

Assignment 5

Programmieren 1 – WiSe 25/26

Prof. Dr. Michael Rohs, Falk Stock, M.Sc.

Alle Assignments (bis auf das erste) müssen in Zweiergruppen bearbeitet werden. Ein Gruppenmitglied kann dabei die Lösung der Zweiergruppe gebündelt abgeben. Einzige Ausnahme ist dabei das erste Assignment, wo jedes Gruppenmitglied einzeln abgeben muss. Namen beider Gruppenmitglieder müssen sowohl in der PDF-Abgabe, als auch als Kommentar in jeglichen Quelltextabgaben genannt werden. Plagiats führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag den 27.11. um 23:59 Uhr über <https://assignments.hci.uni-hannover.de/WiSe2025/Prog1>. Die Abgabe muss aus einer einzelnen ZIP-Datei bestehen, die den Quellcode, eine PDF für Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen auf.

Zum Bestehen der Studienleistung müssen Sie mindestens zwei von vier Punkten pro Assignment erzielen. Möchten Sie zusätzlich den Klausurbonus erreichen, müssen Sie über alle Assignments hinweg 75% der Punkte erreichen.

Aufgabe 1: Talsperrensteuerung (1 Punkt)

Eine Talsperre soll bei einem Wasserstand von unter 20 m kein Wasser mehr abgeben. Im Bereich eines niedrigen Wasserstandes von 20 m bis unter 40 m soll Wasser mit einer Pumpe abgegeben werden. Ab einem Wasserstand von 40 m sollen zwei Pumpen Wasser ableiten. Ist der Wasserstand höher als 67.5 m , soll die Talsperre in einen Notfallmodus gehen und die Notfallablassventile öffnen. Entwickeln Sie eine Funktion `dam_control(double water_level)` zur Regelung der Talsperre, die abhängig von dem Wasserstand alles ausschaltet, eine Pumpe, zwei Pumpen oder zwei Pumpen und das Notfallventil aktiviert.

Die Template-Datei ist `dam_control.c`. Führen Sie die im C-Skript unter Recipe For Intervals (Recipe for Intervals) beschriebenen Schritte durch. Geben Sie mindestens sechs Beispiele (Eingaben und erwartete Ausgaben) in Form von Testfällen an.

Hinweis:

- Nutzen Sie zum Vergleich zweier enum-Werte in Testfällen `test_equal_i`.

Aufgabe 2: Parkhaus (1 Punkt)

Um einem Autofahrer die Parkplatzsuche zu vereinfachen, soll der Zustand eines Parkhauses, d.h. ob es noch freie Parkplätze gibt, auf einen Blick erkennbar sein. Die Template-Datei ist `park_house.c`. Benutzen Sie die in Recipe for Enumerations beschriebene Vorgehensweise, um die Parkhauszustände zu definieren.

- a) Implementieren Sie die Funktion `ParkHouseState det_park_house_state(int free_spots)`, welche die freien Parkhausplätze übergeben bekommt und basierend auf der Anzahl der freien Plätze den Zustand des Parkhauses zurückgibt.

Dabei sollen folgende Grenzwerte gelten:

- 0 freie Plätze → Volles Parkhaus
- Weniger als 10 freie Plätze → Fast volles Parkhaus
- 10 oder mehr freie Plätze → Freies Parkhaus

Fügen Sie (sinnvolle) Tests hinzu. Dokumentieren Sie ihren Code mithilfe von Kommentaren.

- b) Implementieren Sie die Funktion `String print_park_house_state(ParkHouseState state)`, welche einen String zurückgibt, der nähere Informationen zum Zustand des Parkhauses gibt.
- c) (OPTIONAL) Verändern Sie das Programm so, dass man die Anzahl der freien Parkplätze in der Kommandozeile eingeben kann und das Ergebnis auch auf der Kommandozeile ausgegeben wird.

Aufgabe 3: Place-Value Notation (1 Punkt)

In dieser Aufgabe geht es um die Darstellung von Zahlen mit verschiedenen Basen über die Stellenwertsnotation. Das Template für diese Aufgabe ist die Datei `base_converter.c`.

