PR3 - Übungsblatt A.7

(Stand 2021-11-03 18:08)

Prof. Dr. Holger Peine
Hochschule Hannover
Fakultät IV – Abteilung Informatik
Raum 1H.2.60, Tel. 0511-9296-1880
Holger.Peine@hs-hannover.de

Thema

Funktionszeiger, Ein-/Ausgabe, Dynamische Datenstrukturen

Termin

Ihre Arbeitsergebnisse zu diesem Übungsblatt führen Sie bitte bis zum 03.12.2021 vor.

1 Funktionszeiger (1 Punkt)

basiert auf Vorlesung bis einschl. Abschnitt 6.f

Teilaufgabe a)

In der Programmiersprache Ruby gibt es das Konzept des Blocks, mit dem man elegant Funktionsrümpfe definieren kann, die man an andere Funktionen als Parameter übergeben kann. Beispielsweise sorgt die folgende Programmzeile für die dreifache Ausgabe der Nachricht „Hallo“:

```
3.times { puts "Hallo" } # Ruby, kein C!
```

Hier wird die Methode `times` auf dem Objekt 3 der Klasse Integer aufgerufen, und zwar mit einem Code-Block als Parameter.

Wir versuchen, dieses Verhalten in C nachzubilden. Zunächst sei die folgende Funktion vorgegeben:

```
void print(void) { printf("Hallo\n"); }
```

Schreiben Sie in C eine Funktion `times(int n, …)`. Die Funktion `times` soll genau zwei Parameter entgegen nehmen. Als zweiten Parameter übergibt man beim Aufruf einen Funktionszeiger auf `print`. Die Funktion `times` soll die übergebene Funktion `n`-mal ausführen.

Teilaufgabe b)

Ein anderes Beispiel in Ruby:

```
3.times { |i| puts i }   # ruby
```

führt zu der Ausgabe:

```
0
1
2
```

Hier besitzt der Code-Block einen Parameter `i`, der während der Ausführung von `times` mit aktuellen Werten belegt wird.

Schreiben Sie in C eine Funktion `times2(int n, …)`, die einen Funktionszeiger als zweiten Parameter entgegennimmt und die diese Ruby-Funktionalität so gut es geht nachbildet. Auch hier geben wir die als Zeiger zu übergebende Funktion vor: `void print2(int i) { printf("%d\n", i); }`

2 Binärdateien (1 Punkt)

basiert auf Vorlesung bis einschl. Abschnitt 7

L.7 Lesen Sie zunächst das Dokument PR3_07_L.pdf im Vorlesungsordner. Es enthält Informationen zu weiteren Funktionen der Ein- und Ausgabe.

Gegeben sei die Struktur eines Angestellten:

```
struct angestellter {
    char name[NAME_LEN+1];
    int personalnummer;
    float gehalt;
};
```

Schreiben Sie eine `main`-Funktion, die ein Array von 5 Angestellten mit Werten belegt. Mit diesem Array soll Ihre `main`-Funktion eine von Ihnen zu schreibende Funktion

```
void binaer_speichern(struct angestellter arr[],int anz)
```

aufrufen. Die Funktion `binaerSpeichern` schreibt die Angestellten **im Binärmodus** (benutzen Sie also zum Schreiben `fwrite`, nicht `fprintf`) in eine Datei namens `angestellte.dat` im aktuellen Verzeichnis. Eine weitere von `main` aufzurufende Funktion

```
void binaer_laden_und_ausgeben(void)
```

soll die Datei `angestellte.dat` wieder zum Lesen öffnen, den Inhalt einlesen und auf der Konsole vollständig ausgeben. Sie dürfen das Schreiben und Lesen mit einer Schleife implementieren, die jeweils einen Angestellten verarbeitet – es ist aber auch möglich, den gesamten Array mit einer einzigen Operation zu schreiben bzw. zu lesen.

Schreiben Sie in die Datei nicht nur die Angestellten-Daten, sondern auch direkt am Anfang die Anzahl der gespeicherten Angestellten als `int`-Wert, damit Sie beim Lesen einen Speicherbereich passender Größe dynamisch mit `malloc` anlegen können.

3 Textdatei lesen, stdout und stderr (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 7.c

L.7

Lesen Sie zunächst das Dokument PR3_07_L.pdf im Vorlesungsordner. Es enthält Informationen zu weiteren Funktionen der Ein- und Ausgabe.

Schreiben Sie ein Programm, das Text entweder von der Standardeingabe oder aus einer Textdatei einliest. Der Benutzer kann als Kommandozeilenparameter entweder den Dateinamen eingeben oder ein Minuszeichen ("-") mit der Bedeutung, dass von der Standardeingabe gelesen wird.

