Praktische Informatik

Peter Meerwald,
pmeerwald.lba@fh-salzburg.ac.at

MMT, FH Salzburg
Wintersemester 2012

Vorstellung / Wer bin ich?

- Peter Meerwald
  - pmeerwald.lba@fh-salzburg.ac.at
  - http://pmeerw.net
- Seit Sept. 2011: bct electronic GmbH
  - Embedded Linux Entwicklung, Audio-Signalverarbeitung
- Okt. 2010 bis Aug. 2011: Forschungsaufenthalt (PostDoc)
  - INRIA Rennes / Frankreich: Tardos Fingerprinting Codes
- 2008 bis 2010: Lehrbeauftragter an der Univ. Salzburg und FH Salzburg
  - Programmierung, Algorithmen & Datenstrukturen, Unix
- März 2007 bis Sep. 2010: Doktorat Computerwissenschaften
  - Univ. Salzburg: Watermarking & Multimedia Security
- Sept. 2001 bis März 2007: Softwareentwicklung
  - Sony DADC Austria AG: DRM, Kopierschutz
- Mai 2001: Dipl-Ing. Angewandte Informatik, Univ. Salzburg
- Aug. 1999: Master of Science, Computerwissenschaften
  - Bowling Green State Univ., Ohio, USA

Was behandeln wir in der Lehrveranstaltung?

- Teil 1: Algorithmen
  - Balanced Search Trees (AVL), Heaps
- Teil 2: Low-level stuff
  - CPU, Betriebssystem
- Teil 3: C++ Design
  - Refactoring, Patterns

Ablauf / Aufbau

- Vorlesung
  - Anwesenheitspflicht!
  - Fragen und Mitarbeiten
- Übung
  - Anwesenheitspflicht!
  - 2 Gruppen:
    - Heinz Hofbauer
    - Peter Meerwald
  - Lösung und Vorstellen von Beispielen

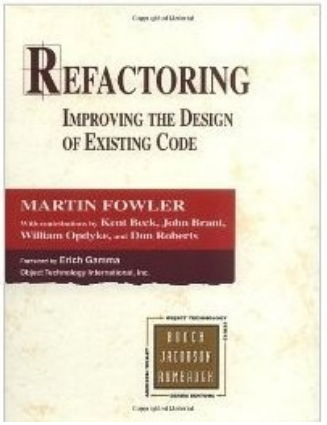
Terminübersicht

- 15. Okt. Einführung, Heap
- 24. Okt. Balanced Search Trees
- 31. Okt. Übung + VL
- 7. Nov. muss ich verschieben
- 14. Nov. C++ Design
- 22. Nov. Übung + VL
- 28. Nov. VL
- 6. Dez. Übung + VL
- 13. Dez. Übung + VL
- 18. Dez. Übung + VL
- 8. Jän. Übung + VL
- 13. Jän. Übung + VL
- 16. Jän. VL
- 24. Jän. Übung + VL
- 1. Feb. Klausur

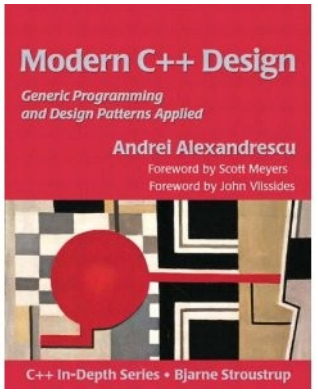
Beurteilung

- Mitarbeit im interaktiven Teil der Vorlesung
- Übung
  - Bereitschaft 66 % der Aufgabenstellungen zu präsentieren
  - Lösung und Präsentation von Übungsaufgaben
- Klausur
- Ohne positive Übung keine positive Gesamtnote; ohne positive Klausur keine positive Gesamtnote.

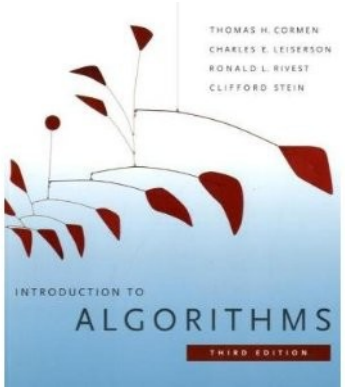
Literatur (für den 1. und 3. Teil)



- Fowler, Refactoring, Addison-Wesley, 1999



- Alexandrescu, Modern C++ Design, Addison-Wesley, 2001



- Cormen, Introduction to Algorithms, MIT Press, 2009

Was brauchen wir?

