Human-Computer Interaction Group

Leibniz Universität Hannover

Assignment 8

Programmieren 1 - WiSe 25/26

Prof. Dr. Michael Rohs, Falk Stock, M.Sc.

Alle Assignments (bis auf das erste) müssen in Zweiergruppen bearbeitet werden. Ein Gruppenmitglied kann dabei die Lösung der Zweiergruppe gebündelt abgeben. Einzige Ausnahme ist dabei das erste Assignment, wo jedes Gruppenmitglied einzeln abgeben muss. 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 18.12. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2025/Prog1. Die Abgabe muss aus einer einzelnen ZIP-Datei bestehen, die den Quellcode, eine PDF für Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen auf.

Zum Bestehen der Studienleistung müssen Sie mindestens zwei von vier Punkten pro Assignment erzielen. Möchten Sie zusätzlich den Klausurbonus erreichen, müssen Sie über alle Assignments hinweg 75% der Punkte erreichen.

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 Dereferenzierung 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: String-Einrückung (1 Punkt)

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 Pointern.

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 2: Random Sort (1 Punkt)

Implementieren Sie einen Sortieralgorithmus, der durch zufälliges Tauschen von zwei Elementen eines Arrays dieses sortiert. Es sollen Autos sortiert werden, die von einer Konstruktorfunktion erzeugt werden. Das Template für diese Aufgabe ist **random_sort.c**. ergänzt werden.

a) Schreiben Sie eine Funktion **int compare(Car car1, Car car2)**, die zwei Autos vergleicht und 1 zurückgibt, wenn car1 jünger ist als car2 und -1, wennc ar1 älter ist als car2. Wenn beide Autos gleich alt sind, dann soll die Funktion auch die Marke überprüfen und diese lexikographisch sortieren. Sind zwei Autos gleich alt und haben die gleiche Marke, so soll 0 zurückgegeben werden. Nutzen Sie für den Vergleich von zwei Zeichenketten die Funktion **int strcmp(char* str1, char* str2)** aus der Standard-C-Bibliothek. Diese liefert als Rückgabe einen ganzzahligen Wert der kleiner als 0 ist, gleich 0 ist, oder größer als 0 ist, abhängig davon ob str1 lexikographisch kleiner, gleich oder größer ist als str2.

b) Implementieren Sie eine Testfunktion **void compare_test(void)**, mit mindestens 5 **sinnvollen** Testfällen, die die korrekte Funktion Ihrer `compare()`-Funktion überprüft.

c) Implementieren Sie eine Funktion **bool sorted(Car* a, int length)**, die testet, ob das Array sortiert ist. Nutzen Sie die bereits implementierte `compare()`-Funktion. Die Funktion soll `true` zurückgeben, wenn das Array sortiert ist, ansonsten `false`.

d) Implementieren Sie die Funktion **int random_sort(Car* a, int length)**, die solange zwei zufällige Autos in dem Array vertauscht, bis die Liste sortiert ist. Nutzen Sie die Funktion `sorted()` nach jedem Tausch, um zu testen ob, das Array nun sortiert ist. Die Funktion soll zunächst nur 0 zurückgeben.

e) Überprüfen Sie, wie oft Ihre Funktion Elemente vertauscht. Je öfter dies passiert, desto ineffizienter ist die Funktion. Fügen Sie dazu eine Zählvariable `swaps` in Ihre `random_sort()`-Funktion ein, die jedes Mal inkrementiert wird, wenn ein Tausch stattfindet. Nach der Sortierung soll `swaps` zurückgeben werden. Können Sie über diese Variable die Anzahl der Aufrufe der `compare()`-Funktion bestimmen? Wenn ja, wie hängt diese von `swaps` ab? Lassen Sie sich die Anzahl der `swaps` und `compares` ausgeben und beantworten Sie danach diese Frage als Kommentar im Quelltext.

