Programmieren in C++

SS 2021

Vorlesung 8, Dienstag 15. Juni 2021

(I/O, Queues, Debugger?)

Prof. Dr. Peter Thiemann
Lehrstuhl für Programmiersprachen
Institut für Informatik
Universität Freiburg
Folienvorlage von Prof. Dr. Hannah Bast

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem Ü7 …
  - Prüfungsanmeldung Erinnerung + Erklärung
- Inhalt
  - Ein- und Ausgabe fopen, fgets, stdin, stdout
  - Warteschlange/verkettete Liste struct xy*
- **Ü8:** Arbeiten mit Files und verketteten Strukturen

Erfahrungen mit dem Ü7 1/3

UNI FREIBURG

- 186 Erfahrungen, 184 mit lesbarer Zeit
- Zeitstatistik (in Stunden)

Min	Q1	Med	Q3	Max	Avg	MD	Var
0.0	5.0	6.5	9.5	48.0	7.72	3.02	23.24

-

#	Schlüsselworte
36	schwer|schwierig|anspruch|aufwendig|fordernd|hart|viel zeit|lange gedauert
0	nicht schwer|nicht schwierig|nicht anspruch|nicht aufwendig|unaufwendig|nicht fordernd|nicht hart|nicht viel zeit|nicht lange gedauert
143	cool|nett|spaß|gut|schön|toll|super
1	nicht cool|uncool|nicht nett|keinen spaß|nicht gut|nicht schön|unschön|nicht toll|nicht super
4	unklar|verwirrend

Erfahrungen mit dem Ü7 2/3

UNI FREIBURG

- Zusammenfassung / Auszüge
  - (Zeitaufwand: 48:00) Es war interessant die Aufgabe zu lösen und was neues dazuzulernen was bei der Spielherstellung so abläuft.

    Da hat jemand alle Extra-Features implementiert?
  - (Zeitaufwand: 6:00) Wie ich letzte Woche schon erwartet hatte, war dieses Blatt das spaßigste bisher. Toll fand ich, wie ein großer Teil des Wissens aus den bisherigen Blättern (besonders der umgang mit malloc()) nützlich wurde. Dass man nicht mehr Unittests zu jeder Funktion schreiben musste, sondern jetzt das Spiel das man programmiert hat spielen kann war auch schön und hat meine Beschwerde, dass man keinen Fortschritt erkannte zunichte gemacht. Die Bonusaufgeben hätten meiner Meinung nach auch ruhig verpflichtend gewesen sein können, aber ich freue hoffentlich mich über die Bonuspunkte.

Erfahrungen mit dem Ü7 3/3

UNI FREIBURG

- Zusammenfassung / Auszüge
  - (Zeitaufwand: 9:30) Sich in den zuerst sehr verwirrenden Code rein zu finden war ganz schön kompliziert. Nach einer halben Stunde oder so war mir dann aber halbwegs klar, was was macht.

    Wie im richtigen Leben: Die Entwicklung startet von vorhandenem Code.
  - (Zeitaufwand: 14:00) … Ich habe immer wieder Heap Overflow errors bekommen und die Vektoren haben mir große probleme bereitet…

    Sehr ungewöhnlich. Speicherleck? Sanitizer benutzt?

Anmeldung Prüfung

UNI FREIBURG

- Sie müssen sich bis zum 11.7. anmelden!
  - Auch wenn die Veranstaltung "nur" eine Studienleistung ist

    Man kann (und muss) sich anmelden für "11LE13SL-BScINFO-1006 Programmieren in C++ - Studienleistung"
  - Fakultätsfremde mögen sich bei Problemen an das eigene Prüfungsamt wenden
  - Für die BOK-ler ist es eine andere Veranstaltung
- Worauf warten Sie?

