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

Aufgabenblatt 3

letzte Aktualisierung: 18. November, 18:02 Uhr

(7fa7639aa243e32832de1da9123841fa89ac5012)

Ausgabe: Mittwoch, 18.11.2015
Abgabe: spätestens Freitag, 27.11.2015, 16:00

Thema: *Listen & Datenstrukturen*

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.

A. Aufgabe (Tut): Datenstruktur struct

Hinweis: Alle Aufgaben die mit *(Tut)* beginnen, sind unbewertet und sollen nicht abgegeben werden.

Was ist ein **`struct`**? Wir erklären das **`struct`** am Beispiel einer Filmdatenbank. Jeder Film ist als eigenes Element gespeichert.

Listing 1: struct _movie

```
1 struct movie {
2     char *title;
3     char *director;
4     int year;
5 };
```

Mithilfe des **`struct`** ist es uns möglich mehr Ordnung in die Datenbank zu bringen. Dabei wird auf Elemente des **`struct`** via dem Punkt-Operator zugegriffen.

Listing 2: Der naive Ansatz

```
1     // matrix
2     char *title1 = "The Matrix";
3     char *director1 = "Wachowski";
4     int year1 = 1999;
```

Listing 3: Besser

```
1     struct movie matrix;
2     matrix.title = "The Matrix";
3     matrix.director = "Wachowski";
4     matrix.year = 1999;
```

B. Aufgabe (Tut): Datenstruktur struct und Pointer

Hinweis: Alle Aufgaben die mit *(Tut)* beginnen, sind unbewertet und sollen nicht abgegeben werden.

Dank des **`struct`** werden Funktionsaufrufe vereinfacht. Dabei kann das **`struct`** entweder direkt oder via Pointer erfolgen. Im zweiten Fall muss erst der Pointer mithilfe des Stern-Operator aufgelöst werden und dann kann erst auf Elemente über den Punkt Operator zugegriffen werden. Eine alternative Schreibweise ist der Pfeil Operator.

Listing 4: Funktionsaufruf

```
1  void print_movie1(struct movie film) {
2      printf("Film\n====\n");
3      printf(" Titel: %s\n", film.title);
4      printf(" Regisseur: %s\n", film.director);
5      printf(" Jahr: %d\n", film.year);
6  }
7  void print_movie2(struct movie* film) {
8      printf("Film\n====\n");
9      printf(" Titel: %s\n", (*film).title);
10     printf(" Regisseur: %s\n", (*film).director);
11     printf(" Jahr: %d\n", (*film).year);
12 }
13 void print_movie3(struct movie* film) {
14     // Statt (*struct).variable geht kürzer struct->variable
15     printf("Film\n====\n");
16     printf(" Titel: %s\n", film->title);
17     printf(" Regisseur: %s\n", film->director);
18     printf(" Jahr: %d\n", film->year);
19 }
```

C. Aufgabe (Tut): Datenstruktur struct und Arrays

Hinweis: Alle Aufgaben die mit *(Tut)* beginnen, sind unbewertet und sollen nicht abgegeben werden.

struct erlauben erlauben die Zusammenfassung von Daten und eine einfachere Weiterverarbeitung. Das wird gerade deutlich wenn wir versuchen die Filme in Arrays abzulegen. Im naiven Fall wird jeder Film auf drei Arrays aufgeteilt. Wenn wir **struct** benutzen, benötigen wir nur eines. Hier wird auch deutlich wie viel Code wir einsparen können.

Listing 5: Naiv: Drei separate Arrays

```
1     char *titles[4];
2     titles[0] = title1;
3     titles[1] = title2;
4     titles[2] = title3;
5     titles[3] = title4;
6     char *directors[4];
7     directors[0] = director1;
8     directors[1] = director2;
9     directors[2] = director3;
10    directors[3] = director4;
11    int years[4];
12    years[0] = year1;
13    years[1] = year2;
14    years[2] = year3;
15    years[3] = year4;
```

Listing 6: Besser: Alles in einem

```
1     struct movie filme[4];
2     filme[0] = matrix;
3     filme[1] = existenz;
4     filme[2] = floor;
5     filme[3] = weltamdraht;
```

D. Aufgabe (Tut): next Pointer und Dynamischer Speicher

Hinweis: Alle Aufgaben die mit *(Tut)* beginnen, sind unbewertet und sollen nicht abgegeben werden.

