Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2018/19

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 29.11. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2018/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: Werte, Variablen, Zuweisungen & Ausdrücke

a) Grenzen Sie die Begriffe *Wert* und *Variable* voneinander ab.

b) Beschreiben Sie was bei einer *Zuweisung* passiert. Nutzen Sie Ihr Wissen über Werte und Variablen. Geben Sie ein Beispiel an.

c) Werten Sie folgende Ausdrücke „auf dem Papier" in einem Auswertungsdiagramm (evaluation diagram) Schritt für Schritt aus. Verwenden Sie hierfür die in der Vorlesung vorgestellte Schreibweise. Beachten Sie die Vorrangreihenfolge (Präzedenz) der Operatoren.

1. x = 1 + 2 * 3 + 4
2. x = (1 * (y = (1 + 2) * (z = 3)))
3. 11 > 2 * 3 && (x = 4 * 5 == 6 / 3.0) || 1

Aufgabe 2: Formatierung von Quelltext

a) Für die Lesbarkeit von Quelltext ist die Formatierung sehr wichtig. Quelltext wird häufiger gelesen, als geschrieben. Gegeben sei folgender unformatierter Quelltext.

```
int f ( int i ) { printf ( "called f\n" ) ; if( i < 0 )
return -i; else return 3*i; }
```

Formatieren Sie diesen Quelltext nach folgenden Regeln:

- { ist das letzte Zeichen einer Zeile und steht nicht alleine in einer Zeile

- } ist das erste Zeichen einer Zeile, evtl. nach Leerzeichen
- der Code der bei einer Fallunterscheidung (if und else) ausgeführt wird, wird mit {} umschlossen
- ; ist das letzte Zeichen einer Zeile und steht nicht alleine in einer Zeile
- kein Leerzeichen vor ;
- Zeilen in einem {...}-Block werden vier Leerzeichen tiefer eingerückt, als der Block selbst
- kein Leerzeichen zwischen Funktionsname und (
- ein Leerzeichen nach if
- kein Leerzeichen nach (
- kein Leerzeichen vor)
- ein Leerzeichen vor und ein Leerzeichen nach einem binären Operator (wie *)

b) Erklären Sie das Verhalten der Funktion f möglichst kurz und prägnant.

c) Übersetzen Sie folgenden Postfix Code in schön formatierten C Code, inklusive des Purpose Statements. Benennen Sie die Variablen genauso wie in Postfix um am Ende genau vergleichen zu können wie sich die Syntax unterscheidet. Nutzen Sie die obigen Regeln für die Formatierung. Stellen Sie sicher, dass ihr Code kompiliert und funktioniert.

```
# calculates the n-th fibonacci number and prints it on the
output
fib: (n :Int){
    fn-1: 1 !
    fn-2: 1 !
    fn: 1 !
    {
        {n 0 <=} {"n too small" println}
        {n 2 <=} {fn println}
        {true}{
            {
                n 2 <= { fn println break} if
                fn: fn-1 fn-2 + !
                fn-1 fn-2!
                fn fn-1!
                n 1 - n!
            }loop
        }
    }cond
}fun
```

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

Implementieren Sie eine Ver-/Entschlüsselung nach dem Caesar Verfahren (https://de.wikipedia.org/wiki/Caesar-Verschl%C3%BCsselung). 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`. Die Funktion `decode` soll zwei Parameter haben: Der erste Parameter ist vom Typ `String` und enthält die Nachricht, die entschlüsselt werden soll. Der zweite Parameter ist von Typ `int` und enthält die Verschiebung der Buchstaben. Der Rückgabewert der Funktion soll vom Typ `String` sein.

b) Schreiben Sie eine Funktion `decode_test`, die mindestens 5 Testfälle für `decode` implementiert. Zeichen, die nicht a-z oder A-Z sind, sollen ignoriert werden und in den entschlüsselten String übernommen werden. Eine Verschiebung um mehr als 26 Stellen soll genauso möglich sein wie die Eingabe einer negativen Verschiebung.

c) Implementieren Sie die Funktion `decode` und entschlüsseln Sie die geheime Nachricht mit einer Verschiebung von 5 Buchstaben:

Uwtlwfrrnjwjs 1 rfhmy Xufxx!

Bsp: Bei einer Verschiebung von 5 wird beim Entschlüsseln aus einem „F“ ein „A“

d) Schreiben Sie einen Funktionsstub sowie ein Purpose Statement für die Funktion `encode`. Die Funktion `encode` soll zwei Parameter haben: Der erste Parameter ist vom Typ `String` und enthält die Nachricht, die verschlüsselt werden soll. Der zweite Parameter ist von Typ `int` und enthält die Verschiebung der Buchstaben. Der Rückgabewert der Funktion soll vom Typ `String` sein. Schreiben Sie eine Funktion `encode_test`, die mindestens 5 Testfälle für `encode` implementiert.

e) Wie können Sie eine nach diesem Verfahren verschlüsselte Nachricht entschlüsseln, wenn Sie nicht wissen wie groß die Verschiebung ist? Beispielsweise bei dieser Nachricht: Apww: Dhz pu kpl lpul Ypjoabun nloa, nloa hbjo pu kpl huklyl. Kpl Svlzbun whzza pu lpul Glpsl.

f) Implementieren Sie die Funktion `encode`. Nutzen Sie den Hinweis in der verschlüsselten Nachricht aus Teilaufgabe e), 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

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`.