Human-Computer Interaction Group

Leibniz Universität Hannover

Programmieren 1 – WS 2020/21

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

Übungsblatt 5

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 19.11. um 23:59 Uhr über https://assignments.hci.uni-hannover.de/WiSe2020/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: Rabatt auf Schokoriegel

Vorbereitung

Hier wird zunächst die Vorbereitung Ihrer „Entwicklungsumgebung" beschrieben. Diese Schritte werden für die weiteren Aufgaben entsprechend sein und werden nur hier ausführlich beschrieben. Das Herunterladen und Kompilieren der Bibliothek ist nur einmalig erforderlich.

Detaillierte Schritte zur Installation der Programming I C Library finden sich unter http://hci.uni-hannover.de/files/prog1lib/index.html. Führen Sie zunächst diese Schritte aus, inklusive dem Kompilieren 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 im MinGW-Home-Verzeichnis. Öffnen Sie die MinGW-Shell. Unter Mac OS X und Linux verwenden Sie das Terminal statt MinGW. Die Kommandos sind identisch. Verwenden Sie das Template dieser Aufgabe wie folgt:

`pwd` ← gibt den Pfad des aktuellen Verzeichnisses aus (sollte /home/MyName sein)

`ls` ← listet den Inhalt des aktuellen Verzeichnisses

(prog1lib und assignment05.zip müssen im aktuellen Verzeichnis liegen)

`unzip assignment05.zip` ← Template-Dateien dieser Übung entpacken

`cd assignment05` ← change directory to assignment05

(`total.c` mit gutem Texteditor (z.B. Notepad++) editieren, speichern)

`make total` ← ausführbares Programm (total bzw. total.exe) erstellen

`./total` ← Programm starten (bzw. `total.exe`)

Die letzten drei Schritte (editieren+speichern, `make total` und `./total`) 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 total && ./total`

Aufgabe

Ein Hersteller von köstlichen Schokoriegeln, vertreibt diese direkt im Internet an seine Kunden. Für größere Bestellungen gewährt er einen Mengenrabatt. Entwickeln Sie eine Funktion, `total`, die für eine Menge an Schokoriegeln den Gesamtpreis berechnet. Ein Schokoriegel kostet pro Stück 50 Cent. Wenn zehn oder mehr Stück gekauft werden sinkt der Preis pro Schokoriegel auf 45 Cent. Bei 100 oder mehr Stück sinkt der Preis auf 40 Cent. Eine negative Menge von Schokoriegeln soll einen Preis von 0 Cent zurückliefern. Der Hersteller verlangt bei einem Gesamtpreis von unter 2000 Cent eine Versandpauschale von 500 Cent. Ab 2000 Cent Wert müssen keine Versandkosten mehr bezahlt werden. Die Funktion soll als Eingabe die Anzahl an Schokoriegeln übergeben bekommen und den Gesamtpreis (Wert der Schokoriegel + ggf. Versandkosten) in Cent zurückgeben.

a) Definieren Sie als erstes Konstanten vom Typ `int`: drei Konstanten für die verschiedenen Preise, eine Konstante für den Versandpreis und eine Konstante für die Grenze, ab der keine Versandkosten mehr anfallen.

b) Schreiben Sie als nächstes einen Funktionsstub für die Funktion `total`. `total` soll eine Anzahl von Schokoriegeln übergeben bekommen und den Gesamtpreis (Wert der Schkoriegel + 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 `total` leistet.

c) Schreiben Sie eine Funktion `total_test`, die die Funktion `total` testet. Erstellen Sie mindestens 6 sinnvolle Testfälle. Nutzen Sie dafür die Funktion `test_equal_i` aus der prog1lib. Schauen Sie sich ggf. das Beispiel `celsius_to_fahrenheit.c` an.

d) Implementieren Sie die Funktion `total`. Ein Schokoriegel kostet 50 Cent. Wenn zehn und mehr Stück gekauft werden sinkt der Preis pro Riegel auf 45 Cent. Bei 100 oder mehr Stück sinkt der Preis auf 40 Cent je Riegel. Der Händler verlangt bei einer Summe von unter 2000 Cent eine Versandpauschale von 500 Cent. Ab 2000 Cent Wert müssen keine Versandkosten mehr bezahlt werden. Eine negative Anzahl von Produkten soll 0 Cent als Rückgabewert liefern. Beispielsweise liefert die Funktion `total` für drei Produkte 650 (Cent) (150 Cent für das Produkt und 500 Cent Versandpauschale) als Rückgabewert. Für 10 Produkte 950 (Cent).

Hinweise:

- 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.
- Windows und MinGW: Damit das unzip Kommando funktioniert muss der „bin"-Eintrag von msys-unzip im MinGW Installation Manager angewählt und installiert sein.
- Mit den Pfeiltasten (hoch, runter) lassen sich frühere Kommandos wieder abrufen.
- Wenn Sie beim Kompilieren eine Fehlermeldung bekommen, dass base.h nicht gefunden wurde, dann weicht Ihre Verzeichnisstruktur von der geforderten ab. Überprüfen Sie erneut die eingangs angegebenen Schritte. Dieser Fehler sieht wie folgt aus:

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

- Kompilieren und Ausführen lassen sich auf der Kommandozeile kombinieren zu: `make total && ./total`

Aufgabe 2: Guess my number

Führen Sie, falls noch nicht geschehen, die vorbereitenden Schritte aus Aufgabe 1 durch. Erstellen Sie eine Datei `guess_my_number.c`. Inkludieren Sie `in guess_my_number.c` die Programmieren 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 ganze Funktionalität in der main Funktion unterbringen.
Implementieren Sie ein Zahlenratespiel. Der „Computer" denkt sich eine zufällige ganze Zahl im Intervall [0, 101)] aus. Der Spieler kann in jeder Runde raten. Wenn die geratene Zahl größer als die Zahl des Computers ist soll „Zahl ist zu gross!" ausgegeben werden. Wenn die geratene Zahl kleiner als die Zahl des Computers ist soll „Zahl ist zu klein!" ausgegeben werden. Wenn die Zahl der Zahl des Computers entspricht soll „Zahl korrekt!" ausgegeben werden. Ein Beispielablauf ist nachfolgend dargestellt:

```
Rate meine Zahl. Sie liegt im Intervall [0, 100].
50
Zahl zu gross!
25
Zahl zu gross!
12
Zahl zu klein!
17
Zahl zu klein!
22
Zahl zu gross!
20
Zahl korrekt!
```

Hinweise:

- Sie können für die Ausgabe von Zeichenketten die Funktion `printsln()` verwenden.
- Ganzzahlige Zufallszahlen können mit der Funktion `i_rnd()` 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 Dokumention der Prog1lib an: https://hci.uni-hannover.de/files/prog1lib/base_8h.html
- Kompilieren und führen Sie Ihr Programm mit folgendem Befehl aus: `make guess_my_number && ./guess_my_number`

Aufgabe 3: 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 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 gefunden Primzahlen im Intervall zurück. Wiederholen Sie die nachfolgenden Schritte solange bis Sie alle Fehler gefunden und behoben haben:

1. Compilieren 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`