Aufgabe 2: Zeiger

In den folgenden Aufgaben sollen Sie den Umgang mit Zeigern üben. **Bitte schauen Sie sich daher die Folien zu Zeigern in *Einführung in C Teil 1 und 3* an.**

Vermutlich kennen Sie aus anderen Veranstaltungen die verkettete Liste (s. Abb. 1). Eine verkettete Liste besteht im Grunde genommen aus einem Zeiger auf das nächste Element (hier `next`) und beliebigen Nutzdaten (hier `data`). Der Anfang der Liste wird durch einen Zeiger (hier `head`) markiert, während das Ende der Liste durch einen Zeiger auf die Adresse `NULL` dargestellt wird. Das ist in C eine sehr elegante Methode, um ein Ende von Datenmengen unbekannter Länge zu markieren, und wird sehr häufig verwendet (s. Zeichenketten).

Da es in der C-Standard-Bibliothek keine Container gibt, wie man sie aus Java kennt, und die verkettete Liste mittels Zeiger sehr einfach zu implementieren ist, wird Ihnen diese Datenstruktur noch oft begegnen. Dies ist ein guter Grund, eine verkettete Liste selbst einmal in C zu implementieren.

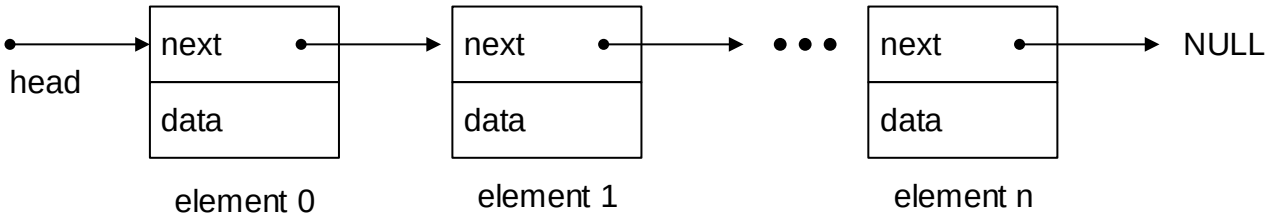

Abbildung 1: Linked List

Für die folgende Implementierung einer einfach verketteten Liste sind im `lili`-Modul alle notwendigen globalen Variablen und Funktionen bereits deklariert. Zusätzlich finden Sie in der `main.c` ein kleines Testprogramm, das Ihre Implementierung überprüft. Ziel der verketteten Liste, die Sie implementieren, soll die Speicherung von nicht-negativen Ganzzahlen sein.

2.1 Vorbereitung (1 Punkt)

Eine Buildumgebung müssen Sie diesmal nicht selber einrichten. Diese ist in Form eines **CMake(1)** Skriptes (in `CMakeLists.txt`) zu finden. CMake ist ein systemunabhängiges Buildsystem, dass die Schritte zum Bau von Programmen abstrahiert und Ihnen eine gut vorkonfigurierte Makefile (oder andere Projektdateien) erzeugt. Um mit CMake eine Makefile zu erstellen, öffnen Sie in der Konsole den Ordner `2R-zeiger/build`, wechseln Sie hinein und geben Sie dort den Befehl `cmake ..` ein. Beachten Sie hierbei bitte die zwei Punkte. Anschließend können Sie wie gewohnt mit dem Befehl **make(1)** Ihr Programm kompilieren.

Sie müssen jedoch nicht weiter auf der Konsole arbeiten. Ab dieser Aufgabe und in allen weiteren Rechnerübungen sollen Sie zum Programmieren die IDE Qt Creator verwenden (im Folgenden einfach nur Qt genannt). Um in Qt ein Projekt für die Aufgabe 2R einzurichten, gehen Sie zuerst auf *File→Open File or Project*. Wählen Sie dann in Ihrer Working-Copy die Datei `CMakeLists.txt`, die Sie im Ordner `2R-zeiger` vorfinden. Anschließend fragt Sie Qt nach dem Build Ordner. Wählen Sie hierfür den Order `2R-zeiger/build` in Ihrer Working-Copy. Im folgenden Dialog klicken Sie einfach auf *Run CMake* und dann auf *Finish*.

Als nächstes sollten Sie noch den Code-Style in Qt an die vorgegebenen Coding-Guidelines anpassen. Gehen Sie hierfür in das Menü *Tools→Options*. Wählen Sie dann im linken Fenster *C++* und klicken Sie dann im Reiter *Coding Style* auf *Import*. Importieren Sie anschließend die Datei `betriebssysteme-code-style.xml`, die Sie im Ordner `2R-zeiger` vorfinden.