Das Programm soll den eingelesenen Text nach Umlauten und dem deutschen ß absuchen und den Text dann mit folgenden Ersetzungen komplett wieder ausgeben:

Von	Nach	Von	Nach	Von	Nach	Von	Nach
ä	ae	Ö	oe	ü	ue	ß	ss
Ä	Ae	Ö	Oe	Ü	Ue		

Die Ausgabe erfolgt auf der Standardausgabe. Als Testdatei können Sie das Gedicht „Erlkönig“ von Johann Wolfgang von Goethe (nicht Göthe ☺) einlesen (s. Anlagen zu diesem Übungsblatt).

Ihr Programm soll Fehler beim Einlesen erkennen und Fehlermeldungen ausgeben. Die Fehlermeldungen sollen nicht auf der Standardausgabe, sondern auf der Standardfehlerausgabe erfolgen.

Prüfen Sie Ihr Programm. Wie können Sie sicher testen, ob Ihr Programm tatsächlich die Standardfehlerausgabe für Fehlermeldungen nutzt?

Hinweise:

- `char`-Konstanten im Quelltext, wie `'ä'` sind vom Typ `char`, d. h. u. U. vorzeichenbehaftet (der Standard spricht hier von „plain char“ – leider ist nicht festgelegt, ob mit oder ohne Vorzeichen). Wenn Sie Zeichen aus der Datei z. B. mit der Funktion `getchar` lesen, erhalten Sie eine Zahl vom Typ `int` aus dem Wertebereich 0-255. Beim Vergleich der eingelesenen `int`-Zahl mit der `char`-Konstante `'ä'` müssen Sie also ggf. eine Umwandlung der Zahl in einen `unsigned`-Typ vornehmen.
- Beachten Sie, dass es beim Einlesen von Textdateien wichtig ist, die richtige Codierung zu verwenden. Wenn Sie in Ihrem Programmtext `char`-Literale wie `'ä'` einsetzen, muss die Codierung der Quelltextdatei mit der Codierung der eingelesenen Datei übereinstimmen. Die Datei `erlkoenig.txt` in den Anlagen ist in der Codierung ISO-8859-1 gespeichert.
- Alternativ können Sie Vergleiche von Zahlencodes vornehmen. Über die Zahlencodes der Umlaute und des ß in der ISO-8859-1-Codierung können Sie sich z. B. hier informieren: http://de.wikipedia.org/wiki/ISO_8859-1.

4 Einfach verkettete Liste mit Speicherung des Listenendes (2 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 8

L.8

Lesen Sie zunächst das Dokument PR3_08_L.pdf im Vorlesungsordner.

In der Vorlesung haben wir C-Funktionen für Operationen auf einer einfach verketteten Liste besprochen. Insbesondere die Einfügung am Listenende war dabei umständlich, weil wir zuerst das Listenende, beginnend vom Kopf her, suchen mussten, um dann dort ein Listenelement anfügen zu können.

Ergänzen Sie den Programmcode derart, dass das Anfügen eines Eintrages am Listenende in konstanter Zeit (O(1)) erfolgt. Dazu repräsentieren wir die Liste nicht mehr nur durch einen Zeiger `kopf` auf den ersten Knoten, sondern führen zusätzlich einen Zeiger `ende` auf den letzten Knoten ein. Dadurch benötigt man für die Listen-Funktionen neben der Parametervariablen `kopfref` einen weiteren Parameter `enderef` (der Listenknotentyp bleibt gleich). Die neue Ende-Referenz müssen Sie (natürlich) auch in einigen anderen der Listen-Funktionen entsprechend pflegen, so dass mehrere Funktionen um den zusätzlichen Ende-Parameter erweitert werden müssen.

Diese Erweiterung der einfach verketteten Liste ist ein typisches Beispiel dafür, dass eine (z.B. für Programmier-Laien) "kleine" Änderung je nach Struktur des Programms einen überraschend großen Aufwand verursachen kann. Solche "Überraschungen" sind der Grund dafür, dass die Aufwandsschätzungen für Softwareentwicklungsprojekte große Erfahrung braucht.

Eine Header-Datei mit den erweiterten Signaturen finden Sie als Datei `liste1ende.h` in den Anlagen zu diesem Übungsblatt. In der Datei `liste1.c` finden Sie den Quelltext der einfach verketteten Liste aus der Vorlesung, den Sie als Grundlage benutzen können. Schreiben Sie Ihren eigenen Code in eine Datei `liste1ende.c`. Zum Testen Ihres Programms können Sie `liste1endetest.c` verwenden. `liste1.c` ist dabei sozusagen eine Bibliothek (wenn auch hier nur aus einer einzigen `.o`-Datei bestehend), die vom Client-Code `liste1endetest.c` (Hauptprogramm) benutzt wird.