Aufgabe 3: Memory (2 Punkte)

In dieser Aufgabe geht es darum, das Spiel "Memory" zu implementieren, sodass man als Einzelspieler auf der Konsole spielen kann. Es geht darum, Paare gleicher, verdeckt liegender Karten durch Aufdecken von jeweils zwei Karten zu finden. Eine detailliertere Spielbeschreibung findet sich z.B. unter https://de.wikipedia.org/wiki/Memory_(Spiel). In der zu implementierenden Variante soll nur ein einzelner Spieler spielen können. Die Versuche und Punkte sollen vom Computer gezählt werden. Die zugehörige Template-Datei ist **memory_game.c**. Hier eine Beispielausgabe:

```
  1 2 3
1 # # #
2 # # #          ←Anfangszustand mit sechs verdeckten Karten
0 points, 0 turns
> 11             ←Eingabe: Spieler möchte Zeile 1, Spalte 1 aufdecken
  1 2 3
1 B # #          ←Karte in Zeile 1, Spalte 1 aufgedeckt
2 # # #
0 points, 1 turns
> 12             ←Eingabe: Spieler möchte Zeile 1, Spalte 2 aufdecken
  1 2 3
1 B C #          ←zwei aufgedeckte Karten
2 # # #
> 0 points, 2 turns

press <return> to continue ← warten auf Eingabe, da B != C wird wieder verdeckt
```

a) Die Funktion **shuffle** mischt die Karten (Einträge des Arrays `a` der Länge `n`) zufällig. Erläutern Sie die Arbeitsweise von **shuffle**. Angenommen der Ausdruck `rand() % (i + 1)` liefere gleichverteilt zufällige ganze Zahlen im Intervall `[0, i]`. Sind dann alle möglichen Permutationen des Arrays gleich wahrscheinlich?

b) Implementieren Sie die Funktion **init_cards**. Nehmen Sie an, dass das übergebene Array bereits existiert und dass lediglich die Zeichen im Array `cards` so gesetzt werden müssen, dass jedes verwendete Symbol genau zweimal vorkommt und dass die Anordnung zufällig ist. Nicht verwendete Plätze auf dem Spielfeld sollen mit Leerzeichen aufgefüllt werden. Als Symbole sollen die ASCII-Zeichen ab `'A'` verwendet werden. Aufgedeckte Karten werden über ihren den ASCII-Wert (z.B. `'A' = 65`), verdeckte Karten über den negativen ASCII-Wert des Symbols (z.B. `-'A' = -65`) repräsentiert.

c) Implementieren Sie die Funktion **print_board**, die den aktuellen Zustand des Spielfelds ausgibt. Die Ausgaben sollen so aussehen, wie in der obigen Beispielausgabe und wie in den Kommentaren im Quelltext beschrieben. Die Zeichen sind im `cards`-Array zeilenweise von oben nach unten abgelegt.

d) Implementieren Sie die Funktion **int array_index(Board *b, int r, int c)**, die den zu den Koordinaten `(row, column)` gehörigen Array-Index in `cards` berechnet. Die Zeichen sind im cards-Array zeilenweise von oben nach unten abgelegt. Die Koordinaten `(0,0)` entsprechen also Index `0`, die Koordinaten `(0,1)` dem Index `1` und die Koordinaten `(1,0)` dem Index `b->cols`. Das Programm muss mit `exit(1);` beendet werden, wenn ungültige Spalten- oder Zeilenwerte übergeben wurden.

e) Implementieren Sie unter Verwendung von **array_index** die Funktionen **get**, **set** und **turn** zum Lesen, Setzen bzw. Umdrehen einer Karte an der Position (row, column).

f) Implementieren Sie die Funktion **int clamp(int x, int low, int high)**, die den Wert von `x` auf das Intervall `[low, high]` beschränkt.

g) Erklären Sie den Aufbau der Funktion **do_move**. Ist die Funktion robust gegenüber Fehleingaben oder kann das Programm durch bestimmte Eingaben in einen inkonsistenten Zustand gebracht werden? Begründen Sie Ihre Antwort.

Stellen Sie sicher, dass Ihre Lösungen die in der Testfunktion **tests** aufgeführten Testfälle erfüllen.