Dr. habil. Bernd Schürmann
Dr. Annette Bieniusa
pinc-support@cs.uni-kl.de

TU Kaiserslautern
Fachbereich Informatik

Übungsblatt 11: Programmieren in C (WS 2020/21)

Abgabe: Montag, 01.02.21, 12:00

Fehlersuche beim Dynamischem Speichermanagement

Da Fehler bei der Verwaltung von dynamischem Speicher in der Regel zu undefiniertem Verhalten führen und oft auch zu Programmabbrüchen durch das Betriebssystem (*Segmentation faults*); manchmal laufen die fehlerhaften Programme aber auch scheinbar fehlerfrei ab, können aber auf anderen Systemen dann zu Probleme führen. Um diese sicherheitskritischen Situationen zu finden, werden beim Ausführen von Tests zusätzliche Überprüfungen gemacht. Diese Überprüfungen werden auch in Exclaim verwendet. Bei der Ausführung von Tests in Exclaim können daher Fehler gemeldet werden, die Sie auf Ihrem System eventuell nicht beobachten können.

Um diese Überprüfungen unter Linux bzw. MacOS zu verwenden, müssen Sie Ihr Programm mit folgendem Befehl kompilieren:

```
clang -Wall -Werror -fsanitize=address -fsanitize=undefined -g mein_programm.c ./a.out
```

Unter Windows werden die benötigen Bibliotheken leider standardmäßig nicht installiert. Nutzen Sie hier bitte die Möglichkeit, Programme mehrfach in Exclaim hochladen zu können.

Die Ausgaben im Fehlerfall sind z.T. schwer verständlich; in den Tutorien werden wir eine kurze Einführung dazu geben. Bitte kontaktieren Sie Ihren Tutor, falls Sie mit den Fehlermeldungen nicht weiterkommen.

Aufgabe 1 *Fehler beim Speichermanagement (3 Punkte)*

Abgabe: `malloc_correct.c`

In folgendem Programm finden sich 3 Fehler. Finden Sie diese Fehler, erklären Sie sie und korrigieren Sie das Programm entsprechend!

```c
#include <stdio.h>
#include <stdlib.h>

// Alloziert ein Array für n double-Werte und initialisiert es mit 0.0, 1.0,
    2.0, etc.
double *make_array(int n)
{
    int i = 0;
    double *array = (double *)malloc(n);
    if (array != NULL) // Allozieren von Speicher
    {
        for (i = 0; i < n; i++)
        {
            array[i] = i;
        }
    }
    return array;
}

int main(void)
{
    int i;
    int size;
    double *dp;
    scanf("%d, &size");
```

```c
    dp = make_array(size);
    if (dp != NULL)
    {
        for (i = 0; i < size; i++) // Ausgeben der Werte
        {
            printf("%4.2f", dp[i]);
        }
        printf("\n");
        free(dp); // Freigeben des Speichers
        free(dp); // Sichergehen, dass es auch wirklich freigegeben ist
    }
    return 0;
}
```

Bitte laden Sie das korrigierte Programm mit den Erklärungen als Kommentare im Programm in Exclaim hoch!

Aufgabe 2 *Verkettete Liste (16 Punkte, jeweils 4 pro Unteraufgabe)*

Abgabe: `linkedlist.c`

Laden Sie sich die Datei `linkedlist.c` herunter, welche eine Implementierung für einfach verketteten Listen mit einigen Funktionen enthält. In dieser Aufgabe sollen Sie diese Implementierung um die folgenden Funktionen erweitern.

Sie können die Header-Datei `linkedlist.h` und das Programm in `main.c` nutzen, um Ihre Implementierung zu testen.

Größtes Element

Schreiben Sie eine Funktion `int max(linked_list_t *ll)`, die den größten in der Liste gespeicherten Wert ermittelt. Falls die Liste leer ist, soll `INT_MIN` zurückgeben werden.

Duplikate

Schreiben Sie eine Funktion `bool list_has_duplicates(linked_list_t *ll)`, welche prüft, ob in der Liste `ll` Werte mehrmals vorkommen. Falls es mehrfache Elemente gibt, soll `true`, andernfalls `false`, zurückgeben werden.

Entfernen aller Elemente

Schreiben Sie eine Funktion `void reset(linked_list_t *ll)`, welche alle Einträge in der Liste entfernt und den Speicherplatz der Listenelemente freigibt. Die Liste soll danach leer sein. Achten Sie darauf keine Speicherlecks zu erzeugen.

Elemente aus Array einfügen

Schreiben Sie eine Funktion `linked_list_t *from_array(int* ar, int size)`, welches eine Liste mit den Einträgen des Arrays `ar` erstellt. Die Reihenfolge der Elemente soll dabei beibehalten werden.

Aufgabe 3 *Texteditor (Teil 2): Modularisierung (**freiwillig**)*

1. Um die Arbeit mit dem Editor etwas benutzerfreundlicher zu machen, werden wir als nächstes falsche Kommandoeingaben behandeln. Erweitern Sie Ihren Editor um eine Funktion zur Fehlerausgabe:

`void error_message (int error_no);`	Gibt eine geeignete Fehlermeldung auf der Konsole aus.

   Je nach Fehlernummer soll dazu eine folgende Fehlermeldung ausgegeben werden:

0 `"Falsches Kommando"`
1 `"Zeilenangabe fehlt"`
2 `"Text fehlt"`

Erweitern Sie das Einlesen von Kommandos entsprechend, damit bei falschen Kommandoeingaben ein geeigneter Fehler ausgegeben wird.

2. Strukturieren Sie Ihren Editorcode vom letzten Übungsblatt um! Dabei sollen folgende Module für Systemfunktionalität, die Ein-/Ausgabe, die Initialisierung und die Fehlerbehandlung angelegt werden:

`main.c`	Hauptprogramm, Einlesen der Kommandos
`globals.h`	Globale Typen, Variablen und Konstanten
`command.c, command.h`	Kommandos
`editor_functions.c, editor_functions.h`	Funktionen des Editor
`in_out.c, in_out.h`	Ein- und Ausgabefunktionen
`error_handling.c, error_handling.h`	Ausgabe von Fehlermeldungen