Praxisorientierte Einführung in C++
Lektion: "Das Schlüsselwort explicit"

Christof Elbrechter

Neuroinformatics Group, CITEC

April 24, 2014

Table of Contents

Übersicht

- Das Schlüsselwort `explicit` ist relativ speziell und wird selten verwendet
- Dennoch gilt: Wer das Schlüsselwort und seine Folgen komplett versteht, der kann kein Laie mehr sein :-)

Ausgangsbeispiel: Simple n-D-Vector-Klasse

```
struct Vector{
   int dim;
   float *data;
   Vector(int dim):dim(dim),data(new float[dim]){}
   ~Vector() { delete [] data; }
   Vector &operator=(const Vector &other){
      delete[] data;
      data = new float[other.dim];
      dim = other.dim;
      for(int i=0;i<dim;++i) data[i] = other.data[i];
      return *this;
   }
   Vector &operator=(float value){
      for(int i=0;i<dim;++i) data[i] = value;
      return *this;
   }
};
```

Problembeschreibung

- **Achtung**: Es ist nicht intuitiv ersichtlich, wann ein Konstruktor-Aufruf stattfindet und wann der Zuweisungs-Operator verwendet wird!
- In Variablen-Deklarationsausdrücken der Form
  `TypName VarName = InitialisierungsAusdruck`
  findet immer ein Konstruktor-Aufruf statt.

Beispiel

```
Vector v = 5; // Equivalent zu Vector v(5); -> Konstruktor
v = 5;        // operator '=' -> Zuweisung
```

- Im oberen Fall (`Vector v=5;`): dim wird 5 und nicht mit Fünfen initialisiert

explicit Konstruktoren

- Um (u.A.) Missverständliche Initialisierungs-Ausdrücke zu verbieten, kann das Schlüsselwort `explicit` verwendet werden
- `explicit` deklarierte Konstruktoren müssen immer in der Klammerschreibweise aufgerufen werden

Beispiel

```
struct Vector{
  ...
  explicit Vector(int dim){...}
};

Vector v = 5; // Fehler: nicht laenger erlaubt
Vector v(5);  // expliziter Konstruktor-Aufruf ok!
Vector v; v = 5; // Leerer Konstruktor geht auch
```

Implizite Konstruktor-Aufrufe

- Das Hauptproblem bei nicht-`explicit` Konstruktoren ist allerdings ein anderes
- Nicht-`explicit`-Konstruktoren dürfen vom Compiler **implizit** aufgerufen werden

Beispiel

```
// Annahme: Vector-Konstruktor ist NICHT explicit
Vector add(const Vector &a, const Vector &b){
   if(a.dim != b.dim){
      throw std::runtime_error("vector dimension missmatch");
   }
   Vector r(a.dim());
   for(int i=0;i<r.dim;++i) r.data[i] = a.data[i]+b.data[i];
   return r;
}
int main(){
  Vector k(4),l(4);
  Vector X = add(k,1); // Kompiliert und wirft eine Exception ??
}
```

Erklärungen ...

- Statt `add(k,1)` hat sich ein Fehler eingeschlichen: `add(k,1)`
- Bei Übergabe von 1 wird implizit der Vector-Konstruktor `Vector(int dim)` aufgerufen
- Dimensionen sind unterschiedlich
- Fehler kann sehr schwer zu finden sein

Implizite Konstruktor Aufrufe

An Stellen wo ein Argument des Typs `X` oder des Typs `const X&` erwartet wird, kann auch ein Typ `Y` übergeben werden falls

- Ein Konstruktor `X(const Y&)` oder `X(Y)` existiert
- Der Typ `Y` einen impliziten Cast nach `X` anbietet

Noch ein Beispiel:

```
#include <cstring>
#include <iostream>
struct String{
  char *m_text; int len;
  String():m_text(0){}
  String(const char *text){
    len = strlen(text);
    m_text = new char[len+1];
    strcpy(m_text,text);
  }
  void append(const String &s){  // sorry hier!
    char *c = new char[s.len+len+1];
    strcpy(c,m_text);
    strcpy(c+len,s.m_text);
    delete [] m_text;
    m_text = c;
    len += s.len;
  }
  // Einiges fehlt noch ...
};
int main(){
  String a = "Hello";
  a.append(" World!"); // implizite Umwandlung von 'const char*'
  // a.append(String(" World")); falls Konstruktor explicit
  std::cout << a.m_text << std::endl;
}
```

Gestaffelte implizite Konstruktoraufrufe

- Implizite Konstruktoraufrufe nicht staffellungsfähig
- (und das ist gut so!)

Beispiel

```
#include <iostream>
struct String{
  const char *text;
  String(const char *text):text(text){}
};
struct Person{
  String name;
  Person(const String &name):name(name){}
};
void show(const Person &p){
  std::cout << p.name.text << std::endl;
}
int main(){
  // show("Florian P. Schmidt");  geht nicht!
  show(Person("Florian P. Schmidt"));
}
```

Temporaries

- Temporäre Ausdrücke *matchen* niemals auf nicht-`const`-Referenzen
- Implizite Konstruktoraufrufe erzeugen in jedem Fall ein *Temporary*

Beispiel

```
#include <iostream>
struct String{
  const char *text;
  String(const char *text):text(text){}
};
void show_a(String text){...}
void show_b(String &text){...}
void show_c(const String &text){...}

int main(){
  show_a("Florian P. Schmidt");
  // show_b("Florian P. Schmidt"); geht nicht
  show_c("Florian P. Schmidt");
  String s("Florian P. Schmidt");
  show_b(s); // geht !
  //show_b(String("Florian P. Schmidt")); // geht wieder nicht
}
```

Also ...

- Auch von Funktionen zurückgegebene Objekte (also nicht-Referenzen) sind Temporaries
- Konstruktoraufrufe erzeugen auch zunächst ein Temporary
- Temporaries werden zu `const`-Referenezen
- Ansonsten könnte man sie sich versehentlich als Referenz merken
- Eine `const`-Referenz kann i.d.R. nur ausgelesen/verwendet und tief kopiert (also zugewiesen) werden