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 1 Ausgabe: 5.5.2008 Besprechung: 7.5.2008

1. **Subobjekte**

   Betrachten Sie folgende Klassen-Hierarchie:

```
1  class A {
2  public: int i;
3  };
4  class B : public A {
5  };
6  class C : public A {
7  };
8  class D1 : public A, public B, public C {
9  };
10 class D2 : public C, public B, public A {
11 };
```

   (a) Zeichnen Sie den Subobjektgraphen dieser Klassenhierarchie

   (b) Wieviele A Subobjekte sind in einem $D1$ bzw. $D2$ Objekt enthalten?

   (c) Durch Typcasts können Sie Mehrdeutigkeiten beim Zugriff auf Members beseitigen. Schreiben Sie die Casts auf, die notwendig sind, um auf das Member i aller A Subobjekte eines $D1$ bzw. $D2$ Objekts zuzugreifen.

2. **Statisches Casten und dynamischer Lookup**

   Betrachten Sie das folgende C++-Fragment:

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

   (a) Zeichnen Sie den Subobjektgraphen und das Objektlayout.

   (b) Wenn man ein neues Objekt vom Typ E kreiert und darauf $f()$ aufruft, was passiert?

   (c) Was passiert, wenn man nach der Objektkreierung noch einen statischen Cast nach B durchführt, bevor man $f()$ aufruft? Können Sie sich dieses Verhalten erklären?

3. **Subobjektgraphen und Static Lookup**

   Betrachten Sie das folgende C++-Fragment:

```
1  class S {
2  public: int m;
3  };
4  class A: public virtual S {
5  public: int m;
6  };
7  class B: public virtual S {
8  public: int m;
9  };
10 class C: public virtual A, public virtual B {
11 public: int m;
12 };
13 class D: public C {};
14 class E: public virtual A, public virtual B, public D {
15 public: void setm() { m=42; }
16 };
```

   (a) Geben Sie den Subobjektgraphen und das Objekt-Layout für die Klassenhierarchie an.

   (b) Berechnen Sie $lookup(E, m)$.

   (c) Lassen Sie dieses Beispiel durch den GNU C++ Compiler laufen. Wie erklären Sie sich das Ergebnis? Wenn Sie Zugriff auf andere C++ Compiler haben, probieren Sie diese auch aus.

4. **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?