A. Hück , alexander.hueck@tu-darmstadt.de, S. Kreutzer, Y. Fischler

Systemnahe und Parallele Programmierung (WS 21/22)

Übungsblatt: 1

Die Lösungen der folgenden Aufgaben müssen bis zum 16. November 2021 um 15:00 Uhr in Moodle eingereicht werden. Die Lösungen werden benotet. Bitte bündeln Sie alle Dateien in einem zip-Archiv. Legen Sie dabei für jede Aufgabe bitte einen eigenen Ordner an.

Aufgabe 1

(4 Punkte) Erstellen sie ein C Programm, das den Text `Hallo Welt!` auf dem Bildschirm ausgibt.

Aufgabe 2

(10 Punkte) Beantworten sie folgende Fragen zum unten gegebenen Quellcode des C Programms `pointer_task.c`.

- (5P) Geben sie den Wert von `a` nach der Ausführung jeder Zeile ab Zeile 16 (`d = b;`) an.
- (5P) Welche Zeilen sind (möglicherweise) fehlerhaft, erläutern und begründen Sie jeden gefundenen Fehler.

```
1  #include <stdlib.h>
2
3  int main() {
4    int *array1 = malloc(sizeof(int) * 2);
5    int array2[2];
6    array1[0] = 0;
7    array2[0] = 0;
8    array1[1] = 1;
9    array2[1] = 1;
10
11   int a = 0;
12   int *b = &a;
13   int *c = array1;
14   int *d = &(array1[0]);
15
16   d = b;
17   *d = 123;
18   a = 42;
19   array1[0] = 42;
20   a = *c + *b;
21   d = 54321;
22   a = *(c + array2[0]);
23   free(array1);
24   free(array2);
25   array2 = 321;
26   array1[0] = a;
27 }
```

Aufgabe 3

(8 Punkte) Ein Entwickler arbeitet gerade an einem Rollenspiel. Jedoch sind in seinem Programm `RPG.c` einige Fehler aufgetreten. Helfen Sie ihm und finden sie insgesamt acht Kompiler- und Laufzeitfehler. Kompilieren und debuggen Sie das Programm. Finden und beschreiben Sie die acht Fehler. Falls Sie mehr als acht Fehlerbeschreibungen liefern, werden nur die ersten acht gewertet. Ein Fehler könnte auch mehrmals vorkommen (pro Korrektur gibt es einen Punkt). Korrigieren Sie die Fehler im Programm damit es wieder korrekt ausgeführt werden kann und die Konsole folgendes zurück gibt

```
        Harald is a Knight
        Lydia is a Thief
```

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

typedef struct {
  char classname;
  int base_strength;
  int base_wisdom;
} Class;

typedef struct {
  char* name;
  int age;
  Class cla
} Character;

typedef struct {
  Character members[2];
} Party;

Class* initClasses(Class* allowed_classes) {
  allowed_classes = (Class*)malloc(2 * sizeof(Class));
  allowed_classes[0].classname = "Knight";
  allowed_classes[0].base_strength = 30;
  allowed_classes[0].base_wisdom = 10;

  allowed_classes[1].classname = "Sorceress";
  allowed_classes[1].base_strength = 10;
  allowed_classes[1].base_wisdom = 30;

  allowed_classes[2].classname = "Thief";
  allowed_classes[2].base_strength = 20;
  allowed_classes[2].base_wisdom = 20;

  return allowed_classes;
}

int main(int i)
{
  Class* classes = NULL;
  classes = initClasses(classes);

