Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2018/19

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

Übungsblatt 9

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 20.12. 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: Operatoren kombinieren

In dieser Aufgabe soll ein Programm implementiert werden, das gegebene Zahlen mit Hilfe der Grundrechenarten zu einer Ergebniszahl verknüpft. Zum Beispiel können die Zahlen [2, 3, 7, 8] mit den binäre Operatoren { +, -, *, / } zur Zahl 9 verknüpft werden durch den Ausdruck 2 * 7 + 3 – 8 (Infix-Notation) bzw. 2 7 * 3 + 8 – (Postfix-Notation). Alle gegebenen Zahlen müssen genau einmal verwendet werden und dürfen beliebig angeordnet werden. Es dürfen beliebige Operatoren aus der Menge { +, -, *, / } verwendet werden. Jeder Operator darf beliebig oft und an beliebiger Position eingesetzt werden. Für n Zahlen werden n-1 binäre Operatoren benötigt, um die Zahlen zu einem Ergebnis zu verknüpfen.

Bei vorgegebenen Zahlen, z.B. [2, 3, 7, 8], und vorgegebenem Ergebnis, z.B. 9, soll Ihr Programm eine mögliche Anordnung der Zahlen z_0 z_1 … z_n und eine mögliche Auswahl von Operatoren op_0 op_1 … op_{n-1} finden, so dass die Auswertung von z_0 z_1 op_0 z_2 op_1 … z_{n-1} op_{n-2} (Postfix-Notation) zum gewünschten Ergebnis führt, sofern das möglich ist. Wenn das nicht möglich ist, ist es erlaubt, dass Ihr Programm nicht terminiert.

Das Template für diese Aufgabe ist `combine_ops.c`. Implementieren Sie folgende Teilaufgaben:

a) Implementieren Sie die Funktion `arrays_equal`, die genau dann wahr zurückgibt, wenn die Elemente der beiden übergebenen double-Arrays `a` und `b` paarweise ungefähr gleich sind. Beide Arrays haben die Länge `n`. Verwenden Sie zum Prüfen der ungefähren Gleichheit der Elemente die Funktion `approx`. Testen Sie die implementierte Funktion mit mindestens zwei weiteren Testfällen.

b) Implementieren Sie die Funktion `array_copy`, die eine dynamisch allokierte Kopie des Arrays a erstellt. Das Array `a` hat `n` Elemente. Verwenden Sie `xcalloc` zum Reservieren

des benötigten Speichers. Testen Sie die implementierte Funktion mit mindestens zwei weiteren Testfällen.

c) Implementieren Sie die Funktion `shuffle`, die die Elemente eines double-Arrays `a` mit `n` Elementen zufällig durcheinanderwürfelt. Das Eingabearray a wird dabei verändert. Jede mögliche Permutation soll die gleiche Auftretenswahrscheinlichkeit haben. Implementieren Sie dazu folgenden Algorithmus:

   1. vertausche das letzte Element (Index n-1) mit einem zufällig gewählten Element aus Indexintervall [0, n-1]
   2. vertausche das vorletzte Element (Index n-2) mit einem zufällig gewählten Element aus Indexintervall [0, n-2]
   3. vertausche das Element an Index n-3 mit einem zufällig gewählten Element aus Indexintervall [0, n-3]

      usw.

   n. vertausche das Element an Index 1 mit einem zufällig gewählten Element aus Indexintervall [0, 1]

d) Implementieren Sie die Funktion `random_ops`, die ein Array `a` der Länge `n` mit zufälligen Operatoren der Grundrechenarten füllt. Jeder Operator darf beliebig oft vorkommen.

Hinweis: Mit printdaln(double* a, int n) kann ein double-Array ausgegeben werden.
Mit printcaln(char* a, int n) kann ein char-Array ausgegeben werden.
Mit printialn(int* a, int n) kann ein int-Array ausgegeben werden.

Hinweis: Die Funktion `i_rnd(int i)` liefert eine zufällige ganze Zahl aus dem Intervall [0,i[mit Intervallgrenze 0 inklusiv und Intervallgrenze i exklusiv, wobei jede ganze Zahl aus dem Intervall gleich wahrscheinlich ist.

Aufgabe 2: Operatoren kombinieren (Fortsetzung)

In dieser Aufgabe wird Aufgabe 1 fortgesetzt. Das Template für diese Aufgabe ist entsprechend ebenfalls `combine_ops.c`.

a) Stellen Sie `ArrayCount* array_counts` grafisch dar, so dass erkennbar ist, dass es sich um ein Array von Strukturen handelt, die jeweils auf Arrays verweisen. In den Vorlesungsfolien finden sich Beispiele dafür, wie eine solche Darstellung aussehen kann.

b) Implementieren Sie die Funktion `update_array_counts`, die für das Array `a` der Länge `n` die Zählerstände in `array_counts` der Länge m erhöht. Schauen Sie sich dazu an, wie die Funktion in `shuffle_frequency` verwendet wird. Falls die Permutation a noch nicht vorhanden ist, muss ein noch nicht genutzter Eintrag in `array_counts` gesetzt werden. Es gilt m = n! (n Fakultät), da es für n Elemente n! Anordnungen gibt.

   Hinweis: Speichern Sie in `array_counts` eine Kopie von a.

c) Rufen Sie die `shuffle_frequency` Funktion in der `main`-Funktion auf und prüfen Sie, ob die `shuffle`-Funktion alle Permutationen in etwa gleich häufig generiert. Beschreiben und begründen Sie, ob das Ihrer Einschätzung nach der Fall ist.

d) Implementieren Sie die Funktion `evaluate`, die n Zahlen und n-1 Operatoren übergeben bekommt und folgenden Postfix-Ausdruck auswertet:

numbers[0] numbers[1] ops[0] numbers[2] ops[1] ... numbers[n-1] ops[n-2]

e) Implementieren Sie die Funktion `print_solution`, die ein Array mit n double Werten und ein gewünschtes Resultat übergeben bekommt. Die Funktion soll durch zufälliges Vertauschen der Anordnung der Zahlen und durch zufällige Wahl der Operatoren +, -, *, / eine solche Anordnung der Zahlen und Wahl der Operatoren finden, dass die Auswertung (approx.) den Wert `result` ergibt, sofern das möglich ist. Geben Sie auch aus, wie viele Durchläufe benötigt wurden, um eine Lösung zu finden. Nutzen Sie die Funktionen, die Sie in den vorherigen Teilaufgaben implementiert haben.

Aufgabe 3: Reversi Zufallsspieler

In dieser Aufgabe soll das Reversi-Spiel weiterentwickelt werden. Diesmal soll ein Computer-Spieler entwickelt werden, der einen zufälligen gültigen Zug mach. 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: ?
  |A|B|C|D|E|F|G|H|
1|_|_|_|_|_|_|_|_|
2|_|_|_|_|_|_|_|_|
3|_|_|_|_|_|_|_|_|
4|_|_|X|X|X|_|_|_|
5|_|*|O|X|O|*|_|_|
6|_|*|*|O|*|*|_|_|
7|_|_|O|*|_|_|_|_|
8|_|_|_|_|_|_|_|_|
```

← Spieler gibt '?' ein

← Das Spielbrett zeigt die möglichen Züge für 'X'

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`