Programmieren in C++

SS 2012

Vorlesung 8, Dienstag 26. Juni 2012

(Templates + alles was dazu gehört, valgrind)

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
- Templates
  - Was ist das und wofür ist das gut?
  - Deklaration, Instanziierung, Spezialisierung
  - Wir machen das am Beispiel einer einfachen Array Klasse
  - Außerdem:
    - die bitweisen Operatoren `&` `|` `^` `~` `<<` `>>`
    - das Speicherzugriffsfehlerentdeckungstool valgrind
  - Übungsblatt: Eine templatisierte String Klasse mit einer Spezialisierung für Bitstrings

Erfahrungen mit dem Ü7 (getopt)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 26. Juni 14:00
  - Machbar, aber für einige länger gedauert als gedacht
  - Man konnte wieder viel übernehmen
  - Erklärung für getopt in der Vorlesung war einigen zu knapp
  - Leichte Verwirrung von Frau Bast hat sich übertragen
  - Probleme mit Leerzeichen bei den kurzen Optionen -i xyz
  - Fehler in der Musterlösung für die String Klasse
  - Bei Benutzereingabe gibt Strg+D ein EOF
  - Leider habe ich seit einer Woche Grippe … nicht nur Sie
  - Die meisten hatten aber trotzdem keine größeren Probleme
  - Alles besser nachdem die Lüftung angeschaltet wurde
  - Es ist vor allem eine Konzentrationssache … genau richtig!

Erfahrungen mit dem Ü7 (getopt)

UNI FREIBURG

- Fortsetzung ...
  - Vorlesungsaufzeichnung extrem asynchron ... stimmt
  - Wieder valgrind benutzt ... heute eine Folie dazu (am Ende)
  - Segmentation faults tauchen vermehrt bei Neumond auf
  - Fehler mit checkstyle bei switch und else ... ist jetzt gefixt
  - Stört: Teile der Aufgabe auf Übungsblatt, Teile in Datei
  - Darf man die Dateien aus der VL verwenden ... aber ja doch!
  - Was hat es mit dem explicit auf sich? ... spätere Vorlesung
  - Noch nicht 100% durchgestiegen bei:
    - Zeigern, Konstruktoren, Destruktoren, Operatoren, ...
  - Mehr Hilfestellung für die "Abgehängten"?
  - Wann gibt es Infos zum Abschlussprojekt?

Templates Grundprinzip 1/2

UNI FREIBURG

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

    class ArrayInt { ... Elemente vom Typ int ... };

    class ArrayFloat { ... Elemente vom Typ float ... };
  - Jetzt kann man natürlich den Code kopieren und in der Kopie überall int durch float 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> Array<T> { ... Elemente vom Typ T ... };
  - Details siehe Codebeispiel in Array.h und Array.cpp

Templates Grundprinzip 2/2

UNI FREIBURG

- Wie benutzt man so eine template Klasse?
  - Indem man in spitzen Klammern angibt, für welches konkrete T man die Klasse haben will

    Array<int> arrayInt;
    Array<float> arrayFloat;
  - Das <int> bzw. <float> ist dabei **Teil des Klassennamens**
  - In der Tat sind bei der Benutzung Array<int> und Array<float> wie zwei völlig **verschiedene** Klassen
  - Wir haben lediglich den Code dafür auf besondere Weise geschrieben (nämlich nur einmal, für beide Klassen zusammen)

Template Instanziierung 1/3

UNI FREIBURG

- Beim Kompilieren einer template Funktion
  - … wird noch **kein** Code erzeugt
  - Siehe nm –C Array.o
  - Um Code zu erzeugen, muss man sagen für welche konkreten T man die Klasse gerne hätte
  - Dafür gibt es zwei Alternativen
    - Alternative 1: Implementierung in der .h Datei
    - Alternative 2: explizite **Instanziierung** in der .cpp Datei
  - Beide Alternativen haben Vor- und Nachteile
  - Wir bevorzugen in dieser Veranstaltung **Alternative 2**

Template Instanziierung 2/3

UNI FREIBURG

- Alternative 1: Implementierung in der .h Datei
  - Die Deklaration wie gehabt in der .h Datei
  - Man schreibt die Implementierung, die normalerweise in der .cpp Datei steht, **auch** in die .h Datei
  - **Vorteil:** Klasse wird erzeugt wenn sie gebraucht wird

```
#include "./Array.h"
...
Array<int> arrayInt;  // Here Array<int> gets compiled.
...
Array<float> arrayFloat;  // Here Array<float> gets compiled.
```

  - **Nachteil:** Wenn in 10 verschiedenen Dateien ein Array<int> benutzt wird, wird der Code dafür 10 mal kompiliert
  - Ok, wenn der Code relativ einfach / schnell zu kompilieren ist

Template Instanziierung 3/3

UNI FREIBURG

- Alternative 2: Explizite Instanziierung
  - Explizite Code-Erzeugung in der .cpp Datei

