

# Inhaltsverzeichnis

|                                       |          |
|---------------------------------------|----------|
| <b>Inhaltsverzeichnis .....</b>       | <b>1</b> |
| <b>Beispiel 1 Ausdrucksbaum .....</b> | <b>2</b> |
| <b>Lösungsidee .....</b>              | <b>2</b> |
| Grammatik/Parser .....                | 2        |
| Scanner .....                         | 3        |
| Syntax Tree .....                     | 4        |
| Name List .....                       | 5        |
| StNode .....                          | 6        |
| Aufbau .....                          | 7        |
| <b>Testfälle .....</b>                | <b>8</b> |
| void test_1(); .....                  | 8        |
| void test_2(); .....                  | 10       |
| void test_3(); .....                  | 11       |
| void test_aufbau(); .....             | 12       |
| void test_4(); .....                  | 13       |

# Beispiel 1 Ausdrucksbaum

## Lösungsidee

Es soll ein Programm implementiert werden welches mit Hilfe des pro-facilities/scanner und einer vorgegeben Grammatik eingaben überprüfen und ausführen soll.

### Grammatik/Parser

```
Programm = { Ausgabe | Zuweisung } .  
Ausgabe  = „print“ „(“ Ausdruck „)“ „;“ .  
Zuweisung = „set“ „(“ Identifier „,“ Ausdruck „)“ „;“ .  
  
Ausdruck = Term { AddOp Term } .  
Term     = Faktor { MultOp Faktor } .  
Faktor   = [ AddOp ] UFaktor .
```

```
UFaktor  = Monom | KAusdruck .  
Monom    = WMonom [ Exponent ] .  
KAusdruck = „(“ Ausdruck „)“ .  
WMonom   = Identifier | Real .  
Exponent = „^“ [ AddOp ] Real .  
AddOp    = „+“ | „-“ .  
MultOp   = „*“ | „/“ .
```

#### 1 Grammatik

Die atomarsten Elemente der Grammatik sind Identifier und Real, so wie AddOp und MultOp. Exponent, welcher ebenfalls ein Operator ist, ist bereits etwas komplexer, da er das Exponenten Zeichen, eventuell einen AddOp beinhaltet und einen Real beinhaltet. Monom ruft ein WMonom auf mit einem Exponenten. Ein WMonom ist entweder ein Identifier oder ein Real. Ein UFaktor kann ein Monom oder ein KAusdruck sein. Ein Faktor unterdessen besteht aus einem UFaktor und kann mit einem AddOp begonnen werden, dadurch kann ein negativer oder ein positiver Faktor entstehen.

Ein Term besteht aus einem oder mehreren Faktoren, wenn es mehrere Faktoren sind werden diese durch MultOps verbunden. Ein Ausdruck besteht aus einem oder mehreren Termen, welche durch AddOp's verbunden werden, wenn mehr als ein Term vorhanden ist.

Die Zuweisung beginnt mit dem Schlüsselwort set gefolgt von einer sich öffnenden Klammer, einem Identifier einem Beistrich, einem Ausdruck, einer sich schließenden Klammer und einem Strichpunkt. Ähnlich ist es bei der Ausgabe, diese beginnt ebenfalls mit einem Schlüsselwort – print – gefolgt von einer sich öffnenden Klammer, einem Ausdruck, einer sich schließenden Klammer und einem Strichpunkt. Schlussendlich gibt es ein Programm, welches eine Ausgabe oder eine Zuweisung ist.

Der Parser im Programm benutzt die eben beschriebene Grammatik. Dieser besteht aus zwei Klassen, einem Interface, welches dem eigentlichen Parser wichtige Funktionen vererbt, und der erbbenden Parser Klasse. Der Parser besitzt einen Konstruktor, einen Destruktor, zwei Parse Funktionen und eine print Funktion, welche dann die Liste, in welcher der Syntax Tree, der beim Parsen erstellt wird, ausgibt.

## Scanner

Der verwendete Scanner verwendet als Datentyp zum Einlesen von Zeichen char. Es wird C++20 für dessen Verwendung vorausgesetzt. Der Scanner hat viele Methoden mit denen er die Daten, die ihm der Parser übergibt, einlesen kann. Siehe Anwender Doku.

