Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2018/19

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

Übungsblatt 5

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 22.11. 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.

Aufgabe 1: Rekursive Listen in Postfix

Eine Liste sei definiert als entweder leer (`null`) oder ein Paar (`pair`) aus einem ersten Element (`pair-first`) und einer Rest-Liste (`pair-rest`). Implementieren Sie die nachfolgend genannten Funktionen rekursiv und ohne Verwendung von Array-Funktionen. Verwenden Sie die in list.pf vorgegebene Datendefinition.

a) Implementieren Sie die Funktion `lprepend`, die `x` vorne vor die Liste `l` einfügt und die resultierende Liste zurückgibt. Schreiben Sie einen Dokumentations-String (purpose statement).

b) Implementieren Sie die Funktion `lappend`, die `x` hinten an die Liste `l` anhängt und die resultierende Liste zurückgibt. Schreiben Sie einen Dokumentations-String (purpose statement).

c) Implementieren Sie die Funktion `has-string?`, die prüft, ob die Liste `l` mindestens einen String enthält. Schreiben Sie einen Dokumentations-String (purpose statement).

d) Implementieren Sie die Funktion `only-numbers?`, die prüft, ob die Liste `l` ausschließlich Zahlen enthält. Schreiben Sie einen Dokumentations-String (purpose statement).

e) Implementieren Sie die Funktion `keep-strings`, die nur die in der Liste `l` vorkommenden Strings als Liste zurückgibt. Schreiben Sie einen Dokumentations-String (purpose statement).

f) Implementieren Sie die Funktion `range`, die eine aufsteigende Liste von ganzen Zahlen zurückgibt. Für `3 7 range` soll die Funktion beispielsweise eine Liste mit den Zahlen 3, 4, 5, 6, 7 (in dieser Reihenfolge) zurückliefern. Schreiben Sie einen Dokumentations-String (purpose statement).

g) Implementieren Sie die Funktion `i-th`, die das i-te Element der Liste `l` zurückgibt. Für Index 0 soll das erste Element der Liste zurückgegeben werden. Für ungültige Indizes soll

`nil` zurückgegeben werden. Schreiben Sie einen Dokumentations-String (purpose statement).

h) Implementieren Sie die Funktion `rev-rec` (für reverse recursively), die von der existierenden Funktion `rev` (für reverse) aufgerufen wird, um eine Liste umzudrehen. Die Variable `result` dient der Speicherung und Übergabe der bereits umgedrehten Teilliste.

Aufgabe 2: Nährwerttabelle als rekursive Liste in Postfix

Eine Liste sei definiert als entweder leer (`null`) oder ein Paar (`pair`) aus einem ersten Element (`pair-first`) und einer Rest-Liste (`pair-rest`). Implementieren Sie die nachfolgend genannten Funktionen rekursiv und ohne Verwendung von Array-Funktionen. Verwenden Sie die in nutrients-list.pf vorgegebene Datendefinition.

a) Implementieren Sie die Funktion `names`, die die Namen der Lebensmittel als Liste zurückgibt. Schreiben Sie einen Dokumentations-String (purpose statement).

b) Implementieren Sie die Funktion `carb-above`, die Lebensmittel zurückgibt, deren Kohlenhydratgehalt (carb) über der angegebenen Grenze liegt. Schreiben Sie einen Dokumentations-String (purpose statement).

c) Implementieren Sie die Funktion `fat-sum`, die die Fettanteile der im Array übergebenen Lebensmittel (Nutrients) summiert. Schreiben Sie einen Dokumentations-String (purpose statement).

d) Implementieren Sie die Funktion `average-fat`, die den durchschnittlichen Fettgehalt der in der Liste übergebenen Lebensmittel (Nutrients) berechnet. Nutzen Sie zur Implementierung dieser Funktion bereits in nutrients-list.pf existierende bzw. von Ihnen implementierte Funktionen. Für `average-fat` ist keine Rekursion erforderlich. Schreiben Sie einen Dokumentations-String (purpose statement).

Aufgabe 3 (Optional): Mergesort in Postfix -> 1 Bonuspunkt

Mergesort (https://de.wikipedia.org/wiki/Mergesort) ist ein interessanter und effizienter Algorithmus zum Sortieren von Listen. Mergesort funktioniert so, dass eine unsortierte Liste solange aufgeteilt wird bis nur noch Teillisten vorhanden sind, die aus einem Element bestehen. Da einelementige Listen immer sortiert sind, kann dann begonnen werden je zwei sortierte Teillisten zu einer sortierten Liste zusammenzufügen. Die `merge-sort` Funktion soll später **Listen** mit Daten vom Typ `:Str` sortieren können.