- Laptop (auch in der Vorlesung)
- C++ - Compiler
- Ubuntu-Installation (12.04 32bit)
  - wenn's damit nicht kompiliert / funktioniert gibt's keine Punkte
- Folien auf https://mediacube.at/wiki/index.php/ILV_Praktische_Informatik_-_WS_2012
- Abgabe in einem git/svn/hg-Repository

Aufgabe #1

- Implementieren Sie einen Zähler, der beliebig oft inkrementiert werden kann! (d.h. ***nicht*** durch den Wertebereich eines primitiven Datentyps beschränkt ist)
  - Wie viel Speicher wird benötigt um bis 2^{100} zu zählen?

Aufgabe #2

- Gegeben sind k sortierte Listen. Schreiben Sie ein Programm, dass daraus ***eine*** sortierte Liste mit allen Elementen erstellt.
  - Welche Komplexität hat ihr Algorithmus?
  - Welche Datenstruktur(en) verwenden Sie?
  - Welche Varianten sind möglich? Vor-/Nachteile?

Lösung #1

```
class Number {
    public:
        Number() : data(0), next(0) {  }

        void incr() {
            if (data == SHRT_MAX) {
                data = 0; // reset current data
                if (!next) // add to list if necessary
                    next = new Number();
                next->incr(); // increment more significant
                              // data recursively
            }
            else data++;
        }

        void print() const {
            print(0); printf("\n");
        }

    private:
        unsigned short data; // primitive data type
        Number *next; // may points to more significant data

        void print(unsigned l) const {
            if (next) next->print(l+1);
            printf("%d ", data);
        }
};
```

Lösung #2 (2)

- Komplexität:

  sei n_i die Anzahl der *i*-ten Listenelemente

  sei $N = \sum_{i=1}^{k} n_i$ die Anzahl der Elemente der Gesamtliste

```
do {
    for (unsigned i = 0; i < k; i++) {
        // inspect k lists for largest element
    }
    // append largest element to merged list
    // runs N times
} while (true);
```

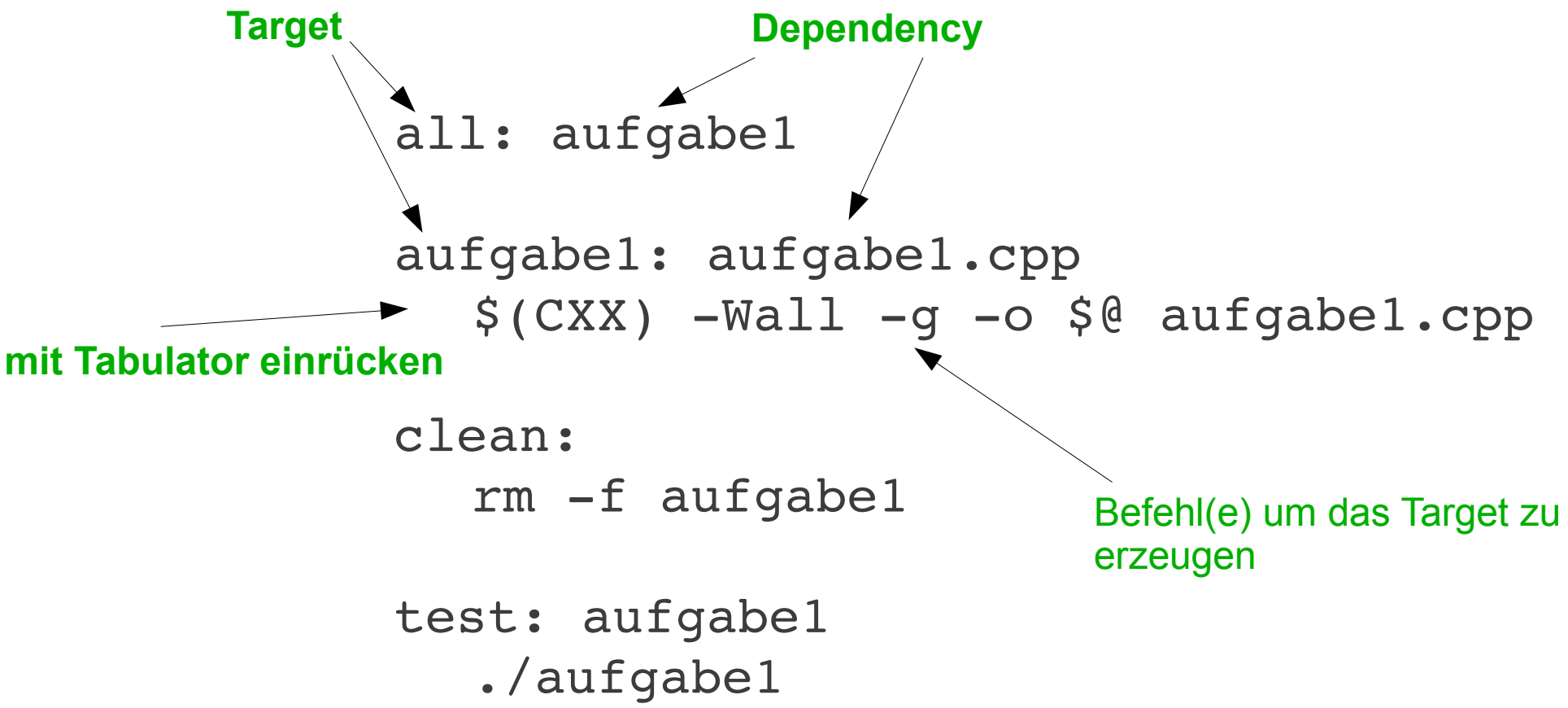
  also O(k N) – Verbesserung möglich?