Schreiben Sie schließlich ein Makefile, das das Programm aus den Dateien `liste1ende.h`, `liste1ende.c` und `liste1endetest.c` baut.

5 Einfach verkettete Liste (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 8

L.8

Lesen Sie zunächst das Dokument PR3_08_L.pdf im Vorlesungsordner.

Betrachten Sie nun wieder die einfach verkettete Liste ohne Speicherung des Listenendes (so wie sie in der Datei `liste1.c` in den Anlagen zu diesem Übungsblatt implementiert ist).

a) Fügen Sie eine weitere Funktion
```
anwenden(struct knoten* kopf, void (*f)(int))
```
hinzu, die `f` auf jeden in der Liste gespeicherten Wert anwendet.

b) Kann man die bereits existierende Funktion `durchlaufen` (die alle Werte ausgibt) durch `anwenden` ersetzen? Falls ja, wie? Falls nein, warum nicht?

c) Fügen Sie eine weitere Funktion
```
void verketten(struct knoten** kopf1Ptr, struct knoten* kopf2)
```
hinzu, die zwei Listen verkettet: Das erste Element der Liste `kopf2` soll der Nachfolger des letzten Elements der Liste `*kopf1Ptr` werden (`kopf1Ptr` ist ein call-by-reference-Parameter, während `kopf2` ein call-by-value-Parameter ist) . Achten Sie darauf, dass Ihre Funktion auch dann funktioniert, wenn eine oder beide der Listen leer sind.

6 Sokoban-Spiel (0 Punkte)

basiert auf Vorlesung bis einschl. Abschnitt 8

In den Anlagen zu diesem Übungsblatt finden Sie ein unvollständiges Sokoban-Spiel, welches Sie ergänzen können. Eine kurze Beschreibung des Spiels finden Sie unter https://de.wikipedia.org/wiki/Sokoban.
Bitte beachten Sie, dass die Grafiken und die Definition des Levels nach §60a UrhG ausschließlich für diese Lehrveranstaltung (und für keine anderen Zwecke) genutzt werden dürfen, weil keine allgemeine Lizenz des Urhebers bzw. der Urheberin vorliegt. Bei der Verwendung von Funktionsbibliotheken sollten Sie im Allgemeinen (auch bei Open Source) darauf achten, dass die Lizenzbedingungen eingehalten werden. Die SDL 2.0 Library steht unter der freizügigen zlib-Lizenz zur Verfügung.

Auf den Rechnern in den PC-Pools ist alles Nötige für diese Aufgabe bereits vorhanden. Wenn Sie stattdessen auf Ihrem eigenen Rechner arbeiten wollen, müssen Sie zunächst die SDL 2.0 SDL-Library auf Ihrem Rechner installieren. Es werden nicht nur die Runtime Binaries sondern u. a. auch die Header-Dateien benötigt. Unter https://www.libsdl.org gibt es im Wiki für unterschiedliche Plattformen Installationsanleitungen. Bevor Sie das Spiel übersetzen können, muss noch das Makefile an die Gegebenheiten auf Ihrem Rechner angepasst werden. Dazu müssen Sie `gcc` über die Optionen `-I` und `-L` mitteilen, wo sich auf Ihrem Rechner die Header- und die Bibliotheksdateien befinden. Wenn auch das Programm `sdl2-config` installiert wurde, könnte dabei ein Aufruf von `sdl2-config --libs --cflags` weiterhelfen.

a) Wenn Sie das Spiel übersetzen können, sollten Sie zunächst die Funktion `sokoban_init_field` in der Datei `sokoban_model.c` implementieren. Wenn Sie alles richtig gemacht haben, sehen Sie nach dem Start des Programms zunächst das Startlevel und nach einigen Sekunden das erste richtige Level.
b) Als nächstes sollten Sie die Funktionen `sokoban_move_north`, `sokoban_move_east`, `sokoban_move_south` und `sokoban_move_west` implementieren. Diese werden aufgerufen, wenn die entsprechenden Pfeiltasten auf der Tastatur gedrückt werden, und bewegen die Spielfigur und ggf. eine Box: Boxen können nur geschoben und nicht gezogen werden. Es kann nur eine Box gleichzeitig geschoben werden. Wände begrenzen das Spielfeld.
c) Schauen Sie sich den übrigen Quellcode an und implementieren Sie weitere Verbesserungen, wie z. B. eine Funktion zum Zurücknehmen von Spielzügen oder ein Zähler der benötigten Züge.