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

Aufgabenblatt 7

letzte Aktualisierung: 22. Dezember, 22:02 Uhr

(78bcf805ab1959e774afa2cf5771b1c7c0da4449)

Ausgabe: Freitag, 22.12.2017
Abgabe: spätestens Mittwoch, 17.01.2018, 18:00

Thema: Merge Sort, Teile und Herrsche

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: 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 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]
31         m ← m + 1
32         i ← i + 1
33     j ← 1                   // Rueckkopieren der gemergten Eintraege
34
35     while ( j < i )
36         A[p + j - 1] ← B[j] // Hinweis: j ist mit 1 initialisiert
37         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
/* === INTROPROG ABGABE ===
 * Blatt 7, Aufgabe 1
 * Tutorium: tXX
 * Gruppe: gXX
 * Gruppenmitglieder:
 *  - Erika Mustermann
 *  - Rainer Testfall
 * ======================
 */

#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);

    return 0;
}
```

Programmmaufruf

Listing 3: Programmbeispiel

```
> gcc -std=c99 -Wall introprog_merge_sort_rekursiv.c input_merge_sort.c
              -o introprog_merge_sort_rekursiv
> ./introprog_merge_sort_rekursiv 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_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`

```c
/* === INTROPROG ABGABE ===
 * Blatt 7, Aufgabe 2
 * Tutorium: tXX
 * Gruppe: gXX
 * Gruppenmitglieder:
 *  - Erika Mustermann
 *  - Rainer Testfall
 * ========================
 */

#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
}

/*
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;
}
```

Programmmaufruf

Listing 6: Programmbeispiel

```
> gcc -std=c99 -Wall introprog_merge_sort_iterativ.c input_merge_sort.c
               -o introprog_merge_sort_iterativ
> ./introprog_merge_sort_iterativ 20 zahlen_unsortiert.txt
```