Programmierung II

Vorlesungsskript

Mitschrift von Falk-Jonatan Strube

Vorlesung von Dr. Arnold Beck

28. April 2016

Inhaltsverzeichnis

Hinweise

Unterlagen unter:

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

Compiler

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

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

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

Empfohlene Literatur: Breymann[1]

Mitschrift von
Falk-Jonatan Strube

1 C++

1.1 Ein- und Ausgabe

(siehe Folie CPP_01_stdio)

```
#include <iostream> // alternativ zu <stdio.h> in C
using namespace std;  // namespace: für bestimmte Abkürzungen (bpsw. cout anstatt std::cout)
// Hinweis: "::" zeigt, dass das davorstehende "static" ist (hier: std)

class integer {}; // class: Vergleichbar mit typedef

int main() {
  integer i0; // i0: Instanz bzw. Objekt der Klasse integer
  cin.get();  // Eingabe abwarten
}
```

(vgl. integer.cpp)

```
#include <iostream>
using namespace std;

class integer { // int - Variable in class verpacken
//  private:    // private ist default
    int i;  // this->i bzw. (*this).i
    // private nur für andere Klassen, andere Instanzen der gleichen Klasse können drauf zugreifen
  public: // wenn nichts steht, wird automatisch das vorherige angenommen. Hier: private
    integer(int i=0):i(i){  // Konstruktor und Defaultkonstruktor
      // i=0 default Wert, wenn keiner Angegeben
      // :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
      cout<<"integer-Objekt i = "<<this->i<<endl;
    }

    int get(){ return i; }

    void set(int i=0){ this->i = i; }

    // statische Methode: aufrufbar ohne Instanziierung der Klasse
    static integer add(integer i1, integer i2){  // Wertkopien von i1 und i2
      // return integer(i1.i + i2.i); // alternativ und explizit: Konstruktor-Aufruf
      // return erstellt eine Kopie (mit Konstruktur erstellt)!
      // return i1.get()+i2.get();    // Umwandlung int nach integer, Aufruf Konstruktor implizit
      return i1.i + i2.i;
      // i1.i Möglich, da innerhalb der Klasse integer und somit privates i sichtbar
    }
};

auto max(int x, int y) -> int { return x>y ? x : y; } // Lambda-Funktion
// auto: Rückgabetyp ergibt sich aus dem Kontext bzw. über das "-> int"

template<typename Typ1, typename Typ2>  // Weiterentwicklung Makro: wählt automatisch Typ aus
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) Initialisiierungen 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
```

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

robust_ea1.cpp:

```
do {
  cout<<"d = "; cin>>d;    // einlesen
  if(cin.eof()) break;     // break bei Strg+D oä.
  if(cin.fail() || (cin.peek() != '\n')){    // ist nächstes Zeichen ungültig?
    cin.clear(); cin.ignore(INT_MAX,'\n');   // Strom zurücksetzen und zum \n gehen
    continue;
  }
  break; % Schleife verlassen, wenn korrekte Eingabe
} while(true);

if(cin.eof()){ cin.clear(); cout<<"eof\n"; }
else {
  cin.clear(); cin.ignore(INT_MAX,'\n');
  cout<<"Wert d = "<<d<<endl;
}
cin.ignore();
```

1.2 Defaultargumente

Defaultargumente müssen immer von rechts angefangen definiert sein:

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

1.3 Überladen

overload.pdf

Hinweis: cast auf zwei Möglichkeiten:

```
int i = 5;
double d;
d = (long) i;
d = long(i);
```

1.4 Typisierte Konstanten

1.5 Referenzen

referenzen.pdf

Ein Speicherplatz wird mit mehreren Variablen-Namen beschrieben.

1.6 String0

1.7 String

1.8 const

const_mutable.pdf
Faustregel: Alle Funktionen, die nichts ändern, immer das const anfügen.

1.9 Folge

Initfolge/Initfolge.pdf

1.10 Kopierkonstrukturen

Kopierkonstruktoren.pdf

1.11 Friend

Zugriff auf private Klassen von außerhalb:

- Funktionen
- Alle Methoden anderer Klassen
- spezifische Methoden anderer Klassen

Muss von Klasse mit privater Funktion festgelegt werden. Freundschaft wird nicht „erwidert“, ist nicht reflexiv (bsp. friendreflex).

1.12 Vererbung

vererben.pdf

Literatur

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