DYNAMISCHES SORTIEREN

INDINF05

6. NOVEMBER 2014

STEFAN ERCEG
4AHITT

Inhalt

Aufgabenstellung ... 1
Zeitabschätzung ... 2
Tatsächlicher Zeitaufwand ... 2
Designüberlegung ... 3
Dateien ... 3
Header-Datei „Sortieren“ ... 3
Bubblesort ... 3
Mergesort ... 3
Quicksort ... 3
Verarbeitung ... 4
Main ... 4
Things I’ve done ... 5
eigenes Headerfile ... 5
dynamische Speicherung ... 6
Lessons learned ... 7
Quellenangaben ... 7

Aufgabenstellung

Basierend auf der Aufgabe "IndInf04 - Mergesort" soll ein Projekt erstellt werden, welches mindestens drei Sortieralgorithmen anbietet (Quick-, Bubble-, Mergesort, etc.). Es soll dabei ein einfaches CLI-Menü erstellt werden, das die Anzahl der zu initialisierenden Elemente ermöglichen soll (dynamische Speicherverw.) sowie den gewünschten Sortieralgorithmus zur Verfügung stellt.

Es soll eine optionale Möglichkeit der Ausgabe der Zahlen (unsortierter, sortierter Vektor) implementiert werden. Wichtig ist aber die Ausgabe der benötigten Zeit der Sortierung (Verwendung von time.h). Das Programm soll erst durch eine entsprechende Eingabe (z.B. 'q') beendet werden.

Die einzelnen Sortieralgorithmen sollen entsprechend aufgeteilt in eigenen .c Files implementiert und über ein Header-File angesprochen werden.

Zeitabschätzung

Teilaufgabe	Benötigte Zeit
Code erstellen	200 Minuten
Protokoll schreiben	60 Minuten
Gesamt	**260 Minuten (4 h 20 min)**

Tatsächlicher Zeitaufwand

Teilaufgabe	Datum	Benötigte Zeit
Code erstellen	24.10.2014	40 Minuten
Code erstellen	03.11.2014	190 Minuten
Protokoll schreiben	06.11.2014	55 Minuten
Gesamt	06.11.2014	**285 Minuten (4h 45 min)**

Designüberlegung

Dateien

Header-Datei „Sortieren“

In der selbst erstellten Header-Datei „Sortieren“ befinden sich neben der Struct ebenfalls die Prototypen der Funktionen von jedem C-File der Aufgabe. Die Struct beinhaltet einen int-Pointer, bei dem die Zahlen in ein Array gefüllt werden, und mehrere Functionpointer, die in der main-Funktion benötigt werden. Natürlich befinden sich auch die benötigten Includes in der Datei, damit sie nicht von jeder Datei hinzugefügt werden müssen.

Bubblesort

In der Datei „Bubblesort“ befindet sich eine Funktion, die diesen Sortieralgorithmus ausführt. Dazu muss das zu sortierende Array und dessen Länge im Parameter übergeben werden. Die Funktion wurde von folgender Seite übernommen: http://de.wikibooks.org/wiki/Algorithmen_und_Datenstrukturen_in_C/_Bubblesort. Zuletzt wurde diese Seite am 24.10.2014 um 8 Uhr abgerufen.

Allgemeines zum Bubblesort

Beim Bubblesort werden immer zwei nebeneinander liegende Elemente verglichen und dann vertauscht, falls das rechte kleiner ist als das linke.

Mergesort

Die Funktion für den Mergesort wurde von folgender Seite übernommen http://de.wikibooks.org/wiki/Algorithmen_und_Datenstrukturen_in_C/_Mergesort und zuletzt am 24.10.2014 um 8 Uhr abgerufen. Als Parameter werden das zu sortierende Array und dessen Länge gefordert.

Allgemeines zum Mergesort

Beim Mergesort werden die Zahlen in einer Liste betrachtet und in kleinere Listen zerlegt, die dann jeweils für sich sortiert werden. Die sortierten kleinen Listen werden danach wieder zu einer Gesamtlist hinzugefügt.

Quicksort

Der in dieser Aufgabe implementierte Quicksort stammt von folgender Seite: http://rosettacode.org/wiki/Sorting_algorithms/Quicksort#C. Diese Seite wurde am 24.10.2014 um 8 Uhr zuletzt abgerufen. Wie bei den anderen beiden Sortieralgorithmen, wird auch hier das zu sortierende Array und dessen Länge als Parameter benötigt.

