Praktikum zu Informatik 2

Freudenreich, Paulus, Schröder

Algorithmen und Datenstrukturen

Im letzten Praktikumsblock wird der Umgang mit komplexeren Konzepten in C vertieft. Ziel ist es, ein bestehendes Spielprojekt zu analysieren, zu verstehen und eigenständig zu erweitern. Dabei stehen sowohl Algorithmen und Datenstrukturen als auch das systematische Testen des eigenen Codes im Mittelpunkt.

Das Spiel selbst ist einfach: Eine Spielerin bzw. ein Spieler gibt an, wie viele Zufallszahlen angezeigt werden sollen. Anschließend wird eine Zahlenreihe generiert, in der genau eine Zahl doppelt vorkommt. Diese doppelte Zahl muss gefunden und eingegeben werden. Die benötigte Zeit wird dabei gemessen und später in einer Highscore-Liste gespeichert.

Für die Umsetzung werden die folgenden Inhalte aus der Vorlesung praktisch angewandt:

- Binärbaum zur Verwaltung und Überprüfung der erzeugten Zufallszahlen
- Stack (als verkettete Liste) zur Traversierung des Binärbaums
- qsort zur effizienten Erkennung der doppelten Zahl im Array

Da Sie dabei intensiv mit dynamischem Speicher arbeiten (z. B. bei Bäumen und verketteten Listen), ist ein sorgfältiger Umgang mit `malloc` und `free` unerlässlich. Stellen Sie sicher, dass Sie keine Speicherlecks erzeugen. Ebenso ist es Ihre Aufgabe, Unit Tests für zentrale Funktionen diesmal selbst zu entwickeln und damit die Korrektheit Ihres Codes sicherzustellen.

Vorbereitungen

- Machen Sie sich mit der Projektstruktur vertraut:
  - Welche Module gibt es und was machen sie?
  - Welche Funktionen sind bereits vorhanden, welche fehlen?
- Kompilieren Sie das Projekt mit dem mitgelieferten Makefile und testen Sie das (noch unvollständige) Programm, indem Sie das Target `doble_initial` bauen.
  - Hinweis: Hierbei wird das (mitgelieferte) Modul `libdoble_complete.a` verwendet. Dies enthält eine lauffähige Version des Programms zum Testen.

```
make doble_initial           // Bauen des Programms
./doble_initial player_name  // Starten des Programms
```

Verkettete Listen & Stack

Inhalte: verkettete Listen, Pointerarithmetik, dynamische Speicherverwaltung, Stack-Operationen.

Aufgaben:

- Implementieren Sie in `stack.c` die grundlegenden Stackfunktionen:
  - `push`: legt ein Element oben auf den Stack,
  - `pop`: entfernt das oberste Element,
  - `top`: liefert das oberste Element zurück,
  - `clearStack`: gibt den gesamten Speicher frei.
- Definieren Sie in `stack.h` die benötigten Datentypen (z.B. `struct StackNode`).
- Schreiben Sie ein Testprogramm `test_stack.c` mit Unit-Tests, das einige Integer-Werte auf den Stack legt und wieder entfernt.
- Überprüfen Sie durch Code-Review, ob alle mit `malloc` reservierten Speicherbereiche am Ende mit `free` freigegeben werden.

Sortierung

Inhalte: Sortieralgorithmen, Vergleichsfunktionen

Aufgaben:

- Implementieren Sie in `numbers.c` folgende Funktionalitäten:
  - Erzeugen eines Arrays mit der vom Nutzer eingegebenen Anzahl an Zufallszahlen.
  - Sicherstellen, dass beim Befüllen keine Duplikate entstehen.

  Hinweis: Diese Funktionalität soll später mit einem Binärbaum gelöst werden. Solange das Thema in der Vorlesung noch nicht behandelt wurde, dürfen Sie eine Übergangslösung verwenden.
  - Duplizieren eines zufälligen Eintrags im Array.
  - Implementieren Sie `getDuplicate()`: Sortieren des Arrays und Erkennen der doppelten Zahl durch Vergleich benachbarter Elemente.
- Schreiben Sie ein Testprogramm test_numbers.c mit Unit-Tests, die die Funktionen überprüfen.

Binärer Suchbaum & Rekursion

Inhalte: Binärer Suchbaum, rekursive Funktionen

Aufgaben:

- Implementieren Sie in `bintree.c` die zentralen Funktionen:
  - `addToTree`: fügt ein neues Element in den Baum ein (rekursiv),
  - `clearTree`: gibt den gesamten Baum frei (rekursiv),
  - `treeSize`: zählt die Knoten im Baum (rekursiv),
  - `nextTreeData`: Traversierung mit Hilfe des zuvor implementierten Stacks.
- Schreiben Sie ein Testprogramm `test_bintree.c` mit Unit-Tests, die die Funktionen überprüfen.
- Binden Sie den Baum in `numbers.c` ein, um sicherzustellen, dass bei der Generierung von Zufallszahlen keine Dopplungen auftreten.

- Testen Sie Ihr Spiel:
  - Lassen Sie sich eine Zahlentabelle anzeigen,
  - finden Sie die doppelte Zahl,
  - überprüfen Sie, ob Zeitmessung und Highscore-Liste korrekt funktionieren.

Feinschliff

- Stellen Sie sicher, dass das gesamte Projekt:
  - ohne Warnungen kompiliert,
  - den dynamisch reservierten Speicher korrekt wieder frei gibt
  - alle (selbstgeschriebenen) Unit-Tests besteht
- Kompilieren und testen Sie das fertige Programm

```
make doble
./doble player_name
```