Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2016/17

Prof. Dr. Michael Rohs, Oliver Beren Kaul, M.Sc.

Übungsblatt 3

Dieses Übungsblatt muss in Zweiergruppen bearbeitet werden. Beide Gruppenmitglieder müssen die Lösung der Zweiergruppe einzeln abgeben. Die Namen beider Gruppenmitglieder müssen in der Abgabe genannt werden. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag, den 10.11. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2016/Programmieren1 . Erstellen Sie alle zur Abgabe gehörigen Dateien im Verzeichnis assignment03. Achten Sie darauf, keine Umlaute in Dateinamen zu benutzen. Entfernen Sie alle vom Compiler generierten Dateien (z.B. .exe-Dateien) aus dem Verzeichnis und komprimieren Sie es zu assignment03.zip. Laden Sie diese Datei dann bitte hoch.

Allgemeine Hinweise zur Kompilierung

- Windows und MinGW bei Benutzung von cmd.exe: Tragen Sie, wie in den Folien zur Hörsaalübung beschrieben, die Pfade zu den Verzeichnissen `C:\MinGW\bin` und `C:\MinGW\msys\1.0\bin` in die PATH-Umgebungsvariable ein. Dies ist notwendig, damit die Tools (`make`, `gcc`, etc.) gefunden werden.
- Mit den Pfeiltasten (hoch, runter) lassen sich auf der Konsole frühere Kommandos wieder abrufen.
- Wenn Sie beim Kompilieren die Fehlermeldung bekommen, dass „base.h" nicht gefunden wurde, dann weicht Ihre Verzeichnisstruktur von der in Übung 2 beschriebenen ab oder Sie haben vergessen, die Library zu kompilieren. Überprüfen Sie erneut die in Übung 2 beschriebenen Schritte. Hilfreich dabei ist die im Stud.IP verfügbare Aufzeichnung der Stundenübung.

Aufgabe 1: Ausdrücke

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.

a) `x = 1 + 2 * 3 + 4`

b) `x = (1 * (y = (1 + 2) * (z = 3)))`

c) `11 > 2 * 3 && (x = 4 * 5 == 6 / 3.0) || 1`

Aufgabe 2: Tempomat

Ein Tempomat hat die Aufgabe, ein gegebenes Tempo möglichst genau zu halten. Entwickeln Sie eine Funktion zur Regelung eines Tempomats, welche abhängig vom aktuellen Tempo und dem Zieltempo entweder bremst, nichts tut, beschleunigt oder eine Notbremsung auslöst.

- Für den Fall, dass das Zieltempo < 0 ist, soll in jedem Fall eine Notbremsung ausgelöst werden.
- Für den Fall, dass das aktuelle Tempo < 0 oder > 300 ist, ist vom einem falschen Eingabewert auszugehen und der Tempomat soll nichts tun.
- Der Tempomat soll bremsen, falls das aktuelle Tempo mehr als 2% größer ist als das Zieltempo.
- Der Tempomat soll beschleunigen, falls das aktuelle Tempo mehr als 2% kleiner ist als das Zieltempo.
- Andernfalls (aktuelles Tempo innerhalb +-2% vom Zieltempo) soll der Tempomat nichts tun.

Verwenden Sie die im Skript unter *Recipe for Intervals* beschriebenen Schritte. Verwenden Sie die Template-Datei `tempomat_control.c`. Passen Sie den Inhalt der Datei an die Aufgabe an, aber behalten Sie das generelle Layout bei. Geben Sie 6 sinnvolle Beispiele (Eingaben und erwartete Resultate) in Form von Testfällen an.

Hinweise:

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

Aufgabe 3: Segelschiff in Spielewelt

In einer Spielewelt sei die Erde eine Scheibe mit den Kontinenten Europa, Afrika und Amerika. Afrika liegt südlich von Europa. Amerika liegt westlich von Europa und Afrika. In der Spielewelt können Nord-, Süd-, Ost- und Westwinde auftreten. In der Spielewelt gibt es genau ein Segelschiff. Wenn das Segelschiff bei Westwind in Europa startet, fährt es nach Amerika. Wenn es bei Ostwind in Amerika startet, landet es (auf Grund einer nördlichen Wasserströmung) in Europa. Wenn es bei Westwind von Amerika aus startet, fällt es ins Nichts. Wenn es bei Südwind in Europa startet, landet es in Afrika. Entsprechendes gilt für die anderen möglichen Kombinationen von Startposition und Windrichtung. Nehmen Sie an, dass sich die Windrichtung nur vor, aber nie während einer Reise des Segelschiffs ändert.

Entwickeln Sie eine Funktion, die abhängig von der Startposition des Segelschiffs und der Windrichtung beim Start die Zielposition bestimmt. Verwenden Sie die im Skript unter *Recipe for Enumerations* beschriebenen Schritte. Verwenden Sie die Template-Datei `sail.c`. Passen Sie den Inhalt der Datei an die Aufgabe an, aber behalten Sie das generelle Layout bei. Geben Sie 8 sinnvolle Beispiele (Eingaben und erwartete Resultate) an.

Hinweise:

- mit Texteditor `sail.c` editieren und speichern
- `make sail` ← ausführbares Programm erstellen
- `./sail` ← Programm starten (evtl. ohne `./`)
- Verwenden Sie in der Testfunktion die Funktion `check_expect_i`.

Aufgabe 4: Skript

Das unter http://hci.uni-hannover.de/files/prog1script/script.html zur Verfügung stehende Skript beschreibt die Vorgehensweisen bei der Lösung von Programmieraufgaben für verschiedene Datentypen. Lesen Sie Kapitel 6 (Recipe for Itemizations) und beantworten Sie folgenden Fragen.

a) Was ist die Aufgabe der Konstruktorfunktionen bei itemizations?

b) Wieso benötigt `struct Train` eine eigene check-Funktion?

c) Erläutern Sie die Funktionsweise der Funktion `check_expect_train.`

d) Was war Ihnen beim Lesen von Kapitel 6 unklar? Wenn nichts unklar war, welcher Aspekt war für Sie am interessantesten?