Human-Computer Interaction Group

Leibniz Universität Hannover

Übungsblatt 5

Programmieren 1 – WiSe 23/24

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

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. Plagiate führen zum Ausschluss von der Veranstaltung.

Abgabe bis Donnerstag den 16.11.um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2022/Prog1. Die Abgabe muss aus einer einzelnen Zip-Datei bestehen, die den Quellcode, ein 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 eine Aufgabe erfolgreich lösen. Sie dürfen natürlich gerne (zum Beispiel zum Erreichen des Klausurbonus) alle bearbeiten und abgeben.

Vorbereitung Für diese und folgende Aufgaben sollten Sie sich eine Entwicklungsumgebung (keine IDE) zum Bearbeiten der Übungsaufgaben einrichten. In den Foliensätzen zur 4. und 5. Hörsaalübung finden Sie detaillierte Anleitungen zur Einrichtung Ihrer Entwicklungsumgebung.

Die Schritte zur Installation der Programming I C Library finden sich unter https://postfix.hci.uni-hannover.de/files/prog1lib/. Führen Sie zunächst diese Schritte aus, inklusive des Kompilierens und Ausführen eines Beispiels in **prog1lib/script_examples**

Laden Sie assignment05.zip mit den Template-Dateien für dieses Übungsblatt aus Stud.IP herunter und speichern Sie assignment05.zip in Ihrem Entwicklungsverzeichnis ab (siehe Foliensatz zur Übung 5). Öffnen Sie ihre Shell und navigieren Sie in das Verzeichnis. Verwenden Sie das Template dieser Aufgabe wie folgt:

```
cd prog 1                       ← In das Entwicklungsverzeichnis navigieren
unzip assignment05.zip          ← Template-Dateien dieser Übung entpacken
cd assignment05                 ← In das frisch entpackte Übungsverzeichnis
                                navigieren price.c editieren und speichern

make price                      ← ausführbares Programm erstellen
./price                         ← Programm starten
```

Die letzten drei Schritte (Editieren und Speichern, **make price** und **./price**) führen Sie nun wiederholt aus, bis das Programm fertig ist. Die ↑ -Taste stellt die vorherige Zeile wieder her. Kompilieren und Ausführen lassen sich kombinieren: **make price && ./price**

Aufgabe 1: Mengenrabatt

Ein Onlineshop bietet Produkte für 1€ pro Einheit an. Für größere Bestellungen wird allerdings ein Mengenrabatt angeboten. Ab einer Bestellmenge von 10 kostet jede Einheit nur noch 90ct. Bei einer Bestellmenge von 50 oder sogar mehr nur noch 75ct. Wird nichts oder eine negative Menge bestellt, beträgt der Preis 0€.

Bei einem Bestellwert von unter 20€ wird eine Versandpauschale von 5€ berechnet. Beträgt der Bestellwert 20€ oder mehr, entfällt die Versandpauschale.

a) Definieren Sie Konstanten vom Typ **int**, welche die Preiskategorien, die Bestellmengengrenzen und die Versandpauschale repräsentieren. Sie sollten am Ende 7 Konstanten definiert haben.

b) Schreiben Sie als nächstes einen Funktionsstub für die Funktion **price**. **price** soll die Anzahl an bestellten Einheiten übergeben bekommen und den Gesamtpreis (Wert + ggf. Versandkosten) in Cent zurückgeben. Nutzen Sie den Datentyp **int** für den Parameter der Funktion und für den Rückgabewert. Schreiben Sie zusätzlich einen Kommentar über die Funktion, der beschreibt, was die Funktion leistet.

c) Schreiben Sie eine Funktion **price_test**, die die Funktion **price** testet. Erstellen Sie mindestens 6 sinnvolle Testfälle. Nutzen Sie die **test_equal_i** Funktion aus der **prog1lib**. Schauen Sie sich ggf. das Beispiel **celsius_to_fahrenheit.c** an.

d) Implementieren Sie die Funktion **price** entsprechend der Aufgabenbeschreibung. Beispielsweise soll die Funktion für die Eingabe 22 den Wert 2480 liefern (22*90 + 500).

