Programmierkurs
Dr. Ole Klein, Dr. Steffen Müthing
IWR, Universität Heidelberg

Aufgabenblatt 11
Abgabe 09. Februar 2018
ipk-exercises:2017-ws-89-g0001213

Aufgabenblatt 11

Musterlösung

Hinweise zur Klausur:

- **Die Klausur wurde um einen Tag verschoben und findet jetzt am 20.02.2018 um 14:00 im grossen Hörsaal der Chemie statt.**

  Melden Sie sich bitte bis zum **12.02.2018** im MÜSLI für die Klausur an. Danach ist der Menüpunkt gesperrt und weder eine An- noch Abmeldung möglich.
- Melden Sie sich bitte vor dem Ende der Anmeldefrist bei uns, wenn bei Ihnen noch Punkte nachgetragen werden müssen.
- Die Klausureinsicht findet am 23.02. von 14:00 bis 15:00 Uhr im Raum 1/414 im Mathematikon statt.
- Sie müssen für die Klausur angemeldet seind und teilgenommen haben, um an der Nachprüfung teilnehmen zu können. Sollten Sie am Klausurtermin krank sein, schicken Sie uns bitte eine kurze Email und reichen baldmöglichst ein ärztliches Attest nach.
- Eventuelle Nachprüfungen werden in der Woche vom 09.04.2018 stattfinden.

Allgemeine Hinweise:

- Dieses Übungsblatt wird **nicht** bewertet.
- Die Aufgaben auf diesem Blatt sollen Ihnen einen Eindruck von Umfang und Schwierigkeit einzelner Klausuraufgaben geben.
- Bei Fragen zu einzelnen Aufgaben können Sie in die Übungen am 12.02. und 16.02. kommen.
- Wir werden eine Musterlösung zu diesem Blatt nach der Übung am 16.02. online stellen.

Aufgabe 1

Geben Sie kurze Antworten auf die folgenden Fragen:

(a) Welche Kontrollstrukturen kennt C++?

Lösung:

`if/else`, `switch`, `for` (normal und vereinfacht), `while`, `do {} while`

(b) Was sind Referenzen, und wofür nutzt man sie (Anwendungsbeispiel)?

Lösung:

Referenzen sind ein "Alias" für eine existierende Variable und verhalten sich so, als ob man direkt auf die unterliegende Variable zugreifen würde.

Sie sind insbesondere wichtig, um Argumente an Funktionen zu übergeben und dabei Kopien zu vermeiden bzw. die Original-Variable in der Funktion zu verändern.

(c) Was sind die zentralen Unterschiede zwischen `std::vector` und `std::map` (konzeptuell und in der Nutzung)?

Lösung:

Ein `std::vector` ist ein *sequence container*, sein Inhalt ist über eine konsekutive Folge von Indizes identifizierbar. Eine `std::map` bildet eine beliebige Menge von Schlüsseln auf Werte ab.

Bei einem `std::vector` muss man eine Grösse vorgeben, Zugriffe mit zu grossen Indizes sind ein Fehler. Er ist aber auch sehr schnell. `std::map` erlaubt Zugriffe mit beliebigen Schlüsseln, der Eintrag wird bei Bedarf angelegt, Zugriff ist jedoch deutlich langsamer und die Einträge können nicht beliebig angeordnet werden.

(d) Was ist der Hauptgrund, Klassen zu definieren, d.h. was ist ihre Aufgabe?

Lösung:

Kapselung von Daten sowie Zusammenführung von Daten und zugehörigem Verhalten (Methoden).

(e) Welche Bedingung sollte beim Nutzen von Vererbung immer eingehalten werden?

Lösung:

Die *is-a*-Regel, die besagt, dass jedes Objekt der abgeleitete Klasse auch ein Repräsentant der Basisklasse sein muss. Hieraus folgt auch, dass Vererbung so gut wie immer `public` sein sollte.

(f) Welchen grossen Vorteil bietet die generische Programmierung (Templates), und welche Nachteile?

Lösung:

Vorteile: Performance und (bei richtigem Design) schwache Kopplung.

Nachteile: Aller Code in Header-Dateien, längere Compile-Zeiten, schwer verständliche Compile-Fehler, schlechter IDE-Support.