  Character char1, char2;
  char1.name = "Harald";
```

```
44    char1.age = 44;
45    char1.cla = classes[0];
46
47    char2.name = "Lydia";
48    char2.age = 28;
49    char2.cla = classes[2];
50
51    Party myparty;
52    myparty->members[0] = char1;
53    myparty.members[0] = char2;
54
55    printf("%s is a %d\n", char1.name, char1.cla.classname);
56    printf("%s is a %s\n", char2.name, char2.cla.classname);
57
58    free(&classes);
59    return 0;
60 }
```

Aufgabe 4

(28 Punkte) In dieser Aufgabe soll eine eine Warteschlange mit fester Länge implementiert werden, die pro Datenelement einen **int** speichert. In der Warteschlange speichert jedes Element einen Zeiger auf das folgende Element. Der Zeiger zum nächsten Element des letzten Listenelements ist `NULL`. Ein Beispiel ist in Abbildung 1 zu sehen.

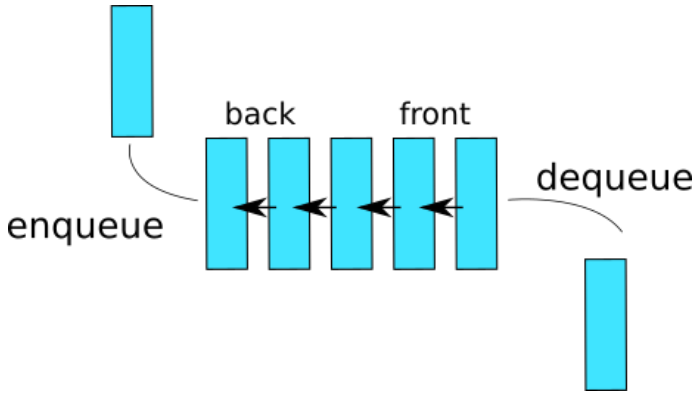

Figure 1: Eine verlinkte Warteschlage

Für die Implementierung sollen zwei Datenstrukturen definiert werden:

- QueueElement

  Diese Struktur repräsentiert ein Element. Es enthält einen Zeiger auf das nächste Listenelement und einen **int**.

- Queue

  Diese Struktur repräsentiert die gesamte Warteschlange und enthält zwei Zeiger, einen auf das erste Element und einen auf das letzte Element der Liste. Ist die Liste leer, sind diese Zeiger NULL. Auch speichert sie ihre maximale Länge max_length, sowie die aktuelle Anzahl an enthaltenen Elementen in actual_length .

Des Weiteren sollen Funktionen zum Zugriff und Bearbeiten der Warteschlange erstellt werden, die zu Beginn der Datei `Queue.c` deklariert und später definiert werden sollen.

- Queue* Queue_init(**int** length)

  Diese Funktion gibt eine neue Warteschlange zurück und setzt deren maximale Länge auf den übergebenen Wert. Sie alloziert den nötigen Speicher und initialisiert alle Felder mit NULL.

- **int** Queue_enqueue(Queue* queue, **int** value)

  Diese Funktion erzeugt ein neues Element, das den Integer data speichert, und fügt das neue Element am Ende der Warteschlange hinzu. Die neue Länge der Warteschlange ist ihr Rückgabewert. Ist die Warteschlange bereits voll, so wird eine Warnung ausgegeben, ein negativer Wert zurück gegebenen und kein neues Element erzeugt und gespeichert.

- QueueElement* Queue_dequeue(Queue* queue)

  Diese Funktion gibt das vorderste Element der Warteschlange zurück und entfernt es aus der Warteschlange. Ist die Warteschlange leer, wird NULL zurückgegeben.

- **void** Queue_print(Queue* queue)

  Die Funktion gibt auf der Konsole die aktuelle Länge der Warteschlange, sowie die in ihr enthaltenen Elemente, beginnend mit front aus.

Folgendes Testprogramm:

```
1  int main() {
2      Queue* my_queue = Queue_init(5);
3      QueueElement* e = Queue_dequeue(my_queue);
4      if (e != NULL) free(e);
5
6      Queue_enqueue(my_queue, 1);
7      Queue_enqueue(my_queue, 2);
8      Queue_enqueue(my_queue, 3);
9      e = Queue_dequeue(my_queue);
10     if(e!=NULL) free(e);
11     Queue_enqueue(my_queue, 4);
12     Queue_enqueue(my_queue, 5);
13     e = Queue_dequeue(my_queue);
14     free(e);
15     Queue_enqueue(my_queue, 6);
16     Queue_print(my_queue);
17
18     return 0;
19 }
```

Soll schließlich diese Ausgabe liefern:

```
1    Queue of size 4: 3 4 5 6
```