Bis jetzt war es nicht möglich für Filme Fortsetzungen anzugeben. In einem neuen Codebeispiel ändern wir das. Im folgenden **struct** wird ein Pointer eingebaut `next`, der auf den nächsten Film zeigt.
Um uns dabei jedes Mal das tippen von **struct** `_movie` zu ersparen, spezifizieren wir dieses Mal mithilfe von **typedef**, dass `movie` dasselbe ist wie **struct** `_movie`. (Den Unterstrich haben wir zur besseren Unterscheidung eingeführt.) Wir können diese Ausdrücke deshalb äquivalent verwenden.

Listing 7: Neues struct

```
1  typedef struct _movie movie;
2
3  struct _movie {
4      char *title;
5      char *director;
6      int year;
7      movie* next;
8  };
```

In der Hausaufgabe wird der Speicher für **struct** meistens dynamisch alloziert und das wollen wir hier auch machen.

Listing 8: Speicher reservieren #1

```
1     movie *matrix = construct_movie("The Matrix", "Wachowski", 1999);
2     movie *matrix2 = construct_movie("The Matrix Reloaded", "Wachowski"
         ↪ , 2003);
3     movie *matrix3 = construct_movie("The Matrix Revolutions", "
         ↪ Wachowski", 2003);
```


Die Zuweisung selbst wird hier von einer Funktion übernommen:

Listing 9: Speicher reservieren #2

```
1  movie* construct_movie(char* title, char* director, int year) {
2      movie* film = (movie*) malloc(sizeof(movie));
3      film->title = title;
4      film->director = director;
5      film->year = year;
6      film->next = NULL; // Pointer wird später gesetzt.
7      return film;
8  }
```

Frage: Was ist der Vorteil davon **struct** dynamisch zu allozieren?

Nun können wir die Fortsetzungen angeben, wie hier am Beispiel der Matrix-Trilogie gezeigt wird.

Listing 10: next Pointer zuweisen

```
1     matrix->next = matrix2; // Pointer zum Sequel
2     matrix2->next = matrix3; // Pointer zum Sequel
3     matrix3->next = NULL; // Letzter Film (hoffentlich...)
```

Natürlich muss der dynamische Speicher auch wieder freigegeben werden.

Listing 11: Speicherfreigabe

```
1     free(matrix);
2     free(matrix2);
3     free(matrix3);
```

E. Aufgabe (Tut): Zugriff auf das nächste Element

Hinweis: Alle Aufgaben die mit *(Tut)* enden, sind unbewertet und sollen nicht abgegeben werden.

Wir wollen nun mit dem Titel eines Film auch den Titel der Fortsetzung ausgeben. Damit müssen wir auf nächste Element zugreifen. Wie machen wir das?
Der Titel des Film ist wie gehabt mit `(*film).titel` oder `film->titel` erreichbar. Die Fortsetzung ist wie der Titel mit `(*film).next` oder `film->next` erreichbar. Damit ist der Titel der Fortsetzung via `(*(*film).next).titel` oder `film->next->titel` ansprechbar. Aufgrund der Kürze entscheiden wir uns für den Pfeil-Operator.

Listing 12: Sequel ausgeben

```
1  void print_sequel(movie* film) {
2      printf("Film\n====\n");
3      printf("Titel: %s\n", film->title);
4      printf("Titel Sequel: %s\n", film->next->title);
5  }
```

Was passiert wenn statt der Fortsetzung von dem ersten oder zweiten Matrix Film, die Fortsetzung des dritten Matrix Film ausgegeben wird?

F. Aufgabe (Tut): Datenstruktur struct und Schleifen

Hinweis: Alle Aufgaben die mit *(Tut)* enden, sind unbewertet und sollen nicht abgegeben werden.

In dieser Aufgabe sind bis jetzt drei Filme miteinander verknüpft. Statt nun auf jede Fortsetzung direkt zuzugreifen können wir auch mithilfe einer Schleife alle Filme ausgeben. Diese Schleife unterscheidet sich von unseren bisherigen, da kein Counter hochgezählt wird, sondern wir uns direkt von einem Film zum nächsten bewegen.

