Programmieren in C++

SS 2010

Vorlesung 8, Mittwoch 16. Juni 2010

(templates, templates, templates)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem 7. Übungsblatt
  - Mein Name in Ihrer copyright notice
- Templates
  - Was ist das und wofür ist das gut?
  - Explizite Instantiierung / explicit instantiation
  - Partielle Instantiierung / partial instantiation
  - Übungsblatt: Sie schreiben eine einfach Klasse Set unter Verwendung von templates

Erfahrungen mit dem 7. Übungsblatt

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Etwas schwerer als das Ü7, aber für die meisten ok
  - Sah erst schwieriger aus als es ist
  - fopen, fgets, etc. wurde nicht genügend erklärt
  - Aufgabenstellung ändert sich ständig
  - Vorlesungs-Code ändert sich ständig
  - Testfälle schreiben nervt / dauert die Hälfte der Zeit
  - Aufgaben wo man mehr denken muss bitte
  - Etwas zu viel Stoff
  - Bitte wieder etwas langsamer vorwärts
  - Hauptlintfehler ist whitespace am Ende
  - Es scheint sich so langsam in zwei Lager zu spalten

Mein Name in Ihrer Copyright Notice

- Was macht der da?
  - Bitte den Namen durch Ihren eigenen ersetzen
  - Sonst ab nächsten Mal 1 Punkt Abzug
  - Ab übernächstem Mal 2 Punkte Abzug
  - Ab überübernächstem Mal 4 Punkte Abzug
  - Ab überüberübernächstem Mal 8 Punkte Abzug
  - Ab überüberüberübernächstem Mal 16 Punkte Abzug
  - ...
  - Bei Übungsblatt 18 sind das schon 1024 Punkte Abzug

Templates

UNI FREIBURG

- Wofür braucht man templates?
  - Oft hat man Klassen, die man fast genauso auch für einen anderen Typ braucht, zum Beispiel

    ```
    class ListOfIntegers { ... Elemente vom Typ int ... };
    class ListOfChars { ... Elemente vom Typ char ... };
    ```

  - Jetzt kann man natürlich den Code kopieren und in der Kopie überall int durch char ersetzen
  - Aber das ist sehr schlechter Stil und fehleranfällig, weil man Änderungen dann immer an zwei Stellen machen muss
  - Genau dafür hat man templates, dann hat man ein und denselben Code für einen beliebigen Typ

    ```
    template<class T> List<T> { ... Elemente vom Typ T ... };
    ```

  - Details siehe Codebeispiel in Array.h und Array.cpp

Explizite Instantiierung

- Mit einem template davor wird erstmal *kein* Code erzeugt
  - Siehe nm –C Array.o
  - Man muss die benötigten Instanzen explizit instantiieren

    template class Array<int>;

    template class Array<char>;
  - Das erzeugt dann erst den Code für die Klasse mit dem entsprechenden Typ
  - Aber man beachte, dass man den ganzen Code trotzdem nur einmal schreiben musste

Template Spezialisierung

UNI FREIBURG

- Manchmal möchte man für einen einzelnen Typ die Sache ganz anders implementieren
  - zum Beispiel für ein Array<bool> 8 bits in einen char packen (defaultmäßig braucht ein bool einen ganzen char)
  - das schreibt man dann so

```
template<> Array<bool>
{
  hier jetzt die speziellen Deklaration für Array<bool>,
  die beliebig anders sein können als die in Array<T>.
}
```

  - Siehe Codebeispiel in Array.h und Array.cpp

```
char x = 5;     //00000101
char y = 1<<4;  //00010000
char z = x | y; //00010101
```

```
  10101100
  00000010
  --------
| 10101110
```

```
  10101100
  00001000
  --------
& 00001000
```

Literatur / Links

- Templates, explicit instantiation, specialization
  - http://www.cplusplus.com/doc/tutorial/templates