Programmieren in C++

SS 2011

Vorlesung 9, Mittwoch 6. Juli 2011

(STL, vector, string, sort, iostream, namespaces)

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 8. Übungsblatt
  - Ein paar Vorab-Infos über das Projekt am Ende
- Die STL = Standard Template Library
  - Klassen für Sachen, die man oft braucht
  - std::vector = ähnlich zu unserer Array Klasse
  - std::string = ähnlich zu unserer String Klasse
  - std::iostream für Ein- und Ausgabe
  - std::stringstream zum "Zusammenbauen" von Strings
  - std::sort zum Sortieren von beliebigen Objekten
  - und was heißt dieses std::... ?
  - **9. Übungsblatt:** Überarbeiten Sie Ihre Abgabe für das Ü4 und verwenden Sie jetzt std::vector, std::string, etc.

Erfahrungen mit dem Ü8 (String<T>)

UNI FREIBURG

- Zusammenfassung / Auszüge **Stand 6.7 15:52**
  - Bitgefriemel war schwierig und / oder hat viel Zeit gekostet
  - Code dafür sehr umständlich
  - Bitte nächstes Mal Umfang wieder geringer
  - Bitte mehr Bitoperatoren und näher an die Hardware, dafür sind doch Hochsprachen gemacht ...
  - Vorlesung zu gehetzt letztes Mal, lieber entspannter
  - Fragen zu Urlaubserlebnissen erst am Ende stellen
  - Was ist dieses explicit nachdem der Linter manchmal fragt?
  - Ich hasse Tests ... die machen nie was ich will
  - Geschnittenes M4V schneller online wäre cool
  - Zeitlicher Rahmen und Umfang des Projektes?

Das Projekt am Ende der Vorlesung 1/3

UNI FREIBURG

- Art des Projektes
  - Zwei Aufgaben zur Auswahl, von verschiedenem Schwierigkeitsgrad
  - Für die Erfahreneren / Unterforderten (aber nur für die) die Möglichkeit ein eigenes Projekt zu definieren
  - Sie schreiben dann ein C++ Programm nach der Art, wie wir es in den Übungsblättern gemacht haben, mit Klassen, Tests, und allem ... nur halt etwas größer
  - Umfang ca. 4 Übungsblätter
  - Siehe die Beispiele von der Vorlesung im SS 2010

    http://ad-wiki.informatik.uni-freiburg.de/teaching

Das Projekt am Ende der Vorlesung 2/3

UNI FREIBURG

- Zeitlicher Rahmen
  - Wir fangen in der vorletzte Vorlesungswoche damit an
  - Das Übungsblatt dazu wird sein, sich ein Design für ihr Programm zu überlegen und die .h Dateien zu schreiben
  - Dazu kriegen Sie dann in der Woche drauf (so schnell wie möglich) Feedback von Ihren Tutoren
  - Wenn Sie sich ranhalten, können Sie Ihr Programm dann in der letzten Vorlesungswoche und in der Woche drauf fertig schreiben … das wäre auch meine Empfehlung, dann haben Sie es hinter sich
  - Finale Deadline für die Abgabe des Projektes ist

    Mittwoch, der 14. September 2011 um 16 Uhr

Das Projekt am Ende der Vorlesung 3/3

UNI FREIBURG

- Muss man das Projekt machen?
  - Ja!
  - Auch wenn man sonst fast alle Punkte in den Übungen hat?
  - Ja!
  - Wirklich?
  - Ja!
  - Zum Bestehen brauchen Sie mindestens 50% der Punkte aus den Übungen **und** mindestens 50% der Punkte aus dem Projekt

Die STL

UNI FREIBURG

- STL = Standard Template Library
  - Voller nützlicher Klassen, die man immer wieder braucht
    - vector (unser Array), string (unser String), etc.
  - Dank templates können alle diese Klassen für alle möglichen Arten von Objekten benutzt werden können
    - vector<int>, string<w_char>, ...
  - Alle Klassen in der STL stehen im std namespace
    - Deswegen muss man schreiben std::vector<int> etc.
    - Erklärung dazu nächste Folie