Jetzt ist das Projekt fertig eingerichtet. Machen Sie sich als nächstes mit dem Inhalt der vorgegebenen Dateien vertraut.

Über die großen Symbole links unten in Qt können Sie Ihren Code kompilieren und starten. Dies schlägt jedoch zunächst fehl, da Funktionen, die in der Main-Funktion benutzt werden, momentan nicht definiert bzw. implementiert sind. Ihr erster Schritt zum Lösen dieses Aufgabenblattes sollte daher sein, das Programm ausführbar zu machen, um weitere Entwicklungsschritte sofort testen zu können. Implementieren Sie zu diesem Zweck die vorgegebenen Funktionen aus der `lili.h` als leere Funktionen in `lili.c`, sodass sie aufgerufen werden können, ohne Fehler zu erzeugen. Eine sinnvolle Aufgabe müssen diese Funktionen jetzt noch nicht erfüllen. Achten Sie darauf, dass die Funktionen einen beliebigen Rückgabewert liefern, wenn dies gefordert ist.

2.2 `insert_element()` implementieren (6 Punkte)

Implementieren Sie nun die Funktion `unsigned int* insert_element(unsigned int value)`. Diese Funktion soll mittels **malloc(3)** dynamisch zur Laufzeit neuen Speicherplatz für genau ein Element der verketteten Liste reservieren. Anschließend soll an das Ende der verketteten Liste ein Listenelement hinzugefügt werden, das den Wert `value` beinhaltet. Im Erfolgsfall gibt `insert_element` dann einen Zeiger auf `value` zurück. Sollte bei der Speicherallokation etwas fehlschlagen, geben Sie den den Grund für den Fehler mittels **perror(3)** aus und geben Sie anschließend einen Zeiger auf `NULL` zurück.

2.3 `print_lili()` implementieren (3 Punkte)

Bisher kompiliert Ihr Programmcode, aber Sie wissen noch nicht, ob `insert_element()` auch wirklich funktioniert. Daher sollen Sie nun `void print_lili()` implementieren, eine Funktion, die die verkettete Liste von Anfang bis Ende sequentiell abläuft und die gespeicherten Daten komma-getrennt in einer Zeile ausgibt.

Wenn Sie alles richtig gemacht haben, sollte Ihre Ausgabe **exakt** so aussehen:

```
remove() => 0
print_lili:
insert(47)
print_lili: 47,
insert(11)
print_lili: 47, 11,
insert(23)
print_lili: 47, 11, 23,
insert(11)
print_lili: 47, 11, 23, 11,
remove() => 0
print_lili: 47, 11, 23, 11,
remove() => 0
print_lili: 47, 11, 23, 11,
insert(18)
print_lili: 47, 11, 23, 11, 18,
insert(43)
print_lili: 47, 11, 23, 11, 18, 43,
```

2.4 `remove_element()` implementieren (5 Punkte)

Als letztes bleibt nur noch die Implementierung von `unsigned int remove_element()`. Diese Funktion soll das *älteste* Element aus der verketten Liste entfernen und den darin gespeicherten Wert `data` zurückgeben. Damit keine Speicherlecks entstehen, soll der zuvor mit **malloc(3)** reservierte Speicher mittels **free(3)** freigegeben werden. Achten Sie darauf, dass diese Funktion keinen Fehler produziert, wenn sie auf eine leere verkettete Liste angewendet wird, und eine entsprechende Warnung ausgegeben wird.

Wenn Sie alles richtig gemacht haben, sollte Ihre Ausgabe **exakt** so aussehen:

```
WARNING: nothing to remove, lili is empty
remove() => 0
print_lili:
insert(47)
print_lili: 47,
insert(11)
print_lili: 47, 11,
insert(23)
print_lili: 47, 11, 23,
insert(11)
print_lili: 47, 11, 23, 11,
remove() => 47
print_lili: 11, 23, 11,
remove() => 11
print_lili: 23, 11,
insert(18)
print_lili: 23, 11, 18,
insert(43)
print_lili: 23, 11, 18, 43,
```

Abgabe bis 09.11.2018, 23:59 Uhr.

Ihre Lösungen sollen bis zur Deadline im GitLab des IBR hochgeladen sein.
Einhaltung der Coding Guidelines: 3 Punkte
Zu erzielende Minimalpunktzahl: 9 Punkte