Fakultät IV Elektrotechnik und Informatik
Einführung in die Programmierung WS 2019/2020
Manfred Hauswirth / Damien Foucard / Christoph Seifert

Aufgabenblatt 10

Ausgabe: *24.01.2020*
Abgabe: *04.02.2020 09:59*

Thema: *Quicksort auf einfach verketteten Listen*

Abgabemodalitäten

1. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des ZE-CM/eecsIT mittels `clang -std=c11 -Wall -g` kompilieren.
2. Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Nur wenn ein Test in Osiris angezeigt wird ist sichergestellt, dass die Abgabe erfolgt ist.
3. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben.
4. Die Abgabe erfolgt in folgendem Unterordner:
   `intoprog-wise1920/Studierende/<L>/<TU-Login>/Abgaben/Blatt<XX>`
   wobei `<L>` durch den ersten Buchstabe des TU-Logins und `<XX>` durch die Nummer des Aufgabenblattes zu ersetzen sind. Die Ordner werden automatisch angelegt sobald die Abgabe freigeschaltet wird.
5. Du darfst den Abgabeordner für das Blatt nicht selbst erstellen. Die Ordner erstellen wir kurz nach der Ausgabe der Aufgabe.
6. Um die Änderungen (z. B. neue Abgabeordner) vom Server abzuholen, musst Du den Befehl `svn update` im Wurzelverzeichnis (also im Verzeichnis `introprog-wise1920`) des Repositories ausführen.
7. Im Abgabeordner gelten einige restriktive Regeln. Dort ist nur das Einchecken von Dateien mit den in der Aufgabe vorgegebenen Namen erlaubt, ausserdem werden die Abgabefristen vom Server überwacht. Beachte eventuelle Fehlermeldungen beim SVN-Commit. Lade nur Dateien hoch, die Du selbst bearbeiten sollst, insbesondere also keine Vorgaben.
8. Der Dateiname bei Abgaben mit C-Code soll dem der Vorgaben entsprechen. Entferne ausschließlich `_vorgabe` aus dem Name. Bei Textaufgaben sind, wenn nicht anders angegeben, `.txt` Dateien zugelassen, die als Plaintext-Datei zu speichern sind (keine Word-Dateien umbenennen).
9. Es gibt einen Ordner `Arbeitsverzeichnis`, in dem Du Dateien für Dich ablegen kannst.
10. Die Ergebnisse der automatischen Tests kannst Du auf OSIRIS einsehen:
    https://osiris.ods.tu-berlin.de/

Aufgabe 1 Implementierung von Quicksort auf verketteten Listen (10 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. Das von `partition()` zurückgegebene 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          // Listen mit 0 oder 1 Element(en) sind per
5          // Definition sortiert
6      else
7          list left, right
8          pivot ← Partition (tosort, left, right)
9
10         QuickSort(left)
11         QuickSort(right)
12
13         if (left.first == NULL)
14             tosort.first ← pivot
15         else
16             tosort.first ← left.first
17             left.last.next ← pivot
18
19         if (right.first == NULL)
20             pivot.next ← NIL
21             tosort.last ← pivot
22         else
23             pivot.next ← right.first
24             tosort.last ← right.last
```

Hinweis: Beachte, dass beim Einfügen in die rechte bzw. linke Teilliste 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
#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)
{
    // 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_list 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
}
```

```
48 // Hauptfunktion des quicksort Algorithmus
49 void qsort_list(list* mylist)
50 {
51     // HIER Code einfügen
52 }
53
54 // Liste ausgeben
55 void print_list(list* mylist)
56 {
57     // HIER Code einfügen:
58     // * Laufe über die list_element in mylist und gebe sie aus.
59 }
```