Institut für Informatik
Prof. Dr. Stefan Conrad

Sommersemester 2020
Übung 1

Hardwarenahe Programmierung

Gruppe 06 (Lars)

In dieser Übung decken wir die Grundlagen der C-Programmierung ab, insbesondere Funktionen, Arrays, Pointer und Unit-Tests.

Wichtig:

- *Sie müssen **mindestens 12 von insgesamt 14 Aufgaben mit Tests lösen**, um die praktische Übung zu bestehen. Auf den Übungsblättern 1 bis 6 gibt es jeweils 2, 3, 2, 2, 3 und 2 Aufgaben mit Tests, die Sie lösen sollten.*
- *Das **Abschlussprojekt muss vollständig gelöst werden**. Hier darf kein einziger Test fehlschlagen! Das Abschlussprojekt finden Sie auf Übungsblatt 7.*
- *Sie müssen **alle 6 Tests zu den Übungen im ILIAS** bestehen.*
- *Um besser auf die Abschlussaufgabe vorbereitet zu sein, sollten Sie sich aber bemühen, alle Aufgaben zu lösen.*
- *Denken Sie daran, dass Sie Ihre **Lösungen im ILIAS abgeben. Ihre Dateien müssen genau in der Form, in der sie hochgeladen werden, kompilierbar sein**. Benennen Sie die Dateien nicht um. Ihre Abgabe wird automatisiert geprüft und der Computer wird nicht raten, was die richtigen Dateinamen gewesen sein könnten. Falls Sie Ihre Daten komprimieren, verwenden Sie kein anderes Dateiformat als `*.zip`.*
- *Denken Sie daran, **Test 1 im ILIAS zu lösen**.*

Aufgabe 0 *Funktionen*

(a) Erstellen Sie eine Funktion `even(int a)`, die testet, ob a eine gerade Zahl ist. Falls a gerade ist, soll 1 als Rückgabewert übergeben werden, andernfalls 0.

(b) Schreiben Sie ein Programm, das mit Ihrer Funktion aus (a) eine Zahl x testet und dem Benutzer am Bildschirm eine Meldung ausgibt, ob die Zahl gerade oder ungerade ist. Dabei soll der Benutzer nach Programmstart aufgefordert werden, die Zahl x einzugeben. Beispiel:

```
Geben Sie eine Zahl ein:      15
-------------------------------------
Die Zahl ist nicht gerade!
```

Aufgabe 1 *Pointer und Arrays*

(a) Schreiben Sie ein Programm, das 12 Messwerte (jeweils vom Typ `double`) von der Konsole in ein Array einliest und den Inhalt des Arrays anschließend ausgibt.

(b) Falls Sie in Ihrer Lösung eckige Klammern, also `[]`, für die Lese- und Schreibzugriffe auf das Array verwendet haben, schreiben Sie Ihr Programm so um, dass es stattdessen Pointerzugriffe verwendet! Verwenden Sie eckige Klammern hier also **nur** noch zur Deklaration des Arrays.

Aufgabe 2 *Pointer und Funktionen*

Schreiben Sie ein Programm, in dem zwei `double` Zahlen von der Tastatur eingelesen und in lokalen Variablen gespeichert werden. Die Werte der Variablen sollen anschließend in einer Funktion namens `swap` miteinander vertauscht werden. Geben Sie die beiden Variablen vor und nach dem Aufruf von `swap` auf dem Bildschirm aus.

Aufgabe 3 *Unittests*

In dieser Aufgabe sollen Sie die Funktionalität von vorgegebenen C-Funktionen mittels des Unit-Test Frameworks *check* überprüfen. Schauen Sie sich dazu zunächst die vorgegebenen Unittests in der Datei `simpleString_tests.ts` zur Implementierung in `simpleString.c` an, welche ihnen zeigen, wie solche Tests aussehen können.

(a) Bauen Sie die Tests und führen Sie sie aus (entsprechend der Anleitung im ILIAS-Lernmodul). Die fehlschlagenden Tests helfen Ihnen, Fehler in der von uns bereitgestellten Implementierung zu erkennen. Sie sollten verstehen, was die Ausgabe der Tests bedeutet.

(b) Korrigieren Sie im Anschluss die Fehler in der Implementierung und demonstrieren Sie, dass nun alle Tests erfolgreich durchlaufen. Die Tests selbst sollten Sie nicht verändern!

Folgende C-Funktionen sind gestellt:

- eine Funktion `int my_strlen(char *string)`, die die Länge des übergebenen Strings zurückgeben soll.
- eine Funktion `int is_string(char *string, int len)`, die 1 zurückgeben soll, wenn das übergebene `char` Array der Länge `len` ein String ist, und 0 sonst.

Aufgabe 4 *Pointer und Funktionen*

In dieser Aufgabe sollen Sie ein Programm schreiben, das einen Apfel-Automaten repräsentiert. In der hochgeladenen Datei *apfel.c* finden Sie einige vorgegebene Funktionsprototypen. Mit Ausnahme der `print_apfel`-Funktion sind diese noch nicht implementiert.

Ein Apfel kostet 1 Euro und der Automat verfügt zu Beginn über 50 Äpfel.

Das Programm soll wie folgt ablaufen:

1. Der Automat fragt den Benutzer, wie viele Äpfel gekauft werden sollen. Sie dürfen davon ausgehen, dass nur ganze Zahlen eingegeben werden.
2. Danach verringert der Automat die Anzahl der Äpfel und erhöht den Geldstand entsprechend. Außerdem gibt der Automat die gekauften Äpfel und die Anzahl der noch verbleibenden Äpfel aus. Für die Ausgabe der Äpfel soll die vorgegebene Funktion verwendet werden. Für den Fall, dass der Benutzer mehr Äpfel anfragt, als übrig sind, wird die Anzahl auf die noch übrige Menge beschränkt.
3. Wenn die Anzahl der angefragten Äpfel negativ ist, soll sich der Automat beenden.
4. Wenn danach die Geldkassette mit 45 Euro EUR gefüllt ist, gibt der Automat zusätzlich eine Meldung aus, dass die Kasse geleert werden soll.
5. Wenn keine Äpfel mehr verfügbar sind, beendet der Automat sich mit einer entsprechenden Meldung. Sonst kehrt er zu Schritt 1 zurück.
6. Überprüfen Sie die Korrektheit ihres Programms, indem sie die Script-Datei `test.sh` ausführen. Eventuell müssen Sie die Datei mit dem Befehl `chmod +x test.sh` erst ausführbar machen.