1. SVN, C++ und STL

AlDaBi Praktikum

David Weese
© 2010/11

René Rahn/Kathrin Trappe
WS 2012/13

VL-Übung II Di 14-16Uhr A3 R. 130

Praktikum

Das Praktikum umfasst:

diese Praktikumsvorlesung

- alle 2 Wochen, donnerstags 8.30 Uhr – 10 Uhr

Praktikumsaufgaben

- stehen auf jedem zweiten Übungszettel
- **Abgabe**: montags, 15 Uhr in den Wochen der P-VL
- Tutorien: dienstags, 12-14 und 14-16 in den Wochen zwischen den P-VLs
- 75% der Praktikumsaufgaben lösen
- Code-Reviews in den Wochen von Review und Examen

Inhalt

- Begleitende Themen zu den Programmieraufgaben: u.a. C++, Parellelprogrammierung, Templates, OOP

Abgabe der Lösungen

Abgabe per Subversion:

- jede Gruppe erhält ein SVN-Verzeichnis https://svn.imp.fu-berlin.de/aldabi/WS12/groupX
- die Lösung jeder Aufgabe muss als Datei mit dem Titel **aufgabeY.cpp** (Kleinschreibung beachten, keine Unterordner!) bis zum Abgabetermin eingecheckt werden
- ihr Programm muss u.a. auf einem Linux-Poolrechner kompilieren mit
  g++ -pedantic -Wall -ansi -o aufgabeY aufgabeY.cpp
- beachten Sie genau die Hinweise zum jeweiligen Ein- und Ausgabeformat auf den Aufgabenblättern und auf den Wiki-Seiten

Abgabe der Lösungen

Erstmaliges Auschecken:

- Eintragen als Gruppe unter

https://www.mi.fu-berlin.de/w/ABI/AlDaBiWS12Groups

- svn checkout

https://svn.imp.fu-berlin.de/aldabi/WS12/gruppeX

Lösung abgeben:

- cd gruppeX
- svn add aufgabeY.cpp
- svn commit aufgabeY.cpp

Literatur

- C++ für Einsteiger
  - *„Der C++ Programmierer“*, Breymann, Hanser Verlag
  - *„Einführung in die Programmierung mit C++“*, Stroustrup, Pearson St.
- C++ für Fortgeschrittene
  - *„C++ Templates“*, Vandevoorde und Josuttis, Addison-Wesley
  - *„Essential C++“*, Lippman, Addison-Wesley
  - *„Moderne C++ Programmierung”*, Schneeweiß, Springer
- STL
  - *„Effective STL“*, Meyers, Addison-Wesley
  - *„Generic Programming and the STL“*, Austern, Addison-Wesley
- Algorithmen
  - *„Algorithmen in C++“,* Sedgewick, Pearson Studium
  - *„Algorithmen und Datenstrukturen“*, Pomberger und Dobler, Pearson Studium

Web

www.cplusplus.com

C++ für Einsteiger

- Häufige Fragen

  http://www.parashift.com/c++-faq-lite/

- Tutorials

  http://code.google.com/intl/de-DE/edu/languages/cpp/basics/

  http://www.mi.fu-berlin.de/en/groups/abi/lectures_past/SS2010/K_CPP_Advanced/

- Referenzen

  http://www.cppreference.com/

  http://www.sgi.com/tech/stl/

  http://gcc.gnu.org/onlinedocs/gcc-4.2.2/gcc/

  http://msdn.microsoft.com/en-us/library/ms864102

C++ für Fortgeschrittene

- Diverse spezifische Artikel

  http://www.gotw.ca/publications/index.htm

- Forum für Entwickler

  http://stackoverflow.com/

Inhalt

Subversion (SVN)

C++

Standard Template Library (STL)

Subversion (SVN)

Kollaboration ...

Bei großen Projekten treten folgende Probleme auf:

- Wie koordiniert und synchronisiert man Code zwischen verschiedenen Entwicklern?
- Wie sichert man Code gegen versehentliches Löschen oder archiviert alte Stände?

Per Mail? Dateien online ablegen? Auf Netzlaufwerken arbeiten?