    ```
    template class Array<int>;  // Compile Array<int> here.
    template class Array<float>;  // Compile Array<float> here.
    ```

  - **Vorteil:** der Code für die entsprechenden Klassen muss jetzt nur einmal kompiliert werden und steht auch nur einmal irgendwo, nämlich in der entsprechenden .cpp Datei
  - **Nachteil:** man muss in der .cpp Datei explizit sagen, für welche T man den Code haben will
    - das ist unmöglich für Bibliotheken, wo der Schreiber der Bibliothek gar nicht wissen kann, für welches T jemand die templatisierte Klasse später mal benutzen will

      ```
      Array<MyPersonalAndStrangeObject> myArray;
      ```

Templates für einzelne Funktionen

UNI FREIBURG

- ... unabhängig davon ob die Klasse "templatisiert" ist

```
template<class T> T cube(T x) {
  return multiply(multiply(x, x), x);
}
...
int n = 3;
printf("n^3 = %d\n", cube<int>(n));  // Prints 27.
```

  - Bemerkung 1: Erst beim **Aufruf** der Funktion wird geschaut, ob die Funktion für den Typ überhaupt kompiliert werden kann
    - hier: ob es überhaupt die multiply Methode für den Typ gibt
  - Bemerkung 2: Man kann beim Aufruf statt cube<int> auch cube schreiben ... der Compiler findet den Typ dann selber heraus

Template Fehlermeldungen

UNI FREIBURG

- ... haben eine charakteristische Form

  FileX:123: instantiated from [some function]
  FileY:456: instantiated from [some other function]
  ...
  FileZ:789: instantiated from [yet another function]
  SomeFile.cpp:666: instantiated from here
  SomeFile.h:555: error: [some error message]

  – Am wichtigsten sind dabei meistens die **letzten** Zeilen:
    - Die Zeile mit dem instantiated from here sagt, an welcher Stelle das template **benutzt** wird, und der Compiler es dann kompilieren wollte
    - Die Zeile mit dem error sagt, wo beim **Kompilieren** in der template **Deklaration** Fehler aufgetreten sind

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 ein Byte** packen (defaultmäßig braucht ein bool ein ganzes Byte)
  - Die Deklaration in der .h Datei 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>.
}
```

  - Die Implementierung in der .cpp Datei dann **ohne** template

```
Array<bool>::append { ... }
```

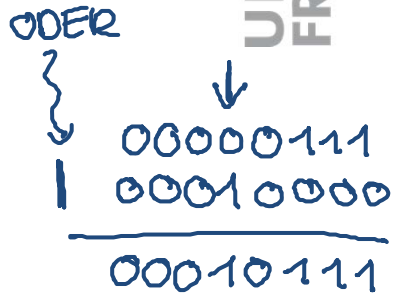
  - Siehe Codebeispiel in Array.h und Array.cpp

Operatoren zur bitweisen Manipulation

UNI FREIBURG

- Hier anhand von ein paar Beispielen erklärt

```
char  x = 7;                // 00000111 in binary
char  y = 18;               // 00010010 in binary.
```



  - Bitweise Operatoren für AND, OR, XOR, NOT

```
char  x_or_y = x | y;       // 00010111 in binary = 23.
char  x_and_y = x & y;      // 00000010 in binary = 2.
char  x_xor_y = x ^ y;      // 00010101 in binary = 21.
char  not_x = ~x;           // 11111000 in binary = 249.
```

  - Bitschiebe-Operatoren << und >>

```
char  x_shifted = x << 2;   // 00011100 in binary = 28.
char  y_shifted = y >> 3;   // 00000010 in binary = 2.
```

  - Das geht genauso mit zum Beispiel int nur sind es dann entsprechend mehr Bits, typischerweise 32

valgrind

UNI FREIBURG

- Zum Aufspüren von Speicherzugriffsfehlern
  - Einfach valgrind vor das ausführbare Programm setzen

    valgrind ./ArrayTest
  - Das "simuliert" den Code und schaut dabei, an welchen Stellen auf unerlaubten Speicher zugegriffen wird, und ob irgendwo Platz, der mit new alloziert wurde, nicht mehr freigegeben wurde
  - Gibt dann einen aussagekräftigen Problembericht ... für Info zu verantwortlichen Zeilen im Code: Option --leak-check=full
  - **Achtung:** das Programm läuft durch diese Simulation um ein Vielfaches langsamer!
  - Installation (Ubuntu/Debian) mit sudo apt-get install valgrind

Literatur / Links

UNI FREIBURG

- Templates, Instanziierung, Spezialisierung
  - http://www.cplusplus.com/doc/tutorial/templates
- Bitweise Operatoren
  - http://www.cplusplus.com/doc/tutorial/operators/
- Valgrind
  - http://valgrind.org/