Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2017/18
Feldmann / Foucard / Lichtblau / Streibelt / Krüger / Enghardt / Palmer

Aufgabenblatt 10

letzte Aktualisierung: 26. Januar, 14:24 Uhr

(b003d903421b1cc6360709919885 6f0d76366c9c)

Ausgabe: Freitag, 26.01.2018
Abgabe: spätestens Mittwoch, 07.02.2018, 18:00

Thema: *Quicksort auf einfach verketteten Listen*

Abgabemodalitäten

- Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des eecsIT mittels `gcc -std=c99 -Wall` kompilieren.
- Für die Hausaufgaben im Semester gibt es ein eigenes Subversion-Repository (SVN). Bitte die Hinweise auf Blatt 1 dazu beachten!
- Die Abgaben für Blatt 1 bis 4 erfolgen als Einzelabgaben. Ab Blatt 5 erfolgen Abgaben in festen Gruppen à 2 Personen.
- Die Gruppen werden vor Ausgabe von Blatt 5 gebildet. Solltet ihr keine Gruppe gebildet haben, werdet ihr einer Gruppe zugewiesen.
- Alle Abgaben müssen an der dafür vorgesehenen Stelle die Namen aller Gruppenmitglieder bzw. bei Einzelabgaben des Autors/der Autorin enthalten!
- Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Die finale Abgabe
  - für Gruppenabgaben erfolgt im Unterordner `Tutorien/t<xx>/Gruppen/g<xx>/Abgaben/Blatt<xx>/`
  - für Einzelabgaben erfolgt im Unterordner `Tutorien/t<xx>/Studierende/<tuBIT-Login>/Abgaben/Blatt<xx>/`

  WICHTIG: Die Abgabeordner werden immer von uns erstellt, sonst kommt es zu Konflikten! Benutzt zum Aktualisieren `svn up` im obersten Verzeichnis des Repositories!
- Benutzt für alle Abgaben die in der jeweiligen Aufgabe angegebenen Dateinamen (die von uns vorgegebenen Dateien haben das Wort "vorgabe" im Dateinamen, du musst die Datei mit deiner Lösung also entsprechend umbenennen.)
- Gib bei den Programmieraufgaben den C-Quellcode ab und achte darauf, dass die Datei mit .c endet
- Bei Textaufgaben sind, wenn nicht anders angegeben, .txt Dateien zugelassen, die als Plaintext-Datei zu speichern sind. (Keine Word-Dateien umbenennen, etc.!)

1. Aufgabe: Implementierung von Quicksort auf verketteten Listen (3 Punkte)

Die Abgabe des Quellcodes erfolgt im SVN unter dem Namen `introprog_quicksort.c`

Implementiere die stabile Variante des in der Vorlesung vorgestellten Sortieralgorithmus `quick_sort` in C. Deine Funktion soll als Eingabe eine verkettete Liste bekommen und die Elemente sortiert zurückgeben.

Dazu findest du hier den Pseudocode für Quicksort, der seine generelle Funktionalität darstellt. Erarbeite dir die Funktionalität der Funktion `partition()` aus den Vorlesungsfolien sowie dem zusätzlichen Foliensatz zu Quicksort auf ISIS. Das von `partition()` zurückgegeben Pivotelement muss für diese Aufgabe das erste Element der Eingabeliste sein.

Listing 1: Pseudocode Quicksort Algorithmus (rekursiv)

```
1  QuickSort (list tosort)
2
3      if (tosort.first == tosort.last)
4          // lists with 0 or 1 elements are sorted by definition
5      else
6          list left, right
7          pivot ← Partition (tosort, left, right)
8
9          QuickSort(left)
10         QuickSort(right)
11
12         if (left.first == NULL)
13             tosort.first ← pivot
14         else
15             tosort.first ← left.first
16             left.last.next ← pivot
17
18         if (right.first == NULL)
19             pivot.next ← NIL
20             tosort.last ← pivot
21         else
22             pivot.next ← right.first
23             tosort.last ← right.last
```

Hinweis: Beachte, dass beim Einfügen in die rechte bzw. linke Teiliste kein neuer Speicher alloziert werden muss, sondern die jeweiligen `next`-Pointer entsprechend gesetzt werden.

Hinweis: Beachte für die Implementierung der Funktion `partition()`, dass sobald in der Liste weitere Elemente vorkommen, die denselben Wert wie das Pivotelement haben, diese in die rechte Teilliste eingefügt werden.
Das Pivotelement selbst ist nicht Teil der rechten bzw. linken Teilliste (s. Zusatzmaterial: Quicksort), sondern tritt nur als Rückgabewert der Funktion `partition` auf.

Die Eingabedatei enthält eine Liste der beliebtesten Passwörter und deren Häufigkeit im Format:
`Passwort Häufigkeit`

Lies die Eingabedatei ein und speichere die Passwörter und ihre Häufigkeiten gemeinsam als Element einer einfach verketteten Liste ab. Dein Programm bekommt den Pfad zur Eingabedatei als erstes Argument.

Deine `quick_sort` Implementierung nimmt diese Liste als Eingabe und gibt am Ende die Passwort-Häufigkeits-Paare nach aufsteigender Häufigkeit sortiert aus. Das Ausgabeformat soll dabei dem Format der Eingabedatei entsprechen.

Du darfst annehmen, dass die Eingabe diesem Format entspricht, und dass die Datei Leseberechtigung hat.

Listing 2: Vorgabe `main_quicksort.c`

```c
#include <stdio.h>
#include "quicksort.h"

int main(int argc, char** args)
{
   if (argc != 2)
   {
      printf("Nutzung: %s <Dateiname>\n",args[0]);
      return 1;
   }
   list mylist;
   init_list(&mylist);
   read_data(args[1],&mylist);
   qsort_list(&mylist);
   printf("Sortierte Liste:\n");
   print_list(&mylist);
   free_list(&mylist);
   return 0;
}
```

Codevorgabe:

Listing 3: Vorgabe `introprog_quicksort_vorgabe.c`

```c
/* === INTROPROG ABGABE ===
 * Blatt 10, Aufgabe 1
 * Tutorium: tXX
 * Gruppe: gXX
 * Gruppenmitglieder:
 *  - Erika Mustermann
 *  - Rainer Testfall
 * ========================
 */

#define _POSIX_C_SOURCE 200809L
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "quicksort.h"

/*****************************************************
 * Die benoetigten structs findet Ihr in quicksort.h *
 *****************************************************/

void init_list(list* mylist)
{
// HIER Liste initialisieren
}



// Diese Funktion fügt Listenelemente an die Liste an
void insert_list(list_element* le, list* mylist)
```



```c
{
    // HIER Code einfügen
}

// Speicher für Listenelemente wieder freigeben
void free_list(list* mylist)
{
    // HIER Code einfügen
}


// Namen, Zahlen Paare in Liste einlesen
void read_data(char* filename, list* mylist)
{
      // HIER Code einfügen:
        // * Speicher allozieren
        // * Daten in list_element einlesen
        // * insert_front benutzen um list_element in Liste einzufügen
}

// Liste teilen. Teillisten werden in left und right zurück gegeben
list_element* partition( list* input, list* left, list* right )
{
    // HIER Code einfügen:
    // parition() Funktion implementieren
}

// Hauptfunktion des quicksort Algorithmus
void qsort_list(list* mylist)
{
    // HIER Code einfügen
}

// Liste ausgeben
void print_list(list* mylist)
{
    // HIER Code einfügen:
    // * Laufe über die list_element in mylist und gebe sie aus.
}
```