Listing 13: Schleife über eine Liste

```
1 void print_movies(movie* film) {
2     printf("Film\n====\n");
3
4     movie* film2 = film;
5     while(film2 != NULL) {
6         printf("␣title:␣%s\n", film2->title);
7         film2 = film2->next;
8     }
9 }
```


1. Aufgabe: Eine Büchersammlung als Liste (2 Punkte)

Ziel der Aufgabe ist es Daten zu Büchern aus einer Datei in eine einfach verkettete Liste zu laden und diese mithilfe einer vorgegebenen Funktion auszugeben.
Bei den Daten handelt es sich um den Titel, den Autor, das Erscheinungsjahr und die ISBN des Buchs. Diese Daten werden dabei aus der Datei `buecherliste.txt` eingelesen.

Listing 14: buecherliste.txt

```
1 Neuromancer;William Gibson;1984;9780006480419;
2 Count Zero;William Gibson;1986;9780441117734;
3 Mona Lisa Overdrive;William Gibson;1988;9780553281743;
```

Es müssen die folgende Datenstruktur und Funktionen implementiert werden:

- `struct element`
- `element *insert_at_begin(element *first, element *new_elem)`
- `element *construct_element(char *title, char* author, int year, long long int isbn)`
- `void free_list(list *alist)`

Dagegen dürfen die `main` Funktion und die übrigen Funktionen nicht geändert werden. Auch dieses Mal wird wieder eine Bibliothek eingebunden (`input_blatt03.c` und `input_blatt03.h`), wie aus dem folgenden beispielhafter Programmaufruf ersichtlich wird:

Listing 15: Programmbeispiel

```
1  > gcc -std=c99 -Wall introprog_blatt03_aufgabe01.c input_blatt03.c \
2        -o introprog_blatt03_aufgabe01
3  > ./introprog_blatt03_aufgabe01
4  Meine Bibliothek
5  ================
6
7  Buch 1
8      Titel: Mars Plus
9      Autor: Frederick Pohl
10     Jahr:  1994
11     ISBN:  9780671876050
12 Buch 2
13     Titel: Man Plus
14     Autor: Frederick Pohl
15     Jahr:  1976
16     ISBN:  9780765321787
17 Buch 3
18     Titel: Brave New World Revisited
19     Autor: Aldous Leonard Huxley
20     Jahr:  1958
21     ISBN:  9780099458234
22 [etc.]
```

Die Aufgabe muss den folgenden Anforderungen genügen:

- Das `struct _element` muss die folgenden Variablen beinhalten:

  - Ein **char** Array `title`, statisch für die Größe 255 (oder `MAX_STR`) reserviert.
  - Ein **char** Array `author`, statisch für die Größe 255 (oder `MAX_STR`) reserviert.
  - Eine **int** Variable `year`.
  - Eine **long long int** Variable `isbn`.
  - Ein Pointer `next` auf das nächste Element.

- Neue Elemente sollen stets an den Anfang der Liste platziert werden, sodass das neue Element jeweils die Stelle von `first` einnimmt.
- Der bestehende Code, außerhalb der geforderten Änderungen, darf nicht verändert werden. Insbesondere dürfen die Funktionen `construct_list`, `read_list` und `main` und die Datenstruktur **struct** `_list` (inkl. dem **typedef**) nicht verändert werden.
- Der Speicher soll dynamisch reserviert und restlos (auch `*list alist`) freigegeben werden.

Benutze die folgende Codevorgabe:

Listing 16: Vorgabe `introprog_blatt03_aufgabe01_vorgabe.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "input_blatt03.h"

/* Bewirkt, dass statt 'struct _element' auch 'element' verwendet werden kann . */
typedef struct _element element;
/* Bewirkt, dass statt 'struct _list' auch 'list' verwendet werden kann. */
typedef struct _list list;

struct _list {      /* Separater Wurzelknoten */
    element *first; /* Anfang/Kopf der Liste */
    int count;      /* Anzahl der Elemente */
};