Besser: Source Code Management Tool (SCM) bspw.:

- CVS
- **Subversion**
- Bazaar
- Mercurial
- Git

... mit einem SCM

SCM besteht aus:

Repository

- Code gespeichert auf zentralem Server oder verteilt bei Usern

Working Copy

- Entwickler checkt eine Kopie des Repositories auf seinem Computer aus

Revision History

- jede Änderung einer Datei wird gespeichert
- kann auch wieder rückgängig gemacht werden

Conflict Handling

- Was passiert wenn zwei Entwickler dieselbe Datei verändern?
- ... in der selben Zeile?

Abläufe

Typischer Entwicklungszyklus

- Entwickler checkt **Repository** initial aus
- Editiert seine lokale **Working Copy**
- Aktualisiert (*update*) seine **Working Copy** mit den Änderungen der anderen Entwickler aus dem **Repository**
- Überträgt (*commit*) seine Änderungen in das **Repository**

Diffs und Patches

- Zwischen **Repository** und **Working Copies** werden nur Datei-Differenzen (*diffs*) hin- und hergeschickt
- Differenzen (*patches*) zweier Dateien liefert das Tool diff
  - diff –u Datei1 Datei2 > MeinBugFix
- Anwenden kann man diese mit patch
  - patch –p0 < MeinBugFix

Subversion - SVN

Kommandos für Kommunikation mit dem Server (Linux):

- svn checkout <URL>
- svn update
- svn commit

Offline Kommandos:

- svn add <Datei>
- svn delete <Datei>
- svn stat
- svn diff
- svn rename
- …
- svn help <Kommando>

TortioseSVN (Windows Client) - http://tortoisesvn.net/ und Tutorial

SVN Homepage - http://subversion.tigris.org/ und Tutorial

C++

Was ist C++ ?

- Objektorientierter Aufsatz auf C
- Maschinennahe Programmiersprache
- STL: viele implementierte Datenstrukturen und Algorithmen

Warum C++?

- Bei gutem Code effiziente und schnelle Programme möglich
  → essentiell in der Bioinformatik
- Viel Hilfe durch STL
- C++ macht Spaß :)

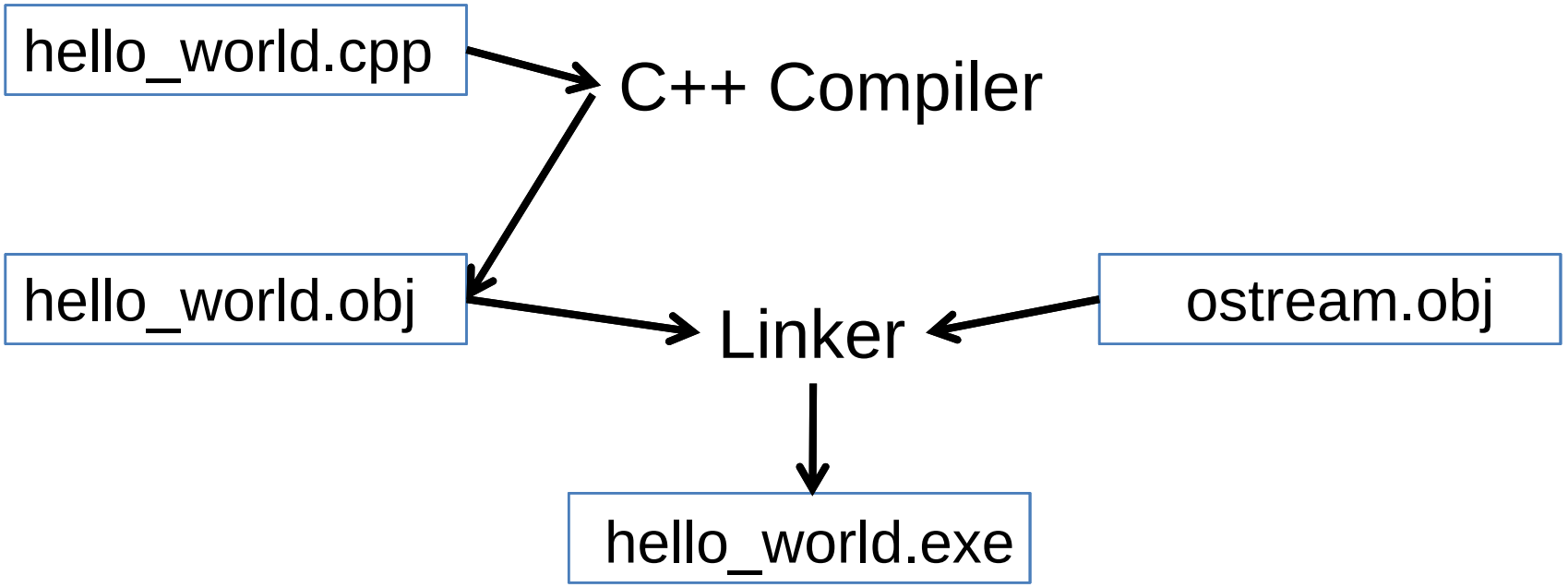
Wie fange ich an?

