# Programmieren in C++

## SS 2010

### Vorlesung 7, Mittwoch 9. Juni 2010

(Ein- und Ausgabe, Optionen, ASSERT_DEATH)

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 6. Übungsblatt
  - Treffen mit Ihrem Tutor
  - Fragetermine im TF-Pool / in 51-030
- Ein paar nützliche Dinge
  - Wie liest man eine Datei bzw. Eingaben vom User
  - Wie berücksichtig man Optionen in der Kommandozeile
  - Testen auf Programmabsturz mit ASSERT_DEATH
  - Type casting, insbesondere const_cast
  - für das neue Übungsblatt sollen sie ein Grep-artiges Programm schreiben = finde alle Zeilen der Eingabe, die zu einem gegeben Muster passen

# Erfahrungen mit dem 6. Übungsblatt

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Leichter als das Übungsblatt vorher
  - Deutlich weniger Bearbeitungszeit
  - Aufgabe war klarer gestellt diesmal
  - Konsolidierung war gut, alte Sachen nochmal klarer geworden
  - "Je mehr man weiß, umso leichter nimmt man Neues auf"
  - Bitte weiter so

  - Mehr Tests von unserer Seite bzw. von Hudson?
  - Warum keine IDE?
  - Bitte ein Forum
  - Wir lernen nicht besonders viel C++ hier

# Treffen mit Ihrem Tutor

- Sie sollten Ihren Tutor kennenlernen ...
  - ... und ihr Tutor Sie
  - Die Gründe habe ich letztes Mal schon erklärt
  - Ein paar Treffen sind schon zustande gekommen, aber noch nicht sehr viele (war wohl zu kurzfristig)
  - Ihre Tutoren werden es einfach diese Woche nochmal versuchen ... und nächste ... und übernächste, ...
  - Ein oder zwei Treffen pro TeilnehmerIn sind Pflicht, also Ausweichen zwecklos

# Fragetermine im TF Pool / in 51-030

UNI FREIBURG

- Die Lage + eine Frage
  - Am Freitag 16 – 18 Uhr ist kaum einer mehr gekommen
  - Der Termin ist deshalb ab jetzt gestrichen
  - Sonst kommen auch immer sehr wenige, warum?
  - Ist eine andere Uhrzeit gewünscht (am Mo oder Di) ?
  - Wenn ja welche?

# Warum lernen wir nicht mehr C++

UNI FREIBURG

- Antwort 1:
  - Die Niveau-Unterschiede im Auditorium sind sehr groß
- Antwort 2:
  - Sie lernen hier in erster Linie gutes Programmieren für die Praxis, und in zweiter Linie C++
- Antwort 3:
  - Das A und O beim guten Programmieren sind die Basics: guter Stil, gute Struktur + Namen, Dokumentation, Tests, Grundkonstrukte wie const und & und *, usw.
  - Die fortgeschrittenen Sachen machen vielleicht 20% von dem aus was in der Praxis wichtig ist, also werden wir uns in der Vorlesung auch nur 20% damit beschäftigen

# Ein- und Ausgabe

UNI FREIBURG

- Wir machen das erstmal C-style
  - Das heißt mit FILE*, fopen, fclose, fgets, fprintf, usw.
  - Das hat diverse Vorteile, u.a. ist es maschinennäher und effizienter, und wegen Folgendem
  - Die Benutzereingabe heißt in C / Linux standard input oder abgekürzt stdin, und ist vom Prinzip her auch eine Datei (auf Kernel-Ebene sind das alles file descriptors)
  - Die Ausgabe auf den Bildschirm heiß standard output oder abgekürzt stdout, und ist vom Prinzip her auch eine Datei
  - Codebeispiele siehe unsere Grep.cpp
  - Es gibt auch noch die Fehlerausgabe stderr, siehe gleich ...

# Parsen von Optionen

- Oft will man in der Kommandozeile Optionen übergeben, zum Beispiel

  ```
  ./GrepMain --input-file=myfile.txt --case-insensitive-match xyz
  ```

  – Optionen sind auch erstmal ganz normale Argumente

  ```
  argv[0] :  ./GrepMain
  argv[1] :  --input-file=myfile.txt
  argv[2] :  --case-insensitive-match
  argv[3] :  xyz
  ```

  – Da man das sehr oft hat, gibt es für das Parsing eine C-Bibliothek getopt, es reicht dafür am Anfang der Datei

  ```
  #include <getopt.h>
  ```

  – Siehe Codebeispiel in Grep.cpp und Referenzen am Ende

# ASSERT_DEATH

UNI FREIBURG

- Manchmal will man testen ob ein Programm "abstürzt" wenn es soll
  - Z.B. mit exit(1) oder ein assert(...) failed
  - Das macht man mit ASSERT_DEATH

    ASSERT_DEATH(list.parse("1,,2"), "ERROR.*");
  - Das erste Argument ist der Befehl der zu einem non-zero exit oder assert failure führen soll
  - Das zweite Argument ist ein regulärer Ausdruck der zu der Fehlermeldung passen muss die dann kommt
  - Wichtig: Diese Fehlermeldung muss nach standard error abgekürzt stderr geschrieben werden, z.B. mit

    fprintf(stderr, "ERROR: digit expected at position %d\n", i);
  - Siehe Codebeispiel in GrepTest.cpp und Referenzen am Ende

# Type casting

- Implizite Typkonvertierung

```
bool flag = true;
int x = flag;  // Will get the value 1 (false = 0).
```

- Explizite Typkonvertierung
  - manchmal muss man einen Typ in einen anderen Typ konvertieren, der eigentlich imkompatibel ist

```
const char* message = "Hi";
char* argv[2];
argv[1] = message;  // This will not compile.
argv[1] = const_cast<char*>(message);  // This will.
```

  - es gibt auch noch static_cast, dynamic_cast und reinterpret_cast aber die brauchen wir alle noch nicht (wen's interessiert, siehe Referenzen am Ende)

# Literatur / Links

- fopen, flclose, feof, fgets, fprintf, fread, frwrite, ...
  - http://linuxmanpages.com/
  - Oder einfach man 3 fopen etc. in ein Terminal eingeben
  - Codebeispiele siehe die Grep.cpp aus der Vorlesung
- Parsen von Optionen
  - http://linuxmanpages.com/man3/getopt.3.php
  - Oder man 3 getopt
- ASSERT_DEATH
  - http://code.google.com/p/googletest/wiki/GoogleTestAdvancedGuide#Death_Tests
- Type casting, insbesondere const_cast
  - http://www.cplusplus.com/doc/tutorial/typecasting