Implementieren Sie `merge-sort` rekursiv in Postfix. Implementieren Sie dazu die folgenden Teilaufgaben.

a) Schreiben Sie eine Funktion `split-list`, die eine Liste erwartet, diese in zwei gleich große Teile aufteilt und beide Hälften auf dem Stack ablegt. Diese Funktion wird nachher genutzt um das zu sortierende Array immer wieder aufzuteilen bis nur noch einelementige Listen vorhanden sind. Schreiben Sie eine Testfunktion, die die Funktion prüft. Prüfen Sie Listen mit gerader Anzahl an Elementen und mit ungerader Anzahl an Elementen. Beispielsweise sollte aus `"A" "B" "C" null cons cons cons split-array` sollte `"A" null cons` und `"B" "C" null cons cons` werden. Tipp: Versuchen Sie die Funktion rekursiv zu implementieren. Ggf. bietet es sich an eine rekursive Hilfsfunktion zu erstellen und diese in `split-list` aufzurufen.

b) Schreiben Sie eine Funktion `merge`, die zwei Listen vom Stack nimmt und diese zu einer sortierten Liste zusammenfügt. Gehen Sie davon aus, dass beide übergebenen Listen sortiert sind. In diesem Fall müssen nur noch 3 grobe Bedingungen geprüft werden. 1. Wenn beide Listen nicht leer sind, dann muss geprüft werden in welcher Liste von beiden das kleinere Element steht. Dieses wird dann aus der Liste entfernt und an die Ergebnisliste angehangen. Dies geschieht solange bis eine der Listen leer ist. 2. Ist eine Liste leer, so wird der Inhalt der anderen Liste an die Ergebnisliste angehangen. Danach sind beide Ausgangslisten leer und in der Ergebnisliste sollten alle Elemente sortiert stehen. 3. Wenn beide Listen leer sind, kann die sortierte Ergebnisliste zurückgegeben werden. Testen Sie auch diese Funktion. Bsp.: `"A" null cons "B" "C" null cons cons merge` sollte das gleiche wie `"A" "B" "C" null cons cons cons` auf den Stack legen. Tipp: Versuchen Sie die Funktion rekursiv zu implementieren.

c) Implementieren Sie die Funktion `merge-sort` unter zu Hilfenahme der bereits implementierten Funktionen `split-list` und `merge` rekursiv (`merge-sort` ruft sich entsprechend immer wieder selbst auf, bis eine Abbruchbedingung erreicht ist). Vergessen Sie nicht, eine passende Abbruchbedingung zu formulieren, damit die Rekursion enden kann. Schreiben Sie auch hierfür Tests. Bspw.: `"C" "B" "A" null cons cons cons merge-sort` sollte das gleiche wie `"A" "B" "C" null cons cons cons` auf den Stack legen.

d) Mergesort ist eigentlich ein sehr effizienter Algorithmus, jedoch nicht mit dieser Postfix-basierten Implementation. Welche Eigenschaft vom Datentyp Liste (in Postfix) führt zu dieser schlechten Effizienz?

Aufgabe 4: C-Compiler installieren

Diese Aufgabe dient der Vorbereitung der C-Aufgaben ab nächster Woche. Installieren Sie auf Ihrem Rechner den GCC-Compiler und einen Texteditor (am besten mit Syntaxhervorhebung). Nähere Erläuterungen zur Installation unter Windows, OS X und Linux finden Sie in den Übungsfolien.

a) Testen Sie die Installation indem Sie `gcc -v` auf der Kommandozeile aufrufen. Fügen Sie die Ausgabe die Sie erhalten in ihre Abgabe ein.

b) Testen Sie Ihre Installation außerdem mit folgendem Programm:

```
1. #include <stdio.h>
2. int main(void) {
3.       printf("hello, world\n");
4.       return 0;
5. }
```

Speichern Sie das Programm als Textdatei unter dem Namen `hello.c`

Öffnen Sie eine Kommandozeile und wechseln Sie in das Verzeichnis, in dem `hello.c` liegt.:

`cd /Pfad/zu/meinem/Verzeichnis` bzw.
`cd C:\Pfad\zu\meinem\Verzeichnis`

Kompilieren Sie das Programm:

`gcc hello.c -o hello` bzw. `gcc hello.c -o hello.exe`

Führen sie das Programm aus:

`./hello` bzw. `hello.exe`

Die Ausgabe sollte lauten:
`hello, world`

c) Erweitern Sie das Programm, so dass statt „world" ihr Name ausgegeben wird. Bspw.:
`hello, Tim Duente`

Aufgabe 5: Rabatt auf Schokoriegel

Vorbereitung

Hier wird zunächst die Vorbereitung Ihrer „Entwicklungsumgebung" beschrieben. Diese Schritte werden für die weiteren Aufgaben entsprechend sein und werden nur hier ausführlich beschrieben. Das Herunterladen und Kompilieren der Bibliothek ist nur einmalig erforderlich.

Detaillierte Schritte zur Installation der Programming I C Library finden sich unter http://hci.uni-hannover.de/files/prog1lib/index.html. Führen Sie zunächst diese Schritte aus, inklusive dem Kompilieren und Ausführen eines Beispiels in `prog1lib/script_examples`.

