C++: Templates

Carsten Gips (HSBI)

Vergleichsfunktion für zwei Integer?

```
bool cmp(const int &a, const int &b) {
    return a<b;
}
```

Vergleichsfunktion für zwei Integer?

```
bool cmp(const int &a, const int &b) {
    return a<b;
}
```

- Und für `double`?
- Und für `string`?
- ...

=> Präprozessor-Makro?

=> Funktionen überladen?

Definition von Funktions-Templates

```
template <typename T>
bool cmp(const T &a, const T &b) {
    return a<b;
}
```

Vorsicht: Im Beispiel oben muss `operator<` für die verwendeten Typen `T` implementiert sein!
(sonst Fehler zur Compile-Zeit)

Bestimmung der Template-Parameter I: Typ-Inferenz

```
template <typename T>
bool cmp(const T &a, const T &b) {
    return a<b;
}

int main() {
    cmp(3, 10);                          // cmp(int, int)
    cmp(2.2, 10.1);                      // cmp(double, double)
    cmp(string("abc"), string("ABC"));   // cmp(string, string)
    cmp(3, 3.4);                         // Compiler-FEHLER!!!
}
```

Bestimmung der Template-Parameter II: Explizite Angabe

```
template <typename T>
bool cmp(const T &a, const T &b) {
    return a<b;
}

int main() {
    cmp<int>('a', 'A');      // cmp(int, int)
    cmp<int>(3, 3.4);        // cmp(int, int)
}
```

Spezialisierung von Funktions-Templates

```
// Primaeres Template
template <typename T>
bool cmp(const T &a, const T &b) {
    return a<b;
}
```

```
// Spezialisiertes Template
template <>
bool cmp<int>(const int &a, const int &b) {
    return abs(a)<abs(b);
}
```

Spezialisierte Templates **nach** “primärem” Template definieren

Konsole: funktionsTemplates.cpp

Klassen-Templates in C++

```
template <typename T>
class Matrix {
    Matrix(unsigned rows = 1, unsigned cols = 1);
    vector<vector<T> > xyField;
};
```

```
int main() {
    Matrix<int> m1;
    Matrix<double> m2(12, 3);
}
```

```
template <typename T>
Matrix<T>::Matrix(unsigned rows, unsigned cols) { ... }
```

Klassen-Templates in C++ spezialisieren

```
template <typename T>
class Matrix {
    Matrix(unsigned rows, unsigned cols);
    vector< vector<T> > xyField;
};
```

```
template <>
class Matrix<uint> {
    Matrix(unsigned rows, unsigned cols);
    vector< vector<uint> > xyField;
};
```

Hinweis auf Implementierung außerhalb der Klasse

Templates: Java vs. C++

- Templates sind nur **Schablonen**!
- **Unterschied zu Java**
  - C++: Für **jeden** Aufruf/Typ eine passende **Instanz** (!)
  - Java: Nur **eine** Klasse mit gemeinsamen Obertyp
- Offener Code: **Templates im `.h`-File implementieren!**
- Bibliotheken und Templates passen nicht recht

Wrap-Up

- Generische Programmierung (Funktions-Templates)
  - `template <typename T>` der Funktionsdefinition voranstellen
  - Funktions-Templates sind spezialisierbar und überladbar
  - Aufruf: Compiler nimmt die am besten "passende" Variante
- Generische Programmierung (Klassen-Templates)
  - Funktionsweise analog zu Funktions-Templates
  - Bei Implementierung außerhalb der Deklaration: Template-Deklaration mitführen!
  - Klassen-Templates lassen sich partiell spezialisieren
- Compiler stellt je instantiiertes Template eine konkrete Funktion/Klasse bereit

LICENSE