Ein- und Ausgabe 1/5

UNI FREIBURG

- Beispielprogramm in C

```
#include <stdio.h>
#define MAXLEN 1000                 // Max. length of a line.
FILE* file = fopen("foo", "r");     // Open "foo" for reading.
char line[MAXLEN + 1];              // +1 for trailing null.
fgets(line, MAXLEN, 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, "I'm a cybernetic organism…\n"); // Write to the console.
  - Außerdem gibt es noch die Fehlerausgabe **standard error**

    fprintf(stderr, "Panic attack\n"); // Write to standard error

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

    Die Standardausgabe erscheint trotzdem!

Ein- und Ausgabe 5/5

UNI FREIBURG

- Testen einer Methode mit Eingabedatei
  - **Variante 1:** als Teil des Tests eine Datei erzeugen

    Am Ende vom Test wieder löschen (mit unlink).

    Dafür gibt es tearDown()!
  - **Variante 2:** Testdatei von Hand schreiben

    Integraler Bestandteil vom Test = muss mit ins git

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

    Zur Erinnerung: Unit Tests sollen grundsätzlich auf kleinen Beispielen laufen und 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, verwenden wir die Eingabeumleitung:

    ./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 …

Mini-Anwendung #1

UNI FREIBURG

- Dienstprogram cat
  - Man 1 cat: concatenate files and print on the standard output
  - **cat** [*OPTION*]... [*FILE*]...
  - With no FILE, or when FILE is -, read standard input.
  - (Wir ignorieren die Optionen.)

Mini-Anwendung #2

UNI FREIBURG

- Dienstprogram tail
  - Man 1 tail: output the last part of files
  - **tail** [*OPTION*]... [*FILE*]...
  - Print the last 10 lines of each FILE to standard output. With more than one FILE, precede each with a header giving the file name.
  - With no FILE, or when FILE is -, read standard input.
  - (Wir ignorieren die Optionen.)
  - Wir benötigen eine geeignete Datenstruktur um jeweils die 10 zuletzt gelesenen Zeilen aufzubewahren…

Queue 1/8

UNI FREIBURG

- Eine Queue (Warteschlange, FIFO = first-in first-out) hat folgende Operationen
  - Einfügen eines neuen Elements am Ende der Warteschlange
  - Entfernen eines Elements vom Beginn der Warteschlange
- Verhalten: Die Elemente werden in der gleichen Reihenfolge entfernt, in der sie eingefügt worden sind.
- Prinzip first-come first-serve!

Queue 2/8

UNI FREIBURG

- Queue-Operationen in queue.h

```
#include ”stdbool.h”
typedef struct queue queue;
queue *create_queue();                    // NULL if error
bool enqueue(queue *q, void *elem);       // false if error
void *dequeue(queue *q);                  // NULL if empty
bool is_empty(queue *q);
bool free_queue(queue *q);                // false if not mt
```

Queue 3/8

UNI FREIBURG

- Implementierung durch verkettete Liste
  - Datentype für die Knoten der Liste:

