Human-Computer Interaction Group

Leibniz Universität Hannover

Übungsblatt 9

Programmieren 1 - WiSe 22/23

Prof. Dr. Michael Rohs, Jan Feuchter, M.Sc., Tim Dünte, M.Sc

Alle Übungen (bis auf die erste) müssen in Zweiergruppen bearbeitet werden. Beide Gruppenmitglieder müssen die Lösung der Zweiergruppe einzeln abgeben. Die Namen beider Gruppenmitglieder müssen sowohl in der PDF Abgabe, als auch als Kommentar in jeglichen Quelltextabgaben genannt werden. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag den 15.12. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2022/Prog1. Die Abgabe muss aus einer einzelnen Zip-Datei bestehen, die den Quellcode, ein PDF für Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen auf.

Hinweis:

Die Dokumentation der prog1lib finden Sie unter der Adresse:
https://postfix.hci.uni-hannover.de/files/prog1lib/

Aufgabe 1: Matrizen

Das Template für diese Aufgabe ist **matrix.c**. In dieser Aufgabe geht es um dynamisch allokierte Matrizen. Eine Matrix wird durch einen Zeiger auf eine struct Matrix repräsentiert. Diese Struktur enthält die Angabe der Anzahl von Zeilen und Spalten, sowie einen Zeiger auf ein **CArray**, das Zeiger auf die Zeilen der Matrix enthält. Jede Zeile der Matrix ist ein C-Array mit Elementen vom Typ double. Verwenden Sie **xmalloc** und **free** zur dynamischen Anforderung bzw. Freigabe von Speicher. Der dynamisch angeforderte Speicher soll vor dem Ende des Programms auch wieder freigegeben werden.

a) Implementieren Sie die Funktion **make_matrix**, die dynamisch eine Matrix mit der entsprechenden Anzahl Zeilen und Spalten erzeugt, die Elemente mit 0 initialisiert und einen Zeiger auf die erzeugte Matrix zurückgibt.

b) Implementieren Sie die Funktion **copy_matrix**. Diese Funktion bekommt einen Zeiger auf ein eindimensionales C-Array mit **n_rows * n_cols** double-Werten übergeben. Es soll nun eine Matrix dynamisch erzeugt werden und die übergebenen double-Werte in diese Matrix kopiert werden. Die Werte sind in der Eingabe zeilenweise angeordnet.

c) Implementieren Sie die Funktion **print_matrix**, die eine Matrix sinnvoll formatiert ausgibt.

d) Implementieren Sie die Funktion **add_matrices**, die zwei Matrizen addiert. Die Eingabematrizen dürfen dabei nicht verändert werden, sondern das Ergebnis soll als neue Matrix dynamisch erzeugt und zurückgegeben werden. Die Funktion soll auch überprüfen, ob die Dimensionen der Argumente kompatibel sind. Geben Sie im Fehlerfall NULL zurück.

e) Implementieren Sie die Funktion **free_matrix**, die eine dynamisch allokierte Matrix freigibt. Wenn allokierter Speicher nicht wieder freigegeben wird, erscheint beim Beenden des Programms folgende Fehlermeldung: **4 bytes allocated in make_matrix (matrix.c at line 123) not freed**

Aufgabe 2: Reversi extended – Zufallsspieler

In dieser Aufgabe soll das Reversi-Spiel weiterentwickelt werden. Diesmal soll ein Computer-Spieler entwickelt werden, der einen zufälligen gültigen Zug macht. Außerdem sollen einige Hilfsfunktionen implementiert werden.

a) Implementieren Sie die Funktion **print_position**, die Ausgaben der Form Buchstabe-Zahl – z.B. "D1" für die Position **(3, 0)** – erzeugt.

b) Auf dem Positions-Stack soll jedes Mal, wenn der Computer am Zug ist, alle in dieser Situation gültigen Züge gespeichert werden. Implementieren Sie die Funktionen **push** und **pop** des Positions-Stacks. Brechen Sie das Programm bei Stack-Überlauf oder Stack-Unterlauf ab.

c) Implementieren Sie die Funktion **random_position**. Diese soll eine zufällige Position vom Stack zurückgeben, ohne den Stack zu verändern. Verwenden Sie **i_rnd(n)**.

d) Implementieren Sie die Funktion **computer_move**. Diese soll für alle Positionen auf dem Spielbrett testen, ob der momentan zu setzende Stein ("my_stone") dort gesetzt werden darf. Alle möglichen Positionen (also alle gültigen Züge) sollen auf dem Stack gespeichert werden. Zuletzt soll eine zufällige gültige Position zurückgegeben werden. Wenn kein gültiger Zug möglich ist, soll **(-1, -1)** zurückgegeben werden.

e) Modifizieren Sie **human_move** so, dass bei Eingabe von **?** in allen Feldern, in denen ein Stein ("my_stone") gesetzt werden darf ein ***** erscheint. Danach ist der Spieler weiterhin am Zug und muss eine gültige Position eingeben. Zur Implementierung bietet es sich an, Hilfsfunktionen zu implementieren.