Allgemeines zum Quicksort

Beim Quicksort wird die Liste in zwei Teillisten unterteilt. Alle Elemente, die kleiner sind als das Pivot-Element, werden in die linke Teilliste, alle größeren Elemente in die rechte Teilliste gegeben. Quicksort zählt zu den schnellsten Sortieralgorithmen.

Verarbeitung

In dieser Datei werden 2 Methoden implementiert.

Die eine, "zufallszahlen", generiert eine vom Benutzer vorgegebene Anzahl an Zufallszahlen. Die andere, "output", gibt alle Zahlen, die im Array enthalten sind, unsortiert oder sortiert aus.

Main

In der Main wird ein CLI-Menü erstellt, welches dem Benutzer ermöglicht, die Anzahl der zu sortierenden Elemente einzugeben und den gewünschten Sortieralgorithmus auszuwählen.

Es werden ebenfalls Abfragen erstellt, bei denen der Benutzer gefragt wird, ob er den unsortierten und/oder den sortierten Zahlenvektor angezeigt haben möchte. Wenn mit "y" (yes) geantwortet wird, kann der Benutzer entscheiden, wie viele Zahlen er pro Zeile angezeigt haben möchte.

Die Ausgabe der benötigen Zeit der Sortierung ist ebenfalls Bestandteil des Programms.

Das Programm kann danach durch die Eingabe von q beendet werden.

Things I've done

eigenes Headerfile

Zu Beginn habe ich alle Includes angegeben, die bei dieser Aufgabe von den Dateien benötigt werden. Das Inkludieren von „time.h" war bei INDINF05 vor allem wichtig, da die benötigte CPU-Zeit für eine bestimmte Sortierung berechnet werden soll. Folgende Includes wurden hinzugefügt:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
```

Danach wurde die Struct erstellt, die uns ermöglicht, mehrere Datentypen in eine Gesamtform zu bringen. Der Functionpointer „sort" bei dieser Aufgabe war sehr praktisch, da man drei Sortieralgorithmen besitzt und diese dem Pointer in der main-Funktion einfach zuweisen kann.

```
struct zahlen {

	int* array;

	void (*sort) (int*, int);
	int* (*zufallszahlen) (int*, int);
	void (*output) (int*, int);

};
```

Damit der Funktionspointer „sort" auf die Funktion „mergesort" zeigt, führt man eine einfache Zuweisung aus:

```
struct zahlen z;
z.sort = mergesort;
```

Bei unseren bisherigen Übungen in „Industrielle Informatik" haben wir die Prototypen am Beginn desselben Files geschrieben. Nun haben wir gelernt, dass man diese einfach in ein eigenes Header-File schreiben kann und somit eine Übersicht für alle Dateien gewährleistet wird.

Bsp. - Prototypen für die Datei „Verarbeitung.c":

```
int* zufallszahlen(int*, int);
void output(int*, int, int);
```

dynamische Speicherung

Der Sinn dieser Übung war, den Umgang mit der dynamischen Speicherung in C zu erlernen. Da wir bei der Aufgabe „IndInf04“ eine fixe Größe für das int-Array im Struct verwendet haben, wollten wir das Gesamte effizienter programmieren.

Der Befehl „malloc“ dient dabei dazu, einen Speicherblock in einer bestimmten Größe zu reservieren.

Mittels „realloc“ kann man diesen entsprechenden Speicher danach verkleinern bzw. vergrößern.

Möchte man den vom Betriebssystem angeforderten Speicherblock erneut wieder freigeben, da man ihn nicht mehr benötigt, verwendet man den Befehl „free“.

Lessons learned

- ➔ Erstellen von eigenen Headerfiles
- ➔ dynamische Speicherreservierung mittels „malloc“
- ➔ Herausfinden der verbrauchten CPU-Zeit bei bestimmten Berechnungen mittels „clock“
- ➔ Wissen über Functionpointer erweitert

Quellenangaben

Funktion „bubblesort“:
http://de.wikibooks.org/wiki/Algorithmen_und_Datenstrukturen_in_C/_Bubblesort

Funktion „mergesort“:
http://de.wikibooks.org/wiki/Algorithmen_und_Datenstrukturen_in_C/_Mergesort

Funktion „quicksort“:
http://rosettacode.org/wiki/Sorting_algorithms/Quicksort#C