Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 - WS 2022/23

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

Präsenzübung 7

Diese Aufgaben sind zur Lösung während der einstündigen Präsenzübung gedacht. Sie können die Aufgaben auf einem mitgebrachten Laptop oder auf Papier lösen.

Aufgabe 1: Formatierte Ausgabe

Erzeugen Sie mit printf die folgenden Ausgaben.

a)

Variablen:

```
String product1 = "Milch";
double price1 = 1.29;
String product2 = "Kaffeemaschine";
double price2 = 252.483145;
String product3 = "iPhone";
double price3 = 1217.50123;
```

Ausgabe (drei Aufrufe von printf, Ausrichtung beachten):

```
   1.29 € für Milch
 252.48 € für Kaffeemaschine
1217.50 € für iPhone
```

b)

Variable:

```
int x = 0x30;
```

Ausgabe (Zeilenumbruch beachten):

```
"30" zur Basis 16 ist "48" zur Basis 10.
```

c)

Variable:

```
double p = 76.5432;
```

Ausgabe (//\\ und Zeilenumbruch beachten):

```
76.5% //\\
```

Aufgabe 2: Vorbedingungen, Nachbedingungen, Zusicherungen

a) Wozu dienen Zusicherungen?

b) Implementieren Sie eine beliebige Zusicherung. Was passiert, wenn die Bedingung erfüllt ist bzw. nicht erfüllt ist? Siehe hierzu auch: https://postfix.hci.uni-hannover.de/files/prog1lib/base_8h.html

c) Was ist der Unterschied zwischen Zusicherungen, Vorbedingungen und Nachbedingungen?

d) Sollten Vorbedingungen in der Dokumentation einer Funktion erwähnt werden?

e) Was ist der Unterschied zwischen Tests (z.B. `test_equal_i`) und Zusicherungen? Was ist eine Gemeinsamkeit beider Mechanismen?

Aufgabe 3: Strukturen, Vorbedingungen und Nachbedingungen

Es sollen Funktionen für die Verwaltung von Konten implementiert werden. Die Template-Datei ist account.c. Ein Konto sei definiert als:

```
typedef struct {
    String owner;
    int balance;
} Account;
```

a) Fügen Sie als Vorbedingungen zu open_account hinzu: Owner darf nicht leer sein. Initial darf nicht negativ sein. Dokumentieren Sie diese Vorbedingungen.

b) Fügen Sie als Vorbedingung zu deposit (einzahlen) hinzu: Amount darf nicht negativ sein. Dokumentieren Sie diese Vorbedingung. Implementieren Sie die Funktion.

c) Fügen Sie als Vorbedingungen zu withdraw (abheben) hinzu: Es muss genügend Guthaben auf dem Konto vorhanden sein. Amount darf nicht negativ sein. Dokumentieren Sie diese Vorbedingungen. Implementieren Sie die Funktion. Fügen Sie als Nachbedingung hinzu: Das resultierende Guthaben darf nicht negativ sein.

d) Ändern Sie die Aufrufe in der main-Funktion bzw. die Funktionsimplementierungen nacheinander so, dass jede der Vor- bzw. Nachbedingungen einmal fehlschlägt.

Aufgabe 4: Varianten

Die Template-Datei ist shape.c. Die Struktur Shape repräsentiert zwei Varianten von geometrischen Formen: Kreise und Rechtecke.

```
typedef struct {
    enum { CIRCLE, RECTANGLE } tag;
    union {
        struct { double x, y, radius; } circle;
```

```
            struct { double x, y, width, height; } rectangle;
        };
    } Shape;
```

a) Implementieren Sie die Funktion `area` zur Berechnung des Flächeninhalts eines Shapes. Verwenden Sie beim Kreis die Konstante `M_PI`.

b) Implementieren Sie die Funktion `scale_area` zum Skalieren der Fläche der Shape mit dem gegebenen positiven Faktor. Schreiben Sie einen Dokumentationskommentar. Schreiben Sie eine Vorbedingung und eine Nachbedingung für die Funktion. Die Nachbedingung soll prüfen, ob der neue Flächeninhalt tatsächlich dem alten Flächeninhalt, skaliert um den Faktor, entspricht. Der alte Flächeninhalt kann zu Beginn der Funktion mittels `ensure_code(double old_area = area(s));` ermittelt werden. Dokumentieren Sie auch die Vorbedingung.