Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2020/21

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 17.12. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2020/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: Zeichenketten Find & Replace

Das Template für diese Aufgabe ist `string_replace.c`. In dieser Aufgabe sollen Sie eine Funktion schreiben, die in einer Zeichenkette `s` nach einer Zeichenkette `find` sucht und dann alle Vorkommen von `find` in `s` durch eine dritte Zeichenkette `replace_by` ersetzt. Beispielsweise sollte die Zeichenkette `"hello, hello world"` nach `"hello"` durchsucht werden und alle Vorkommen durch `"bye"` ersetzt werden: `"bye, bye world"`. Alle anderen Zeichen sollen übernommen werden. Dabei soll die ursprüngliche Zeichenkette erhalten bleiben und **immer** eine neue dynamisch allokierte Zeichenkette erstellt werden. Die Teilaufgaben führen Sie durch die Implementation.

Hinweis: Für die Bearbeitung der nachfolgenden Teilaufgaben dürfen folgende Funktionen **nicht** verwendet werden: xcalloc, calloc, alle String Funktionen aus der Programmieren 1 Bibliothek bis auf s_length(), alle Funktion aus der Standard C Bibliothek für Strings bis auf strlen. Nutzen Sie stattdessen, xmalloc, den Zugriff auf Zeichen über den Index in einer Zeichenkette bspw. s[i] oder über Pointerarithmetik *(s+i).

a) Schauen Sie sich die Testfälle in `void test_next_occurence()` an. Implementieren Sie die Funktion `char* next_occurence(char* s, char* find)`. Diese soll in dem String `s` nach der Zeichenkette `find` suchen und einen Pointer auf das erste Zeichen von s zurückgeben, an dem die Zeichenkette `find` in `s` zu finden ist. Sollte `find` in `s` nicht vorkommen soll `NULL` zurückgeben werden. Bsp.: In der Zeichenkette `"hello world"` ist `"ll"` an Index 2 zu finden, daher würde die Funktion einen Pointer auf das erste `l` zurückgeben bzw. auf die Zeichenkette `"llo world"`.

b) Schauen Sie sich die Testfälle in `void test_count()` an. Implementieren Sie die Funktion `int count(char* s, char* find)`. Diese soll zählen, wie oft die Zeichenkette `find` in `s` vorkommt und dies zurückgeben. Nutzen Sie die Funktion `next_occurence` aus Aufgabenteil a).

c) Schauen Sie sich die Testfälle in `void test_replace()` an. Implementieren Sie den ersten Teil der Funktion `char* replace(char* s, char* find, char* replace_by)`. Implementieren Sie als erstes die dynamische Allokation des Speichers für die Zeichenkette, die Sie zurückgeben möchten. Überlegen Sie sich dazu, wie viel Speicher Sie benötigen und allokieren Sie nur so viel Speicher wie notwendig ist. Nutzen Sie dafür auch die Funktion `count` aus Aufgabenteil b). Beispiel (s.o.): `"hello, hello world"` belegt 19 Bytes an Speicher `"bye, bye world"` nur noch 15 Bytes.

d) Implementieren Sie den zweiten Teil der Funktion `char* replace(char* s, char* find, char* replace_by)`, sodass in die Zeichenkette, die zurück gegeben wird, die Zeichen aus s bzw. aus replace_by eingefügt werden. Eine Vorgehensweise könnte sein, solange alle Zeichen zu kopieren, bis das erste Auftreten von `find` in `s` auftritt, dann fügen Sie stattdessen `replace_by` ein und kopieren dann weiter aus `s` bis Sie wieder auf `find` treffen. Nutzen Sie die Funktion `next_occurence` aus Aufgabenteil a).

Aufgabe 2: 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 C-Array, das Zeiger auf die Zeilen der Matrix enthält. Jede Zeile der Matrix ist ein C-Array mit Elementen vom Typ double.

Die Zeilen und Spalten der Matrix sollen dynamisch allokiert werden. Verwenden Sie `xcalloc/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 `new_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. Sichern Sie die Funktion mit mindestens einer sinnvollen Precondition ab.

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. Sichern Sie die Funktion mit mindestens einer sinnvollen Precondition ab.

c) Implementieren Sie die Funktion `print_matrix`, die eine Matrix sinnvoll formatiert ausgibt. Sichern Sie die Funktion mit mindestens einer sinnvollen Precondition ab.

d) Implementieren Sie die Funktion `transpose_matrix`, die als Eingabe einen Zeiger auf eine Matrix erhält und einen Zeiger auf eine neue transponierte Matrix zurückgibt. Die Eingabematrix darf nicht verändert werden. Sichern Sie die Funktion mit mindestens einer sinnvollen Precondition ab. Erstellen Sie mindestens eine sinnvolle Postcondition.

e) Implementieren Sie die Funktion `mul_matrices`, die zwei Matrizen multipliziert. 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. Sichern Sie die

Funktion mit entsprechenden Preconditions ab. Erstellen Sie mindestens eine sinnvolle Postcondition.

f) 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 new_matrix (matrix.c at line 123) not freed`

Sichern Sie die Funktion mit mindestens einer sinnvollen Precondition ab.

