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

Aufgabenblatt 7

Ausgabe: *20.12.2019*
Abgabe: *14.01.2020 09:59*

Thema: *Mergesort, Teile und Herrsche*

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:
   `introprog-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 Rekursive Implementierung Mergesort (6,5 Punkte)

In dieser Aufgabe soll der **rekursive** "Teile und Herrsche" Algorithmus *Mergesort* 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 Mergesort

```
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 der Laenge r-p+1 zum Mergen
10     k ← p                   // Hilfsvariable fuer linke Seite
11     m ← q + 1               // Hilfsvariable fuer rechte Seite
12     i ← 1                   // Laufvariable fuer gemergtes Array
13
14     // Solange Eintraege in beiden Seiten vorhanden sind
15     while (k ≤ q) and (m ≤ r)
16         if A[k] ≤ A[m] then // Eintrag auf linker Seite kleiner
               ↪ oder gleich
17             B[i] ← A[k]
18             k ← k + 1
19         else                // Eintrag auf rechter Seite kleiner
20             B[i] ← A[m]
21             m ← m + 1
22         i ← i + 1           // Erhoehen der Laufvariable des
               ↪ gemergten Arrays
23
24     while (k ≤ q)           // Kopiere linken "Rest"
25         B[i] ← A[k]
26         k ← k + 1
27         i ← i + 1
28
29     while (m ≤ r)           // Kopiere rechten "Rest"
30         B[i] ← A[m]
```

```
        m ← m + 1
        i ← i + 1
    j ← 1  // Rueckkopieren der gemergten Eintraege

    while (j < i)
        A[p + j - 1] ← B[j] // Hinweis: j ist mit 1 initialisiert
        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_merge_sort_rekursiv 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_merge_sort_rekursiv_vorgabe.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "introprog_input_merge_sort.h"

/*
 * Diese Funktion fügt zwei bereits sortierte Arrays zu einem
 * sortierten Array zusammen
 *
 * array : Pointer auf das Array
 * first : Index des ersten Elements (Beginn) des (Teil-)Array
 * middle: Index des mittleren Elements des (Teil-)Array
 * last  : Index des letzten Elements(Ende) des (Teil-)Array
 */
void merge(int* array, int first, int middle, int last)
{
  // HIER Funktion merge() implementieren
}

/*
 * Diese Funktion implementiert den rekursiven Mergesort
 * Algorithmus auf einem Array. Sie soll analog zum Pseudocode in
 * Listing 1 implementiert werden.
 *
 * array: Pointer auf das Array
 * first: Index des ersten Elements des (Teil-)Array
 * last:  Index des letzten Elements des (Teil-)Array
 */
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_merge_sort_rekursiv <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);
```

```
63
64      return 0;
65  }
```

Programmaufruf:

Listing 3: Programmbeispiel

```
> clang -std=c11 -Wall introprog_merge_sort_rekursiv.c
  input_merge_sort.c -o introprog_merge_sort_rekursiv
> ./introprog_merge_sort_rekursiv 20 zahlen_unsortiert.txt
```

Aufgabe 2 Iterative Implementierung Mergesort (3,5 Punkte)

In dieser Aufgabe soll der **iterative** "Teile und Herrsche" Algorithmus *Mergesort* 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 Mergesort

```
IterativMergeSort(Array A,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_merge_sort_iterativ 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_merge_sort_iterativ_vorgabe.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "introprog_input_merge_sort.h"

/*
 * Diese Funktion fügt zwei bereits sortierte Arrays zu einem
 * sortierten Array zusammen
 *
 * array : Pointer auf das Array
 * first : Index des ersten Elemements (Beginn) des (Teil-)Array
 * middle: Index des mittleren Elements des (Teil-)Array
 * last  : Index des Letzten Elements (Ende) des (Teil-)Array
 */
void merge(int* array, int first, int middle, int last)
{
    // HIER Funktion merge() implementieren
}

/*
 * Diese Funktion implementiert den iterativen Mergesort
 * Algorithmus auf einem Array. Sie soll analog zum Pseudocode in
 * Listing 4 implementiert werden.
 *
 * array:  Pointer auf das Array
 * first:  Index des ersten Elements
 * last :  Index des letzten Elements
 */
void merge_sort(int* array, int first, int last)
{
    // HIER Funktion merge_sort() implementieren
}

```

```c
/*
 * Hauptprogramm.
 *
 * Liest Integerwerte aus einer Datei und gibt diese sortiert im
 * selben Format über die Standardausgabe wieder aus.
 *
 * Aufruf: ./introprog_merge_sort_rekursiv <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;
}
```

Programmaufruf:

Listing 6: Programmbeispiel

```
> clang -std=c11 -Wall introprog_merge_sort_iterativ.c
  input_merge_sort.c -o introprog_merge_sort_iterativ
> ./introprog_merge_sort_iterativ 20 zahlen_unsortiert.txt
```