Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Fortgeschr. Objektorientierung SS 2008		http://pp.info.uni-karlsruhe.de/
Dozent:	Prof. Dr.-Ing. G. Snelting	snelting@ipd.info.uni-karlsruhe.de
Übungsleiter:	Daniel Wasserrab	wasserra@ipd.info.uni-karlsruhe.de
	Andreas Lochbihler	lochbihl@ipd.info.uni-karlsruhe.de

Übungsblatt 2 | Ausgabe: 9.5.2008 | Besprechung: 14.5.2008

1. **Casting in C++**

   Casting ist in C++ (im Gegensatz zu Java) eine nichttriviale Operation, da es das aktuelle Subobjekt verändert. Es gibt dafür zwei Operationen, welche für Up-Casts typsicher sind, `static_cast` und `dynamic_cast` (Bemerkung: C-Style Casts, also z.B. $(C)b$ sind nicht typsicher, sollten also mit äußerster Vorsicht verwendet werden). Die beiden Casts unterscheiden sich in den statischen Checks wie auch im Laufzeitverhalten:

   statische Checks:

   `static_cast` $<C>$ (o)**:** o statischer Typ `Class` D

   (i) $D \leq^* C$ und eindeutiges C-Subobjekt in D oder

   (ii) $C \leq^* D$ und mindestens ein Pfad von C nach D ohne virtuelle Vererbung

   `dynamic_cast` $<C>$ (o)**:** o statischer Typ `Class` D

   (i) $D \leq^* C$ und eindeutiges C-Subobjekt in D oder

   (ii) kein C-Subobjekt in D

   Laufzeitverhalten:

   `static_cast` $<C>$ $[X, Xs.D]$**:**

   in Fall (i): das eindeutige C-Subobjekt in D

   in Fall (ii): falls $Xs = Xs'.C.Xs''$ und $C \notin Xs'' \Longrightarrow$ Resultat $[X, Xs'.C]$
   ansonsten: Fehlerhafter Cast (sollte besser Exception werfen)

   `dynamic_cast` $<C>$ $[X, Xs.D]$**:**

   - wenn möglich in Fall (i) und (ii) Verhalten wie `static_cast`, ansonsten
   - in Fall (ii) eindeutiges C-Subobjekt in X,
   - ansonsten: Fehlerhafter Cast (sollte besser Exception werfen)

   Betrachten sie folgendes Codefragment:

```
1  class D { virtual int f() {} };
2  class C : public D { };
3  class B : public D { };
4  class A : public B, public C { };
5
6  class H { virtual int f() {} };
7  class G : public virtual H { virtual int f() {} };
8  class F : public virtual H { };
9  class E : public F, public G { };
10
11 int main() {
12   A* a = new A(); B* b; C* c; D* d;
13
14   // up-cast
15   d = static_cast<D*>(a);
16   d = dynamic_cast<D*>(a);
17   d = dynamic_cast<D*>(static_cast <C*>(a));
18   d = static_cast<D*>(dynamic_cast <C*>(a));
19
20   // down-cast
21   c = static_cast<C*>(d);
22   b = static_cast<B*>(d);
23   c = dynamic_cast<C*>(d);
24   b = dynamic_cast<B*>(d);
25
26   // cross-cast
27   b = a;
28   c = static_cast<C*>(b);
29   c = dynamic_cast<C*>(b);
30
31   E* e = new E(); F* f; H* h;
32
33   // up-cast
34   h = static_cast<H*>(e);
35   h = dynamic_cast<H*>(e);
36   h = dynamic_cast<H*>(static_cast <G*>(e));
37   h = static_cast<H*>(dynamic_cast <G*>(e));
38
39   // down-cast
40   f = static_cast<F*>(h);
41   f = dynamic_cast<F*>(h);
42
43   // non-related cast
44   a = static_cast<A*>(e);
45   a = dynamic_cast<A*>(e);
46 }
```

Welche Casts werden statisch zurückgewiesen? Bei welchen schlägt der Cast zur Laufzeit fehl?

2. **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  };
```

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

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

(c) Schreiben Sie den Code für die folgenden Funktionsaufrufe auf (Notation wie im Skript, Seite 51ff):

```
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? Was passiert wenn ein Objekt einer Klasse

```
1   class F : public virtual A,
2             public virtual D,
3             public virtual E { };
```

erst nach E und dann nach D gecastet wird?