Bad programmers worry about the code. Good programmers worry about data structures and their relationships.
Linus Torvalds

UNIVERSITÄT DES SAARLANDES

FR Mathematik
Andreas Buchheit

11. Übung zur Vorlesung
Programmierung
im Sommersemester 2019

Abgabe: Mittwoch, den 03.07.2019 bis spätestens 12 Uhr.

Aufgabe 11.1. (12 Punkte) Doubly Linked Lists
In der Vorlesung haben Sie einfach verkettete Listen kennengelernt. Im Folgenden sollen Sie eine Erweiterung dieses Datenstuktur implementieren, eine sogenannte doppelt verkettete Liste (doubly linked list). Diese zeichnet dadurch aus, dass jede Node zusätzlich einen Pointer `node* prev` enthält der auf die vorherige Node zeigt.

(a) **(2 Punkte)** Implementieren Sie alle für den Einsatz der Liste notwendigen Strukturen, sowie eine Prozedur, die die Liste initialisiert. Implementieren Sie eine Prozedur

```
void insert_first(linked_list* list, size_t size, void* data);
```

welche eine neue Head-Node einfügt und für sie Speicher alloziert. Hierbei sollen `size` Bytes an Speicher für die Datenspeicherung innerhalb der Node alloziert werden, und der entsprechende Pointer in der Node wird gesetzt. Daraufhin sollen mittels `memcpy` `size` Bytes an Daten in die Node kopiert werden. Schreiben Sie eine zugehörige Prozedur

```
void delete_first(linked_list* list);
```

welche den Speicher wieder freigibt und die Pointer der Liste aktualisiert.

(b) **(3 Punkte)** Schreiben Sie Prozeduren

```
void insert_next(linked_list* list, size_t size, void* data);
void insert_prev(linked_list* list, size_t size, void* data);
```

die nach, beziehungsweise vor der `current` Node eine neue Node hinzufügen. Schreiben Sie eine Destruktor-Prozedur

```
delete_current(linked_list*);
```

welche die momentane Node aus der verketteten Liste entfernt und den zugehörigen Speicher freigibt. Achten Sie hierbei stets darauf, dass die Liste intakt bleibt, das heißt keine Links vergessen werden.

(c) **(2 Punkte)** Implementieren Sie Prozeduren

```
void go_next(linked_list*);
void go_prev(linked_list*);
void go_num(linked_list*, size_t num);
```

welche den `current` Pointer um eine Node nach rechts oder links, beziehungsweise zur `num`-ten Node updaten (beginnend bei 0). Am Anfang oder am Ende der Liste soll gestoppt werden.

(d) **(2 Punkte)** Mithilfe des Funktionspointers

```
typedef int (*compare_func) (void* data1, void* data2);
```

können Sie einen allgemeinen, datentypunabhängigen Funktionspointer definieren, welcher für eine Funktion steht die zwei Datensätze vergleicht. Schreiben Sie nun eine Funktion

```
node* sorted_insert(linked_list* list,
                    size_t size,
                    void* data2,
                    compare_func compare);
```

welche den Datensatz `data2` in der verketten Liste sucht, welche gemäß der Funktion `compare` geordnet ist, und den Pointer auf die zugehörige Node zurückgibt, falls er den Datensatz findet. Findet er den Datensatz nicht, so soll er eine Node mit dem zugehörigen Datensatz anlegen, sodass die Liste gemäß `compare` geordnet bleibt und den zugehörigen Pointer auf die neue Node zurückgeben.

(e) **(3 Punkte)** Mithilfe ihrer verketten Liste können nun beliebige Daten gespeichert werden. Verwenden Sie die Liste, um eine alphabetisch geordnete Liste an Namen anzulegen. Zum Vergleich der Namen können Sie die Funktion `strcmp` verwenden. Schreiben Sie zwei Prozeduren zur Ausgabe ihrer Liste, eine in alphabetischer Ordnung und eine in umgekehrter Ordnung. Vergessen Sie nicht, den allozierten Speicher wieder freizugeben.

Bonus: Speichern Sie für zwei Bonuspunkte anstatt eines Strings eine selbstdefinierte `person` Struktur in der verketten Liste ab, welche Namen und Vornamen als Felder enthält.