Namespaces 1/2

UNI FREIBURG

- Wer, wie, was, warum
  - Gruppen von Klassen die zu einem bestimmten Projekt oder zu einer Bibilothek gehören macht man oft in einen eigenen sogenannten namespace
  - konkret heißt das, dass einfach um den Code herum steht

```
namespace std
{
  ...
}
```

  - Wenn man etwas aus diesem namespace benutzt, muss man dann den Namen des namespace gefolgt von :: davor schreiben, also z.B. std::vector<int>
  - Grund: Ich will ja vielleicht meine eigene Klasse vector schreiben (und sie genauso nennen)

Namespaces 2/2

UNI FREIBURG

- Muss man das std:: wirklich immer davor schreiben?
  - So ohne Weiteres ja
  - Aber wenn Sie am Anfang der Datei schreiben

    using namespace std;

    dann können Sie es in der Datei weglassen
  - Damit machen Sie aber den Schutz vor Namensgleichheit zunichte, weswegen es namespaces überhaupt gibt
  - Deswegen wird der Linter meckern ... zurecht!
  - Für spezielle Klassen können Sie aber schreiben

    using std::vector;
  - Dann können Sie in dem Rest der Datei schreiben

    vector<int> array; // So glad I didn't have to write std:: !

std::vector

UNI FREIBURG

- Wer, wie, was, warum
  - Ein dynamisches Feld von Objekten, ähnlich zu unserer Klasse Array aus der 8. Vorlesung

```
std::vector<int> numbers;
numbers.push_back(1);
numbers.push_back(2);
for (size_t i = 0; i < numbers.size(); i++)
  printf(%d\n, numbers[i]);
```

  - size() liefert eine Variable vom Typ size_t, die je nach System 32 oder 64 bit hat, und keine negativen Werte annehmen kann. Deswegen an solchen Stellen Zählervariablen als size_t deklarieren und nicht als int
  - Details zu Methoden siehe Referenzen am Ende

std::string

- Wer, wie, was, warum
  - Komfortable Klasse für Zeichenketten, ähnlich zu unserer Klasse String vom 8. Übungsblatt

    ```
    std::string s = Hullo;
    size_t pos = s.find(u);
    if (pos != std::string::npos) s[pos] = a;
    printf(%s\n, s.c_str());
    ```

  - Wie beim std::vector ist die size() vom Typ size_t und ebenso die Positionen in einem std::string
  - Intern benutzt die Klasse einen null-terminierte C-Strings, und den bekommt man mit der c_str() Methode

    Das ist nützlich z.B. wenn man printf benutzt
  - Details zu Methoden siehe Referenzen am Ende

std::sort

- Wer, wie, was, warum
  - Sehr oft will man Objekte sortieren, und das geht einfach und effizient mit std::sort

```
#include <algorithm>
#include <vector>

std::vector<int> numbers;
numbers.push_back(2);
numbers.push_back(1);
numbers.push_back(3);
std::sort(numbers.begin(), numbers.end());
```

  - TODO: eigener Vergleich
  - Details siehe Referenzen am Ende

std::iostream 1/2

UNI FREIBURG

- Oberklasse für die Ein- und Ausgabe
  - std::ofstream für die Ausgabe in eine Datei
  - std::ifstream für das Lesen aus einer Datei
  - Zeige ich Ihnen nicht, weil ich selber immer fopen, etc. benutze … aber siehe die Referenzen am Ende
  - Die Objekte für die Benutzereingabe und die Ausgabe auf dem Bildschirm sind aber manchmal ganz nützlich
    - std::cout für die Ausgabe nach stdout
    - std::cerr für die Ausgabe nach stderr
    - std::cin für die Eingabe von stdin
  - Für das newline ('\n') schreibt man std::endl

std::iostream 2/2

- Beispiel für cin, cout und cerr

```
using std::cin;
using std::cout;
using std::cerr;
using std::endl;
...
int x, y;
cin >> x >> y;  // Reads two ints from stdin.
if (x < 0 || y < 0) cerr << "x and y should be > 0!" << endl;
cout << "x = " << x << "; y = " << y << endl;
```

