Human-Computer Interaction Group

Leibniz Universität Hannover

Assignment 4

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 20.11. 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: C-Compiler installieren (1 Punkt)

Diese Aufgabe dient der Vorbereitung der C-Aufgaben. Installieren Sie auf Ihrem Rechner den gcc-Compiler und eine Programmierumgebung bzw. einen Texteditor mit Syntaxhervorhebung. Nähere Erläuterungen zur Installation unter Windows, macOS und Linux finden Sie in den Folien der Hörsaalübung.

a) Testen Sie die Installation indem Sie **gcc -v** auf der (Linux-)Kommandozeile aufrufen. Fügen Sie die Ausgabe, die Sie erhalten, in Ihre Abgabe ein.

b) Testen Sie Ihre Installation außerdem mit folgendem Programm:

```
#include <stdio.h>
int main(void) {
    printf("hello, world\n");
    return 0;
}
```

Kompilieren und führen Sie das Programm entsprechend der Folien zur Hörsaalübung 4 aus. Nutzen Sie dafür sowohl einmal **gcc**, als auch **make** über die Makefile. Fügen Sie einen Screenshot der Ausführung beginnend mit dem ersten Befehl im Terminal Ihrer Abgabe bei.

c) Welche Funktion haben die folgenden Terminal-Befehle? Geben Sie einen Beispielaufruf an und fügen Sie, falls es eine Ausgabe auf der Konsole gibt, diese der Abgabe bei:

1. **mkdir**
2. **ls**

3. **cd**

d) Kopieren Sie die ZIP-Datei Aufgabe1.zip in Ihr Arbeitsverzeichnis, entpacken Sie es und navigieren Sie mit dem Terminal dorthin.

1. Geben Sie zwei verschiedene Wege an, wie Sie mit **cd** in das Verzeichnis **folder/A01** gelangen können.
2. Geben Sie zwei verschiedene Wege an wie Sie mit **cd** vom Verzeichnis **A01** in das Verzeichnis **A02** gelangen können.
3. Was passiert, wenn Sie in Ihrem ursprünglichem Verzeichnis „**cd folder/../folder/A01**" eingeben?
4. Was passiert, wenn Sie statt **ls ls -al** aufrufen?

Aufgabe 2: Guess my number (1 Punkt)

Hinweis: Diese Aufgabe ist nahezu identisch mit „Zahlenraten" aus Assignment 02, allerdings wird hier der Code in C geschrieben.

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, sondern 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 einer Zeichenkette gefolgt von einem Zeilenumbruch die Funktion **println(String s)** verwenden.
- Ganzzahlige Zufallszahlen im Intervall [0, i) 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 (1 Punkt)

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: Die Syntax des Modulo-Operators ist in C: `x % y`

Aufgabe 4: Compiler-Fehlermeldungen

Manchmal gibt 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**