Übungsblatt 2

Abgabe: 13. Mai 2020

Aufgabe 2.1: Ein Stack in C (1.5+3 = 4,5 Punkte)

Zeiger sind gut zum Aufbau von Datenstrukturen geeignet. Ein Beispiel dafür sind **Stacks**. Ein Stack ist eine einfache, aber dynamische Datenstruktur, um Datenelemente übereinander zu stapeln und sie anschließend in umgekehrter Reihenfolge wieder abzurufen. Dieses Prinzip wird als Last-In-First-Out (LIFO) bezeichnet. In dieser Aufgabe betrachten wir Stacks, um die `Undo`-Funktion eines Dateimanagers umzusetzen. Jede Aktion des Dateimanagers wird geloggt und auf dem Stack abgelegt (`push`$(\cdot)$). Muss eine Aktion rückgängig gemacht werden, da beispielsweise eine Datei versehentlich gelöscht wurde, können die ausgeführten Aktionen in umgekehrter Reihenfolge durch `pop`$(\cdot)$ vom Stack genommen werden. Im Lernraum finden Sie die Datei `stack.c`, die allerdings Lücken aufweist.

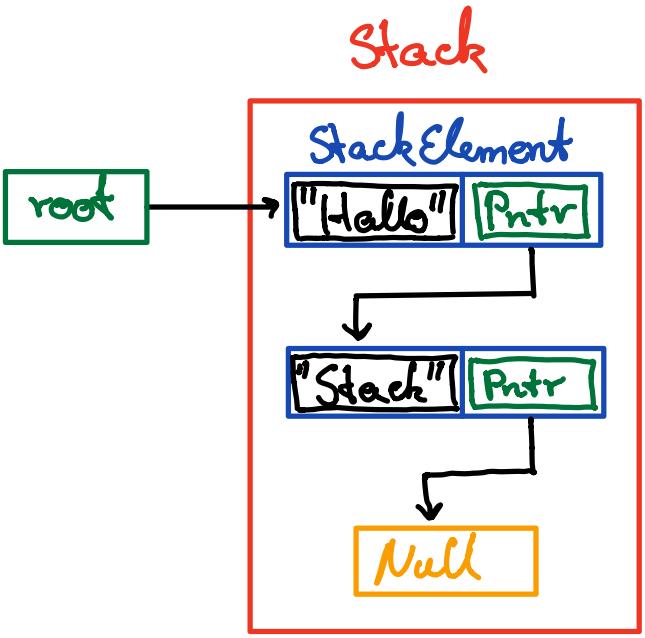

Abbildung 1: **Skizzierte Darstellung eines Stacks.** Der root-Zeiger zeigt stets auf das oberste Element eines Stacks (oder `Null` im Falle eines leeren Stacks). Stack-Elemente sind Strukturen, die aus einem beliebigen Datenfeld bestehen (bspw. `int`, `char*`, eine weitere Struktur, ...) und einem Zeiger, der auf das nächste Stack-Element verweist. Zeigt der Pointer eines Stack-Elements auf `Null`, ist es das letzte Element des Stacks.

a) Implementieren Sie die Funktion `push`$(\cdot)$:
Gehen Sie dabei so vor, dass der Zeiger `root` immer auf das oberste Element des Stacks verweist. Die Struktur `StackNode` besitzt einen Zeiger namens `next_node`, der auf das folgende Element des Stacks verweisen soll.

b) Implementieren Sie die Funktion `pop`$(\cdot)$:
Die Funktion gibt `false` zurück, falls der Stack leer ist (siehe Funktion `isEmpty`$(\cdot)$). Andernfalls soll der letzte Befehl zurückgegeben werden und das oberste Element des Stacks gelöscht werden. Stellen Sie sicher, dass es sich nach der Operation, um einen validen Stack handelt und geben Sie freien Speicherplatz auf dem Heap wieder frei.

Hinweis: Für die Rückgabe des letzten Befehls ist der Parameter `**command` vorgesehen.

Aufgabe 2.2: Bash (1+2+1+2 = 6 Punkte)

a) Geben Sie einen Kommandozeilen-Befehl an, mit dem in einer über Standard-Eingabe gelesenen Zeichenkette die erste (und nur die erste!) '1' durch eine '2' ersetzt wird. Beispiel (`mein-befehl` ist der hier zu definierende Befehl):

```
$ echo "BuS 1020: Abgabe der 2. Uebung am 11.5."| mein-befehl
BuS 2020: Abgabe der 2. Uebung am 11.5.
```

b) Schreiben Sie ein Shell-Skript, dass alle `x` Sekunden ausgibt ob ein durch eine `PID` identifizierter Prozess läuft oder nicht. Übergeben Sie die Anzahl an Sekunden und die PID als Parameter an das Skript, sodass es wie folgt aufgerufen werden kann: `$ script [PID] [SECONDS]`. Ihnen sind keine Einschränkungen gegeben, welche Befehle Sie dafür verwenden.