- vim hello_world.cpp (Linux)
  - Life demo…

- Kompilieren der hello_world.cpp
- Ausführen des Programms hello_world

Wer oder was ist ein Compiler?

Übersetzt Quellcode in Maschinencode oder Objektcode



Linker bindet alle zu einem Programm gehörenden Objekte in ein ausführbares Programm

Ausführen des Programmes

```
knecht:~/aldabi$
knecht:~/aldabi$ vim hello_world.cpp
knecht:~/aldabi$ g++ -pedantic -Wall -ansi -o hello_world hello_world.cpp
knecht:~/aldabi$ ./hello_world
Hello World!
knecht:~/aldabi$
```

Hello World

Was macht unser Programm eigentlich?

```
#include <iostream>

int main() {
   std::cout << ”Hello World!” << std::endl;
   return 0;
}
```

Die STL ist dein Freund und Helfer!

Quiz

Funktioniert dieser Code?

```
#include <iostream>

int main() {
   std::cout << ‘Hello World!’ << std::endl;
   return 0;
}
```

```
      @knecht:~/test$ g++ -pedantic -Wall -ansi -o hello_world hello_world.cpp
hello_world.cpp:6:10: warning: character constant too long for its type
      @knecht:~/test$ ./hello_world
1466264692
      @knecht:~/test$
```

Quiz

Funktioniert dieser Code?

```
#include <iostream>

int main() {
   cout << ”Hello World!” << endl;
   return 0;
}
```

Zeiger (Pointer)

Pointer sind Variablen, deren Inhalt auf andere Variablen verweist.

```
int * pointerAufInt;
double * pointerAufDouble;

int foo = 37; // foo zeigt auf Zahl 37 im
Speicher
int * bar = &foo; // bar zeigt auf foo bzw.
den Speicher, wo foo abgelegt wurde
```

STANDARD TEMPLATE LIBRARY (STL)

Warum die STL benutzen?

- Feld *f* von `int` der Größe *n* erzeugen:
  - malloc/free (klassisches C, stdlib):

```
int *f = (int *)malloc(sizeof(int) * n);
...
free(f);
```

  - new/delete (C++):

```
int *f = new int[n];
...
delete[] f;
```

  - vector (C++, STL):

```
std::vector<int> f(n);
```

Warum die STL benutzen? (II)

- An *f* eine 23 anfügen:
  - malloc/free (klassisches C, stdlib):

```
int *f_new = (int *)realloc(f, sizeof(int)*(n+1));
if (f_new != NULL) f = f_new;
f[n] = 23;
```

  - new/delete (C++):

```
int *f_new = new int[n+1];
for (int i = 0; i < n; ++i)
    f_new[i] = f[i];
delete[] f;
f = f_new;
f[n] = 23;
```

  - vector (C++, STL):

```
f.push_back(23);
```

Warum die STL benutzen? (III)