- a) Erstellen Sie ein Purpose Statement für die Funktion `int length_for_base(int number, int base)` inklusive Parameterbeschreibung und erklären Sie die Funktionalität. Schauen Sie sich dazu auch die Verwendung dieser Funktion in `String get_string_for_number_and_base(int number, int base)` an.
- b) Implementieren Sie Funktion `String convert_to_base(int number, int base)`. Die Funktion bekommt eine Zahl `number` übergeben sowie eine Basis mit der diese dargestellt werden soll. Bspw. sollte `number = 5` und `base = 2` zu folgender Ausgabe führen: `101`. Basen sollen nur aus dem Intervall $[2, 36]$ gewählt werden können. Für die Darstellung von Stellen mit einer Wertigkeit von mehr als 9 sollen wie im Hexadezimalsystem Buchstaben verwendet werden. Bspw. Wäre „Z“ eine gültige Zahl in einem 36er-Zahlensystem und würde die Dezimalzahl 35 repräsentieren. Die Funktion `String get_string_for_number_and_base(int number, int base)` gibt Ihnen eine Zeichenkette passender Länge zurück, in der Sie mit der `s_set` Funktion aus der `prog1lib` die Ziffern einsetzen können. Beispiele für die Verwendung der benötigten Hilfsfunktionen finden Sie im Template.

Hinweise:

- Nutzen Sie die Funktionen `s_get` und `s_set` zum zeichenweisen Zugriff auf Strings.
- Nutzen Sie den String `characters`.
- Nutzen Sie den Modulo Operator `%` und die Integer Division `/`.

Aufgabe 4: Operationen auf einzelnen Bits (1 Punkt)

Diese Aufgabe baut auf der vorherigen Aufgabe auf und nutzt auch das Template `base_converter.c`. Die vorherige Aufgabe kann Ihnen helfen, da Sie mit der Funktion `convert_to_base` eine einfache Funktion haben, um die Zahlen binär darzustellen. Lösen Sie daher zuerst Aufgabe 3.

- a) Schauen Sie sich die Funktion `void bit_operations()` an und beschreiben Sie die Funktionsweise der folgenden Operatoren `&`, `|`, `^`, `<<` und `>>` als Kommentar im Quelltext.
- b) Implementieren Sie die Funktion `bool get_bit(int value, int index)` ohne die Nutzung von externen Funktionen oder Macros, mit der Sie ein einzelnes Bit aus einem Integer extrahieren können. Ist das Bit an Stelle `index` gleich 1, soll `true` zurückgegeben werden ansonsten `false`. Zum Beispiel gibt der Aufruf von `get_bit(5, 0)` den Wert des niedrigwertigsten Bits und der Aufruf `get_bit(-1, 31)` das höchstwertige Bit als `bool` zurück.
- c) Implementieren Sie die Funktion `int set_bit(int value, int index, bool bit)` ohne die Nutzung von externen Funktionen oder Macros, mit der Sie ein einzelnes Bit in einem Integer setzen können. Beispielsweise soll der Aufruf `set_bit(0, 31, true)` das höchstwertige Bit auf 1 setzen.
- d) (OPTIONAL) Implementieren Sie die Funktion `int extract_bits(int value, int start, int end)`. Die Funktion soll einen beliebigen Bereich innerhalb eines Integers extrahieren. Dabei soll der Bereich `[start, end]` extrahiert werden und zurückgegeben werden. Schauen Sie sich auch die Testfälle an. Beispielsweise sollte der Aufruf `extract_bits(0xf0, 4, 8)` (0xf0 entspricht 11110000) 0x0f bzw. 1111 zurückgeben. Die Bits werden extrahiert und verschoben. Die Funktion kann die Funktionen aus b) und c) verwenden, muss dies aber nicht. Ansonsten ist die Nutzung von externen Funktionen oder Macros nicht erlaubt. Nutzen Sie `&`, `|`, `^`, `<<` und `>>`