

C++

Templates

STL

**Spezialisierung
von Templates**

**Code
Organisation**

**Fortgeschrittene
Templates**

Folgerung

C++Templates

Proseminar Programmiersprachen

C++ Templates

**Betreuer: Steven Obua
Referentin: Johanna Witte**

8. November 2006, Johanna Witte


C++

Templates

STL

Spezialisierung
von TemplatesCode
OrganisationFortgeschrittene
Templates

Folgerung

- Bjarne Stroustrup
- „C with Classes“  C++
- C++ bildet die Obermenge von C
- Vergleich von C++ mit Java und C# ist nahe liegend

Wo findet C++ Verwendung

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Allgemein verwendbare Sprache
- Weit verbreitet und industriell bedeutend
- Effiziente, maschinennahe Programmierung
- Hauptanwendungsgebiet Systemprogrammierung

Was unterstützt C++

C++

Templates

STL

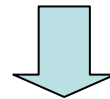
Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Multiparadigmen-Sprache
- Datenabstraktion
- Generische, prozedurale, modulare und strukturierte Programmierung
- Objektorientierte Programmierung



Flexibilität und Wiederverwendbarkeit

Besonderheiten von C++

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- hocheffizienter Code
- Flexibel, hohe Ausdrucksstärke
- Kompatibilität mit C



Vorteile und Nachteile

- Breites Leistungsspektrum
- Vielfältige Gestaltungsmöglichkeiten

Generische Programmierung

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Programmierparadigma
- Templates
- Parametrische Polymorphie
- Paradebeispiel C++
Standardbibliothek

Was sind Templates

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Schablone
- Anfang der 90er in C++ eingeführt
- Großer Nutzen
- Äquivalente Konzepte in anderen Sprachen

Wozu benutzt man Templates

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Verallgemeinerung
- Reduzierung des Aufwandes für den Programmierer

Wie definiert man ein Template?

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

Sagt dem
Kompiler, dass
folgendes als
Schablone zu
behandeln ist

Platzhalter =
Templateparameter

```
template<class T>
void swap(T&a, T&b) {
    T tmp= a;
    a = b;
    b = tmp;
};
```

Welche Arten von Templates gibt es

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Funktionstemplates
 - verhält sich ähnlich wie eine Funktion
- Klassentemplates
 - Gleiche Prinzip wie Funktionstemplates nur auf Klassen angewendet

Überladen von Templates

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Mehrere Funktions-Templates mit demselben Namen
- Auflösung von Überladungen

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Abkürzung für „Standard Template Library“
- Teil der Standardbibliothek von C++
- Wesentliche Komponenten:
 - Algorithmen
 - Container
 - Iteratoren
 - Funktionsobjekte
 - Adaptoren

Spezialisierung von Templates

C++

Templates

STL

Spezialisierung
von Templates

Code

Organisation

Fortgeschrittene
Templates

Folgerung

- Ermöglicht effiziente Implementierung für bestimmte ausgewählte Datentypen, ohne Veränderung der Schnittstelle des Templates
- Viele Implementierungen der Standardbibliothek machen davon Gebrauch

Explizite Spezialisierung

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

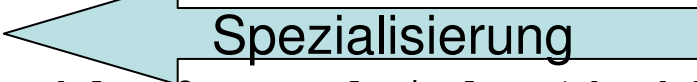
Folgerung

Standardimplementierung:

```
template<class T, int Size>
void MyVector::multiply (int d) {
    for (int i=0; i<Size; i++){
        data[i] *=T(d);
    }
};
```

Spezialisierte Implementierung:

```
template<>
void MyVector<double,2>::multiply (double d){
    data[0] *=d;
    data[1] *=d;
};
```



Partielle Spezialisierung

C++

Templates

STL

Spezialisierung
von Templates

Code

Organisation

Fortgeschrittene
Templates

Folgerung

- Behandlung von Spezialfällen innerhalb eines Klassen-Templates
- Beispiel:

```
template<class T>  
void MyVector<T, 2>::multiply  
(double d) {...};
```

