Praxisorientierte Einführung in C++
Lektion: "Aufzählungen"

Christof Elbrechter

Neuroinformatics Group, CITEC

May 28, 2014

Table of Contents

Motivation

- Oft benötigt: Bezeichner für paarweise unterschiedliche konstante Werte mit paarweise unterschiedlichen Bezeichnern
- Möglichkeiten:
  - Makros
  - Konstanten
- Aufwändig! Fehlerträchtig!

```
const int RotX = 0, RotY = 1, RotZ = 2, None = 3;
```

- Wenn man Konstanten entfernt/hinzufügt muss man selber auf Konsistenz achten

Aufzählungen

- Lösung: Aufzählungstypen: `enum` (Enumeration)
- Beispiel

```
enum { RotX, RotY, RotZ };
```

- `RotX, RotY, RotZ` sind jetzt Bezeichner für drei unterschiedliche Ganzzahlwerte
- `RotX` ist gleich 0
- `RotY` ist gleich 1
- `RotZ` ist gleich 2

Alternativer Anfang

- Wenn man nicht mit 0 anfangen will:

```
enum { RotX = 1, RotY, RotZ };
```

- Man kann auch zwischendurch "hüpfen":

```
enum { RotX = 1, RotY = 4, RotZ };
```

- Aber Vorsicht! Damit kann man sich auch selbst in den Fuss schiessen

```
enum { RotX = 1, RotY = 0, RotZ };
```

Aufzählung mit Typbezeichner

- Typdefinition mit Bezeichner

```
enum RotationAxis { RotX, RotY, RotZ };
```

- `RotationAxis` kann jetzt wie jeder andere Typ benutzt werden

```
RotationType foo() { return RotX; }

int main() {
  RotationType rot = foo();
}
```

Typdefinition, Deklaration und Initialisierung

- Typdefinition und Variablendeklaration in einem

```
enum RotationAxis {RotX, RotY, RotZ} r;
```

- Plus Initialisierung

```
enum RotationAxis {RotX, RotY, RotZ} r = RotX;
```

Klassen/Strukturen und Aufzählungen

- Können natürlich auch in Klassen/Strukturen verwendet werden

```
struct Joint {
  enum RotationType { RotX, RotY, RotZ };
  RotationType m_type;

  Joint(RotationType type) : m_type(type) { }
};

int main() {
  Joint j(Joint::RotX);
}
```

Konvertierung nach int

- enum-Werte können implizit nach int konvertiert werden
- Aber nicht anders herum
- Was Sinn macht, wegen der evtl. Nichteindeutigkeit

```
int main() {
  enum { A, B, C } x;

  x = 5;
}
```

Switch/Case

- Über Enums kann mann 'switchen'
- Bei nicht 'ge-case-ten' Werten warnt der Compiler

```
Matrix create_rotation_matrix(RotationType t, float value){
    switch(t){
       case RotX:  ... break;
       case RotY:  ... break;
       // compiler Warnung: RotZ wurde nicht beruecksichtig!
    }
```

Tipp

- Manchmal will man auch nach und nach Werte hinzufügen!

```
enum Features{
     Color,
     Speed,
     Length,
     Power,
     NUM_FEATURES   /// hat automatisch den richtigen Wert
}

bool features_activated[NUM_FEATURES];

int main(){
    if(features_activated[Speed]){
         ...
    }
}
```

enum class in C++-11

- Manchmal möchte man nicht, dass einem alle enum Werte den entsprechenden Namensraum *polluten*
- Ausweg: sog. *Strongly types Enumerations*

```
// gloaler namensraum
enum Axis { X, Y, Z };
// von nun an ist X (genau genommen ::X) belegt
Axis a = X;
```

- Ausweg in C++-11 *enum class*

```
enum class Axis { X,Y,Z };
// X ist nur durch Voranstellen des types gueltig
Axis a = Axis::X;
```

- Entsprechener Namensraum bleibt *sauber*