```
typedef struct queue_item {
  void *q_item;      /* NOT responsible for freeing */
  struct queue_item *q_next;
} queue_item;
```

  - Wichtig und neu: q_next enthält einen Zeiger auf Listenknoten, d.h. auf den struct-Typ, der gerade definiert wird!
  - C erlaubt die Verwendung eines Zeigertyps auf einen noch nicht fertig definierten struct-Typ.
  - So werden rekursive Datentypen (Listen, Bäume, etc) in C abgebildet.

Queue 4/8

UNI FREIBURG

- Implementierung durch verkettete Liste … und einen Verwaltungsknoten
  - Datentyp für die Queue-Struktur:

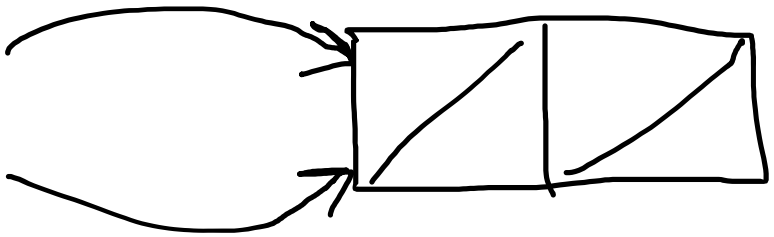
```
typedef struct queue {
  queue_item *q_front;
  queue_item *q_back;
} queue;
```

  - Einfügen durch Anhängen eines neuen Items an q_back
  - Entfernen durch Abhängen von q_front
  - Trick: füge zu Beginn ein leeres sogenanntes Wächteritem (sentinel) ein
  - Test auf leere Schlange: q_front == q_back

Queue 5/8

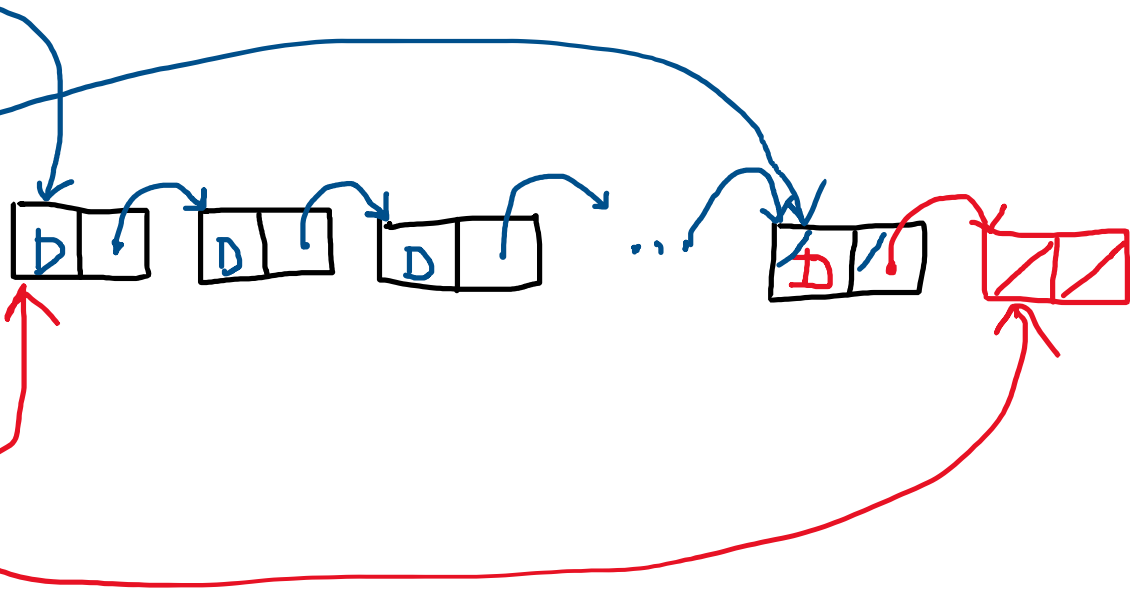
- Start: der Wächterknoten (leer)

- q_front
- q_back



Queue 6/8

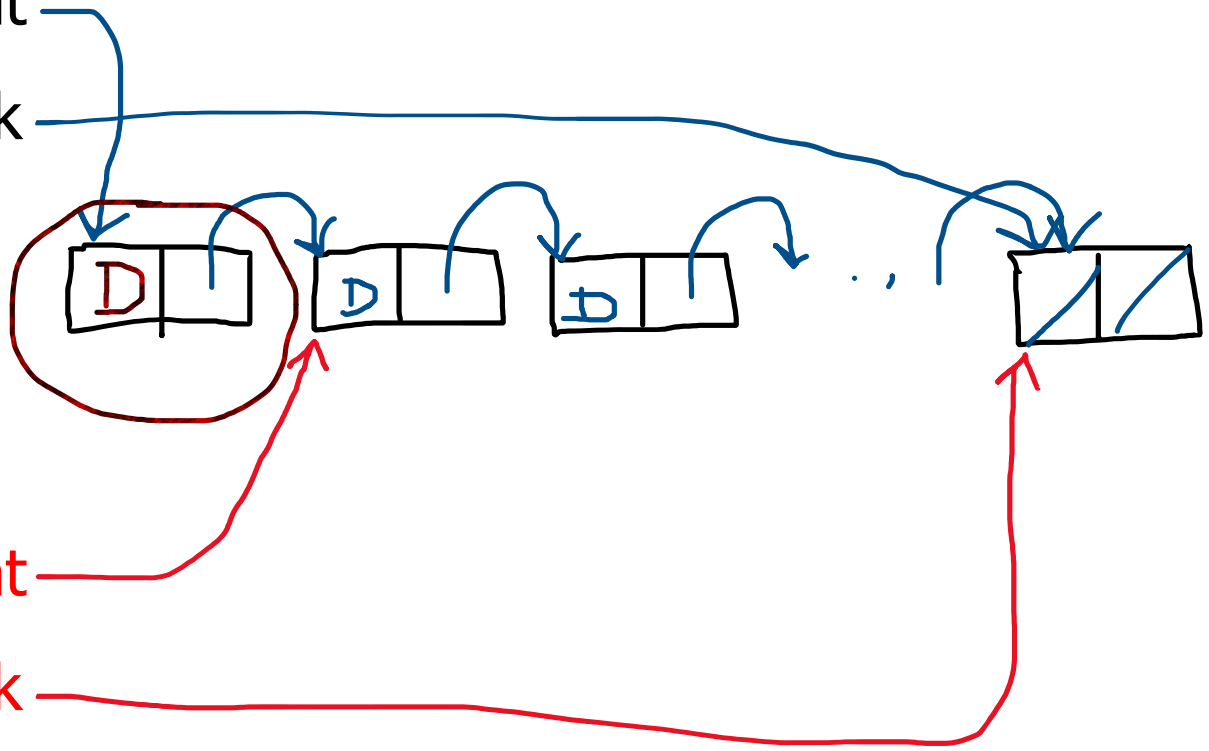
- Einfügen vorher/nachher
- q_front
- q_back



- q_front
- q_back

Queue 7/8

- Entfernen vorher/nachher
- q_front
- q_back



- q_front
- q_back

Queue 8/8

UNI FREIBURG

- Implementierungsidee von tail
  - Einfügen jeder neuen Zeile ans Ende der Warteschlange
  - Sobald 10 Zeilen in der Warteschlange sind, entferne jeweils eine Zeile vom Beginn der Warteschlange
  - Wenn alle Zeilen verarbeitet sind, drucke die maximal 10 Zeilen aus der Warteschlange aus

Literatur / Links

- C-style input / output
  - man 3 fopen
  - Dito für fgets, fprintf, feof, fread, fwrite, fclose, …
- Warteschlangen
  - https://de.wikipedia.org/wiki/First_In_–_First_Out
- Wächter
  - https://de.wikipedia.org/wiki/Sentinel_(Programmierung)