Studnet Managemant System

Finale Aufgabe

Gruppenmitglieder

Marcel Anders
Simon Baur
Marian Mutschler

Programmieren in C
Frau Goertz

2021

Inhaltsverzeichnis

Kapitel 1

Allgemein

1.1 Allgmeines

Im Rahmen des Programmieren in C Unterrichts an der DHBW Mannheim, sollten wir in Gruppen eine Aufgabe erledigen. Diese Gruppenarbeit hatte einen größeren Umfang um somit ein messbare Leistung der Studenten zu erheben. Die Aufgabe war es ein Programm zu entwickeln welches den Anforderungen in *1.2 Intension / Aufgaenstellung* entspricht.
Für das Projekt wurde ein Deadline festgelegt der 30 Dez. 2021. Innerhalb dieses Zeitraums müssen alle Anforderungen erfüllt sein.

1.2 Intension / Aufgaenstellung

- Erstelle eine Studentenstruktur mit folgenden Inhalten: Nachname, Matrikelnummer, Start Datum, End Datum, Geburstdatum
- Erstelle eine Datumsstruktur (day month year) für das Geburstdatum und einschreibungsdatum etc.
- Schreibe eine Funktion inputStudent: Bei dem der Benutzer alle relevanten Daten zu einem Studenten eingeben kann.
- Schreibe eine Funktion addStudent: bei der ein Student hinzugefügt wird.
- Schreibe eine Funktion in der die Anzahl der gespeicherten Studenten zurück gegeben werden soll.
- Schreibe eine Funktion printStudent(Matrikelnummer), mit der ein Student auf dem Bildschirm ausgegeben werden soll.
- Schreibe eine Funktion printAllStudents, mit der alle Studenten Alphabetisch auf dem Bildschirm ausgegeben werden sollen.

- Schreibe eine Funktion menue, über die die Verschienen funktionen aufgerufen werden sollen.
- Schreibe eine funktion deleteStudent(matrikelnummer), welche einen Studenten löscht.
- Schreibe eine Funktion save, die alle gespeicherten Studenten in eine Datei speichert. Diese Funktion soll beim beenden des Programms automatisch aufgerufen werden und soll somit nicht im menue auftauchen
- Schreibe eine Funktion read die eine Datei ausliest und alle Studenten daraus ins Programm laed. Diese Funktion soll beim start des Programms automatisch aufgerufen werden und soll somit nicht im menue auftauchen
- Die Aufgabe soll mithilfe von Verketteten Listen gelöst werden.

Kapitel 2

Student Management System

In diesem Kapitel werden wir auf die einzelnen Projektschritte eingehen. Jede verwendete bzw. entwickelte Funktion wird im folgenden erleutert. Die einzelnen Funktionen können in der *main.c* eingesehen werden. Diese liegt der Abgabe bei.

2.1 Includes

In diesem Schritt binden wir alle Bibliotheken ein. Wir haben verschiedene Hilfsbibliotheken eingbeunden. Im folgenden sind alle Bibliotheken zu sehen.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include <windows.h>
```

2.2 Umlaute

Im folgenden werden wir alle Umlaute definieren damit diese in der Konsole dargestellt werden können. In diesem Fall haben wir die Umlaute "ß, ö, ä, ü, Ü"deklariert.

```
#define sss 0xe1
#define oe 0x94
#define ae 0x84
#define ue 0x81
#define UE 0x9a
```

2.3 Farben

Im folgenden werden alle Farben deklariert, damit diese in der Konsole ausgegeben werden können.

```
#define BLACK "\x1b[30m"
#define RED "\033[0;31m"
#define GREEN "\x1b[32m"
#define YELLOW "\x1b[33m"
#define BLUE "\x1b[34m"
#define MAGENTA "\x1b[35m"
#define CYAN "\x1b[36m"
#define WHITE "\x1b[37m"
#define RESET "\033[0m"
```

2.3.1 Farbzuweisungen

Hierbei ist zu beachten das *ERR RED* in der *.exe Anwendung nur mit <windows.h> funktioniert. Ebenfalls muss man system(color...) verwenden. Ansonsten ist dies nicht möglich.

```
#define ERR RED
#define SUCCESS GREEN
#define IMPORTANTTEXT CYAN
```

2.4 Struct Student

Wir haben hier das Struct Student angelegt.

```
struct student{
    char *surname;
    int matrikelnummer;
    struct date startdate;
    struct date exitdate;
    struct date birthdate;
    struct student *previous;
    struct student *next;
}*start=NULL, *end=NULL;
```

2.5 Sonstige Sonderzeichen

Im folgenden werden die Elemente für eine saubere und auch schöne Darstellung der Tabellen in der Konsole definiert.