- Enthält Datenstrukturen (Container), die:
  - die (fehlerprovozierende) Verwaltung des Speichers abnehmen
    - Speicherfreigabe vergessen
    - delete statt delete[] verwendet
    - doppelte Speicherfreigabe
  - auch kompliziertere Klassen speichern können
    - malloc, realloc, free (C) konstruiert/destruiert Objekte nicht
  - **generisch** sind (templates)
    - Elementtyp ist template-Parameter
    - Vergleichsoperator von sortierten Containern lässt sich frei definieren
  - **effizient** implementiert sind
    - Container-Funktionen haben Laufzeitgarantien

Die Standard Library – Überblick

Überblick: Die Header-Dateien

C++ Library Header Files

<algorithm>	<iomanip>	<list>	<ostream>	<streambuf>
<bitset>	<ios>	<locale>	<queue>	<string>
<complex>	<iosfwd>	<map>	<set>	<typeinfo>
<deque>	<iostream>	<memory>	<sstream>	<utility>
<exception>	<istream>	<new>	<stack>	<valarray>
<fstream>	<iterator>	<numeric>	<stdexcept>	<vector>
<functional>	<limits>			

C Library Header Files

<cassert>	<ciso646>	<csetjmp>	<cstdio>	<ctime>
<cctype>	<climits>	<csignal>	<cstdlib>	<cwchar>
<cerrno>	<clocale>	<cstdarg>	<cstring>	<cwctype>
<cfloat>	<cmath>	<cstddef>		

STL = „Standard Template Library“

Programmbibliothek:

Datenstrukturen & Algorithmen.

„Generisch“:

- Formuliert über „Concepts“
- Hochgradige Typisierung (über Template-Argumente)
- Container und Algorithmen sind voneinander „entkoppelt“

Quiz: Wo ist der Unterschied?

Frage: Was unterscheidet folgende Ausdrücke voneinander? (**`obj`** sei ein Containerobjekt)

```
if ( obj.size() == 0 ) ...
```

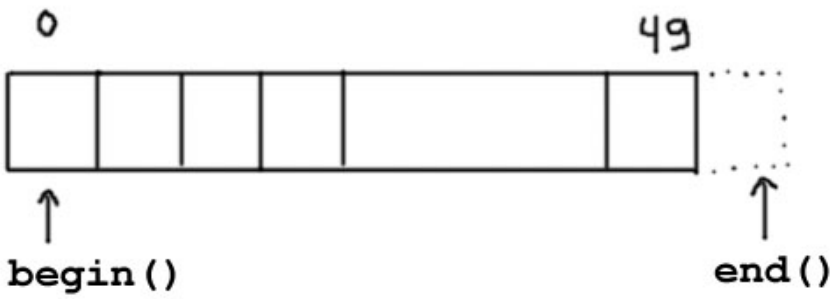
```
if ( obj.empty() ) ...
```

Iteratoren

Iteratoren = verallgemeinerter Pointer

```
vector<char> vec(50);
vector<char>::iterator anfang = vec.begin();
vector<char>::iterator ende = vec.end();

cout << ende - anfang;   //Ausgabe: 50
```



Warum Iteratoren benutzen?

- summiere alle Elemente von Feld *f*

```
int sum = 0;
for (int i = 0; i < n; ++i)
   sum += f[i];
```

- wird vom Compiler genauso übersetzt wie:

```
int sum = 0;
for (int i = 0; i < n; ++i)
   sum += *(f + i);
```

- effizienter ist (ohne Optimierung):

```
int sum = 0;
int *f_end = f + n;
for (int *p = f; p < f_end; ++p)
   sum += *p;
```

Warum Iteratoren benutzen? (II)

- Warum sorgen, wenn der Compiler optimiert?
  - die Optimierung funktioniert nur bis zu einer gewissen Komplexität
- Was passiert, wenn *f* eine verkettete Liste ist?
  - **`f[i]`** hat dann eine Laufzeit von O(i)
  - die ganze Schleife hat dann also $O(n^2)$ Laufzeit

```
std::list<int> f(n);
for (int i = 0; i < n; ++i)
    sum += f[i];
```

- Iteratoren sind optimal für sequentielle Zugriffe
  - Schleife mit Iteratoren hat O(n) Laufzeit

```
std::list<int> f(n);
std::list<int>::iterator i = f.begin();
std::list<int>::iterator i_end = f.end();
for (; i != i_end; ++i)
    sum += *i;
```

Iteratoren (II)

Mit Iteratoren kann man durch Container traversieren.

```
template<typename TIn, typename TOut>
inline TOut copy(TIn first, TIn last, TOut dest)
{
 while ( first != last )
 {
    *dest = *first;
    ++dest; ++first;
 }
 return (dest);
}
```

Quiz: Was läuft hier schief?