- Datenstruktur: STL list, erlaubt effizientes Anfügen und Löschen am Anfang
- Varianten?

Hausaufgabe

- Installieren Sie Ubuntu 12.04 32-bit in einer VM.
- Erstellen Sie ein git/svn/hg-Repository und schicken Sie die Zugangsdaten an pmeerwald.lba@fh-salzburg.ac.at.
- Implementieren Sie die mergelist() Funktion und ein Testprogramm und laden Sie beides in Ihr Repo in das Verzeichnis `aufgabe1/`. Testen nicht vergessen!
- Erstellen Sie ein einfaches Makefile (`aufgabe1/Makefile`).
- `make -C aufgabe1` soll Ihr Testprogramm bauen, `make -C aufgabe1 test` soll Ihr Testprogramm ausführen.

Makefile

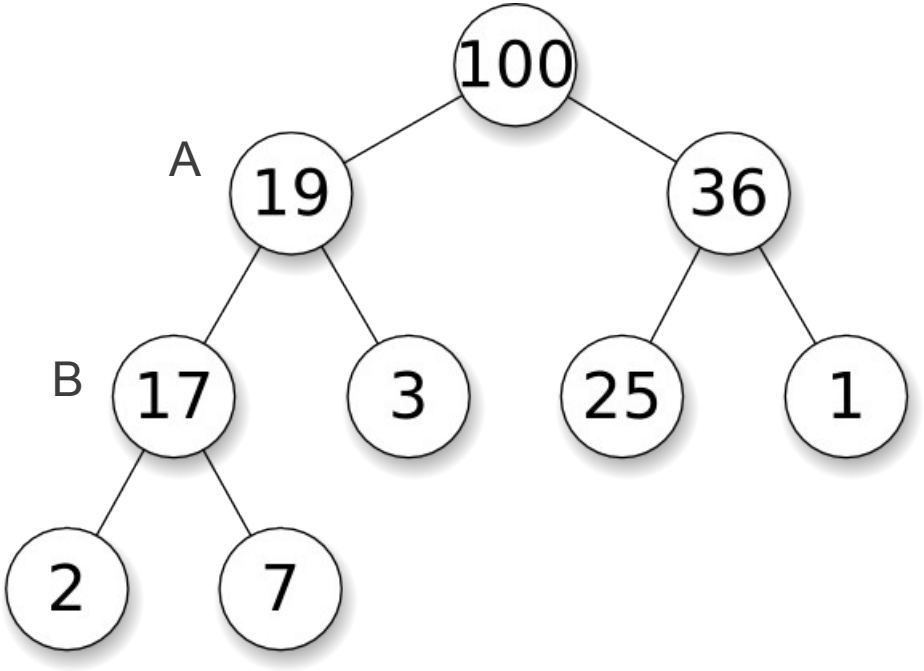

Target
Dependency
all: aufgabe1
aufgabe1: aufgabe1.cpp
$(CXX) -Wall -g -o $@ aufgabe1.cpp
mit Tabulator einrücken
clean:
rm -f aufgabe1
Befehl(e) um das Target zu erzeugen
test: aufgabe1
./aufgabe1


Heap

(Max)-Heap

- Ein Max-Heap ist eine baum-artige Datenstruktur, die folgende Eigenschaft erfüllt:

  Knoten B ist Kind von Knoten A → key(A) >= key(B)



  → Größter Knoten ist ganz oben

- Achtung: ist ***kein*** binärer Suchbaum!

(Min)-Heap

- Ein Min-Heap ist eine baum-artige Datenstruktur, die folgende Eigenschaft erfüllt:

Knoten B ist Kind von Knoten A → key(A) <= key(B)

Eigenschaften / Operationen

- find_max() / top() in O(1)
- delete_max() / pop() in O(log n)
- insert() / push() in O(log n)
- merge() in O(n)