```
#define MENU_ARROW 0x1a
#define SPACE 0x20
#define HORIZONLINE 0xcd
#define VERTICALLINE 0xba
#define CROSS 0xce
#define CORNERDOWNLEFT 0xc8
#define CORNERDOWNRIGHT 0xbc
#define CORNERUPLEFT 0xc9
#define CORNERUPRIGHT 0xbb
#define TCROSSUP 0xca
#define TCROSSDOWN 0xcb
#define TCROSSRIGHT 0xcc
#define TCROSSLEFT 0xb9
```

2.6 Struct Date

In diesem Schritt haben wir das Struct Date erstellt.

```
struct date{
      unsigned int day;
      unsigned int month;
      unsigned int year;
};
```

2.7 void wait

Wartet auf eine interaktin des Users.

2.8 int getLength

Checking the length of a integer. A simple example: 10 = 2, 100=3. Also it checks the integer isn't eqal 0 beacuse in this case we get some errors. Überprüft die länge des *integer*. Als einfaches Beispiel 10 = 2, 100 = 3. Ebenfalls wird überprüft ob der *integer* ungleich 0 ist. Falls der *integer* 0 wäre würde hier ein Fehler abgefangen werden.

2.9 checkDate

Hier überprüfen wir den Userinput für das Datum. Das Ziel ist es dafür zu sorgen, dass es sich um ein echtes Datum handelt. Somit können Fehler des Users ausgeschlossen werden.

2.10 void addStudent()

Diese Funktion bekommt lediglich Pointer übergeben. Die übergebenen Pointer haben die Aufgabe die Eingabe des User in die Liste hinzuzufügen. Im folgenden gehen wir auf signifikante Codebausteine ein:

```
struct student *now, *before;
```

Dabei ist *now* ein Pointer der auf den aktuellen Listeneintrag zeigt. Ebenfalls haben wir hier noch den Pointer *before* der auf den Eintrag vor dem aktuellen Eintrag in der Liste zeigt.

2.11 void inputstudent(void)

Diese Funktion erwartet eine *void* Übergabe. Die Funktion gibt dem User die Möglichkeit alle Studentendaten einzugeben. Durch Eingabeeinschränkungen wird dafür gesorgt das der User möglichst wenig Fehler macht.
Bei der Eingabe der Matrikelnummer wird überprüft ob diese sich bereits in der Liste befindet.
Ebenfalls wird noch geprüft ob die Bedingung
Geburtstag < *Eintrittsdatum* < *Austrittsdatum* Zutrifft. Falls hierbei ein Wiederspruch vorliegt wird ein Fehler für den User Ausgeworfen.

2.12 int countStudent(void)

Hierbei wird die Anzahl der gespeicherten Studenten zurückgegeben.

2.13 void printStudent(struct student *now)

Hierbei wird die Übergabe eines *struct student* erwartet. Wenn dies der Fall ist wird der gewünschte Student mit der gesuchten Matrikelnummer ausgegeben.

2.14 void deleteStudent(struct student *del)

Diese Funktion hat die Aufgabe einen Studenten mit der gewünschten Matrikelnummer aus der Liste zu löschen. Hierbei wird der Funktion der Pointer des gewünschten Studenten übergeben. Damit wird sichergestellt das auch der richtige Student entfernt wird.

2.15 struct student

```
struct student *merge(struct student *list1, struct student *list2){}
```

Hier benutzen wir die *merge sort* in der *top down* variante. Dazu bekommt man zwei Pointer auf den Start zweier Listen, dabei sollen diese beisen Listen sortiert und zu einer Liste zusammengesetzt werden. Ist das geschehen wird ein Pointer zurückgegeben, welcher auf den Start der neuen Liste zeigt.

2.15.1 struct student // msort

```
struct student *msort(struct student *sort_start)
```

Bekommt einen Pointer übergeben. Ab diesem Pointer wird die Liste sortiert. Hierbei ist zu beachten, dass wenn der Pointer gleich *NULL* ist oder die Liste auf die der Pointer verweist nur ein Eintrag ist, die Liste bereits sortiert ist.

```
if ((sort_start==NULL) || (sort_start->next==NULL)) return sort_start
  ;
```

Wie folgt wird die Liste geteilt, damit man Sie sortieren kann. Daher suchen wir sie mit einer *for* Schleife ab. Dabei geht *now* immer einen Eintrag weiter, währen *after* immer zwei Einträge weiter geht, wenn z.B. *after->next==NULL* ist wird die Schleife abgebrochen, weil after dann am letzen Eintrag angekommen ist.

