



# **Programmierung II**

**Vorlesungsskript**

Mitschrift von Falk-Jonatan Strube

Vorlesung von Dr. Arnold Beck

24. März 2016

# Inhaltsverzeichnis

<b>1</b>	<b>C++</b>	<b>4</b>
1.1	Ein- und Ausgabe . . . . .	4
1.2	Defaultargumente . . . . .	6
1.3	Überladen . . . . .	6
1.4	Typisierte Konstanten . . . . .	6
1.5	Referenzen . . . . .	6

## Hinweise

Unterlagen unter:

```
1 cd /home/rex/fi1/nestler/Programmierung_II_2016/
```

## Compiler

- Intel i16, i13 (für Linux oder Visual Studio) [www.hocomputer.de](http://www.hocomputer.de) (kostenpflichtig)
- gcc 5.3, 4.85 [gcc.gnu.org](http://gcc.gnu.org)

Zugriff auf Windows-Programme (Visual Studio 2013) in Linux-Laboren:

```
1 rdesktop -f its56 # oder its59
```

Empfohlene Literatur: Breymann[1]

# 1 C++

## 1.1 Ein- und Ausgabe

(siehe Folie CPP\_01\_stdio)

```
1 #include <iostream> // alternativ zu <stdio.h> in C
2 using namespace std; // namespace: für bestimmte Abkürzungen (bspw. cout anstatt std::
  cout)
3 // Hinweis: "::" zeigt, dass das davorstehende "static" ist (hier: std)
4
5 class integer {}; // class: Vergleichbar mit typedef
6
7 int main() {
8     integer i0; // i0: Instanz bzw. Objekt der Klasse integer
9     cin.get(); // Eingabe abwarten
10 }
```

(vgl. integer.cpp)

```
1 #include <iostream>
2 using namespace std;
3
4 class integer { // int – Variable in class verpacken
5     // private: // private ist default
6     int i; // this->i bzw. (*this).i
7     // private nur für andere Klassen, andere Instanzen der gleichen Klasse können drauf
  zugreifen
8 public: // wenn nichts steht, wird automatisch das vorherige angenommen. Hier: private
  integer(int i=0):i(i){ // Konstruktor und Defaultkonstruktor
9     // i=0 default Wert, wenn keiner Angegeben
10    // :i(i) übergebenes i wird dem i der Klasse zugewiesen: i_1(i_2) i_1 ist this->i,
    und i_2 ist das übergebene i
11    cout<<"integer-Objekt_i_="<<this->i<<endl;
12  }
13
14  int get(){ return i; }
15
16  void set(int i=0){ this->i = i; }
17
18  // statische Methode: aufrufbar ohne Instanziierung der Klasse
19  static integer add(integer i1, integer i2){ // Wertkopien von i1 und i2
20      // return integer(i1.i + i2.i); // alternativ und explizit: Konstruktor-Aufruf
21      // return erstellt eine Kopie (mit Konstruktor erstellt)!
22      // return i1.get()+i2.get(); // Umwandlung int nach integer, Aufruf Konstruktor
    implizit
23      return i1.i + i2.i;
24      // i1.i Möglich, da innerhalb der Klasse integer und somit privates i sichtbar
25  }
26 };
27
28
29 auto max(int x, int y) -> int { return x>y ? x : y; } // Lambda-Funktion
30 // auto: Rückgabetyt ergibt sich aus dem Kontext bzw. über das "-> int"
31
32 template<typename Typ1, typename Typ2> // Weiterentwicklung Makro: wählt automatisch Typ
  aus
33 auto quotient(Typ1 a, Typ2 b) -> decltype(a/b) { return a/b; }
```

```

34
35 auto main() -> int {
36     auto k = 0; // C++11: da 0 vom Typ int ist auch k vom Typ int
37     decltype(k) j = 5; // C++11: da k vom Typ int ist auch j vom Typ int
38     char *c = nullptr; //C++ 11: Zeigerliteral
39     int *ip = NULL;
40
41     integer i0(5), i1=4; // 2 (alternative) Initialisierungen von Objekten
42     // i1=4 nur möglich, wenn 1 Parameter gefordert ist.
43     cout<<"i0.i_="<<i0.get()<<endl;
44     // cout im Vergleich zu printf() typsicher.
45     cout<<"i0.i_+i0.i_="<<integer::add(i0, i0).get()<<endl; // Aufruf static-Methode add
46     integer i3 = integer::add(i0, i0); // Initialisierung von i3
47     cout<<"i3.i_="<<i3.get()<<endl;
48     i0.set(22);
49     cout<<"i0.i_="<<i0.get()<<endl;
50     cout<<"max(3,5)_="<<max(3,5)<<endl;
51     cout<<"5/3_="<<quotient(5, 3)<<endl;
52     cout<<"5.0/3.0_="<<quotient(5.0, 3.0)<<endl;
53     cout<<"b/_1_="<<quotient('b', '1')<<endl;
54     cin.get();
55 }

