Human-Computer Interaction Group

Leibniz Universität Hannover

Assignment 6

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. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag den 04.12. 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: Gewichte umrechnen (1 Punkt)

In dieser Aufgabe werden Gewichtsangaben zwischen verschiedenen Einheiten umgerechnet. Die Struktur `Weight` repräsentiert ein Gewicht in einer bestimmten Einheit. Die möglichen Einheiten (Gramm g, Kilogramm kg, Tonne t, Pfund lb) sind über die Aufzählung **Unit** definiert. Es gilt $1\,\text{lb} = 0.453\,592\,37\,\text{kg}$. Auch ist eine Testfunktion **test_within_weight** für Gewichte gegeben. Verwenden Sie die Template-Datei **weights.c** und bearbeiten Sie die mit **todo** markierten Stellen.

a) Implementieren Sie die Konstruktorfunktion **make_weight**.

b) Implementieren Sie die Funktion **print_weight**, die Gewichte in folgendem Format ausgibt:
```
1234.00 g
4.75 kg
3.10 t
5.40 lbs
```
Runden Sie auf maximal zwei Nachkommastellen.

c) Implementieren Sie die Funktion **to_unit**. Diese soll ein Gewicht in eine Zieleinheit konvertieren. Fügen Sie zunächst mindestens 5 sinnvolle Tests zu **to_unit_test** hinzu.

d) Implementieren Sie die Funktion **compare**. Diese soll zwei Gewichte vergleichen und `0` zurückgeben, wenn `w` und `v` das gleiche Gewicht repräsentieren, `-1` zurückgeben, wenn `w` ein kleineres Gewicht als `v` repräsentiert und `1` zurückgeben, wenn `w` ein größeres Gewicht als `v` repräsentiert. Fügen Sie zunächst mindestens 5 sinnvolle Tests zu **compare_test** hinzu.

Aufgabe 2: Oberflächen geometrischer Körper

In einem Programm sollen verschiedene Formen von geometrischen Körpern – nämlich Zylinder, Kugeln und Quader – repräsentiert werden. Entwickeln Sie eine Funktion, die diese geometrischen Körper verarbeiten kann und die zugehörige Oberfläche berechnet. Verwenden Sie die im Skript unter Recipe for Variant Data beschriebenen Schritte. Verwenden Sie die Template-Datei **surface.c**.

a) Implementieren Sie die Struktur GeomObject zur Repräsentation eines geometrischen Objekts. Die möglichen Varianten sind **Cylinder, Sphere und Cuboid**. Nutzen Sie die entsprechenden Strukturen.

b) Implementieren Sie die Konstruktorfunktionen **make_cylinder**, **make_sphere** und **make_cuboid**.

c) Implementieren Sie die Funktion **surface_area**. Diese berechnet die Oberfläche des übergebenen Objekts. Die mathematischen Formeln sind als Kommentar im Quelltext gegeben. Der Wert von π ist über die Konstante `M_PI` definiert.

Aufgabe 3: Verschachtelte Schleifen (1 Punkt)

Implementieren Sie Funktionen, die Ausgaben der angegebenen Form erzeugen. Der Parameter **n** beschreibt die Größe der Ausgabe. Erlaubte Werte für den Parameter **n** liegen im Intervall **[0, 9]**. In den Beispielen gilt **n = 5**. Verwenden Sie in dieser Aufgabe verschachtelte for-Schleifen. Die Template-Datei für diese Aufgabe ist **loops.c**.

a) Implementieren Sie die Funktion **void loops_a(int n)**, die Ausgaben der folgenden Form erzeugt:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

b) Implementieren Sie die Funktion **void loops_b(int n)**, die Ausgaben der folgenden Form erzeugt:

```
        1
      1 2
    1 2 3
  1 2 3 4
1 2 3 4 5
```

c) Implementieren Sie die Funktion **void loops_c(int n)**, die Ausgaben der folgenden Form erzeugt:

```
    1
  1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

d) Implementieren Sie die Funktion **void loops_d(int n)** für ungerade n, die Ausgaben der folgenden Form erzeugt:

```
0           0
  1       1
     2
  1       1
0           0
```

e) Implementieren Sie die Funktion **void loops_e(int n)**, die Ausgaben der folgenden Form erzeugt (für **n ≥ 3**). Sie dürfen beliebige Hilfsfunktionen implementieren.

```
    +--------+
   /        /
  /        /
 /        /
+--------+
```

f) **(OPTIONAL)** Implementieren Sie die Funktion **void loops_f(int n)**, die Ausgaben der folgenden Form erzeugt (für **n ≥ 3**). Sie dürfen beliebige Hilfsfunktionen implementieren.

