Human-Computer Interaction Group

Leibniz Universität Hannover

Übungsblatt 12

Programmieren 1 – WiSe 21/22

Prof. Dr. Michael Rohs, Jan Wolff, 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 20.01.um 23:59 Uhr über *https://assignments.hci.uni-hannover.de/WiSe2021/Programmieren1*. 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: prog1lib

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

Aufgabe 1: Text-Statistik in C

In dieser Aufgabe geht es um die Entwicklung eines Programms zur Erstellung von Text Statistiken. Eine vergleichbare Aufgabe haben Sie bereits in PostFix bearbeitet. Die Statistik eines Texts enthält folgende Komponenten: Anzahl der Zeichen (inkl. Leerraum), Anzahl der Buchstaben (`a-z` und `A-Z`), Anzahl der Ziffern (`0-9`), Anzahl der Sätze (Sätze enden mit „`.`", „`!`" oder „`?`") und die Anzahl der Zeilen. Zum Whitespace (Leerraum) gehören folgende Zeichen: Leerzeichen („ "), Tabulator („`\t`"), Zeilenumbruch („`\n`") und Wagenrücklauf („`\r`"). In der Eingabe sind nur Zeichen mit einem ASCII-Code zwischen 32 (Leerzeichen) und 126 (Tilde) sowie Whitespace erlaubt. Also insbesondere keine Umlaute.

Die Template-Datei für diese Aufgabe ist `text_statistic.c`. Bearbeiten Sie alle mit `TODO` markierten Stellen.

a) In dieser Aufgabe soll die Erstellung von Statistiken durch *Zeiger auf Funktionen* ermöglicht werden. Grundsätzlich wird jedes Element der Statistik erzeugt, indem gezählt wird wie viele Zeichen der Zeichenkette ein als Funktionszeiger gegebenes *Prädikat* erfüllen.

   Definieren Sie zuerst den Typen `CharacterTestFunction` für den Funktionszeiger. Funktionen dieses Typens erwarten als Parameter ein einzelnes Zeichen (`char`) und geben als `bool` zurück, ob dieses Zeichen die Funktion erfüllt oder nicht.

b) Implementieren Sie die Funktion `calculate_statistic`. Diese soll über die Zeichenkette `text` iterieren und die Anzahl an Zeichen zählen, die die Prädikatenfunktion `predicate` erfüllen.

c) Definieren und implementieren Sie die Prädikatenfunktionen `is_valid_char`, `is_letter`, `is_digit`, `is_sentence_end` und `is_newline`. Beachten Sie, dass die Funktionen kompatibel mit dem Typen `CharacterTestFunction` sein müssen.

d) Implementieren Sie die Funktion `make_text_statistic` in der eine `TextStatistic` Struktur erzeugt wird und die Statistiken mithilfe der `calculate_statistic` Funktion und Ihren Prädikatenfunktionen berechnet werden.

Aufgabe 2: Zeigerliste: Warenkorb

In dieser Aufgabe werden Operationen einer Zeigerliste verwendet. Die Zeigerliste ist in `pointer_list.c` implementiert. Die dazugehörige Header-Datei ist `pointer_list.h`. Die Beispiele aus der Vorlesung finden sich in `book_list.c`. Machen Sie sich zunächst mit `pointer_list` und `book_list` vertraut und lesen Sie den Beispielcode gründlich. Implementieren Sie dann die unten angegebenen Operationen auf einer Liste von Waren. Geben Sie nicht mehr benötigten Speicher jeweils frei. Verwenden Sie für diese Aufgabe ausschließlich die Listenimplementierung in `pointer_list.h` bzw. `pointer_list.c`.

Die Template-Datei für diese Aufgabe ist `shopping_cart.c`. Bearbeiten Sie alle mit `TODO` markierten Stellen.

a) Implementieren Sie die Funktion `copy_item`, die einen übergebenen Warenkorb Gegenstand dupliziert. Verwenden Sie die Konstrukturfunktion `new_item`. Implementieren Sie ebenfalls die Funktion `free_item`, die den belegten Speicherplatz freigibt.

b) Implementieren Sie die Funktion `is_electronics`, die genau dann *true* zurückgibt, wenn der Gegenstand zu der Kategorie Elektronik (`C_ELECTRONICS`) gehört. Die Funktion soll im Zusammenhang mit der Funktion `find_list` verwendet werden (siehe Aufruf in der `main`).