```

(vgl iostream.pdf)

- nach jeder cin Eingabe: „cin.clear();“, damit Fehler ignoriert werden um weiter cin's abhandeln zu können (vgl. robust\_ea)

Einlesen:

```

1 char sc;
2 cout << "sc=";
3 cin >> sc;
4 cin.clear(); // clear, um die Eingabezeile freizumachen, damit man nicht an Falscher
    Eingabe hängen bleibt
5 cin.ignore(INT_MAX, '\n'); // braucht #include <limits.h>
6 cout << "sc" << dec << (int) sc << endl;
7
8 // alternativ:
9 char vb[128];
10 cout << "s=";
11 cin.getline(vb, sizeof(vb), '\n'); // lesen als String, dann wieder umwandeln (liest
    auch Leerzeichen ein)
12 sc = atoi(vb); // braucht #include <cstdlib>
13
14 // alternative zu getline:
15 cin.get(...); // lässt aber \n im Strom
16 cin.get();
17
18 // alternativ
19 cin >> setw(sizeof(vb)) >> vb; // verhindert Überlauf
20 sc = atoi(vb);
21
22 // alternativ
23 String s; // braucht #include <string>
24 size_t ie=0;
25 cin >> s;
26 unsigned int ni = stoi(s, &ie, 10);
27
28 // alternativ
29 getline(cin, s, '\n');
30 double d = stod(s, &ie);
31

```

```
32 // zum compilieren: g++ p2a1.cpp -std=c++11 -o a.out
```

robust\_ea1.cpp:

```
1 do {
2   cout<<"d_="; cin>>d;    // einlesen
3   if (cin.eof()) break;    // break bei Strg+D oä.
4   if (cin.fail() || (cin.peek() != '\n')){ // ist nächstes Zeichen ungültig?
5     cin.clear(); cin.ignore(INT_MAX, '\n'); // Strom zurücksetzen und zum \n gehen
6     continue;
7   }
8   break; % Schleife verlassen, wenn korrekte Eingabe
9 } while(true);
10
11 if (cin.eof()){ cin.clear(); cout<<"eof\n"; }
12 else {
13   cin.clear(); cin.ignore(INT_MAX, '\n');
14   cout<<"Wert_d_="<<d<<endl;
15 }
16 cin.ignore();
```

## 1.2 Defaultargumente

Defaultargumente müssen immer von rechts angefangen definiert sein:

```
1 myFunc(int i = 5, int j = 7) // korrekt
2 myFunc(int i, int j = 7)    // korrekt
3 myFunc(int i = 5, int j)    // falsch !!!
```

## 1.3 Überladen

overload.pdf

Hinweis: cast auf zwei Möglichkeiten:

```
1 int i = 5;
2 double d;
3 d = (long) i;
4 d = long(i);
```

## 1.4 Typisierte Konstanten

## 1.5 Referenzen

referenzen.pdf

Ein Speicherplatz wird mit mehreren Variablen-Namen beschrieben.

# Literatur

- [1] Ulrich Breymann und Ulrich Breymann. „Der C++ Programmierer“. In: *C++ lernen, professionell anwenden, Lösung* (2009).