```
for (now=sort_start, after=sort_start->next; after && after->next;
  after=after->next->next) now=now->next;
```

2.16 void sort(void)

Hier wird die Liste sortiert. Dabei wird sie wie eine einfach verkettete Liste behandelt.

2.17 void printAllStudents(void)

Diese Funktion gibt alle Studenten in einer Liste aus und formatiert diese.

2.18 void read(void)

Hierbei handelt es sich um eine Kernfunktion der Anwendung. Diese Funktion ermöglicht es eine *CSV* Datei einzulesen. Hierbei wird klassisch zwischen der ersten und der letzten Zeile unterschieden. Die erste Zeile stellt die Headerzeile dar. Es wird immer eine ganze Zeile eingelesen. Jeder Teil der Zeile wird immer als *String* eingelesen. Als Trennzeichen wird das *Komma* genommen. Ebenfalls wird der *String* wie in *inputStudent()* auf seine Richtigkeit überprüft. Bei dieser Funktion hatten wir das problem das wenn die CSV Datei eine leere letzte Zeile hatte, die letze Zeile doppelt eingelesen haben. Um das zu beheben schauen wir ob die letze und vorletze die selbe Martikelnummer haben. Falls dies der Fall ist wird die letzte Zeile wieder rausgelöscht.

2.19 void save(void)

Hier werden alle Studenten in der *CSV* Datei gespeichert. Wenn erfolgreich gespeichert wurde wird der Speicher wieder mit dem Aufruf von *free()* freigegeben.

2.20 struct student *inputSearch(void)

Diese Funktion lässt den Nutzer eine Matrikelnummer eingeben, es wird dann in der Liste gesucht ob es einen Eintrag gibt, es wird ein Pointer auf das Element zurück gegeben, wenn eins gefunden wurde, NULL wenn die Liste leer ist oder nichts gefunden wurde.

2.21 int menu(void)

Hat die schlichte Aufgabe das Menü abzubilden.

Kapitel 3

Main

3.1 Grundlegendes

Die main Funktion verbindet alle Teilfunktionen und übernimmt die Gestaltung in der Konsole.

```
int main(void){
  system("color");
  read();
  system("cls");
  int select;
  struct student *now;
  do{
      select=menu();
      switch(select){
      case 0:
      system("cls");
          printf("\t\t%c", CORNERUPLEFT); for(int i=1;i<=MENUMAX;i++)
   printf("%c", HORIZONLINE); printf("%c\n", CORNERUPRIGHT);
          inputStudent();
      break;
      case 1:
          system("cls");
          printf("\t\t%c", CORNERUPLEFT); for(int i=1;i<=MENUMAX;i++)
   printf("%c", HORIZONLINE); printf("%c\n", CORNERUPRIGHT);
          int len=3;
          int count=countStudent();
          len=4-getLength(count);
          printf("\t\t%c Es befinden sich " IMPORTANTTEXT "%d" RESET,
   VERTICALLINE, count);  printf(" Eintr%cge in der Datenbank!     ",
   ae); for(int i=0;i<len;i++) printf(" "); printf("%c\n",
  VERTICALLINE);
          printf("\t\t%c", CORNERDOWNLEFT); for(int i=1;i<=MENUMAX;i
  ++) printf("%c", HORIZONLINE); printf("%c\n", CORNERDOWNRIGHT);
          wait();
      break;
      case 2:
          system("cls");
```

```
            printf("\t\t%c", CORNERUPLEFT); for(int i=1;i<=91;i++)
    printf("%c", HORIZONLINE); printf("%c\n", CORNERUPRIGHT);
28          printf("\t\t%c Studenten suchen!\n", VERTICALLINE);
            now=inputSearch();
30          if(now){
                printStudent(now);
32          }
            wait();
34      break;
        case 3:
36          system("cls");
            printAllStudents();
38          wait();
        break;
40      case 4:
            system("cls");
42          printf("\t\t%c", CORNERUPLEFT); for(int i=1;i<=91;i++)
    printf("%c", HORIZONLINE); printf("%c\n", CORNERUPRIGHT);
            printf("\t\t%c Studenten l%cschen!\n", VERTICALLINE, oe);
44          now=inputSearch();
            if(now){
46              deleteStudent(now);
            }
48          wait();
            select=0;
50      break;

52      case 5:
        break;
54

        default:
56          printf("\t\t%c " ERR "!Fehler select hat den ung%cltigen
    Wert '%d'!\n" RESET, VERTICALLINE, ue, select);
        break;
58      }
    }while(select!=5);
60  save();
    system("cls");
62  return 0;
}
```