Der Scanner erkennt die folgenden Terminalsymbole:

| Symbol             | Symbol                      | Symbol              |
|--------------------|-----------------------------|---------------------|
| = assign           | ^ caret                     | : colon             |
| , comma            | / division                  | " double quote      |
| ( left parenthesis | - minus                     | * multiply          |
| . period           | + plus                      | ) right parenthesis |
| ; semicolon        | ␣ space                     | \t tabulator        |
| _ underscore       | \n <i>end of line</i> (eol) |                     |

Darüber hinaus wird auch *end of file* (eof) als Terminalsymbol interpretiert.

### 2 Vorlesungsunterlagen

Der Scanner erkennt die folgenden Terminalklassen:

| Symbol              | Attributabfrage                 | Datentyp        |
|---------------------|---------------------------------|-----------------|
| <i>identifizier</i> | get_identifizier()              | std::string     |
| <i>integer</i>      | get_integer() oder get_number() | int oder double |
| <i>keyword</i>      |                                 |                 |
| <i>real</i>         | get_real() oder get_number()    | double          |
| <i>string</i>       | get_string()                    | std::string     |

### 3 Vorlesungsunterlagen

Der Scanner verwendet intern die folgenden Grammatiken:

```
Identifier = Alpha { Alpha | Digit } .  
Integer   = Digit { Digit } .  
Real      = ( Integer „ . “ [ Integer ] ) | ( „ . “ Integer ) .  
String    = „ “ { any char except quote, eol, or eof } „ “ .
```

```
Alpha = „a“ | ... | „z“ | „A“ | ... | „Z“ | „-“ .  
Digit = „0“ | „1“ | „2“ | „3“ | „4“ | „5“ | „6“ | „7“ | „8“ | „9“ .
```

```
BlockComment = „/*“ { any char except eof } „*/“ .  
LineComment  = „//“ { any char except eol or eof } „eol“ .
```

#### 4 Vorlesungsunterlagen

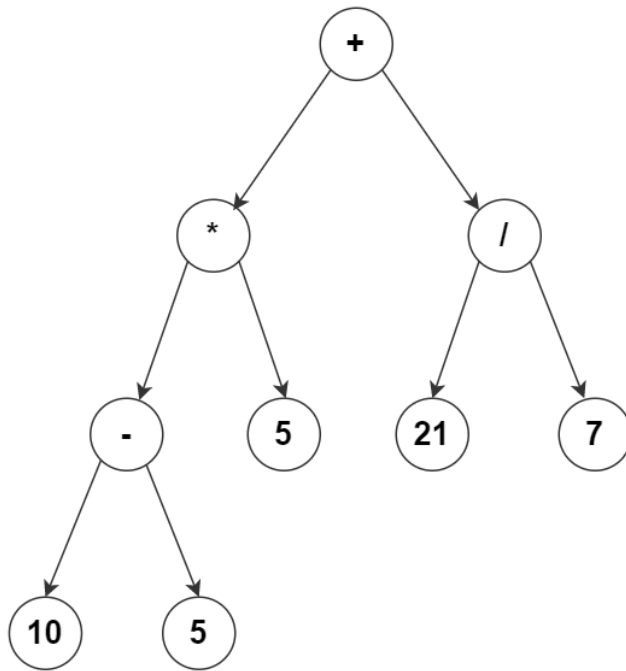
Auch der Scanner verwendet das Digit, Zahlen von 0-9, und etwas, dass hier Alpha heißt und das chars beinhaltet. Der Ausdruck Integer besteht aus mindestens einem oder mehreren Digits. Ein Identifier ist mindestens ein Alpha gefolgt von optional einem weiteren Alpha oder einem Digit.

Ein String beginnt mit „, umfasst dann alles an chars was der Compiler auswerten kann und endet mit „. Ein Real besteht aus einem Integer. Integer oder einem. Integer.