  - Details dazu, siehe Referenzen am Ende

std::stringstream 1/4

UNI FREIBURG

- Nützlich für das Zusammenbauen von strings
  - Insbesondere in folgendem Kontext
  - Nehmen wir an, wir haben unsere eigene Klasse, z.B. SetOfIntegers, und wollen ein Objekt davon ausgeben
  - Bisher haben wir dafür eine print() Methode geschrieben
  - Vielseitiger ist aber eine asString() Methode, die ein Objekt in einen string schreibt, in menschen-lesbarer Form
  - Den string können wir dann ausgeben

```
cout << set.asString() << endl;          // If you prefer cout.
printf("%s\n", set.asString().c_str());  // If you prefer printf.
```

  - Oder in einem Test auf Gleichheit testen

```
ASSERT_EQ("{1, 3, 7, 12}", set.asString());
```

std::stringstream 2/4

- Beispielimplementierung (in der .cpp Datei)

```
#include <sstream>
...
// Return object as human-readable string.
string SetOfIntegers::asString()
{
  std::ostringstream os;  // Stream for assembling a string.
  os << "{";  // Can be used just like cout.
  for (int i = 0; i < _size; i++)
    os << (i > 0 ? ", " : "") << _elements[i];
  os << "}";
  return os.str();  // The assembled string;
}
```

std::stringstream 3/4

UNI FREIBURG

- Warum lassen wir asString einen string zurückgeben

  … und nicht einfach einen (const) char* ?

  – Wir müssen in der Funktion asString Speicherplatz für die Zeichenkette, die wir da zusammenbauen, allozieren

  – Wenn wir das mit new char[…] machen, muss derjenige, der asString() aufruft, den Speicher irgendwann wieder freigeben

  – Wenn wir das mit einem string Objekt machen, wird die Allokation von dem Objekt übernommen, und sobald das Objekt nicht mehr gebraucht wird, wird der Destruktor aufgerufen und der Speicher wieder freigegeben

std::stringstream 4/4

UNI FREIBURG

- Man kann auch den << Operator überladen
  - Das heißt, eine Funktion operator<< mit geeigneten Argumenten schreiben, so dass man schreiben kann

    cout << "My set is " << set << endl;

    im Gegensatz zu

    cout << "My set is " << set.asString() << endl; (*)
  - Für die string Klasse aus der STL brauchen wir das, sonst könnten wir (*) gar nicht schreiben
  - Ansonsten brauchen wir es nicht … im Gegenteil, es führt oft zu schwer verständlichen Fehlermeldungen; die Variante mit asString ist vielseitiger und klarer
  - Wen's trotzdem interessiert, siehe Referenzen

explicit

- Bei Konstruktoren mit genau einem Argument

  … möchte der Linter, dass wir explicit davor schreiben

```
// Stupid constructor that creates singleton set.
explicit SetOfIntegers(int x);
```

  – Das liegt daran, dass ohne dieses explicit, Konstruktoren mit einem Argument auch zur automatischen Typkonvertierung verwendet werden

```
SetOfIntegers.add(const SetOfIntegers& set)
...
SetOfIntegers set;
set.add(5);  // Will call the constructor above.
```

