Human-Computer Interaction Group

Leibniz Universität Hannover

Übungsblatt 8

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 08.12.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: Pointer auf Strukturen

Wenn Sie einen Pointer auf eine Struktur vorliegen haben, können Sie auf ein Attribut der Struktur mit dem Pfeil-Operator (->) zugreifen.

```
typedef struct test_s {
    int one;
    int two
} Test;
```

Zugriff über Pointer mit Pfeiloperator oder über Derefenzierung und wie bekannt unter Nutzung des Punktoperators:

```
void function_with_pointer(Test * test){
    // Pfeiloperator (Dereferenzierung und Zugriff in einem):
    test->one = 3;
    test->two = 4;

    // Manuelle Dereferenzierung und Zugriff auf Attribute:
    (*test).one = 3;
    (*test).two = 4;

    // Beide Varianten sind äquivalent
}
```

Hinweis: prog1lib

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

Aufgabe 1: Operationen auf Arrays

In dieser Aufgabe sollen Sie verschiedene Operationen für Integer Arrays implementieren. Das Template für diese Aufgabe ist **array_operations.c**. Testfälle für die verschiedenen Aufgabenteile sind bereits vorhanden und können gegebenenfalls ergänzt werden.

a) Implementieren Sie als erstes die Funktion **bool compare()**. Diese soll zwei Arrays `array_a` und `array_b` vergleichen und `true` zurückgeben, wenn der Inhalt beider Arrays identisch ist. Andernfalls soll `false` zurückgegeben werden. Diese Funktion ist relevant für die weiteren Teilaufgaben, da sie auch in den Testfällen der späteren Teilaufgaben verwendet wird. Bearbeiten Sie die weiteren Teilaufgaben erst, wenn Sie sicher sind, dass **compare** funktioniert (Alle Testfälle werden erfüllt).

b) Implementieren Sie die Funktion **int remove_negatives()**, die ein Array übergeben bekommt und in diesem in-place die negativen Zahlen entfernt. Dafür werden die verbleibenden positiven Zahlen an den Anfang des Arrays verschoben und die Anzahl an positiven Elementen als neue Länge zurückgegeben. Sie allokieren also keinen neuen Speicher, sondern verändern das bestehende Array.

c) Machen Sie sich mit der Struktur `BetterArray` vertraut und implementieren Sie danach die Funktion **BetterArray intersect()**. Diese bekommt zwei Arrays übergeben und bildet die Schnittmenge aus beiden. Die Schnittmenge wird auch hier in-place in einem der übergebenen Arrays gespeichert. Wählen Sie dazu das Array aus, dass eine kleinere Länge hat. Sollten beide Arrays gleich lang sein, so überschreiben Sie `array_a`. Geben Sie dann eine entsprechend initialisierte Struktur `BetterArray` mit dem Ergebnis und der Länge des Ergebnisses zurück.

d) Implementieren Sie die Funktion **void merge_sorted_arrays()**, die die beiden sortierten Arrays `array_a` und `array_b` zu einem sortierten Array `result` zusammenfügt.

e) Welche Vorteile bietet die Struktur `BetterArray`? Wie werden Strings in C repräsentiert? Wie ist in C die Länge eines Strings gespeichert bzw. wie wird sie bestimmt? Ist so etwas auch für `int []` möglich? Beantworten Sie diese Fragen als Kommentar in Ihrer Quelltextdatei.

Aufgabe 2: String Einrückung

In dieser Aufgabe sollen eingerückte Strings eingelesen werden und die Einrückung entfernt werden. Einrückung (Indentation) bezeichnet hier die Leerzeichen am Anfang eines Strings. In dieser Aufgabe werden Zeichenketten nicht mehr als Typ `String` betrachtet, sondern als Pointer auf das Zeichen am Anfang der Zeichenkette (`char *`).

Das Template für diese Aufgabe ist `indentation.c`. Testfälle für die verschiedenen Aufgabenteile sind bereits vorhanden und können gegebenenfalls ergänzt werden.

a) Implementieren Sie die Funktion `indentation()`. Diese soll die Anzahl an Leerzeichen (`" "`) bis zum ersten nicht-Leerzeichen des Strings zählen. Befinden sich allerdings Tabs (`"\t"`) vor dem ersten nicht-Leerzeichen, so soll die Funktion den Fehlerwert -1 zurückgeben.

b) Implementieren Sie die Funktion `left_trim()`. Diese soll eine Zeichenkette zurückgeben, die der Eingabe ohne Einrückung entspricht. Erzeugen Sie hier **keine** neue Zeichenkette sondern arbeiten Sie mit Zeigern.

c) Implementieren Sie die Funktion `extract_comment()`. Diese soll innerhalb einer Zeichenkette ein C-artiges Kommentar extrahieren können, also den Text hinter dem ersten Auftreten von `"//"`. Am Ende soll ihre Funktion in der Lage zu sein die Zeile
`"int i = 0; // a new integer"`
abzubilden auf:
`"a new integer"`
Erzeugen Sie auch hier **keine** Kopie der Eingabe und legen Sie **keinen** neuen String an.