## Syntax Tree

Der Syntax Tree stellt eine Verbindung zwischen Parser und Syntax Tree durch einen Name\_List<SyntaxTree<double>\*> dar. Diese Klasse ist für den Aufbau des Syntax Tree zuständig. Sie evaluiert den Tree, der beim Parsen entsteht und stellt Funktionen zur Verfügung, durch welche dieser ausgegeben werden kann. Außerdem kümmert sich diese Klasse um den Operator <<, welchen der Parser braucht.

In dieser Klasse werden die Funktionen print\_2d und print\_2d\_upright aus dem letzten Semester wieder verwendet.



5 Beispiel Binärer Ausdrucksbaum

Ein binärer Ausdrucksbaum ist eine bestimmte Art eines Binärbaums, der zur Darstellung von Ausdrücken verwendet wird. Zwei gebräuchliche Arten von Ausdrücken, die ein binärer Ausdrucksbaum darstellen kann, sind algebraische Ausdrücke und boolesche Ausdrücke. Diese Bäume können Ausdrücke darstellen, die sowohl unäre als auch binäre Operatoren enthalten.

## Name List

Name List besteht ebenfalls aus zwei Klassen. Wieder einem Interface, welches Funktionen und Member an eine weitere Klasse vererbt, welche in diese Programm die meiste Arbeit erledigt. NameListMap erbt vom Interface den Destruktor, einen Konstruktor erstellen sich die Klassen alles jeweils selbst, falls einer vorhanden ist. Außerdem erbt sie auch noch die Funktionen look\_up\_var, register\_var und print.

Diese Funktionen fügen die jeweilige Variable und den dazugehörigen Wert in einer Map ein, aus der dann die Variable und der Wert hergenommen werden kann für den Parser. Die verwendeten Variablen können durch die print Funktion ausgegeben werden.



6 [https://www.google.at/search?q=map+c%2B%2B&hl=de&sxsrf=AJOqlzW1aatBa2MJbgqfhomBT2A-pbTq5A:1676217557438&source=lnms&tbn=isch&sa=X&ved=2ahUKEwjm886ArZD9AhWJQvEDHRUXAx4Q\\_AUoAXoECAEQAw&biw=768&bih=730&dpr=1.25#imgrc=1PW18z-u99eVfM](https://www.google.at/search?q=map+c%2B%2B&hl=de&sxsrf=AJOqlzW1aatBa2MJbgqfhomBT2A-pbTq5A:1676217557438&source=lnms&tbn=isch&sa=X&ved=2ahUKEwjm886ArZD9AhWJQvEDHRUXAx4Q_AUoAXoECAEQAw&biw=768&bih=730&dpr=1.25#imgrc=1PW18z-u99eVfM)

Die `std::map` ist ein sortierter assoziativer Container, der Schlüssel-Wert-Paare mit eindeutigen Schlüsseln enthält. Die Sortierung der Schlüssel erfolgt über die Vergleichsfunktion `Compare`. Such-, Entfernungs- und Einfügeoperationen haben eine logarithmische Komplexität. Maps werden normalerweise als Rot-Schwarz-Bäume implementiert.

