UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing

IPI: Einführung in die Praktische Informatik
Wintersemester 2018
Prof. Björn Ommer
ommer@uni-heidelberg.de

Übungsblatt 11: Klassen, Vererbung und Virtuelle Methoden

Abgabe am 23.01.2019, 13:00.

Hinweise:

- Dies ist der letzte Übungszettel dieser Vorlesung. Die Zulassung zur Prüfung wird auf Basis aller Übungszettel (1. bis 11.) berechnet.
- Die Anmeldung zur Klausur wird über das Moodle System erfolgen und ca. am 24.01.2019 freigeschaltet. Wir werden euch davor nochmal Informationen zur Prüfungsanmeldung und genenerell zur Klausur per Email zukommen lassen. Hierfür werden wir die im Moodle von euch hinterlegte Email Adresse verwenden. Stellt deshalb bitte sicher, dass ihr regelmäßig das Postfach euer Moodle Email Adresse abruft, damit wir euch auch erreichen können.

Aufgabe 1: Vererbung (5P)

Finden Sie die Fehler im folgenden Programmfragment. Begründen Sie kurz, warum der jeweilige Ausdruck falsch ist.

```
3  class A {
4  public:
5      int ap;
6      void X();
7  private:
8      int aq;
9      void aX();
10 };
11
12 class B : public A {
13 public:
14     int bp;
15     void Y();
16 private:
17     int bq;
18     void bY();
19 };
20
21 class C : public B {
22 public:
23     int cp;
24     void Z();
25 private:
26     int cq;
27     void cZ();
28 };

   ...

38 void B::Y()
39 {
40     bq = bp;
41     aq = ap;
42     bY();
43 }
44
45 void C::cZ()
46 {
47     ap = 1;
48     bp = 2;
49     cq = 3;
50     X();
51     Y();
52     aX();
53 }
54
55 int main()
56 {
57     A a;  B b;  C c;
58
59     a.X();
60     b.bY();
61     c.cp = 4;
62     c.bp = 1;
63     c.ap = 2;
64     c.aq = 5;
65
66     b.ap = c.ap;
67
68     return 0;
69 }
```

Aufgabe 2: Virtuelle Methoden **(5P)**

Betrachten Sie folgende Hierarchie von drei Klassen.

```
class A {
public:
    void a();
    virtual void va() = 0;
    virtual void a(int n);
private:
    void c();
};
```

```
class B: public A {
public:
    void b();
    virtual void vb();
    void a(double d);
    void a(int i);
    virtual void va();
};
```

```
class C: public B {
public:
    virtual void c();
    void a(float);
    virtual void a();
    virtual void va();
};
```

Eine Implementierung aller deklarierten Methoden existiert. Was passiert, wenn Sie den folgenden Programmteil compilieren wollen? Welche Ausdrücke sind nicht korrekt, und warum?

```
A a;  B b;   C c;  A* pa=&b;  B* pb=&c; float x = 1.2;
pa->a();  pa->va();  pa->a(1);  pa->c();  pa->b();  pa->vb();  pa->a(x);
pb->a();  pb->va();  pb->a(1);  pb->c();  pb->b();  pb->vb();  pb->a(x);
pa = &c;
pa->a();  pa->va();  pa->a(1);  pa->c();  pa->b();  pa->vb();  pa->a(x);
```

Wenn Sie die falschen Ausdrücke entfernt haben und das Programm ausführen, von welcher Klasse werden dann die Methoden aufgerufen? Geben Sie für jeden Methodenaufruf an, ob die Methodenimplementierung von Klasse `A`, `B` oder `C` verwendet wird.

Aufgabe 3: Die Standard Template Library (STL) **(10P)**

Hinweis: Containerklassen werden in der Vorlesung Donnerstag (17.01.2019) genauer behandelt.

Die C++ Standard Library, die in der Regel mit dem Akronym ihrer Vorgängerin STL bezeichnet wird, stellt Container und Algorithmen zur Verfügung. Fast jedes größere Programm profitiert von der Verwendung der STL. Nicht nur erspart man sich auf diese Weise viel Programmieraufwand, weil die abstrakten Datentypen nicht selbst implementiert werden müssen, man reduziert auch die Wahrscheinlichkeit, subtile Bugs zu produzieren.

(a)
Schreiben Sie eine Vektorklasse, die einen `std::vector` zur Verwaltung der Einträge verwendet. Übergeben Sie den Typ der Elemente und die Länge des Vektors

als Templateparameter. Die Klasse soll Methoden bereit stellen, die folgende Operationen erlauben:

- Addition zweier Vektoren
- Multiplikation mit einem Skalar
- Berechnung von Maximum (größter Eintrag im Vektor)

Das genaue Interface ist Ihnen überlassen. Übergeben Sie den Methoden ihre Argumente templatisiert, z. B. muss der Skalar bei der Skalarmultiplikation nicht den Typ der Vektoreinträge haben. Benutzen Sie jeweils in mindestens einer Methode

- den Elementzugriff über `operator[]`, durch den sich der `std::vector` wie ein C-Array verwenden lässt,
- den Zugriff über die Containeriteratoren, der auch mit anderen Containern (z. B. `std::list`) funktioniert und
- die vordefinierten STL-Algorithmen, die ebenfalls mit anderen Containern funktionieren.

Die in der STL definierten Container und die Komplexität ihrer Methoden können Sie z. B. unter `http://www.cplusplus.com/reference/stl/` finden, während die Algorithmen unter `http://www.cplusplus.com/reference/algorithm/` zu finden sind.

(b)
Schreiben Sie ein Programm, das jede der in Aufgabenteil (a) implementierten Funktionalitäten (Vektoraddition, Skalarprodukt, Maximum) ihrer Klasse testet.