/* HIER struct element implementieren. */

/* Fuege ein Element am Anfang der Liste an, sodass das neue Element immer das
 * erste Element der Liste ist.
 * Wenn die Liste leer ist soll das Element direkt an den Anfang platziert
 * werden.
 *
 * first    - Erstes Element (bzw. Anfang) der Liste
 * new_elem - Neues Element das in die Liste eingefuegt werden soll.
 *
 * Gib einen Pointer auf den neuen Anfang der Liste zurueck.
 */
element *insert_at_begin(element *first, element *new_elem) {
    /* HIER implementieren. */
}

/* Kreiere ein neues Element mit dynamischem Speicher.
 *
 * title     - Der Titel des Buches
 * author    - Autor des Buches
 * year      - Erscheinungsjahr des Buches
 * isbn      - ISBN des Buches
 *
 * Gib einen Pointer auf das neue Element zurueck.
 */
element *construct_element(char *title, char* author, int year, long long int isbn) {
    /* HIER implementieren. */
}

/* Gib den der Liste und all ihrer Elemente zugewiesenen Speicher frei. */
void free_list(list *alist) {
    /* HIER implementieren. */
}

/* Lese die Datei ein und fuege neue Elemente in die Liste ein
 * _Soll nicht angepasst werden_
 * */
void read_list(char* filename, list *alist) {
    element* new_elem;

    char title[MAX_STR];
    char author[MAX_STR];
    int year;
    long long int isbn;
    while(read_line(filename, title, author, &year, &isbn) == 0) {
        new_elem = construct_element(title, author, year, isbn);
        alist->first = insert_at_begin(alist->first, new_elem);
        alist->count++;
    }
}

/* Erstelle die Liste:
 *  - Weise ihr dynamischen Speicher zu
 *  - Initialisiere die enthaltenen Variablen
 * _Soll nicht angepasst werden_
 */
list* construct_list() {
    list *alist = malloc(sizeof(list));
    alist->first = NULL;
    alist->count = 0;
    return alist;
}

/* Gib die Liste aus:
 * _Soll nicht angepasst werden_
 */
void print_list(list *alist) {
    printf("Meine Bibliothek\n================\n\n");
    int counter = 1;
    element *elem = alist->first;
    while (elem != NULL) {
        printf("Buch %d\n", counter);
```

```
88          printf("\tTitel:␣%s\n", elem->title);
89          printf("\tAutor:␣%s\n", elem->author);
90          printf("\tJahr:␣␣%d\n", elem->year);
91          printf("\tISBN:␣␣%lld\n", elem->isbn);
92          elem = elem->next;
93          counter++;
94      }
95  }
96
97  /* Main Funktion
98   * _Soll nicht angepasst werden_
99   */
100 int main() {
101     list *alist = construct_list();
102     read_list("buecherliste.txt", alist);
103     print_list(alist);
104     free_list(alist);
105     return 0;
106 }
```

2. Aufgabe: Eine Büchersammlung als sortierte Liste (1 Punkt)

Die Elemente sollen nun aufsteigend sortiert nach ISBN in die einfach verkettete Liste eingetragen werden. Übernimm die in der letzten Aufgabe entwickelten Funktionen. Es muss nun folgende Funktion implementiert werden:

- `element* insert_sorted(element *first, element *new_elem)`

Der Rest des Codes soll nicht angepasst werden. Auch dieses Mal wird wieder eine Bibliothek eingebunden (`input_blatt03.c` und `input_blatt03.h`), wie aus dem folgenden beispielhafter Programmaufruf ersichtlich wird:

Listing 17: Programmbeispiel

```
1  > gcc -std=c99 -Wall introprog_blatt03_aufgabe02.c input_blatt03.c \
2        -o introprog_blatt03_aufgabe02
3  > ./introprog_blatt03_aufgabe02
4  Meine Bibliothek
5  ================
6
7  Buch 1
8      Titel: Neuromancer
9      Autor: William Gibson
10     Jahr:  1984
11     ISBN:  9780006480419
12 Buch 2
13     Titel: Burning Chrome
14     Autor: William Gibson
15     Jahr:  1986
16     ISBN:  9780060539825
17 Buch 3
18     Titel: Interface
19     Autor: Neal Stephenson
20     Jahr:  1994
```



