Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2018/19

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

Übungsblatt 8

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. Wenn Sie für Übungsblatt 1 noch keinen Gruppenpartner haben, geben Sie alleine ab und nutzen Sie das erste Tutorium dazu, mit Hilfe des Tutors einen Partner zu finden. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag den 13.12. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2018/Programmieren1. Die Abgabe muss aus einer einzelnen zip-Datei bestehen, die den Quellcode, ein pdf bei Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen bitte auf.

Die Dokumentation der Prog1lib finden Sie unter: https://hci.uni-hannover.de/files/prog1lib/index.html

Aufgabe 1: Meine erste eigene Postfix REPL

In dieser Aufgabe geht es darum einen kleinen Teil eines Postfix Laufzeitsystems zu implementieren. Dieser soll Ausdrücke wie „1 2 + 3 –" evaluieren können. Ihre Postfix REPL soll am Ende dieser Aufgabe die 4 Grundrechenarten + - * und / mit Ganzzahlen umsetzen können. Das Template für diese Aufgabe ist `postfix_repl.c`. Die zu verwendenden Strukturen sind schon gegeben. Ein StackElement ist dabei von dreierlei Art: Es kann leer sein `tag == EMPTY`, einen Wert enthalten `tag == VALUE` oder eine Operation enthalten `tag == OPERATION`. Die Struktur `Stack` speichert ein Array vom Typ `StackElement` sowie einen Verweis auf den Index des zuletzt hinzugefügten Elements.

a) Implementieren Sie folgende Funktionen: `StackElement make_StackElement(Tag tag)`, `StackElement make_Value(int value)` und `StackElement make_Operation(char operation)`. Diese sollen die verschiedenen Typen von `StackElement` initialisieren und zurückgeben.

b) Implementieren Sie die Funktion `void print_stackElement(StackElement ele)`, die ein `StackElement` sinnvoll auf der Konsole ausgibt. Bspw. [109] für die Zahl 109 oder [+], wenn der Operator + ist oder [ ] für EMPTY.

c) Implementieren Sie die Funktion `void clear_stack(Stack* stack, int n)`, die den Stack mit EMPTY StackElementen initialisiert. Der Parameter `n` gibt die Größe des Stacks an. Der Stackpointer soll auf -1 gesetzt werden, da noch kein Element dem Stack hinzugefügt wurde.

d) Implementieren Sie die Funktion `void print_stack(Stack* stack, int n)`, die einen Stack auf der Konsole ausgibt. Der Parameter n soll dabei angeben, wie viele Element ausgegeben werden. Bspw. sollte Folgendes nach dem Befüllen des Stacks mit den Werten 1, 2 und 3 ausgegeben werden:

[3]<- auf dieses Element zeigt der Stackpointer

[2]

[1] <- unterstes Element

e) Schreiben Sie als Kommentar, was die Methoden `pop` und `push` leisten. Welche Parameter der Funktionen sind Eingabe- und welche Ausgabeparameter?

f) Implementieren Sie die Funktion `void compute(Stack* stack, int n)`, die überprüft, ob oben auf dem Stack ein Element vom Typ `OPERATION` liegt und dieses ausführt. Bspw.:

[+]<- auf dieses Element zeigt der Stackpointer

[2]

[1] <- unterstes Element

Sollte zu nachfolgendem evaluieren.

[3] <- unterstes Element, Stackpointer zeigt auf dieses Element.

Wenn die Stackhöhe zu klein ist, also zu wenige Operanden zur Verfügung stehen, dann soll `compute` den Stack nicht verändern und eine aussagekräftige Fehlermeldung ausgeben.

g) Implementieren Sie in der main() Methode das Einlesen und Auswerten der Eingabe. Sie können die Hilfsfunktionen `is_operation(char c)` und `is_digit(char c)` nutzen. Erstellen Sie beim Einlesen einer ganzen Zahl ein neues `StackElement` und fügen Sie es dem gegebenen Stack hinzu. Beim Einlesen einer Operation soll diese als erstes auf den Stack gelegt werden und dann mithilfe der `compute` Funktion ausgewertet werden. Nach jeder Änderung des Stacks soll dieser auf der Konsole ausgeben werden. Hinweis: Sie müssen keine Fehlerbehandlung implementieren. Gehen Sie davon aus, dass nur ganze Zahlen oder die vier Operatoren getrennt von Leerzeichen eingegeben werden können.

h) OPTIONAL: Erweitern Sie Ihre REPL um beliebige Features. Bspw. Support von Doubles.

Aufgabe 2: Sortieren durch zufälliges Vertauschen

Implementieren Sie aufbauend auf dem Template `random_sort.c` einen Sortieralgorithmus, der durch zufälliges Tauschen von zwei Elementen eines Arrays dieses sortiert. Es sollen Autos sortiert werden, die von einer Konstruktorfunktion erzeugt 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, wenn car1 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)`. Diese liefert -1, 0 oder 1 zurück, abhängig davon ob `str1` lexikographisch kleiner, gleich oder größer ist als `str2`.

b) Schreiben Sie eine Testfunktion `void compare_test(void)`, mit mindestens 5 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, später (e) soll die Anzahl der Vertauschungen zurückgegeben werden.

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?

f) Testen Sie Ihren Algorithmus für mindestens 5 verschieden große zufällige Arrays jeweils 100-mal (Arraygrößen im Bereich von 3 – 10). Geben Sie jeweils den Durchschnittswert von `swaps` für die verschiedenen Arraylängen auf der Konsole aus, sowie die Anzahl der stattgefundenen Aufrufe von `compare` (falls möglich). Ein Array mit beliebiger Länge und zufälligen Autos können Sie mit der `create_car_park(int car_count)` Methode erstellen.

Aufgabe 3: Reversi

In dieser Aufgabe geht es darum, das Spiel Reversi zu implementieren, so dass 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 ein eine beispielhafte Programmausgabe an Hand 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 (`dir`ection) 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.

Hinweise zum Editieren, Compilieren und Ausführen:

- mit Texteditor `file.c` editieren und speichern
- `make file` ← ausführbares Programm erstellen
- `./file` ← Programm starten (evtl. ohne `./`)
- Die letzten beiden Schritte lassen sich auf der Kommandozeile kombinieren zu: `make file && ./file`