7. Übungsblatt: Programmierpraktikum II (SS 2003)

Abgabe: 23. Juni 2003

Bemerkungen:

- Es sind alle Abgabehinweise auf der Übungswebseite zu beachten!
- Als Programmbibliotheken sind `stdio.h`, `stdlib.h`, `string.h`, `time.h` und (für die Cracks) `errno.h` zugelassen.

Aufgabe 1: Minesweeper

(8 Punkte, Abgabedatei: mines.c)

Das Spiel "Minesweeper" kennen vermutlich alle Windows-Nutzer. Unser Minesweeper-Feld soll folgende Vorgaben beachten:

- Ein Spielfeld ist 30 Felder breit und 16 Felder hoch. Dieses Spielfeld wird intern durch ein Feld von `char`-Werten simuliert.
- Auf das Spielfeld werden 100 Minen zufällig verteilt. Ein Spielfeld mit Mine wird durch das Zeichen 'M' gekennzeichnet.
- Alle Spielfelder, auf denen keine Mine liegt, werden mit der Zahl der angrenzenden Minen beschriftet. Dabei zählt horizontale, vertikale und diagonale Nachbarschaft, ein Feld kann also an 0 bis 8 Minen grenzen. Grenzt ein freies Spielfeld nicht an eine Mine, so ist es mit '-' zu kennzeichnen, andernfalls mit den Zeichen '1' bis '8'.

Schreibe ein C-Programm, das bei jedem Aufruf ein neues deartiges Minesweeper-Feld generiert und am Bildschirm ausgibt, z.B.:

```
M 1 - 1 M 1 1 M 1 1 2 M 1 1 M 1 1 1 2 M 4 M 2 - 2 M M M M 1
3 3 2 2 2 1 1 1 2 2 M 2 2 2 2 2 2 M 3 3 M M 2 - 2 M 4 3 2 1
M M 2 M 1 - - - 1 M 2 1 1 M 2 2 M 2 2 M 4 3 2 1 2 2 1 - - -
2 3 3 2 1 1 1 1 1 1 2 2 3 3 M 2 1 1 1 2 3 M 1 1 M 1 - - 1 1
- 1 M 1 - 1 M 2 2 2 2 M M 3 2 3 2 1 - 2 M 4 2 1 1 1 1 2 3 M
- 1 1 2 1 2 1 2 M M 2 3 M 3 3 M M 2 1 2 M M 1 - - - 1 M M 3
1 1 1 1 M 1 - 1 3 3 3 2 3 M 3 M 5 M 1 1 2 2 2 2 2 1 1 2 3 M
2 M 2 2 2 1 - - 1 M 2 M 2 1 2 2 M 3 3 1 1 - 2 M M 1 - - 1 1
M 3 4 M 2 1 1 1 1 1 2 1 2 1 1 1 3 M 3 M 1 - 2 M 3 2 1 1 - -
2 M 3 M 2 1 M 1 - - - - 1 M 2 1 3 M 3 1 1 - 1 1 2 2 M 1 1 1
3 3 3 1 1 2 2 3 1 2 2 2 2 1 2 M 2 1 1 - - - 1 1 2 M 3 2 1 M
M M 1 - - 1 M 2 M 2 M M 2 2 3 2 1 - 1 1 1 - 1 M 2 3 M 3 2 1
2 2 1 - - 1 1 2 1 2 2 2 2 M M 1 - - 1 M 1 - 2 3 3 4 M M 1 -
2 3 2 2 1 1 - - - - - - - 2 3 4 2 1 1 3 3 3 1 2 M M 3 M 3 1 -
M M M 3 M 1 - - - - - - - 1 M 3 M 1 2 M M 3 M 2 2 2 3 3 4 3 2
M 4 3 M 2 1 - - - - - - - 1 2 M 2 1 2 M M 3 1 1 - - 1 M M M M
```

Aufgabe 2: Einfache Printer-Queue

(12 Punkte, Abgabedatei: queue.c)

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. */
```