Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2015/16
Feldmann / Semmler / Lichtblau / Streibelt / Pujol / Rost

Aufgabenblatt 6

letzte Aktualisierung: 16. Dezember, 18:33 Uhr

(614067d8c02993c60479bd07c695b6535ce413ea)

Ausgabe: Mittwoch, 9.12.2015
Abgabe: spätestens Freitag, 18.12.2015, 20:00

Thema: *Merge Sort, Teile und Herrsche*

Abgabemodalitäten

- Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des tubIT/IRB mittels `gcc -std=c99 -Wall` kompilieren.
- Abgaben erfolgen prinzipiell immer in Gruppen à 2 Personen, welche in den Tutorien festgelegt wurden. Einzelabgaben sind explizit als solche gekennzeichnet.
- Die Abgabe erfolgt ausschließlich über SVN. 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>/`

  wobei die Ordner von uns erstellt werden.
- Benutze für alle Abgaben von Programmcode das folgende Namensschema: `introprog_blatt0X_aufgabe0Y_Z.c`, wobei X durch die Blattnummer, Y durch die Aufgabe und Z durch die Unteraufgabe zu ersetzen ist.
  Beispiel: Aufgabe 1.2 wird zu: `introprog_blatt01_aufgabe01_2.c`
  Gib für jede Unteraufgabe maximal eine Quellcodedatei ab, es sei denn, die Aufgabenstellung erfordert explizit die Abgabe mehrerer Dateien pro Aufgabe.

  Benenne alle anderen Abgaben (Pseudocode, Textaufgaben) wie oben beschrieben. Die zugelassenen Abgabeformate sind PDF, ODT und Text (txt). Verwende auch hier eine Datei pro Aufgabe, nicht jedoch pro Unteraufgabe.

1. Aufgabe: Rekursive Implementierung Merge Sort (2 Punkte)

In dieser Aufgabe soll der **rekursive** "Teile und Herrsche" Algorithmus *Merge Sort* implementiert werden.

Implementiere die C Funktion `merge_sort()` sowie alle nötigen Hilfsfunktionen anhand des Pseudocodes, der in der Vorlesung vorgestellt wurde. Die Funktion `merge_sort` bekommt als Argumente die Startadresse eines Integerarrays sowie den Index des ersten Elements und die Länge des Arrays. Orientiere dich am Pseudocode aus Listing 1.

Listing 1: Pseudocode Merge Sort

```
1  MergeSort(Array A,p,r)        // Sortiere A von index p bis r
2      if p < r then
3          q ← floor((p+r)/2)    // Mitte mit Abrunden finden
4          MergeSort(A, p, q)    // Linke Seite sortieren
5          MergeSort(A, q+1, r)  // Rechte Seite sortieren
6          Merge(A, p, q, r)     // Seiten Zusammenfuehren
7
8  Merge(A,p,q,r)
9      Array B                               // Hilfsarray zum Mergen
10                                           // mit Laenge r-p+1
11     k←p                    // Hilfsvariable fuer linke Seite
12     m←q + 1                // Hilfsvariable fuer rechte Seite
13     i←1                    // Laufvariable fuer gemergtes Array
14
15     // Solange Eintraege in beiden Seiten vorhanden sind
16     while ( k ≤ q ) and ( m ≤ r)
17         if A[k] ≤ A[m] then // Eintrag auf linker Seite kleiner oder gleich
18             B[i] ← A[k]
19             k ← k+1
20         else                // Eintrag auf rechter Seite kleiner
21             B[i] ← A[m]
22             m ← m+1
23         i ← i+1             // Erhoehen der Laufvariable des gemergten Arrays
24
25     while ( k ≤ q )         // Kopiere linken "Rest"
26         B[i] ← A[k]
27         k ← k+1
28         i ← i+1
29
30     while ( m ≤ r )         // Kopiere rechten "Rest"
31         B[i] ← A[m]
32         m ← m+1
33         i ← i+1
34     j ← 1                   // Rueckkopieren der gemergten Eintraege
35
36     while ( j < i )
37         A[p + j - 1] ← B[j]  // Hinweis: j ist mit 1 initialisiert
38         j ← j+1
```

Verwende die Sortierfunktion in einem lauffähigen Programm. Das Programm bekommt als Argumente die maximale Länge des zu sortierenden Arrays und einen Dateinamen mit zu sortierenden Werten. Die Werte sollten mittels der vorgegebenen Funktion `read_array_from_file()` in ein Array eingelesen werden. Beachte, dass Du vor dem Einlesen entsprechend Speicher für das Array dynamisch allozieren musst.

Der Programmaufruf für ein Array mit maximal 20 Werten soll wie folgt aussehen:

```
./introprog_blatt06_aufgabe01 20 zahlen_unsortiert.txt
```

Ausgabe des Programms sind die sortierten Werte, wobei jede Zahl durch ein Leerzeichen getrennt sein muss. Also im Format:

```
1 2 3 ...
```

Die Implementierung soll sich an folgender Vorgabe orientieren:

Listing 2: Vorgabe `introprog_blatt06_aufgabe01_vorgabe.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "input_blatt06.h"
/*
Diese Funktion implementiert den rekursiven Mergesort Algorithmus auf einem
    ↪ Array.
Sie soll analog zum Pseudocode in Listing 1 implementiert werden.

    array:  Pointer auf den Beginn des Arrays
    first:  Index des ersten Elements
    len  :  Index des letzten Elements
*/

void merge(int* array, int start, int middle, int end)
{
// HIER Funktion merge() implementieren
}

void merge_sort(int* array, int first, int last)
{
 // HIER Funktion merge_sort() implementieren
}

/*
Hauptprogramm.

