Human-Computer Interaction Group

Leibniz Universität Hannover

Assignment 11

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 22.01. 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: prog1lib

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

Aufgabe 1: Flattersatz (1 Punkt)

In dieser Aufgabe geht es um die Formatierung von Text auf eine vorgegebene Breite. Der Text wird aus `mytext.txt` eingelesen und soll im linksbündigen Flattersatz (https://de.wikipedia.org/wiki/Flattersatz) ausgegeben werden. Das erste Wort einer Zeile wird immer ausgegeben, auch wenn es breiter ist als die geforderte Breite. Das gewünschte Ergebnis für Breite 40 ist in `mytext_formatted.txt` zu sehen.

Die Lösung soll den Datentyp `Str` verwenden, der in der Template-Datei definiert wird. Dieser ermöglicht es, auf Teile von C-Strings zuzugreifen. Ein `Str` wird also nicht durch ein Nullbyte terminiert, sondern speichert explizit die Länge. Außerdem wird eine Kapazität gespeichert. Dies ermöglicht es, einen Puffer einer benötigten Kapazität zu allokieren und dann nach und nach Zeichen anzufügen (wordurch die Länge vergrößert wird), bis die Kapazität erreicht ist.

Die Template-Datei für diese Aufgabe ist `alignment.c`.

a) Machen Sie sich mit dem Datentyp `Str` vertraut. Was geschieht, wenn beim Anfügen (`append`) an einen `Str` die Kapazität nicht ausreicht? Welche Einschränkung hat der gewählte Ansatz? Schreiben Sie die Antwort als Kommentar in die C-Datei.

b) Implementieren Sie die Funktion **split_words()**. Diese bestimmt die in einem Text vorkommenden Wörter als Array von `Strs`. Die Eingabe ist unformatierter Text, bei dem Wörter durch Leerzeichen und Zeilenumbrüche (siehe `skip_word()`) voneinander getrennt sind. Der Ausgabeparameter `word` muss auf ein Array der Länge `words_length` zeigen. Die Elemente des Arrays sollen von **split_words** entsprechend gesetzt werden. Die Funktion soll die Anzahl der gesetzten Elemente zurückgeben. Die Funktion muss abbrechen, wenn die Arraylänge `words_length` erreicht ist.

   Tipp: Die Implementierung kann ähnlich zur Implementierung von `count_words()` realisiert werden.

c) Implementieren Sie die Funktion **words_align_left()**. Diese erhält ein Array mit Wörtern als Eingabe und soll daraus einen formatierten Text der Breite `width` generieren und in `dst` (destination) ablegen. Die Kapazität von `dst` muss genügend groß sein.

Aufgabe 2: Pointerliste: Warenkorb (2 Punkte)

In dieser Aufgabe werden Operationen einer Pointerliste verwendet. Die Pointerliste ist in `pointer_list.c` implementiert. Die dazugehörige Header-Datei ist `pointer_list.h`. Machen Sie sich zunächst mit `pointer_list.c` vertraut. 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`.

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()`-Funktion).

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.

e) Was sind die Vorteile von Pointerlisten? Ist es sinnvoll verschiedene Daten in ein und dieselbe Pointerliste einzufügen, beispielsweise eine Liste, die gleichzeitig Pointer auf Zahlen und Pointer auf Zeichenketten enthält?

   Beantworten Sie diese Fragen als Kommentar im Quelltext.

Aufgabe 3: Zeigerliste: Erweiterung (1 Punkt)

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`.

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 verändert 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. Die Reihenfolge der Gruppen und der Elemente in den Gruppen ist dabei beliebig.

   Beispiel: Für die Liste `["x", "yy", "zzz", "f", "gg"]` ergeben sich nach Aufruf von `group_list()` die Gruppen `[["x", "f"], ["yy", "gg"], ["zzz"]]`.

Aufgabe 4: Bonusaufgabe: Öffnende und Schließende Klammern (1 Bonuspunkt)

Diese Bonusaufgabe liefert bei erfolgreicher Bearbeitung einen Bonuspunkt, um möglicherweise versäumte Punkte für den Klausurbonus nachzuholen. **Zum Bestehen dieses Assignments und somit der Studienleistung müssen dennoch zwei Aufgaben aus dem Aufgabenpool von Aufgaben 1–4 erfolgreich bearbeitet werden!**

In dieser Aufgabe geht es um die Implementierung einer Syntaxprüfung von öffnenden und schließenden Klammern. Zu jeder öffnenden Klammer muss in der Zeichenkette ein passendes schließendes Gegenstück vorkommen. Schließende Klammern müssen zu der jeweiligen öffnenden Klammer passen. Folgende Klammernpaare sollen unterstützt werden: `( )`, `[ ]`, `{ }`, `< >`.

Die Zeichenkette „`<{[()]}>`" ist bspw. valide. Die Zeichenketten „`(Test`" und „`([)]`" nicht.

Die Template-Datei für diese Aufgabe ist `parentheses.c`.

a) Zum Lösen dieser Aufgabe ist es hilfreich einen `Stack` zu nutzen. Implementieren Sie diese Datenstruktur mit der für diese Aufgabe benötigten Funktionalität. Allokieren Sie Speicher dynamisch.

   Tipp: Sie können dafür auch Stack-Implementierungen aus vergangenen Assignments nutzen. Womöglich müssen diese dann an diese Aufgabenstellung angepasst werden.

b) Implementieren Sie die Funktion **verify_parentheses()**, die `true` zurückgibt, wenn die Klammerung syntaktisch korrekt ist und `false` wenn nicht. Die Funktion muss Zeichenketten beliebiger Länge behandeln können.

c) Stellen Sie sicher, dass dynamisch allokierter Speicher wieder freigegeben wird.