Hinweis:
Wenn Sie beim Kompilieren die Fehlermeldung bekommen, dass base.h nicht gefunden wurde, dann weicht Ihre Verzeichnisstruktur von der geforderten ab. Überprüfen Sie erneut die angegebenen Schritte. Dieser Fehler sieht wie folgt aus:

```
total.c:7:10: fatal error: 'base.h' file not found
#include "base.h"
         ^~~~~~~~
```

Aufgabe 2: Guess my number

Erstellen Sie eine Datei **guess_my_number.c**. Inkludieren Sie in **guess_my_number.c** die Progammieren 1 Bibliothek. Erstellen Sie eine Funktion **int main(void)** und geben Sie in **main** immer 0 zurück. Kompilieren Sie die Datei. Sie müssen in dieser Aufgabe keine weiteren Funktionen implementieren. Sie können die gesamte Funktionalität in der **main** Funktion unterbringen.
Implementieren Sie ein Zahlenratespiel. Der Computer würfelt eine zufällige ganze Zahl im Intervall [0, 100]. Der Spieler kann in jeder Runde raten. Wenn die geratene Zahl größer als die Zahl des Computers ist, soll „**Too Large!**" ausgegeben werden. Wenn die geratene Zahl kleiner als die Zahl des Computers ist, soll „**Too Small!**" ausgegeben werden. Wenn die Zahl der Zahl des Computers entspricht, soll „**Match!**" ausgegeben werden. Ein Beispielablauf ist nachfolgend dargestellt:

```
Guess my Number!
50
Too Small!
75
Too Small!
83
Too Large!
79
Match!
```

Hinweise:

- Sie können für die Ausgabe von Zeichenketten die Funktion **printsln(String s)** verwenden.
- Ganzzahlige Zufallszahlen können mit der Funktion **i_rnd(int i)** erzeugt werden.
- Um eine Zahl von der Terminaleingabe zu lesen, können Sie die Funktion **i_input()** verwenden.
- Schauen Sie sich zur Verwendung der Funktionen die Vorlesungsfolien bzw. auch die Dokumentation der Prog1lib an: https://postfix.hci.uni-hannover.de/files/prog1lib/base_8h.html

Aufgabe 3: Formatierung von Quelltext

a) Für die Lesbarkeit von Quelltext ist auch in C 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 den PostFix Code auf der folgenden Seite in schön formatierten C Code, inklusive Purpose Statement und Kommentaren. 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. Inkludieren Sie die Programmieren 1 Bibliothek und erstellen Sie die Funktion **int main(void)**. Nutzen Sie **test_equal_i** um mindestens 4 sinnvolle Testfälle für die Funktion **number_of_days** zu definieren. Fügen Sie das Programm in der Datei **leap_years.c** Ihrer Abgabe hinzu.

```
#<Return the number of days in the given year.A leap year has 366 days, a
non-leap year has 365 days.The input represents any year A. D. as an integer, the
return value is the number of days.>#
number_of_days: (year :Num -> :Num) {
    # Leap Years are any year that can be exactly divided by 4
    year 4 mod 0 = multiple_of_four!
    # except if it can be exactly divided by 100, then it isn't
    year 100 mod 0 = multiple_of_hundred!
    # except if it can be exactly divided by 400, then it is
    year 400 mod 0 = multiple_of_fourhundred!
    multiple_of_four not
    multiple_of_hundred multiple_of_fourhundred not and
    or {
        365
    } {
        366
    } if
} fun
```

Hinweis:
Der Modulo Operator mod aus PostFix wird in C durch % dargestellt.

Aufgabe 4: Compiler Fehlermeldungen

Manchmal spuckt der Compiler seltsame Fehlermeldungen aus. Hier sollen Sie das Programm in **primes.c** lauffähig machen. Die Funktion **int print_primes_in_intervall(int lower, int upper)** gibt auf der Konsole alle Primzahlen im Intervall [lower, upper) aus und gibt als Rückgabewert die Anzahl an gefundenen Primzahlen im Intervall zurück. Wiederholen Sie die nachfolgenden Schritte solange, bis Sie alle Fehler gefunden und behoben haben.

1. Kompilieren Sie die Datei **primes.c**
2. Kopieren Sie die Fehlermeldung oder Warnung aus der Konsole und beschreiben Sie kurz in 1-2 Sätzen was die Fehlermeldung bedeutet und was das eigentliche Problem ist.
3. Beheben Sie den Fehler.

Hinweise:

- Sie können die Ausgaben des Compilers sowie deren Interpretation in Ihren Quelltext schreiben. Nutzen Sie dafür Blockkommentare: /* ... */
- Kompilieren und führen Sie Ihr Programm mit folgendem Befehl aus: **make primes && ./primes**