Programmieren in C++

SS 2019

Vorlesung 8, Dienstag 18. Juni 2019

(I/O, Valgrind, Graphik)

Prof. Dr. Peter Thiemann
Lehrstuhl für Programmiersprachen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem Ü6 — last Wumpus & Graphik
  - Prüfungsanmeldung — Erinnerung + Erklärung
- Inhalt
  - Ein- und Ausgabe — fopen, fgets, stdin, stdout
  - Speicherfehler — valgrind
  - Graphik — Open GL secrets #1
  - **Ü7:** Vorbereitungen für Conways Game of Life

Erfahrungen mit dem Ü6 1/2

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Viele Kommentare zum Wumpus
  - Die Aufgabe war als einfach empfunden. Allerdings können inzwischen viele keinen Wumpus mehr sehen (muss wohl daran liegen, dass es in der Höhle zu dunkel ist :).
  - Probleme mit OpenGL ES 1.1

    Siehe zweiter Teil der Vorlesung heute
  - Einige haben die Zeit über Pfingsten genutzt, um auf den Stand der Vorlesung aufzuschließen.

    👍
  - „erst die Aufgaben gemacht, später die Hinweise gelesen, andersrum wäre es einfacher gewesen…"

Erfahrungen mit dem Ü6 2/2

UNI FREIBURG

- Zusammenfassung / Auszüge
  - „In der Vorlesung wurde nicht gut erklärt.“

    Was können wir verbessern?
  - "Erneut sehr gute Vorlesung."

    
  - "Die Aufgaben wurden nicht präzise gestellt.“

    (Rechtzeitig) im Forum fragen.

    Übrigens wie im richtigen Leben
  - "Sinnvoll und klar gestellte Aufgaben."

Erfahrungen mit dem Ü6 3/2

- Zusammenfassung / Zeitaufwand

3 × 8h 3 × 9h 2 × 10h 3 × 12h 1 × 18h 1 × 40h	

Anmeldung Prüfung

UNI FREIBURG

- Sie müssen sich anmelden!
  - Auch wenn die Veranstaltung "nur" eine Studienleistung ist

    Man kann (und muss) sich anmelden für "11LE13SL-BScINFO-1008 Programmieren in C - Studienleistung"
  - Fakultätsfremde sollen sich bei Problemen an das Prüfungsamt der Technischen Fakultät wenden
  - Für die BOK-Teilnehmenden ist es eine andere Veranstaltung

Ein- und Ausgabe 1/5

UNI FREIBURG

- Beispielprogramm in C

```
#include <stdio.h>

FILE* file = fopen("foo", "r");      // Open "foo" for reading.
const int max = 1000;                // Max. length of a line.
char line[max + 1];                  // +1 for trailing null.
fgets(line, max, file);              // Read until next newline.
if (feof(file)) { … }                // End of file reached
fclose(file);                        // Close the file.
```

  - **Achtung:** beim Lesen einer Zeile muss eine Obergrenze für die Anzahl der Zeichen angegeben werden, damit nicht andere Speicherbereiche überschrieben werden

    Ein "buffer overflow" war die Hauptursache hinter vielen Sicherheitslücken in Software; auch heute noch relevant

Ein- und Ausgabe 2/5

UNI FREIBURG

- Die wichtigsten C-Befehle im Überblick
  - fopen öffnet eine Datei, liefert FILE* zurück
  - fclose schließt die Datei wieder
  - feof sagt, ob das Ende der Datei erreicht it
  - fread liest eine gegebene Anzahl Bytes aus einer Datei
  - fwrite schreibt eine gegebene Anzahl Bytes in eine Datei
  - fprintf schreibt formatiert in eine Datei, analog zu printf
  - fgets liest die nächste Zeile aus einer Datei
  - Details dazu mit man, z.B. man 3 fopen

Ein- und Ausgabe 3/5

UNI FREIBURG

- Ein paar Besonderheiten
  - Wenn fopen fehlschlägt, wird NULL zurückgegeben

    ```
    FILE* file = fopen(...);
    if (file == NULL) { perror("..."); exit(1); }
    ```

    fgets o.ä. mit file == NULL testen gibt einen seg fault
  - Das Ende der Datei wird behandelt wie ein eigenes Zeichen (EOF = end of file)
  - Nach dem Lesen des letzten "richtigen" Zeichens aus einer Datei, ist das Dateiende noch **nicht** erreicht
  - Sondern erst nach dem nächsten Lesezugriff

