Programmierkurs
Dr. Steffen Müthing
IWR, Universität Heidelberg

Probeklausur
Abgabe 08. Februar 2019
ipk-exercises:2018-ws-55-gc187ae0

Probeklausur

Musterlösung

Allgemeine Hinweise:

- Dieses Übungsblatt wird **nicht** bewertet.
- Die Aufgaben auf diesem Blatt sollen Ihnen einen Eindruck von Umfang und Schwierigkeit einzelner Klausuraufgaben geben, die Klausur selbst wird aber mehr Aufgaben enthalten.
- Wir werden eine Musterlösung zu diesem Blatt nach der Vorlesung am 08.02. online stellen.

Aufgabe 1

Geben Sie kurze Antworten auf die folgenden Fragen:

(a) Welche Sichtbarkeits-Arten gibt es in einer Klasse, und was bedeuten sie?

Lösung:

`public`: Jeder kann auf diese Klassenmitglieder zugreifen.
`protected`: Nur die Klasse und abgeleitete Klassen können auf diese Klassenmitglieder zugreifen.
`private`: Nur die Klasse selbst kann auf diese Klassenmitglieder zugreifen.

(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) Worin unterscheiden sich `std::array` und `std::vector`?

Lösung:

Ein `std::array` ist ein *sequence container*, dessen Grösse bereits zur Compile-Zeit festgelegt wird und zur Programmlaufzeit nicht verändert werden kann.

Ein `std::vector` ist ein *sequence container*, dessen Speicher dynamisch verwaltet wird; man kann zur Laufzeit Elemente hinzufügen und entfernen.

(d) Was ist der Hauptgrund, Templates zu verwenden, d.h. was ist ihre wichtigste Aufgabe?

Lösung:

Erstellen von Algorithmen und Datenstrukturen, die auf unterschiedlichen Datentypen optimal arbeiten.

(e) Welche Regeln sollten beim Nutzen von virtuellen Methoden immer eingehalten werden?

Lösung:

- Wenn eine Klasse eine virtuelle Method enthält, muss der Destruktor auch `virtual` deklariert sein.
- Wenn man in einer abgeleiteten Klasse eine virtuelle Methode überschreibt, sollte man immer `override` an die Deklaration anhängen, damit der Compiler bei Fehlern in der Funktions-Signatur einen Fehler wirft.

(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;
9  }
10
11 int main(int argc, char** argv)
12 {
13   vector<int> v = {{ 1, 2, 3, 4, 5 }};
14   v = multiply(v,3);
15   // output all entries in the vector together with their index
16   for (int i = 1 , i <= v.size() , i++)
17     std::cout << i << ": ";
18     std::cout << v[i] << std::endl;
19   return 0
20 }
```

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.

```
1  #include <iostream>
2  // <-- missing include <vector> (ct)
3
4  // changes the vector v in place by multiplying all its entries by n
5  // does not return anything
6
7  // next line: missing reference, should be
8  // void multiply(std::vector<int>& v, int n) (rt)
9  void multiply(std::vector<int> v, int n)
10 {
11   for (auto& i : v)
12     i = i + n; // <-- should be *, not + (rt)
13 }
14
15 int main(int argc, char** argv)
16 {
17   // next line: missing "std::" before vector (ct)
18   vector<int> v = {{ 1, 2, 3, 4, 5 }};
19   // next line: remove "v = ", function does not return anything,
20   // but modifies argument in place (ct)
21   v = multiply(v,3);
22   // output all entries in the vector together with their index
23   // next line:
24   // - indexing starts at 0 (i = 0) (rt)
25   // - test should use "<", not "<=" (rt)
26   // - for loop requires semicolon, not comma (ct)
27   for (int i = 1 , i <= v.size() , i++)
28     // loop body is missing {} (ct, because i does not exist in
29     // second line)
30     std::cout << i << ": ";
31     std::cout << v[i] << std::endl;
32   return 0 // <-- missing semicolon (ct)
33 }
```

Aufgabe 3

Gegeben sei folgende Klasse (nicht vollständig!):

```
1  class Building
2  {
3  public:
4    std::string name() const;
5    std::string city() const;
6    int height() const;
7  };
```

Sie sollen folgende Funktion implementieren:

```
std::vector<Building> sortedByHeight(const std::vector<Building>& vec);
```

Diese Funktion soll eine Kopie des Vektors zurückgeben, in der die Gebäude absteigend nach Höhe sortiert sind.

Implementieren Sie die Funktion!

- Zum Sortieren verwenden Sie `std::sort()`.
- Schreiben Sie eine vollständige Funktion inklusive der oben angegebenen Funktions-Signatur, nicht nur den Funktions-Body.

Lösung:

```
std::vector<Building> sortedByHeight(const std::vector<Building>& vec)
{
  auto result = vec;
  std::sort(result.begin(),result.end(),[](auto& a, auto& b) {
    return a.height() > b.height();
  });
  return result;
}
```

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.
- Es soll möglich sein, ein Objekt vom Typ `Student` auf das Terminal auszugeben:

```
Student s("Max","Mustermann",321542);
std::cout << s << std::endl;
```

  Dies soll folgende Ausgabe auf dem Terminal produzieren:

  ```
  Mustermann, Max: 321542
  ```

- Beachten Sie die Richtlinien für Klassendesign aus der Vorlesung.
- Denken Sie an eventuell erforderliche includes!

Lösung:

```
#include <string>
#include <iostream>

class Student
{

public:

  Student(
```

```
    const std::string& firstName,
    const std::string& lastName,
    int idNumber)
    : _firstName(firstName)
    , _lastName(lastName)
    , _idNumber(idNumber)
  {}

  const std::string& firstName() const
  {
    return _firstName;
  }

  const std::string& lastName() const
  {
    return _lastName;
  }

  int idNumber() const
  {
    return _idNumber;
  }

  friend std::ostream& operator<<(std::ostream& os, const Student& student)
  {
    os << student._lastName << ", "
       << student._firstName << ": "
       << student._idNumber;
    return os;
  }

private:

  std::string _firstName;
  std::string _lastName;
  int _idNumber;

};
```