```
21     ISBN:  9780099427759
22 [etc.]
```

Die Aufgabe muss den folgenden Anforderungen genügen:

- Die Anforderungen aus der vorherigen Aufgabe gelten, mit Ausnahme der Regel zur Ordnung der Elemente in der Liste, hier ebenfalls.
- Neue Elemente sollen so in die Liste eingefügt werden, dass die Element aufsteigend nach ISBN sortiert sind. (D.h. zu jedem Zeitpunkt gilt das erste Buch hat die kleinste ISBN und jedes darauffolgende Buch hat eine größer werdende ISBN.)

Benutze die folgende Codevorgabe:

Listing 18: Vorgabe `introprog_blatt03_aufgabe02_vorgabe.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "input_blatt03.h"
5
6  /* Bewirkt, dass statt 'struct _element' auch 'element' verwendet werden kann
       ↪ . */
7  typedef struct _element element;
8  /* Bewirkt, dass statt 'struct _list' auch 'list' verwendet werden kann. */
9  typedef struct _list list;
10
11 struct _list {      /* Separater Wurzelknoten */
12     element *first; /* Anfang/Kopf der Liste */
13     int count;      /* Anzahl der Elemente */
14 };
15
16 /* HIER struct element implementieren. */
17
18 /* Fuege ein Element in die Liste ein, sodass die Liste aufsteigend nach ISBN
19  * sortiert ist.
20  * Dafür muss das erste Element ermittelt werden, dass in der
21  * bisher sortierten Liste eine ISBN besitzt die grö ß er ist als die des
       ↪ neuen
22  * Elements. Wenn die Liste leer ist soll das Element direkt an den Anfang
23  * platziert werden.
24  *
25  * first      - Erstes Element (bzw. Anfang) der Liste
26  * new_elem   - Neues Element das in die Liste eingefuegt werden soll.
27  *
28  * Gib einen Pointer auf den neuen oder alten Anfang der Liste zurueck.
29  */
30 element* insert_sorted(element *first, element *new_elem) {
31     /* HIER implementieren. */
32 }
33
34 /* Kreiere ein neues Element mit dynamischem Speicher.
35  *
36  * title     - Der Titel des Buches
37  * author    - Autor des Buches
38  * year      - Erscheinungsjahr des Buches
```

```
 * isbn      - ISBN des Buches
 *
 * Gib einen Pointer auf das neue Element zurueck.
 */
element *construct_element(char *title, char* author, int year, long long int
    ↪ isbn) {
    /* HIER implementieren. */
}

/* Gib den der Liste und all ihrer Elemente zugewiesenen Speicher frei. */
void free_list(list *alist) {
    /* HIER implementieren. */
}

/* Lese die Datei ein und fuege neue Elemente in die Liste ein
 * _Soll nicht angepasst werden_
 * */
void read_list(char* filename, list *alist) {
    element* new_elem;

    char title[MAX_STR];
    char author[MAX_STR];
    int year;
    long long int isbn;
    while(read_line(filename, title, author, &year, &isbn) == 0) {
        new_elem = construct_element(title, author, year, isbn);
        alist->first = insert_sorted(alist->first, new_elem);
        alist->count++;
    }
}

/* Erstelle die Liste:
 *  - Weise ihr dynamischen Speicher zu
 *  - Initialisiere die enthaltenen Variablen
 * _Soll nicht angepasst werden_
 */
list* construct_list() {
    list *alist = malloc(sizeof(list));
    alist->first = NULL;
    alist->count = 0;
    return alist;
}

/* Gib die Liste aus:
 * _Soll nicht angepasst werden_
 */
void print_list(list *alist) {
    printf("Meine␣Bibliothek\n================\n\n");
    int counter = 1;
    element *elem = alist->first;
    while (elem != NULL) {
        printf("Buch␣%d\n", counter);
        printf("\tTitel:␣%s\n", elem->title);
        printf("\tAutor:␣%s\n", elem->author);
```



```
        printf("\tJahr:␣␣%d\n", elem->year);
        printf("\tISBN:␣␣%lld\n", elem->isbn);
        elem = elem->next;
        counter++;
    }
}

/* Main Funktion
 * _Soll nicht angepasst werden_
 */
int main() {
    list *alist = construct_list();
    read_list("buecherliste.txt", alist);
    print_list(alist);
    free_list(alist);
    return 0;
}
```