4. Übungsblatt:
Programmierpraktikum II (SS 2002)

Abgabe: 24. Juni 2002

Erlaubte Bibliotheken:

- `stdio.h`
- `stdlib.h`
- `string.h`

Makefile: Ein vollständiges Makefile erzeugt die Ziele `aufgabe1` bis `aufgabe3`.

Aufgabe 1: Taschenrechner

(8 Punkte)

Schreibe ein Linux-Programm, das mit mehreren Argumenten aufgerufen wird. Dazu soll ein (einfacher) Taschenrechner programmiert werden. Das Programm soll `compute` heißen und von der Linux-Konsole in der Form

`compute` *zahl1 operator zahl2*

aufgerufen werden. Dabei sollen `zahl1` und `zahl2` positive ganze Zahlen und `operator` $\in \{+, -, /, \%\}$ sein.[1]

- Entspricht einer der drei Werte nicht dem korrekten Format, so soll das Programm eine Fehlermeldung ausgeben und sich beenden.
- Ist die Eingabe dagegen korrekt, so soll das Programm die beschriebene Rechnung ausführen, das Ergebnis am Bildschirm ausgeben und sich beenden.

Bsp.: Einige Beispiele für das Verhalten des Programmes:

```
> compute 17 + a
Fehlerhafte Eingabe! Korrektes Eingabeformat:
compute zahl1 operator zahl2
> compute 17 / 5
17 / 5 = 3
> compute 17 +
Fehlerhafte Eingabe! Korrektes Eingabeformat:
compute zahl1 operator zahl2
```

[1] Unbepunktete Zusatzaufgabe: Was passiert, wenn man den Operator '*' mit dazu nimmt? Hast du eine Erklärung für das Verhalten des Programmes?

Aufgabe 2: Spielkarten aufnehmen

(10 Punkte)

Datenstruktur: Wir verwenden ein Kartenblatt mit 32 Karten, in dem jede Kombination aus Farbe (Karo, Herz, Pik, Kreuz) und Wert (7, 8, 9, 10, Bube, Dame, König, As) genau einmal vorkommt. Es bleibt Dir überlassen, welche Darstellung Du für eine solche Spielkarte wählst.

Karten aufnehmen: Das Programm soll nacheinander 10 zufällige Karten ziehen, wobei nicht zweimal die gleiche Karte vorkommen darf. Eine gezogene Karte wird sofort in eine verkettete Liste einsortiert. Dabei soll zunächst nach Farbe und dann nach Wert sortiert werden, wobei die obige Reihenfolge der Farben und Werte gilt (also z.B. $Karo < Herz$ und $10 < Bube$).

Liste ausgeben: Wenn alle zehn Karten gezogen wurden, soll der Inhalt der sortierten Liste am Bildschirm ausgegeben werden. Dabei sind in jedem Fall die oben angegebenen Bezeichnungen für Spielkarten zu verwenden (also z.B. "Herz Dame" oder "Pik 4")!

Erzeugung von Zufallszahlen

Der Standard-Zufallsgenerator von C wird in der Datei `stdlib.h` beschrieben. Auch wenn er für "ernsthafte" Anwendungen (z.B. Simulationen oder Verschlüsselung) oft zu schlecht ist, reicht er für unsere Zwecke doch aus.

Initialisierung: Der Zufallsgenerator muss initialisiert werden, sonst liefert er bei jedem Programmstart die gleiche Folge von Zufallszahlen. Die Initialisierung nehmen wir über eine Variable `SEED` vor. Wir weisen `SEED` über eine `#define`-Anweisung eine ganzzahligen Wert zu (z.B. `#define SEED 17446`) und initialisieren den Generator zu Beginn des Hauptprogrammes mit dem Befehl `srand(SEED)`.[2]

Zahlenerzeugung: Der Befehl `rand()` liefert eine Pseudo-Zufallszahl im Bereich zwischen 0 und einem Wert `RAND_MAX` zurück, der ebenfalls in `stdlib.h` definiert ist. Wer kleinere Zufallszahlen benötigt, kann mittels `x = rand()%n` Zahlen zwischen 0 und $n-1$ erzeugen.

[2] Eine beliebte Variante besteht darin, zur Initialisierung die aktuelle Uhrzeit zu verwenden. Dazu muss man noch die Datei `time.h` mit einbinden und den Befehl `srand(time(0))` aufrufen. Das soll aber bitte auf dem aktuellen Übungsblatt nicht gemacht werden!

Aufgabe 3: Einfache Printer-Queue

(12 Punkte)

Schreibe ein C-Programm, das eine Printer-Queue simuliert. Der Einfachheit halber repräsentieren wir den Print-Job dazu ausschließlich durch eine positive ganze Zahl - die Job-ID.

Datenstruktur: Printjobs werden im *first in - first out* Modus (FIFO) abgearbeitet. Als Datenstruktur soll eine verkettete Liste verwendet werden. Dabei erhält das Programm zwei Arten von Befehlen aus der Standardeingabe:

Der Put-Befehl: Bei einem Befehl der Form "`put` *job-id*" wird die Job-ID hinten an die Printer-Queue angehängt.

Der Get-Befehl: Bei einem Befehl der Form "`get`" arbeitet das Programm den ältesten Printjob in der Printer-Queue ab, d.h. es gibt die Job-ID aus und löscht ihn aus der Liste.

Der End-Befehl: Um eine elegante Beendigung des Programmes zu ermöglichen, soll es sich bei Eingabe des Befehls "`end`" selbst terminieren.

Beispiel: Das folgende Beispiel zeigt eine typische Ein-/Ausgabefolge des Programmes:

```
> put 15
> put 4
> put 1
> get
> 15
> put 44
> get
> 4
> get
> 1
> end
```

Hilfestellung: Es ist vermutlich hilfreich, beim Befehls-Parsen die Funktion `strcmp` aus der `string`-Bibliothek zu verwenden. Diese hat den Prototypen

```
int strcmp(const char*, const char*);
```

und gibt 0 zurück, falls die beiden Eingabestrings gleich sind. Wenn man also prüfen will, ob der Befehl `get` eingegeben wurde, kann man wie folgt vorgehen:

```
char command[4];     /* Alle Befehle bestehen aus 4 Zeichen */
scanf("%s", command);
if(strcmp(command, "get")==0)     /* Falls command = "get": */
  {...}                           /* Tue etwas. */
```