```
n = 5:
    +--------+
   /0 1 2 3 /
  /4 5 6 7 /
 /8 9 0 1 /
+--------+

n = 9:
        +----------------+
       /0 1 2 3 4 5 6 7 /
      /8 9 0 1 2 3 4 5 /
     /6 7 8 9 0 1 2 3 /
    /4 5 6 7 8 9 0 1 /
   /2 3 4 5 6 7 8 9 /
  /0 1 2 3 4 5 6 7 /
 /8 9 0 1 2 3 4 5 /
+----------------+
```

Aufgabe 4: Dateien einlesen und verarbeiten (1 Punkt)

In dieser Aufgabe soll die Tabelle **people.txt** eingelesen und verarbeitet werden. Die Tabelle enthält eine Zeile mit Spaltenbeschreibungen und dann eine größere Anzahl Zeilen mit entsprechenden Werten. Die erste Spalte enthält Geburtsjahre (Typ int), die zweite die Geschlechter ('m', 'f', 'd') (Typ char), die dritte Körpergrößen in Meter (Typ double) und die letzte die Namen (Typ String).

Im Template ist bereits Code enthalten, der die Datei in einen String einliest. Gehen Sie diesen zeilenweise durch und errechnen Sie damit folgende statistische Angaben:

- Das durchschnittliche Geburtsjahr (gerundet auf ganze Jahre)
- Die Anzahl Personen pro Geschlecht
- Die durchschnittliche Körpergröße pro Geschlecht
- Die durchschnittliche Länge der Namen (inklusive Leerzeichen sofern vorhanden)

Vervollständigen Sie die Funktion **compute_statistics**, welche die in einer Zeichenkette (Typ: String) vorliegende Tabelle verarbeitet und dafür eine Statistik berechnet und zurückgibt. Schauen Sie sich zunächst an, wie der Inhalt von **people.txt** formatiert ist. Ihre Funktion muss nur für dieses Format funktionieren. Eine Fehlerbehandlung soll nicht implementiert werden.

Hinweise:

- Benutzen Sie zur Verarbeitung des Strings die Funktionen `s_length`, `s_get` und `s_sub`.
- In der **people.txt** sind Zeilen durch ein `newline` ('\n') getrennt, einzelne Daten durch einen Tabulator ('\t').
- Benutzen Sie zur Konvertierung eines Strings in ein `int` die Funktion `i_of_s` und zur Konvertierung eines Strings in ein `double` die Funktion `d_of_s`.

Aufgabe 5: Bonusaufgabe: Quersummen (1 Bonuspunkt)

Diese Bonusaufgabe liefert bei erfolgreicher Bearbeitung einen Bonuspunkt, um möglicherweise versäumte Punkte für den Klausurbonus nachzuholen. **Zum Bestehen dieses Assignments und somit der Studienleistung müssen dennoch zwei Aufgaben aus dem Aufgabenpool von Aufgaben 1-4 erfolgreich bearbeitet werden!**

Um die Quersumme einer Zahl zu bilden werden die einzelnen Ziffern der Zahl aufaddiert. Die Quersumme von 123 ist also $6 = 1 + 2 + 3$. Bei der alternierenden Quersumme werden von Links nach Rechts die Ziffern an ungeraden Stellen addiert und die Ziffern an geraden Stellen subtrahiert. Die alternierende Quersumme von 537591 ist also $12 = 5 - 3 + 7 - 5 + 9 - 1$.

Die Template-Datei für diese Aufgabe ist **digit_sum.c**. Bearbeiten Sie die mit **todo** markierten Stellen. Die **digit_sum** Funktion soll die Quersumme der positiven Ganzzahl **number** berechnen. Über den Parameter **alternating** wird bestimmt ob die Quersumme alternierend ist (**true**) oder nicht (**false**).

a) Schreiben Sie ein Purpose Statement.

b) Implementieren Sie die Funktion **digit_sum** und definieren Sie mindestens 6 sinnvolle Testfälle.

c) Eine Zahl ≥ 11 ist genau dann durch 11 teilbar, wenn ihre alternierende Quersumme 0 ist oder selbst durch 11 teilbar ist. Implementieren Sie die Funktion **divisible_by_eleven**. Nutzen Sie hier **nicht** den Modulo Operator. Negative Quersummen werden normalisiert, indem man sie wiederholt mit 11 addiert bis sie positiv sind. Definieren Sie sinnvolle Testfälle.