c) Was bewirkt das folgende Skript?

```
S=0
for f in $(find . -name "*.c"); do S=$(($S + $(wc -l $f | awk '{ print $1 }'))); done
echo $S
```

d) Schreiben Sie ein Shell-Skript, das die Verzeichnisstruktur des aktuellen oder eines als Parameter angegebenen Verzeichnisses auflisten kann. Schreiben Sie dazu eine rekursive Funktion, die in Unterverzeichnisse herabsteigt. Geben Sie das Verzeichnis und die Dateien auf dem Bildschirm aus. Nutzen Sie Einrückungen um die Zugehörigkeit von Dateien und Verzeichnissen zueinander zu kennzeichnen. Dies kann dann zum Beispiel so aussehen:

```
File: linux-3.14/COPYING
File: linux-3.14/CREDITS
Directory: linux-3.14/Documentation
  File: linux-3.14/Documentation/00-INDEX
  Directory: linux-3.14/Documentation/ABI
    File: linux-3.14/Documentation/ABI/README
    Directory: linux-3.14/Documentation/ABI/obsolete
      File: linux-3.14/Documentation/ABI/obsolete/proc-sys-vm-nr_pdflush_threads
      File: linux-3.14/Documentation/ABI/obsolete/sysfs-bus-usb
      File: linux-3.14/Documentation/ABI/obsolete/sysfs-class-rfkill
      File: linux-3.14/Documentation/ABI/obsolete/sysfs-driver-hid-roccat-koneplus
      File: linux-3.14/Documentation/ABI/obsolete/sysfs-driver-hid-roccat-kovaplus
      File: linux-3.14/Documentation/ABI/obsolete/sysfs-driver-hid-roccat-pyra
    Directory: linux-3.14/Documentation/ABI/removed
      File: linux-3.14/Documentation/ABI/removed/devfs
      File: linux-3.14/Documentation/ABI/removed/dv1394
      ...
```

Hinweis: es ist möglich, in der bash Funktionen zu definieren. Die Syntax finden Sie z.B. hier:

`http://tldp.org/LDP/abs/html/functions.html`

Aufgabe 2.3: Bash systemnah (0.5+2.5+1+1.5 = 5.5 Punkte)

a) Beschreiben Sie knapp, was ein Systemcall (Systemaufruf, Syscall) ist.

b) Beschreiben Sie in je einem Satz, was die folgenden vier wichtigen Syscalls tun:

`accept, open, write, mmap, brk`

Tipp: Die zweite Section der man-Pages beschreibt Syscalls, siehe: `man man`

c) Wozu dient das Programm `strace`? Beschreiben Sie seine Funktion.

Tipp: `strace` hat eine man-page!

d) Wir wollen `strace` nutzen, um das Core Util[1] `ls` zu analysieren. Nutzen Sie dabei die Option `-C`. Da dies viel Output erzeugt, schränken wir unsere Betrachtung außerdem noch mit `-e trace=stat,lstat,fstat,open,openat` auf die Syscalls `stat`,`lstat`,`fstat` (zwei Varianten von `stat`) und `open` bzw. die Variante `openat` ein. Analysieren Sie die folgenden Kommandos:

```
ls /etc
ls -la /etc
```

Vergleichen Sie die Ausgabe von `strace` bei Anwendung auf die beiden Kommandos. Was fällt Ihnen in Bezug auf die Anzahl und Art der auftretenden Systemaufrufe auf? Erklären Sie den Sachverhalt kurz.

Aufgabe 2.4: Bash: Textverarbeitung (0.5+0.5+2+1 = 4 Punkte)

Im Folgenden wird die Textbearbeitung mittels bash betrachtet. Beantworten Sie dazu die folgenden Fragen. Alle diese Probleme können und sollen als "Einzeiler"-Shell-Skripte durch das Kombinieren verschiedener Kommandozeilenprogramme, aber ohne Schleifen und bash-Variablen gelöst werden!

a) Schreiben Sie ein Skript, welches zu allen Dateien (und Verzeichnissen) im aktuellen Verzeichnis die Dateigröße und den Dateinamen (und auch nur diese Angaben) ausgibt. Die Ausgabe braucht nicht schön formatiert zu sein, beide Angaben können einfach durch ein Leerzeichen getrennt ausgegeben werden.

b) Modifizieren Sie die Ausgabe aus dem vorherigen Aufgabenteil so, dass Dateigröße und Dateiname in umgekehrter Reihenfolge ausgegeben werden (also "Dateigröße Dateiname" wird zu "Dateiname Dateigröße").

c) Bei der Anmeldung zu den Übungen konnte man einen String als "Teamname" festlegen. Personen mit gleichem String wurden bevorzugt der gleichen Übungsgruppe zugeordnet. Im Lernraum finden Sie eine Datei `teamnamen.txt`, die in jeder Zeile einen Teamnamen enthält, wie er von einem Studierenden vergeben worden sein könnte. Schreiben Sie ein Skript, das ausgibt, wie viele Dreiergruppen sich aus den Teamnamen ergeben. Wie müssten Sie ihr Skript ändern, um die Anzahl der Zweiergruppen auszugeben? Wie viele Einergruppen (Teamname, der nur einmal auftaucht) gibt es? Wie viele Studierende haben keinen Teamnamen (Leerzeile) angegeben?

Hinweis: `uniq`

d) Schreiben Sie ein Skript, das mit einem Aufruf die Anzahl der Einer-, Zweier- und Dreiergruppen, wie sie im vorherigen Aufgabenteil definiert wurden, ausgibt. Dabei ist sowohl eine Ausgabe der Form "<Gruppengröße> <Anzahl>" (oder umgekehrt) in drei Zeilen erlaubt, als auch die reine Ausgabe der Anzahlen in aufsteigender Gruppengröße.

Hinweis: Die Anzahl der Studierenden ohne Teampräferenz ("Nullergruppen") darf ausgeben werden, muss aber nicht.

[1] `http://www.gnu.org/software/coreutils/coreutils.html`