Programmieren in C++

SS 2010

Vorlesung 10, Mittwoch 30. Juni 2010

(std::sort mit eigener Sortierreihenfolge)

Jens Hoffmann
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem 9. Übungsblatt
- Sortieren mit frei gewählter Sortierordnung
  - Für die letzte Vorlesung / Übung hat uns Sortieren nach der Standardreihenfolge von int / char gereicht
  - Oft will man aber seine eigene Sortierreihenfolge festlegen

  Rückgabe von komplexen Objekten in einer Methode

  - Übungsblatt: Berechnen Sie die k größten Anagrammgruppen mit Hilfe eines Sortierens mit eigener Sortierreihenfolge

Erfahrungen mit dem 9. Übungsblatt

UNI FREIBURG

- Zusammenfassung / Ausüge
  - Zeitaufwand weit gestreut
  - Aufgabe hat Spass gemacht
  - Laufzeitoptimierungen haben vielen gefallen
  - Laufzeitrekord etwa bei etwa 0.8 Sekunden.
  - „Endlich stl, […] so machts gleich mehr Spass."
  - „Wie soll man das aber testen?"

Überladen von Operatoren

UNI FREIBURG

- Motivation: Eine Klasse Scientist habe zwei Memberfelder: _name und _age. Zwei Instanzen der Klasse Scientist (x und y) seien gleich wenn

  x._name == y._name

- Statt bei jedem Vergleich zweier Wissenschaftler diesen Ausdruck aufzuschreiben, kann man den Operator '==' neu definieren (überladen), denn
- Operatoren sind Funktionen
- Dann lässt sich auch so vergleichen: x == y
- Nicht nur '==' kann überladen werden, auch: '[]', '()', '=', '+', '-', ...

std::sort über komplexen Objekten

UNI FREIBURG

- Sortiere Buchstaben in einem String:

  ```
  std::sort(aString.begin(), aString.end());
  ```

- aString nicht lexikographisch sortieren, sondern anders?
- Eigene Datentypen sortieren?
- std::sort versteht einen dritten Parameter, der beschreibt wie sortiert werden soll.
- Dieser dritte Parameter ist eine Vergleichsfunktion.

std::sort über komplexen Objekten

UNI FREIBURG

- class Scientist: Eine Klasse bestehend aus Memberfeldern: _name und _age.
- Scientists sollen nun nach ihrem Alter sortiert werden.
- 1. Schritt: In der Header, in der auch 'class Scientist' definiert ist, definieren wir:

```
class ScientistCompare
{
   public:
     bool operator()(const Scientist& a, const Scientist& b)
};
```

std::sort über komplexen Objekten

```
class ScientistAgeCompare
{
   public:
     bool operator()(const Scientis& a, const Scientist& b)
};
```

- Zwei Klassen in einer Header? Ist das möglich? - Na klar doch! Soviel Sie wollen, nach dem Motto: zusammen was zusammen gehört.

Kleine Frage Zwischendurch ...

- ... wir sind ja jetzt unter uns ...

```
class ScientistCompare
{
   public:
     bool operator()(const Scientis& a, const Scientist& b);
 };
```

- Wer hat nur eine blasse/oder gar keine Ahnung was das '&' im obigen Codeschnipsel bedeutet?
- Bitte um Handzeichen ...

*(*Einschub)[(&Referenzen)] & Pointer

- class A { ... }; eine Klasse
- A object; ein Objekt der Klasse A

→ Dieses Objekt purzelt jetzt im Speicher herum. Wir wollen wissen wo im Speicher?

→ Objekte haben eine Adresse:

- &object; die Adresse des Objekts object

→ Der &-Operator vor einem Objekt wirkt Adressbeschaffend.

*(*Einschub)[(&Referenzen)] & Pointer

- A* pointer = &object;

→ Pointer ist ein Zeiger auf das Objekt object

→ Pointer speichern Adressen

- Warum Pointer?

→ Pointer sind handlicher als Objekte.

- Wir arbeitet man mit Pointern?
- (*pointer).somePublicMemberFunction();

→ Um auf ein Memberfeld des Objects zugreifen zu können, muss der Pointer mit dem *-Operator dereferenziert werden. Dereferenzieren = Adresse auflösen.

(*Einschub)[(&Referenzen)] & Pointer

UNI FREIBURG

- Referenzen sind auch Zeiger, zumindest intern.
- Unterschied: Referenzen müssen nicht mit dem
- *-Operator dereferenziert werden.
- Beispiel:
  - `class A { … };` wie gehabt eine Klassendefinition.
  - `A object;` Ein Objekt vom Typ A.
  - `A& ref1 = object;`
  - `const A& ref2 = object;`

→ Die Referenz ref1 kann wie object behandelt werden.

→ Über die konstante Referenz ref2 darf object nicht verändert werden.

std::sort über komplexen Objekten

- Wir waren stehen geblieben bei ...

```
class ScientistCompare
{
  public:
    bool operator()(const Scientis& a, const Scientist& b);
 };
```

- ... und wir wollten damit zwei Scientists vergleichen.

std::sort über komplexen Objekten

```
class ScientistCompare
{
   public:
     bool operator()(const Scientis& a, const Scientist& b);
 };
```

- ScientistCompare compare;
- compare.operator()(a, b);

  compare(a, b)
- compare(a,b) stellt die Frage: „Sind a und b in order?“
- Was „in order“ bedeutet definieren wir in der Implementierung der operator()-Funktion

std::sort über komplexen Objekten

```
class ScientistCompare
{
  public:
    bool operator()(const Scientis& a, const Scientist& b);
 };
```

- ScientistCompare compare;
- vector<Scientist> scientists;
- std::sort(scientists.begin(), scientists.end(), compare);

Literatur / Links

UNI FREIBURG

- Operatoren überladen:
  - http://www.willemer.de/informatik/cpp/cppovrld.htm
- Wie immer good-old sgi:
  - http://www.sgi.com/tech/stl/sort.html
- Zum Unterschied: Referenzen & Pointer:
  - http://www.dgp.toronto.edu/~patrick/csc418/wi2004/notes/PointersVsRef.pdf