## StNode

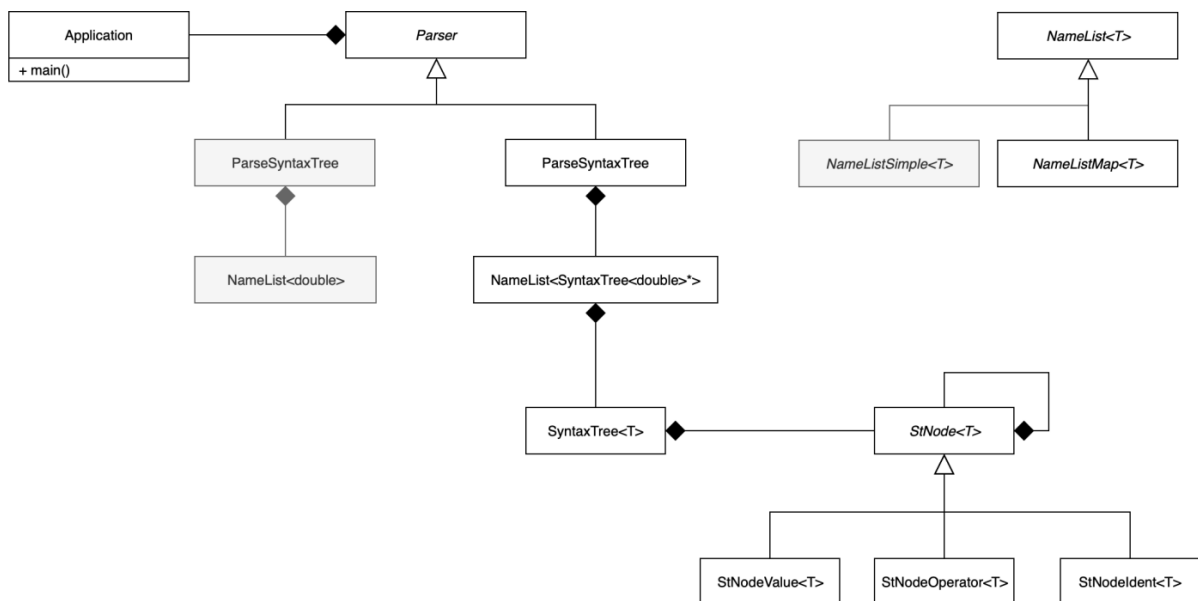
`StNode` besteht aus einem Interface und drei verschiedenen Klassen, welche vom Interface erben. Das Interface beinhaltet einen linken und einen rechten Knoten, so wie Funktionen, um diese zu setzen und diese nach Aufforderung wieder zu erhalten. Außerdem besitzt es einen Konstruktor, einen Destruktor, verschiedene print Funktionen die evaluate Funktion. Diese Funktion sorgt bei den jeweiligen erbenden Klassen, dafür, dass die Liste für den Syntax Tree entsprechend gefüllt wird.

Vor den Klassen gibt es eine Forward Deklaration, damit der Compiler arbeiten kann. In der Computerprogrammierung ist eine Vorwärtsdeklaration eine Deklaration eines Bezeichners für die der Programmierer noch keine vollständige Definition gegeben hat.

Es ist erforderlich, dass ein Compiler bestimmte Eigenschaften eines Bezeichners kennt oder Definition.

Forward-Deklaration wird in Sprachen verwendet, die vor der Verwendung eine Deklaration erfordern; Es ist für die gegenseitige Rekursion in solchen Sprachen erforderlich, da es unmöglich ist, solche Funktionen (oder Datenstrukturen) ohne eine Vorwärtsreferenz in einer Definition zu definieren: Eine der Funktionen (bzw. Datenstrukturen) muss zuerst definiert werden. Es ist auch nützlich, eine flexible Codeorganisation zu ermöglichen, zum Beispiel wenn man den Hauptteil oben und aufgerufene Funktionen darunter platzieren möchte.

## Aufbau



### 7 Programm aufbau als Diagramm

Der interne Aufbau des Programmes funktioniert wie im obigen Bild zu sehen. Die Beschreibungen der einzelnen Bestandteile finden sich unter den jeweiligen Überschriften. Im Hintergrund verborgen gibt es auch noch Exceptiones für Problem, damit der Parser oder andere Bestandteile des Programmes entsprechende Fehlermeldungen ausgeben können. Es gibt keine extra Klasse Application. Es wurde so verstanden, dass der Aufruf im Main mit Application gemeint ist. Der Parser wird also im Main aufgerufen.

Vanessa Wahlmüller

## Testfälle

`void test_1();`

Es wird getestet ob die eingaben aus der Angabe funktionieren, durch die übergabe an den Parser als Datei.

