vorteile der EBNF

```
1  seq = term | term "_" seq
2  term = "A" | "A" lowerterm | lowerterm
3  lowerterm = "a" | "a" lowerterm
4
5
6  seq = term | term "_" seq
7  term = "A" { "a" } | "a" { "a" }
8
9  seq = term [ "_" seq ]
10 term = "A" { "a" } | "a" { "a" }
```

struct

```
1  struct rational{
2    int n;
3    int d;
4  };
5
6  int main (){
7    rational r;
8    r.n = 1;
9    r.d = 2;
10
11   return 0;
12 }
```

struct - Direkte Instantiierung

```
1  struct rational{
2    int n;
3    int d;
4  }r,s;
5
6  int main (){
7    r.n = 1;
8    r.d = 2;
9
10   return 0;
11 }
```

struct - Als Funktionsargument

```
1 //POST: deliver solution for quadratic
     equation and return number of solutions
2 int quad_solve(double a, double b, double
     c, double & x1, double & x2);
```

struct - Als Funktionsargument

```
1 struct solution{
2   double x1;
3   double x2;
4 };
5
6 //POST: return solution as struct
7 solution quad_solve(double a, double b,
    double c);
```

Funktionsüberladung

```
1  void print_variable(int a){
2    cout<<"This is an int."<<endl;
3  }
4
5  void print_variable(double a){
6    cout<<"This is a double."<<endl;
7  }
8
9  int print_variable(int a, int b){
10   cout<<"Two ints."<<endl;
11   return 2;
12 }
```

Für Enthusiasten

Operatorüberladung

```
1 rational& operator+= (rational& a, const
     rational b){
2   a.n = a.n * b.d + a.d * b.n;
3   a.d *= b.d;
4   return a;
5 }
```

$$\frac{a_n}{a_d} \leftarrow \frac{a_n}{a_d} + \frac{b_n}{b_d} = \frac{a_n \cdot b_d}{a_d \cdot b_d} + \frac{b_n \cdot a_d}{b_d \cdot a_d}$$

Schönes Tutorial, Beispiele am Ende der Seite

Operatorüberladung

```
1 rational& operator+= (rational& a, const
      rational b){
2   a.n = a.n * b.d + a.d * b.n;
3   a.d *= b.d;
4   return a;
5 }
6
7 rational operator+ (rational a, const
      rational b){
8   return a += b;
9 }
```

Operatorüberladung: ++

```
1  //pre-increment
2  rational& operator++ (rational& r){
3    rational s = {1,1};
4    return r += s;
5  }
6
7  //post-increment
8  rational operator++ (rational& r, int i){
9    rational s = {1,1};
10   rational r_0 = r;
11   r += s;
12   return r_0;
13 }
```

Operatorüberladung: <<

```
1  std::ostream& operator<<
2  (std::ostream& o, rational r){
3    o<< r.n << "/" << r.d;
4    return o;
5  }
6
7  int main(){
8    rational r = {3,2};
9    cout<<r<<r<<endl;
10 }
```

Operatorüberladung auf codeboard

Operatorpräzedenz

Const reference

```
1 int a = 5;
2 int& b = a;
3 const int& c = a;
4
5 c++; // runtime error
6 b++; // a is now 6
7 a++; // a is now 7
```

Const reference

```
1 void print_result (const double& result){
2   cout<<result;
3 }
```

Const reference

```
1 const int& a = 5; //Referenz zu r-value
2
3 void print_result (const double& result){
4   cout<<result;
5 }
6
7 print_result(5); //funktioniert!
```