Institut für Informatik
Daniel Neugebauer

Wintersemester 2019/20
Tag 6

Hardwarenahe Programmierung

Gruppe 6 (Stefan)

In den bisherigen Übungen haben Sie Nutzereingaben zur Laufzeit des Programms über Tastatureingaben implementiert. In dieser Übung schreiben Sie C-Programme, die ihren Input als Kommandozeilen-Parameter und -Optionen oder über einzulesende Dateien erhalten, und die Ausgabedaten ebenfalls in Dateien schreiben können.

Wichtig: *Achten Sie auf dem gesamten Übungsblatt auf sinnvolle Fehlerabfragen! Falsch formatierte Eingabedateien müssen Sie nicht abfangen.*

Aufgabe 1 *Kleiner Taschenrechner*

Wir haben Ihnen für die Funktion `int calc(int x, int y, int interval, int mod10)` eine Implementierung vorgegeben, die jede `interval`-te Zahl des Intervalls $[x, y]$ quadriert und die Quadrate summiert. Wenn `mod10` $\neq 0$ gilt, wird nur die letzte Stelle des Ergebnisses zurückgegeben.

(a) Schreiben Sie ein Programm *main.c*, welches die Zahlen `x` und `y` über die Kommandozeile übergeben bekommt, und das die Funktion `calc` mit diesen Werten für `x` und `y` sowie mit `interval=1` und `mod10=0` aufruft.

(b) Erweitern Sie Ihre `main` so, dass Sie folgende Optionen verarbeiten können.

- `-m i`: Setzt `interval` für den Funktionsaufruf auf `i`.
- `-d`: Setzt `mod10` auf `1`.
- `-h`: Gibt eine Hilfestellung zur Benutzung des Taschenrechners aus.

Ändern Sie den Funktionsaufruf entsprechend. Das Programm könnte also zum Beispiel so `./taschenrechner 2 10` oder so `./taschenrechner -m 2 2 10` aufgerufen werden.

Aufgabe 2 *Personal I*

In dieser Aufgabe sollen Sie ein Programm schreiben, das die Wochenarbeitszeit von Mitarbeitern verwalten kann. Das Programm erhält über die unten beschriebenen Eingabemethoden das Wochensoll aller Mitarbeiter und die Daten der einzelnen Mitarbeiter, und soll anschließend die Namen und den neuen Stand des Arbeitszeitkontos aller Mitarbeiter ausgeben. Der neue Stand des Arbeitzeitkontos berechnet sich aus der Differenz von persönlicher Wochenarbeitszeit und allgemeinem Wochensoll, plus den persönlichen Stand des Vormonats. Falls ein Beschäftigter aktuell krank oder im Urlaub ist (Spalte 'Bemerkungen'), soll der Wochensoll *nicht* abgezogen werden!

- Die Wochenarbeitszeit in Stunden (integer), die jeder Mitarbeiter leisten soll, wird Ihrem Programm als Kommandozeilen-Parameter übergeben.
- Die Personaldaten bestehen aus einem 4-Tupel pro Mitarbeiter, dieses enthält den Nachnamen, die geleistete Wochenarbeitszeit, den Stunden-Stand des Vormonats und eine Bemerkung. Ihr Programm erhält diese Daten zur Laufzeit über `stdin`. Beispieldaten haben wir Ihnen in der Datei *Mitarbeiter.csv* bereitgestellt, diese können z.B. so aussehen:

```
Schneider  15  -30  Urlaub
Fischer    50  -60  Dienstreise
Weber      40   0   leer
```

- Zusätzlich soll Ihr Programm folgende Optionen anbieten:
  - `-m` Gibt nur diejenigen Beschäftigten aus, deren neuer Stand des Arbeitszeitkontos negativ ist.
  - `-a` Berechnet den Vormonatswert nicht mit ein.
  - `-b bem` Gibt nur die Beschäftigten aus, bei denen in der Bemerkungsspalte `bem` steht.

Beispielaufrufe:

`./main 40 < Mitarbeiter.csv`

Gibt die Namen und neuen Stunden-Stand aus *Mitarbeiter.csv* aus, wenn pro Woche 40h gearbeitet werden soll.

`./main 40 -b Dienstreise < Mitarbeiter.csv > Stundensoll.csv`

Gibt für 40 Soll-Wochenstunden die Namen und den neuen Stunden-Stand derjenigen Mitarbeiter aus *Mitarbeiter.csv* aus, in deren Bemerkung '`Dienstreise`' steht, und schreibt das Ergebnis in *Stundensoll.csv*.

`./main 35 -m -a < Mitarbeiter.csv`

Gibt die Namen und die Differenz Wochenarbeitszeit-Wochensoll derjenigen Mitarbeiter aus *Mitarbeiter.csv* aus, die diese Woche weniger als das Soll von 35h gearbeitet haben.

Die Header-Datei *personal.h* enthält einige Funktionsprototypen, die Sie implementieren sollen, um diese Aufgabe zu lösen. Testen Sie Ihre Implementierungen mit den Unit-Tests in *personal_tests.ts*. Schreiben Sie ein Makefile, um den Kompilier- und Testprozess zu automatisieren.

Aufgabe 3 *Personal II*

Als nächstes sollen Sie ein Programm schreiben, das die in der vorherigen Aufgabe erstellten Listen weiter verarbeitet. Ihr Programm soll eine Datei mit Namen und Stundensoll (Output der vorherigen Aufgabe) sowie eine beliebige Anzahl an Namen von Projekt-Dateien mit deren verfügbaren Mitarbeiter-Plätzen als Eingabe erhalten, und soll die Mitarbeiter auf die Projekte verteilen. In jede der angegebenen Projekt-Dateien sollen dabei die Namen der Mitarbeiter geschrieben werden, die in diesem Projekt arbeiten sollen. Wenn bereits Daten in den Projekt-Dateien stehen, sollen diese nicht gelöscht werden! Hängen Sie neue Daten stattdessen an. Wie Sie die Mitarbeiter konkret verteilen ist Ihnen überlassen, Sie müssen lediglich die Kapazitäten der Projekte einhalten.

Beispielaufruf:

```
./aufteilung Stundensoll.csv P1.csv 5 P2.csv 10 P3.csv 2
```

Verteilt die Mitarbeiter aus *Stundensoll.csv* auf die drei Projekte P1 (5 Plätze), P2 (10 Plätze) und P3 (2 Plätze).

Fügen Sie eine Regel für das Kompilieren Ihres Programms aus dieser Aufgabe zu Ihrem Makefile hinzu.

Aufgabe 4 *Personal III*

Geben Sie einen Kommandozeilenbefehl an, der Ihre Programme aus den vorherigen beiden Aufgaben verwendet, um damit aus der Datei *Mitarbeiter.csv* diejenigen Beschäftigten auf die zwei Projekte P1 (2 verbleibende Plätze) und P2 (3 verbleibende Plätze) zu verteilen, die nach Aktualisierung der Arbeitszeit im Minus sind. Dabei soll

- der Wochensoll 40h betragen, **und**
- nur Beschäftigte betrachtet werden, für die die Bemerkung 'leer' eingetragen wurde **und** deren Name mit M beginnt.

Fügen Sie in Ihrem Makefile eine Regel namens '`loesung_aufgabenteil_III`' hinzu, die den gesuchten Kommandozeilenbefehl ausführt. Verwenden Sie das Make-Feature *prerequisites* in der neuen Regel, um sicher zu stellen, dass alle Voraussetzungen (wie das Kompilieren der benötigten Programme) vorher erfüllt sind.