Aufgabe 3: Reversi

In dieser Aufgabe geht es darum, das Spiel Reversi zu implementieren, sodass von menschlichen Spielern Züge eingegeben werden können. Reversi ist ein Brettspiel für zwei Spieler, das auf einem Spielbrett mit 8x8 Feldern gespielt wird. Die verwendeten Spielsteine haben zwei unterschiedliche Seiten ('X' für Spieler 1 und 'O' für Spieler 2). Die Grundregeln lauten:

- 'X' beginnt.
- Horizontal, vertikal oder diagonal in einer Reihe liegende Steine einer Farbe werden umgedreht, wenn sie am Anfang und Ende von gegnerischen Steinen eingerahmt sind.
- Ein Zug ist nur dann gültig, wenn er zum Umdrehen mindestens eines gegnerischen Steins führt.
- Wenn keiner der Spieler einen gültigen Zug machen kann, endet das Spiel. Der Spieler mit den meisten Steinen hat gewonnen. Bei Gleichstand endet das Spiel unentschieden.

Details finden sich unter: https://de.wikipedia.org/wiki/Othello_(Spiel)

Hier eine beispielhafte Programmausgabe anhand einiger Züge.

```
|A|B|C|D|E|F|G|H|
1|_|_|_|_|_|_|_|_|
2|_|_|_|_|_|_|_|_|
3|_|_|_|_|_|_|_|_|
4|_|_|_|O|X|_|_|_|
5|_|_|_|X|O|_|_|_|
6|_|_|_|_|_|_|_|_|
7|_|_|_|_|_|_|_|_|
8|_|_|_|_|_|_|_|_|
X's turn: D3 ← Eingabe Spieler X: D3

|A|B|C|D|E|F|G|H|
1|_|_|_|_|_|_|_|_|
2|_|_|_|_|_|_|_|_|
3|_|_|_|X|_|_|_|_|
4|_|_|_|X|X|_|_|_|
5|_|_|_|X|O|_|_|_|
6|_|_|_|_|_|_|_|_|
7|_|_|_|_|_|_|_|_|
8|_|_|_|_|_|_|_|_|
Score for X: 3
O's turn: c5 ← Eingabe Spieler O: C5

|A|B|C|D|E|F|G|H|
1|_|_|_|_|_|_|_|_|
2|_|_|_|_|_|_|_|_|
3|_|_|_|X|_|_|_|_|
4|_|_|_|X|X|_|_|_|
5|_|_|O|O|O|_|_|_|
6|_|_|_|_|_|_|_|_|
7|_|_|_|_|_|_|_|_|
8|_|_|_|_|_|_|_|_|
Score for O: 0
X's turn: b6 ← Eingabe Spieler X: B6

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

Das Template für diese Aufgabe ist **reversi.c**. Zunächst soll nur nach jedem Zug durch einen Spieler das Spielbrett dargestellt werden. Das Programm soll jeweils prüfen, ob der Zug korrekt ist und die betroffenen Steine umdrehen. Das Programm soll in den nächsten Übungen weiter ausgebaut werden.

a) Implementieren Sie die Funktion **Game init_game(char my_stone)**, so dass die Anfangsaufstellung erzeugt und das übergebene Zeichen ('X' oder 'O') als eigener Stein gespeichert wird.

b) Implementieren Sie die Funktion **print_board**, so dass der Spielzustand in der oben gezeigten Form

ausgegeben wird.

c) Implementieren Sie die Funktionen **legal_dir** und **legal**. Diese sollen prüfen, ob ein Stein der Farbe my_stone an Position (x, y) gesetzt werden darf. Dies ist dann der Fall, wenn das Feld noch leer ist, die Position sich in den Grenzen des Spielbretts befindet und in mindestens einer Richtung (direction) gegnerische Steine umgedreht werden können. Die Funktionen **legal_dir** und **legal** sollen den Spielzustand **nicht** ändern.

d) Implementieren Sie die Funktionen **reverse_dir** und **reverse**. Diese sollen einen eigenen Stein (my_stone) auf Position (x,y) setzen und alle dadurch eingerahmten gegnerischen Steine umdrehen. Wenn das Setzen an Position (x,y) kein legaler Zug wäre, soll nicht gesetzt werden.

e) Implementieren Sie die Funktion **count_stones**, die die Anzahl der Steine einer bestimmten Farbe auf dem Spielbrett zählt.