Liest Integerwerte aus einer Datei und gibt diese sortiert
im selben Format über die Standardausgabe wieder aus.

Aufruf: ./introprog_blatt06_aufgabe01 <maximale anzahl>  <dateipfad>
*/
int main (int argc, char *argv[])
{
    if (argc!=3){
        printf ("usage: %s <maximale anzahl> <dateipfad>\n", argv[0]);
        exit(2);
    }

    char *filename = argv[2];

    // Hier array initialisieren

    int len = read_array_from_file(array, atoi(argv[1]), filename);

    printf("Eingabe:\n");
    print_array(array, len);

    // HIER Aufruf von "merge_sort()"

    printf("Sortiert:\n");
    print_array(array, len);

    return 0;
}
```


Programmmaufruf

Listing 3: Programmbeispiel

```
> gcc -std=c99 -Wall introprog_blatt06_aufgabe01.c input_blatt06.c
              -o introprog_blatt06_aufgabe01
> ./introprog_blatt06_aufgabe01 20 zahlen_unsortiert.txt
```

2. Aufgabe: Iterative Implementierung Merge Sort (1 Punkt)

In dieser Aufgabe soll der **iterative** "Teile und Herrsche" Algorithmus *Merge Sort* implementiert werden.

Implementiere die C Funktion `merge_sort()` sowie alle nötigen Hilfsfunktionen anhand des Pseudocodes, der in der Vorlesung vorgestellt wurde. Die Funktion `merge_sort` bekommt als Argumente die Startadresse eines Integerarrays sowie den Index des ersten Elements und die Länge des Arrays. Orientiere dich am Pseudocode aus Listing 4.

Listing 4: Pseudocode Insertion Sort

```
IterativMergeSort(Array A,s,n)
    step←1
    while  step ≤ n do
        left←1
        while left≤n−step do
            middle←left + step − 1
            middle←min(middle,n)
            right←left + 2 ∗ step − 1
            right←min(right,n)
            merge(A, left, middle, right)
            left←left + 2∗step
        step←step∗2
```

Verwende die Sortierfunktion in einem lauffähigen Programm. Das Programm bekommt als Argumente die maximale Länge des zu sortierenden Arrays und einen Dateinamen mit zu sortierenden Werten. Die Werte sollten mittels der vorgegebenen Funktion `read_array_from_file()` in ein Array eingelesen werden. Beachte, dass Du vor dem Einlesen entsprechend Speicher für das Array dynamisch allozieren musst.

Das Programm soll für ein Array mit maximal 20 Einträgen wie folgt aufgerufen werden:

```
./introprog_blatt06_aufgabe02 20 zahlen_unsortiert.txt
```

Ausgabe des Programms sind die sortierten Werte, wobei jede Zahl durch ein Leerzeichen getrennt sein muss. Also im Format:

```
1 2 3 ...
```

Die Implementierung soll sich an folgender Vorgabe orientieren:

Listing 5: Vorgabe `introprog_blatt06_aufgabe02_vorgabe.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "input_blatt06.h"
/*
Diese Funktion implementiert den iterativen Mergesort Algorithmus auf einem
   ↪ Array.
Sie soll analog zum Pseudocode in Listing 4 implementiert werden.

    array:  Pointer auf den Beginn des Arrays
    first:  Index des ersten Elements
    len  :  Index des letzten Elements
*/

void merge(int* array, int start, int middle, int end)
{
// HIER Funktion merge() implementieren
}

void merge_sort(int* array, int first, int last)
{
 // HIER Funktion merge_sort() implementieren
}

/*
Hauptprogramm.

Liest Integerwerte aus einer Datei und gibt
diese sortiert im selben Format über die Standardausgabe wieder aus.

Aufruf: ./introprog_blatt06_aufgabe01 <maximale anzahl>  <dateipfad>
*/
int main (int argc, char *argv[])
{
    if (argc!=3){
        printf ("usage: %s <maximale anzahl>  <dateipfad>\n", argv[0]);
        exit(2);
    }

    char *filename = argv[2];

    // Hier array initialisieren

    int len = read_array_from_file(array, atoi(argv[1]), filename);

    printf("Eingabe:\n");
    print_array(array, len);

    // HIER Aufruf von "merge_sort()"

    printf("Sortiert:\n");
    print_array(array, len);

    return 0;
}
```


Programmmaufruf

Listing 6: Programmbeispiel

```
> gcc -std=c99 -Wall introprog_blatt06_aufgabe02.c input_blatt06.c
               -o introprog_blatt06_aufgabe02
> ./introprog_blatt06_aufgabe02 20 zahlen_unsortiert.txt
```