Aufgabe 2

Lesen Sie sich folgendes Programm durch:

```
1  #include <iostream>
2
3  // changes the vector v in place by multiplying all its entries by n
4  // does not return anything
5  void multiply(std::vector<int> v, int n)
6  {
7    for (auto& i : v)
8      i = i + n;
```

```
}

int main(int argc, char** argv)
{
  vector<int> v = {{ 1, 2, 3, 4, 5 }};
  v = multiply(v,3);
  // output all entries in the vector together with their index
  for (int i = 1 , i <= v.size() , i++)
    std::cout << i << ": ";
    std::cout << v[i] << std::endl;
  return 0
}
```

Dieses Programm enthält insgesamt 10 Fehler, von denen manche zu Compile-Fehlern führen und andere zu Laufzeit-Fehlern, bei denen das Programm sich nicht so verhält wie erwünscht.

Finden Sie alle 10 Fehler (Sie können die Fehler im Quellcode markieren) und benennen Sie kurz (wenige Worte), wo der Fehler liegt. Schreiben Sie ausserdem dazu, ob der Fehler zur Compile- oder zur Laufzeit auftritt.

Lösung:

Im folgenden steht `ct` für Compilezeit-Fehler und `rt` für Laufzeit-Fehler.

```
#include <iostream>
// <-- missing include <vector> (ct)

// changes the vector v in place by multiplying all its entries by n
// does not return anything

// next line: missing reference, should be
// void multiply(std::vector<int>& v, int n) (rt)
void multiply(std::vector<int> v, int n)
{
  for (auto& i : v)
    i = i + n; // <-- should be *, not + (rt)
}

int main(int argc, char** argv)
{
  // next line: missing "std::" before vector (ct)
  vector<int> v = {{ 1, 2, 3, 4, 5 }};
  // next line: remove "v = ", function does not return anything,
  // but modifies argument in place (ct)
  v = multiply(v,3);
  // output all entries in the vector together with their index
  // next line:
  // - indexing starts at 0 (i = 0) (rt)
  // - test should use "<", not "<=" (rt)
  // - for loop requires semicolon, not comma (ct)
  for (int i = 1 , i <= v.size() , i++)
    // loop body is missing {} (ct, because i does not exist in
    // second line)
    std::cout << i << ": ";
    std::cout << v[i] << std::endl;
```

```
32    return 0 // <-- missing semicolon (ct)
33  }
```

Aufgabe 3

Gegeben sei folgende Funktions-Signatur:

```
1  int longestString(const std::vector<std::string>& vec);
```

Diese Funktion soll den längsten String in dem übergebenen `vector` finden und die Länge dieses Strings ausgeben.

Implementieren Sie die Funktion!

- Um die Länge eines Strings herauszufinden, können Sie die Member-Funktion `size()` der String-Klasse verwenden.
- Schreiben Sie eine vollständige Funktion inklusive der oben angegebenen Funktions-Signatur, nicht nur den Funktions-Body.

Lösung:

```
1  int longestString(const std::vector<std::string>& vec)
2  {
3    int length = 0;
4    for(auto& s : vec)
5      if (s.size() > length)
6        length = s.size();
7    return length;
8  }
```

Aufgabe 4

Schreiben Sie eine Klasse `Student`, die die drei Attribute Vorname, Nachname und Matrikelnummer speichert (die ersten beiden als String, das letzte als Integer). Die Variablen in der Klasse sollen gekapselt und nicht extern zugänglich sein. Beachten Sie folgende Punkte:

- Einen Konstruktor, der Werte für die Attribute als Parameter akzeptiert und in der Klasse speichert.
- Öffentliche Zugriffsmethoden für Lese-Zugriff, die die Namen `firstName()`, `lastName()` und `idNumber()` tragen sollen. Methoden für Schreibzugriff sind nicht erforderlich.
- Beachten Sie die Richtlinien für Klassendesign aus der Vorlesung.
- Denken Sie an eventuell erforderliche includes!

Lösung:

```
1  #include <string>
2
3  class Student
4  {
5
6  public:
7
```

```
8     Student(
9       const std::string& firstName,
10      const std::string& lastName,
11      int idNumber)
12      : _firstName(firstName)
13      , _lastName(lastName)
14      , _idNumber(idNumber)
15    {}
16
17    const std::string& firstName() const
18    {
19      return _firstName;
20    }
21
22    const std::string& lastName() const
23    {
24      return _lastName;
25    }
26
27    int idNumber() const
28    {
29      return _idNumber;
30    }
31
32  private:
33
34    std::string _firstName;
35    std::string _lastName;
36    std::string _idNumber;
37
38  };
```