Input Datei Test.txt

Erwartet:

```
1 -7.08971
2 3
3 1
4 30
5 32
6 6
7 16
8 21
9 5
10 10.5
```

+ Ausgaben der Syntax Trees

Arbeitsaufwand: 11 h



## Output:

```
Test 1
-7.08971

3
1
30
32
6
16
21
5
10.5

a:
Print tree in pre-order noation: 1 Print tree in in-order noation: 1 Print tree in post-order noation: 1

2d print:
1

2d upright print:
1
k:
Print tree in pre-order noation: * - 0 + a 3 ^ pi 0.5 Print tree in in-order noation: 0 - a + 3 * pi ^ 0.5 Print tree in post-order noation: 0 a 3 + - pi 0.5 ^ *

2d print:
      0.5
     ^
    pi
   *
    +
   -
  0
```

```
2d upright print:
      *
     / \
    /   \
   /     \
  /       \
 /         \
0         +
         / \
        a  3
      pi  0.5

pi:
Print tree in pre-order noation: 3.1415 Print tree in in-order noation: 3.1415 Print tree in post-order noation: 3.1415

2d print:
3.1415

2d upright print:
3.1415
x:
Print tree in pre-order noation: - ^ y 2 4 Print tree in in-order noation: y ^ 2 - 4 Print tree in post-order noation: y 2 ^ 4 -

2d print:
  4
 -
  ^
  y
```

```
2d upright print:
  -
 / \
^   4
 / \
y  2
y:
Print tree in pre-order noation: 5 Print tree in in-order noation: 5 Print tree in post-order noation: 5

2d print:
5

2d upright print:
5
```

`void test_2();`

Es wird getestet ob die Eingaben von Blödsinn funktionieren, durch die übergrabe an den Parser als Datei. Dateien dürfen für eine ordentliche Ausführung keine Fehler beinhalten.

Input test1.txt

Erwartet:

Error Ausgabe

Output:

```
Test 2
No keyword!
```

Vanessa Wahlmüller

`void test_3();`

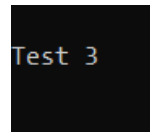
Es wird getestet ob die Eingaben von einer leeren Datei funktionieren.

Input empty.txt

Erwartet:

Keine

Output:

A screenshot of a terminal window with a black background. The text "Test 3" is displayed in a light blue or cyan monospaced font.

Arbeitsaufwand: 11 h

## `void test_aufbau();`

Test aus der Übungsstunde. Aufbau von Syntax Tree ohne Parser.

Input:

```
void test_aufbau(){  
    cout << "Test aufbau (aus uebung uebernommen\n";  
  
    StNodeOp<double>* root = new StNodeOp<double>(StNodeOp<double>::ADD);  
  
    StNodeValue<double>* left = new StNodeValue<double>(17.0);  
    StNodeIdent<double>* right = new StNodeIdent<double>("mehh");  
  
    root->set_left(left);  
    root->set_right(right);  
  
    SyntaxTree<double>* st = new SyntaxTree<double>(root);  
    NameList<SyntaxTree<double>*>* nl = new NameListMap<SyntaxTree<double>*>();  
  
    StNodeValue<double>* x_value = new StNodeValue<double>(4);  
    SyntaxTree<double>* x_tree = new SyntaxTree<double>(x_value);  
  
    nl->register_var("mehh", x_tree);  
  
    cout << "Addition: \n";  
  
    cout << st << "\nResult = " << st->evaluate(nl) << "\n";  
    st->print_2d(cout);  
    cout << "\n\n";  
    st->print_2d_upright(cout);  
}
```

Erwartet:

Ausgabe von Syntax Tree, normalem Print und Rechnung (17+mehh). mehhs ist mit 4 initialisiert daher erwartet 21.

Vanessa Wahlmüller

Output:

```
Test aufbau (aus uebung uebernomenen)
Addition:
Print tree in pre-order noation: + 17 mehh Print tree in in-order noation: 17 + mehh Print tree in post-order noation: 17 mehh +
Result = 21
mehh
+
  17
  +
 /  \
17   mehh
```

**void test\_4();**

Alle Rechen Operationen die der Parser kann. +-\*/^

Input Datei test2.txt

Erwartet:

-3

0

-14

4.5

11

3.25

2

-0.571429

-3.5

14

3.5

1

Error div by zero!

Arbeitsaufwand: 11 h

Vanessa Wahlmüller

Output:

```
Test 4
-3
0
-14
4.5
11
3.25
2
-0.571429
-3.5
14
3.5
1
Error div by zero!
```

Arbeitsaufwand: 11 h