Aufgabe 3: Mensa

Das Template für diese Aufgabe ist mensa.c. In dieser Aufgabe soll eine interaktive Mensasimulation geschrieben werden. Die Mensa bietet auf der Tageskarte fünf verschiedene Gerichte an (`const String menu[]`). Bevor die ersten Studierenden kommen, kocht die Küche bereits fünf zufällige Gerichte von der Tageskarte. Jedes Mal, wenn ein Gericht an einen Studierenden ausgegeben wurde, kocht die Küche ein weiteres zufälliges Gericht. Die fertigen Gerichte sind in der String-Liste `food` gespeichert. Es stehen zunächst 3 Studierende in der Schlange und warten auf Essen. Die Studierenden sind in der String-Liste `students` durch ihre Essenswünsche repräsentiert. Der Benutzer der Simulation spielt die Rolle einer bzw. eines Mensa-Bediensteten und hat das Ziel, durch Ausgabe der gewünschten Gerichte die Reputation der Mensa zu maximieren. Wenn ein Essenswunsch erfüllt wurde, steigt die Reputation um eins. Jedes Mal, wenn die Reputation der Mensa steigt, kommt ein weiterer Studierender dazu. Wenn ein falsches Gericht ausgegeben wurde, sinkt die Reputation um eins und das Essen wird zurückgenommen. Wenn ein Essenswunsch nicht erfüllt werden kann, weil das entsprechende Gericht gerade nicht fertig ist, sinkt die Reputation der Mensa um 2. Die/der Studierende verlässt daraufhin die Mensa ohne gegessen zu haben. Hier ein möglicher Simulationsablauf:

```
fertige Essen: [Salat, Vegi, Vegi, Spaghetti, Eintopf]
nächster Essenswunsch: Spaghetti (3 hungrige Studierende warten)
Reputation der Mensa: 0
> 3 (← Mitarbeiter/in nimmt fertiges Essen an Position 3, Spaghetti entspricht dem Wunsch)
Vielen Dank! Ich liebe die Mensa!
fertige Essen: [Salat, Vegi, Vegi, Eintopf, Eintopf]
nächster Essenswunsch: Spaghetti (3 hungrige Studierende warten)
Reputation der Mensa: 1
> -1 (← Mitarbeiter/in antwortet „haben wir gerade nicht“)
Spaghetti ist nicht da? Schade.
fertige Essen: [Salat, Vegi, Vegi, Eintopf, Eintopf]
nächster Essenswunsch: Salat (2 hungrige Studierende warten)
Reputation der Mensa: -1
> 0 (← Mitarbeiter/in nimmt fertiges Essen an Position 0, Salat entspricht dem Wunsch)
Vielen Dank! Ich liebe die Mensa!
fertige Essen: [Vegi, Vegi, Eintopf, Eintopf, Eintopf]
nächster Essenswunsch: Eintopf (2 hungrige Studierende warten)
Reputation der Mensa: 0
> 1 (← Mitarbeiter/in versucht fälschlicherweise Vegi statt Eintopf auszugeben)
Vegi möchte ich nicht! Ich möchte Eintopf!
fertige Essen: [Vegi, Vegi, Eintopf, Eintopf, Eintopf]
nächster Essenswunsch: Currywurst (1 hungrige Studierende warten)
Reputation der Mensa: -1
> 1 (← Mitarbeiter/in versucht fälschlicherweise Vegi statt Currywurst auszugeben)
Vegi möchte ich nicht! Ich möchte Currywurst!
Fertig für heute. Die Mensa schließt.
Finale Reputation der Mensa: -2
```

a) Implementieren Sie die Funktion `length_list`, die die Anzahl der Elemente der Liste zurückgibt.

b) Implementieren Sie die Funktion `get_list`, die das Listenelement an der Indexposition zurückgibt. Das erste Listenelement befindet sich an Indexposition 0.

c) Implementieren Sie die Funktion `free_list`, die eine List mitsamt Inhalt freigibt. Beachten Sie, dass die Liste Besitzer („owner") der Listenelemente (`value`) ist und alle Listenelemente dynamisch allokiert sind. Diese müssen also auch freigegeben werden.

d) Implementieren Sie die Funktion `append_list`, die ein neues Listenelement hinten an die Liste anfügt.

e) Implementieren Sie die Funktion `print_situation`, die den aktuellen Zustand der Simulation wie im obigen Beispiel ausgibt. Beispielausgabe:

```
fertige Essen: [Spaghetti, Vegi, Salat, Salat, Vegi]
nächster Essenswunsch: Currywurst (3 hungrige Studierende warten)
Reputation der Mensa: -1
```

f) Implementieren Sie die Funktion `finish`, die eine abschließende Meldung ausgibt, allen dynamisch allokierten Speicher freigibt und das Programm beendet. Achten Sie darauf, dass keine Speicherlecks auftreten. Beispielausgabe:

```
Fertig für heute. Die Mensa schließt.
Finale Reputation der Mensa: 3
```

g) Implementieren Sie die Funktion `run_mensa`, die es erlaubt, interaktiv gewünschte Essen auszugeben. Es gibt nur eine Essensausgabe vor der die Studierenden in einer Schlange warten und nacheinander bedient werden. Dazu wird der Index des gewünschten Gerichts in der Liste der fertiggestellten Essen eingegeben. Falls ein Essenswunsch nicht erfüllt werden kann, soll -1 eingegeben werden. Die Eingabe von -2 beendet das Programm. Implementieren Sie die Funktion entsprechend dem oben gezeigten Beispiel.

Hinweise:

- Lesen Sie eine ganze Zahl von der Standardeingabe mit `int i_input(void);`

Erzeugen Sie eine Zufallszahl mit `int i_rnd(int n); // 0, 1, …, n-1`