c) Erklären Sie als Kommentare im Quelltext jede Zeile der Funktion `price_less_than`. Implementieren Sie in der `main`-Funktion einen geeigneten Aufruf der `find_list`-Funktion, sodass der erste Gegenstand mit einem Preis geringer als 10€ zurückgegeben wird und geben Sie aus, um welchen es sich handelt.

d) Implementieren Sie die Funktion `item_name`, die den Namen des übergebenen Gegenstandes extrahiert. Die Funktion soll im Zusammenhang mit `map_list` verwendet werden (siehe Aufruf in der `main`-Funktion).

e) Implementieren Sie die Funktion `reduce_price`, die im Zusammenhang mit `map_list` verwendet werden soll (siehe Aufruf in der `main`-Funktion). Die zu implementierende Funktion soll eine Kopie des in `element` übergebenen Gegenstandes erstellen, den Preis in der Kopie um den übergebenen Faktor (im Bereich von $[0, 1]$) reduzieren und die so veränderte Kopie zurückgeben. Bsp. Preis vorher: 200€, übergebener Faktor von $0.1 \rightarrow$ Preis nachher: 180€

f) Erstellen Sie in der `main`-Funktion unter Benutzung von `filter_list` eine Liste aller Gegenstände (Preis geringer als 79€) und geben diese aus.

g) Erklären Sie als Kommentare im Quelltext jede Zeile der Funktionen `item_stats` und `add_prices`, die im Zusammenhang mit `reduce_list` verwendet werden (siehe Aufrufe in der `main`-Funktion).

h) Sortieren Sie den Warenkorb in `list` aufsteigend nach deren Preis durch sortiertes Einfügen per `insert_ordered`. Der entsprechende Aufruf ist in der `main`-Funktion zu finden. Implementieren Sie dazu die Vergleichsfunktion `cmp_item_price`.

i) Was sind die Vorteile von Zeigerlisten? Ist es sinnvoll verschiedene Daten in ein und dieselbe Zeigerliste einzufügen, beispielsweise eine Liste, die gleichzeitig Zeiger auf Zahlen und Zeiger auf Zeichenketten enthält?

   Beantworten Sie diese Fragen als Kommentar im Quelltext.

Aufgabe 3: Zeigerliste erweitern

In dieser Aufgabe soll die Zeigerliste um weitere Operationen erweitert werden. Geben Sie nicht mehr benötigten Speicher jeweils frei. Verwenden Sie für diese Aufgabe ausschließlich die Listenimplementierung in `pointer_list.h` bzw. `pointer_list.c`.

Die Template-Datei für diese Aufgabe ist `pointer_list_ext.c`. Bearbeiten Sie alle mit `TODO` markierten Stellen.

a) Implementieren Sie die Funktion `take_list`, die eine neue Liste mit den ersten `n` Elementen der Eingabeliste erzeugt. Die Elemente selbst sollen nicht dupliziert werden.

b) Implementieren Sie die Funktion `drop_list`, die eine neue Liste mit den Elementen der Eingabeliste außer den ersten `n` Elementen erzeugt. Die ersten Elemente der Eingabeliste werden also nicht verwendet. Die Elemente selbst sollen nicht dupliziert werden.

c) Implementieren Sie die Funktion `interleave`, die eine neue Liste erzeugt, indem sie abwechselnd ein Element aus der ersten und der zweiten Liste nimmt. Die Eingabelisten sollen nicht veränert werden. Die Elemente selbst sollen nicht dupliziert werden.

d) Implementieren Sie die Funktion `group_list`, die äquivalente Elemente der Eingabeliste gruppieren soll. Für äquivalente Elemente liefert die Funktion `equivalent` den Wert *true*. Für die Beispielfunktion `group_by_length` sind Strings äquivalent, wenn sie die gleiche Länge haben. Das Resultat von `group_list` ist eine Liste von Gruppen. Jede Gruppe ist eine Liste von äquivalenten Elementen. Beispiel: Für die Liste `["x", "yy", "zzz", "f", "gg"]` ergeben sich nach Aufruf von `group_list` die Gruppen `[["x", "f"], ["yy", "gg"], ["zzz"]]`. Die Reihenfolge der Gruppen und der Elemente in den Gruppen ist dabei beliebig.