Laden Sie assignment05.zip mit den Template-Dateien für dieses Übungsblatt aus Stud.IP herunter und speichern Sie assignment05.zip im MinGW-Home-Verzeichnis. Öffnen Sie die MinGW-Shell. Unter Mac OS X und Linux verwenden Sie das Terminal statt MinGW. Die Kommandos sind identisch. Verwenden Sie das Template dieser Aufgabe wie folgt:

`pwd` ← gibt den Pfad des aktuellen Verzeichnisses aus (sollte /home/MyName sein)

`ls` ← listet den Inhalt des aktuellen Verzeichnisses

(prog1lib und assignment05.zip müssen im aktuellen Verzeichnis liegen)

`unzip assignment05.zip` ← Template-Dateien dieser Übung entpacken

`cd assignment05` ← change directory to assignment05

(`total.c` mit gutem Texteditor (z.B. Notepad++) editieren, speichern)

`make total` ← ausführbares Programm (total bzw. total.exe) erstellen

`./total` ← Programm starten (bzw. `total.exe`)

Die letzten drei Schritte (editieren+speichern, `make total` und `./total`) führen Sie nun wiederholt aus, bis das Programm fertig ist. Die ↑-Taste stellt die vorherige Zeile wieder her. Kompilieren und Ausführen lassen sich kombinieren: `make total && ./total`

Aufgabe

Ein Hersteller von köstlichen Schokoriegeln, vertreibt diese direkt im Internet an seine Kunden. Für größere Bestellungen gewährt er einen Mengenrabatt. Entwickeln Sie eine Funktion, `total`, die für eine Menge an Schokoriegeln den Gesamtpreis berechnet. Ein Schokoriegel kostet pro Stück 50 Cent. Wenn zehn oder mehr Stück gekauft werden sinkt der Preis pro Schokoriegel auf 45 Cent. Bei 100 oder mehr Stück sinkt der Preis auf 40 Cent. Eine negative Menge von Schokoriegeln soll einen Preis von 0 Cent zurückliefern. Der Hersteller verlangt bei einem Gesamtpreis von unter 2000 Cent eine Versandpauschale von 500 Cent. Ab 2000 Cent Wert müssen keine Versandkosten mehr bezahlt werden. Die Funktion soll als Eingabe die Anzahl an Schokoriegeln übergeben bekommen und den Gesamtpreis (Wert der Schokoriegel + ggf. Versandkosten) in Cent zurückgeben.

a) Definieren Sie als erstes Konstanten vom Typ `int`: 3 Konstanten für die verschiedenen Preise,1 Konstante für den Versandpreis und eine Konstante für die Grenze, ab der keine Versandkosten mehr anfallen.

b) Schreiben Sie als nächstes einen Funktionsstub für die Funktion `total`. `total` soll eine Anzahl von Schokoriegeln übergeben bekommen und den Gesamtpreis (Wert der Schkoriegel + ggf. Versandkosten) in Cent zurückgeben. Nutzen Sie den Datentyp `int` für den Parameter der Funktion und für den Rückgabewert. Schreiben Sie zusätzlich einen Kommentar über die Funktion, der beschreibt, was die Funktion `total` leistet.

c) Schreiben Sie eine Funktion `total_test`, die die Funktion `total` testet. Erstellen Sie mindestens 6 sinnvolle Testfälle. Nutzen Sie dafür die Funktion `test_equal_i` aus der prog1lib. Schauen Sie sich ggf. das Beispiel `celsius_to_fahrenheit.c` an.

d) Implementieren Sie die Funktion `total`. Ein Schokoriegel kostet 50 Cent. Wenn zehn und mehr Stück gekauft werden sinkt der Preis pro Riegel auf 45 Cent. Bei 100 oder mehr Stück sinkt der Preis auf 40 Cent je Riegel. Der Händler verlangt bei einer Summe von unter 2000 Cent eine Versandpauschale von 500 Cent. Ab 2000 Cent Wert müssen keine Versandkosten mehr bezahlt werden. Eine negative Anzahl von Produkten soll 0 Cent als Rückgabewert liefern. Beispielsweise liefert die Funktion `total` für drei Produkte 650 (Cent) (150 Cent für das Produkt und 500 Cent Versandpauschale) als Rückgabewert. Für 10 Produkte 950 (Cent).

Hinweise:

- Windows und MinGW bei Benutzung von cmd.exe: Tragen Sie, wie in den Folien zur Hörsaalübung beschrieben, die Pfade zu den Verzeichnissen `C:\MinGW\bin` und `C:\MinGW\msys\1.0\bin` in die `PATH`-Umgebungsvariable ein. Dies ist notwendig, damit die Tools (`make`, `gcc`, etc.) gefunden werden.
- Windows und MinGW: Damit das unzip Kommando funktioniert muss der „bin"-Eintrag von msys-unzip im MinGW Installation Manager angewählt und installiert sein.
- Mit den Pfeiltasten (hoch, runter) lassen sich frühere Kommandos wieder abrufen.
- Wenn Sie beim Kompilieren eine Fehlermeldung bekommen, dass base.h nicht gefunden wurde, dann weicht Ihre Verzeichnisstruktur von der geforderten ab. Überprüfen Sie erneut die eingangs angegebenen Schritte. Dieser Fehler sieht wie folgt aus:
```
total.c:7:10: fatal error: 'base.h' file not found
#include "base.h"
         ^~~~~~~~
```
- Kompilieren und Ausführen lassen sich auf der Kommandozeile kombinieren zu:
```
make total && ./total
```