Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2020/21

Prof. Dr. Michael Rohs, Tim Dünte, M.Sc.

Übungsblatt 6

Alle Übungen (bis auf die erste) müssen in Zweiergruppen bearbeitet werden. Beide Gruppenmitglieder müssen die Lösung der Zweiergruppe einzeln abgeben. Die Namen beider Gruppenmitglieder müssen sowohl in der PDF Abgabe als auch als Kommentar in jeglichen Quelltextabgaben genannt werden. Wenn Sie für Übungsblatt 1 noch keinen Gruppenpartner haben, geben Sie alleine ab und nutzen Sie das erste Tutorium dazu, mit Hilfe des Tutors einen Partner zu finden. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag den 26.11. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2020/Programmieren1. Die Abgabe muss aus einer einzelnen zip-Datei bestehen, die den Quellcode, ein pdf bei Freitextaufgaben und alle weiteren nötigen Dateien (z.B. Eingabedaten oder Makefiles) enthält. Lösen Sie Umlaute in Dateinamen bitte auf.

Aufgabe 1: Place-Value Notation

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. Bpsw. Wäre „Z" eine gültige Zahl in einem 36er-Zahlensystem und würde die Dezimalzahl 35 repräsentieren. Die Funktionen `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.

Benötigte String-Funktionen:

- `int s_length(String s);` // Anzahl Zeichen in einem String
- `void s_set(String s, int i, char c);` // Zeichen an Position i auf c setzen
- `char s_get(String s, int i);` // Zeichen an Index i zurückgeben

Hinweise:

- Nutzen Sie den String characters.
- Nutzen Sie den Modulo Operator % und die Integer Division /.

Aufgabe 2: Operationen auf einzelnen Bits (Fortsetzung von Aufgabe 1)

Diese Aufgabe baut auf der vorherigen Aufgabe auf und nutzt auch das Template `base_converter.c`. Die vorherige Aufgabe hilft Ihnen, da Sie mit der Funktion `convert_to_base` eine einfache Funktion haben, um die Zahlen binär darzustellen. Bearbeiten Sie daher zuerst Aufgabe 1:

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`. Bspw. gibt der Aufruf von `get_bit(5, 0)` gibt den Wert des niederwertigsten 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 aus einem Integer setzen können. Bspw. soll der Aufruf `set_bit(0, 31, true)` das höchstwertige Bit auf 1 setzen.

d) 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 >>.

Aufgabe 3: Climate Control

Eine Klimaanlage soll bei Temperaturen unter 21 °C heizen, bei 21-23.7 °C nichts tun und bei Temperaturen ab 23.7 °C kühlen. Entwickeln Sie eine Funktion `climate_control` zur Regelung der Klimaanlage, die abhängig von der Temperatur heizt, ausschaltet oder kühlt.

Führen Sie die im C-Skript unter Recipe for Intervals beschriebenen Schritte durch. Verwenden Sie die Template-Datei `climate_control.c`. Bearbeiten Sie die mit `todo` markierten Stellen. Geben Sie fünf weitere Beispiele (Eingaben und erwartete Resultate) in Form von Testfällen an.

Editieren, compilieren und ausführen:

- mit Texteditor `climate_control.c` editieren und speichern
- `make climate_control` ← ausführbares Programm erstellen
- `./climate_control` ← Programm starten (evtl. ohne `./`)
- Die letzten beiden Schritte lassen sich auf der Kommandozeile kombinieren zu: `make climate_control && ./climate_control`
- Verwenden Sie in der Testfunktion die Funktion `test_equal_i`.

Aufgabe 4: Die geheimnisvolle Nachricht

Lösen Sie das in Abbildung 1 beschriebene Rätsel. Implementieren Sie dazu die Funktionen `encode` und `decode`, die eine Nachricht ver- bzw. entschlüsseln können. Benutzen Sie dafür das template `secret_message.c`. Nutzen Sie für diese Aufgabe die unten angegebenen String-Funktionen aus der Programmieren 1 Bibliothek (http://hci.uni-hannover.de/files/prog1lib/string_8h.html).

a) Schreiben Sie einen Funktionsstub sowie ein Purpose Statement für die Funktion `decode`. Schreiben Sie eine Funktion `decode_test`, die mindestens 5 Testfälle für `decode` implementiert.

b) Implementieren Sie die Funktion `decode` und entschlüsseln Sie die geheime Nachricht.

c) Schreiben Sie einen Funktionsstub sowie ein Purpose Statement für die Funktion `encode`. Schreiben Sie eine Funktion `encode_test`, die mindestens 5 Testfälle für `encode` implementiert.

d) Implementieren Sie die Funktion `encode`. Nutzen Sie den Hinweis in der verschlüsselten Nachricht um sich Implementierungsarbeit zu sparen.

Benötigte String-Funktionen:

- `int s_length(String s);` // Anzahl Zeichen in einem String
- `String s_copy(String s);` // String s kopieren
- `char s_get(String s, int i);` // Zeichen an Index i zurückgeben
- `void s_set(String s, int i, char c);` // Zeichen an Position i auf c setzen

Abbildung 1: Rätsel

Editieren, compilieren und ausführen:

- mit Texteditor `secret_message.c` editieren und speichern
- `make secret_message` ← ausführbares Programm erstellen
- `./ secret_message` ← Programm starten (evtl. ohne `./`)
- Die letzten beiden Schritte lassen sich auf der Kommandozeile kombinieren zu: `make secret_message && ./ secret_message`
- Verwenden Sie in der Testfunktion die Funktion `test_equal_s`.