Folgerung

C++

Templates

STL

Spezialisierung
von Templates

Code

Organisation

Fortgeschrittene
Templates

Folgerung

- Keine Wiederholungen von analogem Code
- Nicht weniger oder mehr kompilierter Code als von Hand
- Geschwindigkeitsgewinn

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Präprozessor
 - Makrosubstitution
 - Einfügen von Headerdateien
- Kompilieren und Binden
- Fehler die durch Templateparameter entstehen können nicht vor der ersten Benutzung erkannt werden!

Inclusion Model

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Heute gängigste Lösung
- Implementierung findet in der *.h Datei statt

Separation Model

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Export
- Von fast keinem Übersetzer unterstützt
 - Zukunft noch unklar

Template Instanziierung

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

= Umwandlung des Templates in eine konkrete, übersetzbare Einheit

- Spezialisierung des Templateparameters

Implizite Template Instanziierung

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

• Automatische Spezialisierung

Beispiel Minimum Berechnung von zwei Werten:

```
template<class T>  
T min (T a, T b) {  
    return ((a<b) ? a : b);  
}
```

```
double d1=5.0, d2=3.0;  
double m = min (d1, d2);
```

Geht, da
T = double

```
double d1=5.0;  
float f1=10.0f;  
double m = min(d1, f1);
```

No matching
function for
call to min
(double&,float&)

Explizite Template Instanziierung

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Ausdrückliche Spezifikation des Templateparameter

- Erstes Beispiel:

```
template<class T>
void swap(T&a, T&b) {
    T tmp= a;
    a = b;
    b = tmp;
};
```

- Explizite Instanziierung:

```
double d1=5.0, d2=3.0;
swap<double>(d1, d2);
```

Code Organisation ohne Templates

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Separate Dateien (*.h, *.cpp)
- Linker

Code Organisation mit Templates

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Bei Bedarf
(durch Bindung an spezifische
Templateparameter)

Wo soll die Implementierung untergebracht werden?

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Drei Möglichkeiten:
 - Instanziierung erzwingen
 - Behandlung wie Inline-Funktion
 - Behandlung wie Inline-Funktion aber mit separatem Header

Fortgeschrittene Templates

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Template Metaprogramme
 - Gezielte Code-Generierung
- Introspection
 - Statische Analyse der vorhandenen Typen

Template Metaprogramme

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Programmcode von Programmcode
- Laufzeitverkürzung,
Kompilierungsverlängerung
- Mächtige Technik
- Boost-Bibliothek
- Turing-Vollständig
- Nachteile!

Beispiel Template Metaprogramme

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

```
#include <iostream>
template<long B,unsigned long E>
struct pow_helper {
static const double value;
};
template <long B,unsigned long E>
const double pow_helper<B,E>::value=B*pow_helper<B,E-
1>::value;
template<long B>
struct pow_helper<B,0> {
static const double value;
};
template <long B>
const double pow_helper<B,0> ::value=1;
template<long B,long E> struct power {
static const double value;
};
template <long B,long E>
const double power<B,E>::value= E<0 ? 1.0/pow_helper<B,
-E> :
pow_helper<B,E>::v
```

Exponent = 0
Spezialisierung pow_helper
Ergebnis = 1

power wird aufgerufen wenn
Exponent negativ:
1.0/B^-E wird berechnet

$P(B,E) := B * P(B,E-1)$ $P(B,0) := 1$

C++Templates

Folgerung

C++

Templates

STL

Spezialisierung
von Templates

Code
Organisation

Fortgeschrittene
Templates

Folgerung

- Generischer Code
- Zahlreiche Bibliotheken
- Hohe Komplexität

C++

Templates

STL

**Spezialisierung
von Templates**

**Code
Organisation**

**Fortgeschrittene
Templates**

Folgerung

**Vielen Dank für die
Aufmerksamkeit**