   Es folgt ein Beispiel:

```
|A|B|C|D|E|F|G|H|
1|_|_|_|_|_|_|_|_|
2|_|_|_|_|_|_|_|_|
3|_|_|_|_|_|_|_|_|
4|_|_|X|X|X|_|_|_|
5|_|_|O|X|O|_|_|_|
6|_|_|_|O|_|_|_|_|
7|_|_|O|_|_|_|_|_|
8|_|_|_|_|_|_|_|_|
Score for O: 0

X's move: ?          ← Spieler gibt ''? ein
|A|B|C|D|E|F|G|H|
1|_|_|_|_|_|_|_|_|
2|_|_|_|_|_|_|_|_|
3|_|_|_|_|_|_|_|_|
4|_|_|X|X|X|_|_|_|
5|_|*|O|X|O|*|_|_| ← Das Spielbrett zeigt die möglichen Züge für ''X
6|_|*|*|O|*|*|_|_|
7|_|_|O|*|_|_|_|_|
8|_|_|_|_|_|_|_|_|
```

Aufgabe 3: (OPTIONAL) Bug Resolving

Die Firma SoLalaSoftware hat für die Sternenflotte ein Konsolenprogramm entwickelt, mit dem die Admiralität ihre Schiffe verwalten kann. Die Firma hatte das günstigste Angebot. Die Admiralität ist leider nicht begeistert von dem Endprodukt. Helfen Sie ihr und finden Sie die Fehler in dem Programm. Sie sollen Bugs finden, die Speicherallokation funktionsfähig machen und logische Fehler im Programm beheben. Die einzelnen Aufgabenteile geben Ihnen Hinweise dazu.

Das Template für diese Aufgabe ist **bug_resolving.c**.

a) Führen Sie das Programm aus. Prüfen Sie ob die Eingaben funktionieren. Eine Eingabe von `q` soll das Programm beenden. Eine Eingabe von `n<index> <name>` soll das Umbenennen eines Raumschiffes ermöglichen (Beispieleingabe: "`n0 Enterprise`"). Das Schiff an Stelle `<index>` soll den Namen `<name>` erhalten. Mit der Eingabe von `s<index>` soll ein Schiff auf Erkundungsmission geschickt werden (Beispieleingabe: "`s1`"). Es wird damit aus der Flotte entfernt. Mit der Eingabe `a <name> <passengers> <reach> <load_capacity>` soll ein Schiff zur Flotte hinzugefügt werden können. Es wird unten an die ausgegebene Tabelle angehangen. Korrigieren Sie erkannte Fehler.

b) Sie haben in der Vorlesung das DRY Prinzip kennen gelernt. Untersuchen Sie die Funktion **Fleet* read_spacecrafts(char* file_name)** auf wiederholende Teile und lagern Sie diese in Hilfsfunktionen aus. Ziel ist es die Funktion verständlicher und übersichtlicher zu machen. Schauen Sie sich auch in der **main** Funktion das Hinzufügen eines neuen Schiffes an. Können Sie Ihre Hilfsfunktionen auch hierfür nutzen?

c) Beschreiben Sie in einem Kommentar, was passiert, wenn Sie ein Schiff entfernen? Ist das Entfernen sinnvoll gelöst? Verbessern Sie das Entfernen eines Schiffes so, dass keine Lücken in dem Array aus Raumschiffen entstehen. Nicht nur die Ausgabe soll gut aussehen, auch im zugrunde liegenden Array sollen keine Lücken sein.

d) Ist es sichergestellt worden, dass nicht mehr Raumschiffe zur Flotte hinzugefügt werden können als die maximale Kapazität `N`? Falls nicht implementieren Sie eine Prüfung und verhindern Sie dies.

e) Beschreiben Sie in einem Kommentar, was passiert, wenn Sie ein Raumschiff an Index 3 entfernen wollen, dort aber kein Raumschiff existiert. Sollte hier vor dem Entfernen geprüft werden, ob ein gültiges Raumschiff an der Position vorhanden ist? Entfernen Sie ein paar Schiffe an nichtexistierenden Indices testweise und fügen Sie Schiffe wieder hinzu. Verhindern Sie ggf. das Entfernen von nicht vorhandenen Raumschiffen, wenn Sie der Meinung sind, dass dies zu Problemen führen kann.

f) Aktivieren Sie die Speicherkontrolle, indem Sie am Anfang der `main` Funktion folgende Zeilen einfügen:

```
report_memory_leaks(true);
```

Finden Sie alle Stellen, an denen allokierter Speicher nicht mehr freigegeben wird und geben Sie entsprechend Speicher frei. Nach dem Beenden des Programms sollen keine Memory Leaks

existieren. Das Programm soll weiterhin funktionsfähig bleiben. Fügen Sie zum Testen, ob Sie alle Memory Leaks gefunden haben erst 2 Schiffe hinzu zu Ihrer Flotte und entfernen Sie dann 3 beliebige Schiffe aus Ihrer Flotte. Benennen Sie die verbleibenden Schiffe um und beenden dann das Programm über die Eingabe von `q`.