Frage: Wieso produziert dieses Programm einen Laufzeitfehler?

```
vector<int> vec1;
vector<int> vec2;

for (int i = 0; i < 10; ++i) vec1.push_back(i);

copy(vec1.begin(), vec1.end(), vec2.begin());
```

Pattern Matching Beispiel

Gegeben langer Text und kürzeres Pattern:

```
text:       This is an awesome tutorial to get to know some C++!
pattern:    some
```

Berechne die Anzahl gleicher Zeichen zwischen Text und Pattern an jeder Stelle des Textes:

```
score:      000.....  ....400...       ..01001..    ....400...
text:       This is an awesome tutorial to get to know some C++!
pattern:    some           some           some           some
             some           some           some           some
              some           some           some           some
               .....          .....          .....          .....
```

Naives Pattern Matching

```
#include <iostream>
#include <string>
#include <vector>

int main() {
     // Initialisierung
     std::string text = "This is an awesome tutorial to get to know some C++!";
     std::string pattern = "some";
     std::vector<int> score(text.length());
     // Computation of the similarities
     // Iteration über den Text (outer loop)
     for (unsigned i = 0; i < (text.length() – pattern.length() + 1); ++i) {
          int localScore = 0;
          // Iteration über das Pattern (inner loop)
          for (unsigned j = 0; j < pattern.length(); ++j) {
               // Zeichenvergleich und Scoreberechnung
               if (text[i + j] == pattern[j])
                    ++localScore;
          }
           // Speichern des localScore im Vector score
          score[i] = localScore;
     }
     // Ergebnisausgabe
     for (unsigned i = 0; i < score.size(); ++i)
          std::cout << score[i];
     return 0;
}
```

Naives Pattern Matching

```
#include <iostream>
#include <string>
#include <vector>

int main() {
    // Initialisierung
    std::string text = "This is an awesome tutorial to
                        get to know some C++!";
    std::string pattern = "some";
    std::vector<int> score(text.length());
    // Computation of the similarities
    …
}
```

Naives Pattern Matching

```
int main() {
    …
    // Computation of the similarities
    // Iteration über den Text (outer loop)
    for (unsigned i = 0; i < (text.length() – pattern.length()
    + 1); ++i) {
        int localScore = 0;
        // Iteration über das Pattern (inner loop)
        for (unsigned j = 0; j < pattern.length(); ++j) {
            // Zeichenvergleich und Scoreberechnung
            if (text[i + j] == pattern[j])
                ++localScore;
        }
         // Speichern des localScore im Vector score
        score[i] = localScore;
    }
```

Naives Pattern Matching

```
int main() {
    …
    // Ergebnisausgabe
    for (unsigned i = 0; i < score.size(); ++i)
        std::cout << score[i];
    return 0;
    …
}
```

Naives Pattern Matching

```
#include <iostream>
#include <string>
#include <vector>

// Iteration über das Pattern (inner loop)
int computeLocalScore(std::string const & text, std::string const & pattern,
                      int const &  textPos) {...}
// Iteration über den Text (outer loop)
std::vector<int> computeScore(std::string const & text, std::string const &
                              pattern){...}
int main() {
    // Initialisierung
    std::string text = "This is an awesome tutorial to get to know some C++!";
    std::string pattern = "some";
     // Computation of the similarities
    std::vector<int> score = computeScore(text, pattern);
    // Ergebnisausgabe
    for (unsigned i = 0; i < score.size(); ++i)
        std::cout << score[i];
    std::cout << std::endl;
    return 0;
}
```