Universität Karlsruhe (TH)
Lehrstuhl für Programmierparadigmen

Fortgeschr. Objektorientierung SS 2009 http://pp.info.uni-karlsruhe.de/
Dozent: Prof. Dr.-Ing. G. Snelting snelting@ipd.info.uni-karlsruhe.de
Übungsleiter: Andreas Lochbihler lochbihl@ipd.info.uni-karlsruhe.de
Dennis Giffhorn giffhorn@ipd.info.uni-karlsruhe.de

Blatt 4 Ausgabe: 14.05.2009 Besprechung: 20.05.2009

1. **vtables / Type Casts**

   Betrachten Sie folgende Klassenhierarchie:

```
1   class A {
2     virtual void f() { ... }
3     virtual void g() { ... }
4   };
5   class B : public A {
6     virtual void f() { ... }
7   };
8   class C {
9     virtual void g() { ... }
10    virtual void h() { ... }
11  };
12  class D : public B, public C {
13    virtual void f() { ... }
14  };
15  class E : public virtual D, public virtual A {
16    virtual void f() { ... }
17  };
```

   (a) Erstellen Sie die vtables mit Methodennamen und Subobjekt-Deltas

   (b) Wie verändern sich die v-tables, wenn E D nicht mehr virtuell, sondern nichtvirtuell erbt?

   (c) Schreiben Sie den Code für die folgenden Funktionsaufrufe auf (Notation wie im Skript, Kap. 5):

```
1   D *d=new D();
2   d->f();
3   d->g();
4   d->h();
```

(d) Für welchen der folgenden Type-Casts muss Code erzeugt werden? Schreiben Sie ihn in ähnlicher Notation wie zuvor auf.

```
1          D* d=new D();
2          E* e=new E();
3          A* pa; B* pb; C* pc; D* pd; E* pe;
4          pa=d; pd=(D*)pa;
5          pb=d; pd=(D*)pb;
6          pc=d; pd=(D*)pc;
7          pa=e; pe=(E*)pa;
8          pb=e; pe=(E*)pb;
9          pc=e; pe=(E*)pc;
```

(e) Das Subobjekt-Layout von E lässt die Vermutung zu, man könnte auf die virtuellen D- und A-Subobjekte nicht nur über den in E enthaltenen Pointer auf diese Subobjekte zugreifen, sondern auch über ein entsprechendes δ. Warum geht das nicht? Wie sieht es aus, wenn folgende Unterklassen später definiert werden?

```
1          class F {}
2          class G : public F, public E {}
```

2. **vtables / statische Information und final overrider**

Betrachten Sie folgendes Programm:

```
1  class A {
2  public: virtual void foo() {}
3  };
4  class B: public A {
5  public: virtual void foo() {}
6  };
7  class C: public A {
8  };
9  class D: public C, public B  {
10 };
11
12 int main() {
13   C* c = new D();
14   c->foo();
15 }
```

(a) Erstellen Sie die vtables für die Subobjekte in D.

(b) Welche $foo()$-Methode ruft `main()` auf?

(c) Betrachten Sie nun folgendes, leicht modifiziertes Programm. Wie ändert sich das Verhalten?

```
1  class A {
2  public: virtual void foo() {}
3  };
4  class B: public virtual A {
5  public: virtual void foo() {}
6  };
7  class C: public virtual A {
8  };
9  class D: public C, public B  {
10 };
11
12 int main() {
13   C* c = new D();
14   c->foo();
15 }
```

(d) Fügen Sie nun in das Programm in (c) eine Methodendefinition von foo in C ein. Was meldet der Compiler? Ändert sich das Resultat, wenn man die `main`-Methode auskommentiert? Erläutern Sie dies anhand der vtables.