Ein- und Ausgabe 4/5

UNI FREIBURG

- Benutzereingabe / Bildschirmausgabe
  - Das sind in der Unix/Linux–Welt auch "Dateien" !
  - Die "Datei" für Benutzereingabe heißt **standard input**

    ```
    fgets(line, max, stdin);  // Read line from user input.
    ```
  - Die "Datei" für Bildschirmausgabe heißt **standard output**

    ```
    fprintf(stdout, "Doof\n");  // Write to the console.
    ```
  - Außerdem gibt es noch die Fehlerausgabe **standard error**

    ```
    fprintf(stderr, "Falsch\n");  // Write to standard error
    ```

    Geht normalerweise auf den Bildschirm, umleiten in bash geht zum Beispiel mit ./InputOutputMain 2> error.log

Ein- und Ausgabe 5/5

UNI FREIBURG

- Testen einer Methode mit Eingabedatei
  - **Variante 1:** als Teil vom Test eine Testdatei erzeugen

    Dann am Ende vom Test wieder löschen (mit unlink)
  - **Variante 2:** Testdatei von Hand schreiben

    Integraler Bestandteil vom Test = muss mit ins SVN

  - **Achtung:** in jedem Fall soll die Testdatei klein sein

    Zur Erinnerung: Unit Tests sollen grundsätzlich nur auf kleinen Beispielen laufen und nur wenig Zeit benötigen

Hinweise zum Testen 1/1

UNI FREIBURG

- Füttern einer Datei als Standardeingabe
  - Um eine Datei als Standardeingabe für ein Programm zu nutzen, kann man Folgendes schreiben

    ./InputOutputMain < mycuteinput.txt …
  - Alternativ geht auch:

    cat kawaii.txt | ./InputOutputMain …
  - Um die Standardausgabe in eine Datei umzulenken:

    ./InputOutputMain > sugoi.txt …
  - Um sowohl stdout und stderr umzulenken (bash):

    ./InputOutputMain **2&>1** > out_and_err.txt …

Speicherfehler 1/2

UNI FREIBURG

- valgrind
  - Mit **malloc** und **free** kann folgendes schnell passieren:

    Zugriff auf Speicher, der nicht mit malloc alloziert oder schon wieder mit free freigegeben wurde

    Schwer zu finden, weil sich der Fehler oft nicht an der Stelle äußert, wo er passiert, sondern erst (viel) später

    Speicher, auf den kein Zugriff mehr möglich ist, wurde nicht mit free freigegeben

    Ein sogenanntes **Speicherleck** bzw. memory leak
  - Solche Fehler lassen sich mit **valgrind** erforschen

Speicherfehler 2/2

UNI FREIBURG

- Benutzung von valgrind
  - Einfach vor das ausführbare Programm schreiben, z.B.

    valgrind ./StringMain
  - Das findet allerdings nur Zugriffsfehler auf Speicher, der dynamisch (mit malloc) alloziert wird, auf dem **Heap**

    Alle anderen Variablen liegen auf dem **Stack**
  - Achten Sie auf die LEAK SUMMARY, insbesondere etwa:

    definitely lost: 4 bytes in 1 blocks
  - Details mit valgrind –leak-check=full ./StringMain

Mini-Anwendung

UNI FREIBURG

- Verschlüsselung mit One-Time-Pad (OTP)
  - Ultimative Verschlüsselung, kann nur durch Verrat des Schlüssels geknackt werden (bei geeigneter Wahl)
  - Kennzeichnendes Merkmal ist die *einmalige* Verwendung eines *zufälligen* Schlüssels, der (mindestens) die gleiche Länge wie die zu verschlüsselnde Nachricht aufweist.
  - Chiffrat = Klartext ^ Schlüssel (exklusives Oder)
  - Entschlüsselung

    Chiffrat ^ Schlüssel =

    (Klartext ^ Schlüssel) ^ Schlüssel =

    Klartext ^ (Schlüssel ^ Schlüssel) =

    Klartext

Hinweise zum Übungsblatt

- Grafik mit Open GL

Literatur / Links

- C-style input / output
  - man 3 fopen
  - Dito für fgets, fprintf, feof, fread, fwrite, fclose, …
- OpenGL ES 1.1-Standard
  - The OpenGL Graphics System