Aufgaben 1: Shell Skripting

(a) Schreiben Sie ein Skript, welches die letzten 10 Zeilen
des Kernel Message Buffers sekündlich ausgibt (siehe dmesg).
Zwischen den Ausgaben soll die Konsole wieder geleert werden (siehe clear).
Falls der Benutzer das Skript mittels <strg>+<c> beenden möchte, so
soll ein Fragedialog erscheinen, in dem er die Beendigung bestätigen muss.

Beispielausgabe:

```
Ihr Kernel Message Buffer:
[ 8158.114960] sd 5:0:0:0: [sdb] Write Protect is off
[ 8158.114965] sd 5:0:0:0: [sdb] Mode Sense: 23 00 00 00
[ 8158.116811] sd 5:0:0:0: [sdb] No Caching mode page present
[ 8158.116815] sd 5:0:0:0: [sdb] Assuming drive cache: write through
[ 8158.119430] sd 5:0:0:0: [sdb] No Caching mode page present
[ 8158.119433] sd 5:0:0:0: [sdb] Assuming drive cache: write through
[ 8158.195013] sdb: sdb1
[ 8158.199931] sd 5:0:0:0: [sdb] No Caching mode page present
[ 8158.199936] sd 5:0:0:0: [sdb] Assuming drive cache: write through
[ 8158.199940] sd 5:0:0:0: [sdb] Attached SCSI removable disk
^C
Möchten Sie das Programm wirklich beenden? [J/N]
```

(b) Schreiben Sie ein Skript. Es sollen alle Dateien aus Ihrem Home-Verzeichnis, die älter als sieben Tage sind, in der Datei archiv-$DATUM.tar.gz Datei archiviert werden. Außerdem sollen die Namen aller archivierten Dateien in eine Textdatei index-$DATUM.txt geschrieben werden. ($DATUM ist jeweils durch durch das aktuelle Datum zu ersetzen.)

Aufgabe 2: C-Programme

(a) Schreiben Sie zunächst zwei kleine C-Programme "`add.c`" und "`mult.c`", die jeweils zwei Kommandozeilenparameter akzeptieren.
Diese Kommandozeilenparameter sollen anschließend addiert (in add.c) bzw. multipliziert (in mult.c) und das Ergebnis jeweils mit `exit()` zurückgegeben werden.
Für die Umwandlung der Kommandozeilenparameter (Strings/Character) zu entsprechenden Integerwerten: siehe `atoi()` (benötigt #include <stdlib.h>).

(b) Nun soll ein C-Programm geschrieben werden, welches ebenfalls zwei Kommandozeilenparameter akzeptiert.
Das Programm soll zwei Kindprozesse erzeugen. In einem Kind soll "`add`", im anderen Kind soll "`mult`" ausgeführt werden.
Beide sollen als Parameter jeweils auch die zwei Parameter des aktuellen Programms bekommen.
Wurden die Kindprozesse erzeugt, soll auf sie gewartet und ihr exit-Status, das Berechnungsergebnis, ausgelesen und auf dem Bildschirm ausgegeben werden. (Siehe `fork, exec, getpid, wait/waitpid` und die zugehörigen Makros)

Achtung: Weil unter UNIX/Linux die Rückgabewerte der Kinder lediglich als 8-Bit-Werte verfügbar sind, funktioniert das Programm nur mit sehr kleinen Zahlen (bei Ergebnissen die größer als 255 sind, kommt es zu einem Überlauf),
Dies ist für den Zweck der Übung jedoch zweitrangig.