Praxisorientierte Einführung in C++
Lektion: "C++ Cast Operatoren"

Christof Elbrechter

Neuroinformatics Group, CITEC

April 24, 2014

Table of Contents

Typumwandlung

- Typumwandlung:
  - Objekte von Unterschiedlichen Typen werden einander zugewiesen
  - Instanzierung eines Objektes vom Typ A *von* einem Objekt des Typs B (Copy-Konstruktor)
  - Auch implizite Umwandlung bei Parameterübergabe
- Machmal ist muss die Typ-Umwandlung explizit erfolgen
- Hierfür stehen in C++ verschiedene *Cast-Operatoren* zur Verfügung

Übersicht Cast-Operatoren

C-Style-Cast

```
(ZielTyp)Ausdruck
```

Statischer Cast

```
static_cast<ZielTyp>(Ausdruck)
```

Hinzufügen/Enfernen der `const`-ness

```
const_cast<ZielTyp>(Ausdruck)
```

Uminterpretieren von Binärmustern

```
reinterpret_cast<ZielTyp>(Ausdruck)
```

Zu Laufzeit dynamischer Cast

```
dynamic_cast<ZielTyp>(Ausdruck)
```

C-Style-Cast

Beispiel

```
int i = (int)4.3432;      // Objekte werden umgewandelt
int ai[10] = {1,2,3,4,5,6,7,8,9,0};
float *pf = (float*)ai;   // reinterpretiert daten
```

- Pointer und Referenzen werden *reinterpretiert*
- Objekte werden *konvertiert* (evtl. gesliced)
- Kaum Plausibilitäts-Checks
- Kann leicht zu Fehlern führen
- Teilweise ist schwer zu erkennen warum eigentlich *gecastet* wird
- Sollte sparsam verwendet werden

static_cast

- `static_cast` wird selten explizit verwendet; ist aber für alle impliziten Casts zuständig
- Abhängig von Typinformationen die **zur Übersetzungszeit** feststehen

Anwendungen für static_cast

- Unterklassen-Pointer -> Basisklassen-Pointer
- Unterklassen-Referenzen -> Basisklassen-Referenzen
- Arithmetische Konvertierungen (z.B. `float`→`int` oder `int`→`enum`)
- Implizite Umwandlungen aller Art sind auch statische Casts

const_cast

- Ermöglicht das Enfernen/Hinzufügen von `const` und `volatile` Qualifiern

Beispiel

```
void add(const Point &p){
   Point &k = const_cast<Point&>(p);
   k.setX(7);
}
```

- Ziel-Typ muss Pointer- oder Referenz-Typ sein
- **Sehr gefährlich!!!**
  - Falls Quell-Instanz tatsächlich `const` war, kann es sein, *const-ness* zur Übersetzungszeit vorausgesetzt wird
  - Ergebnis ist dann undefiniert

reinterpret_cast

- Umwandeln von Typen ohne Bezug
- Reinterpretation des Binärmusters
- Kann auch z.B. einen Pointer in einen int konvertieren
- Mal sehr gefährlich, mal sehr nützlich

Beispiele

```
void add(int a, int b){ return a+b; }
int anything[3] = {
  42,
  reinterpret_cast<int>(add),
  reinterpret_cast<int>(anything)
};
void write_to_file(void *data, int len, const std::string &filename){
   std::ofstream str(filename.c_str());
   str.write(reinterpret_cast<char*>(data),len);
}
void write_point_to_file(const Point &p, const std::string &filename){
   write_to_file(&p,sizeof(Point),filename);
}
```

dynamic_cast

- Konvertierung innerhalb einer Vererbungshierarchie (nun auch abwärts!)
- Zur Laufzeit dynamisch
- ZielTyp muss Pointer oder Referenz sein
- Falls ZielTyp ein Pointer ist, so gibt `dynamic_cast` einen NULL-Pointer zurück, falls die Konvertierung nicht erfolgreich war
- Bei Referenzen wird eine `bad_cast`-exception geworfen

Beispiele für dynamic_cast

Beispiel

```
struct Point {
   int x,y;
   ...
};
struct TextLabel : public Point {
   std::string text;
   ...
};
struct Intlabel : public TextLabel {
   ...
};
void set_label_to_4(Point *p){
   TextLabel *l = dynamic_cast<TextLabel*>(p);
   if(l){
      l->setLabel("4");
   }else{
      IntLabel *i = dynamic_cast<IntLabel*>(p);
      if(i) i->setLabel(4);
  }
}
```

Einschränkungen von dynamic_cast

Wichtig!

- `dynamic_cast` funktioniert nur für Typen die Polymorph sind

Wann ist ein Typ polymorph?

- Typen sind Polymorph, wenn sie **mindestens eine virtuelle Funktion** aufweisen
- Falls keine solche Funktion existiert, kann ein (notfalls leerer) virtueller Destruktor eingebaut wird

- Dynamische casts sind allerdings *etwas langsam*
- Intern muss bei jedem `dynamic_cast` die **vtable**-analysiert werden