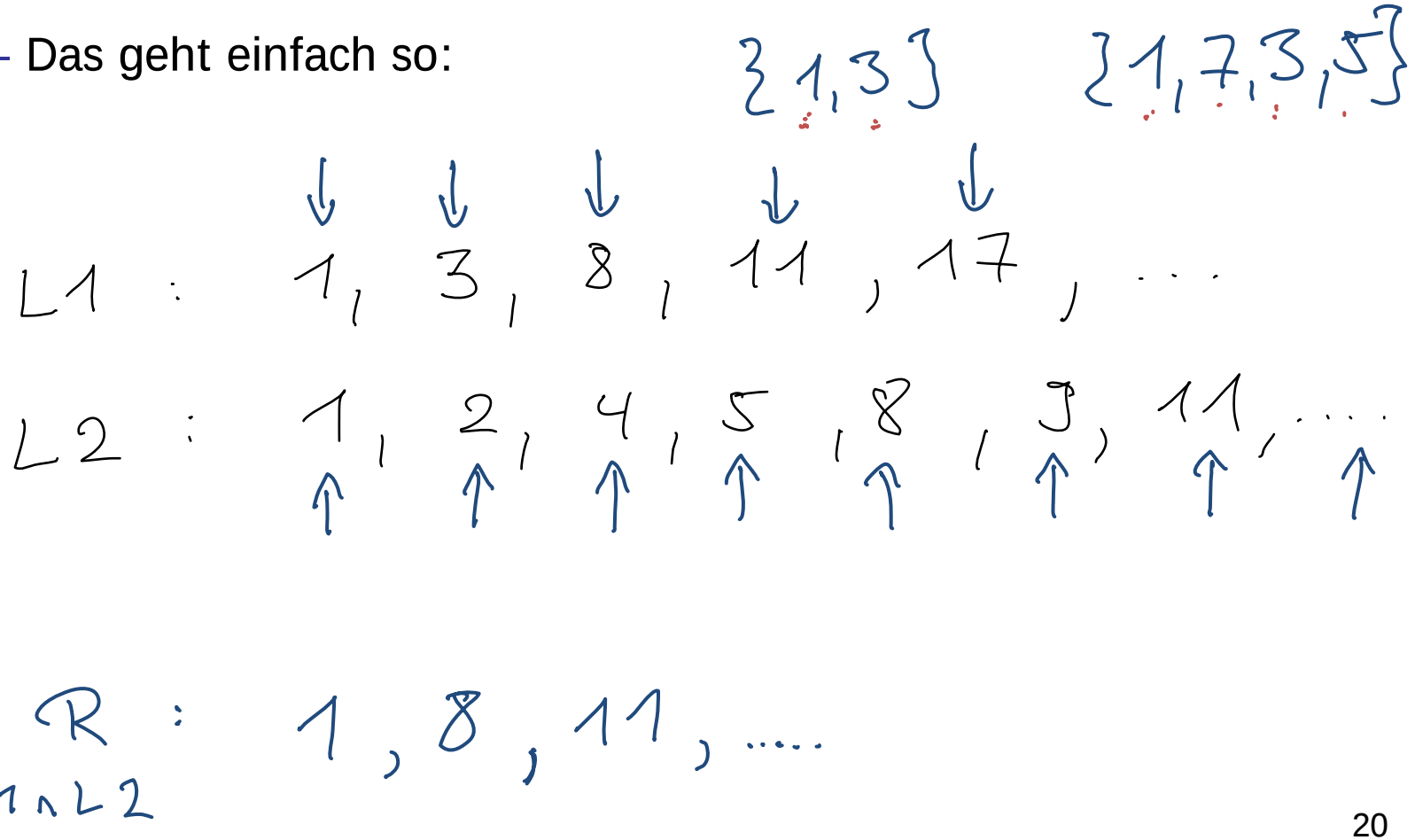
  – Das kann nett sein, aber auch zu sehr schwer zu debuggenden Fehlern führen, deswegen meckert Lint

intersectTwoSetsOfIntegers

- Kann man mit **linearer** Laufzeit implementieren

  ... wenn die Listen intern sortiert sind

  – Das geht einfach so:



Literatur / Links

UNI FREIBURG

- Standard Template Library (STL)
  - http://www.sgi.com/tech/stl
- Namespaces
  - http://www.cplusplus.com/doc/tutorial/namespaces
- vector, string, sort, iostream, stringstream
  - http://www.sgi.com/tech/stl/Vector.html
  - http://www.sgi.com/tech/stl/basic_string.html
  - http://www.sgi.com/tech/stl/sort.html
- Ein- und Ausgabe
  - http://www.cplusplus.com/doc/tutorial/basic_io/
  - http://www.cplusplus.com/doc/tutorial/files/