Stefan Kurtz **Universität Hamburg**
Mathias von Behren **Zentrum für Bioinformatik**

Programmierung in der Bioinformatik
Wintersemester 2015
Übungen zur Vorlesung: Ausgabe am 30.11.2016

Dies ist das erste Übungsblatt mit Aufgaben zur Programmiersprache Ruby. Es gibt für Ruby eine umfangreiche Sammlung von Software-Bibliotheken mit vielen Funktionen, die Ihnen beim Lösen der Aufgaben helfen können. Unter `http://ruby-doc.org/core-2.0.0/` finden Sie die entsprechende Dokumentation für die am ZBH installierte Version von Ruby. Diese Version ist auch unter Ubuntu verfügbar.

Punktevergabe:

- Aufgabe 8.1: 7 Punkte
- Aufgabe 8.3: 3 Punkte

Aufgabe 8.1 Schreiben Sie ein Modul `linkedlist.c`, welches Datenstrukturen und Operationen für eine doppelt verkettete Liste realisiert. Die Liste soll Integer vom Typ **`unsigned long`** speichern.

Definieren Sie einen eigenen Typ `List`, welcher Zeiger auf das erste und das letzte Element der Liste enthält. Deklarieren Sie des weiteren einen eigenen Typ `List_element`, um einen Knoten der Liste darzustellen. Jeder Knoten der doppelt verketteten Liste soll die Variablen `prev` (Zeiger auf den Vorgängerknoten), `next` (Zeiger auf den Nachfolgerknoten) und `value` (im Knoten gespeicherter Wert) haben.

Ihr Modul soll die folgenden Funktionen enthalten:

```
List* list_new(void);
```
Initialisierung einer neuen Liste. Die Zeiger auf das erste und letzte Element der Liste werden auf `NULL` gesetzt. Falls der Speicherplatz für die Liste nicht reserviert werden konnte, wird abgebrochen.

```
unsigned long list_element_get_value(const List_element *element);
```
Liefert den in `element` gespeicherten Wert.

```
List_element* list_first(const List* list);
```
Liefert einen Zeiger auf das erste Element der durch `list` referenzierten Liste. Für eine leere Liste wird `NULL` zurückgegeben.

```
List_element* list_last(const List* list);
```
Liefert einen Zeiger auf das letzte Element der Liste, referenziert durch `list`. Für eine leere Liste wird `NULL` zurückgegeben.

```
List_element* list_element_next(const List_element* element);
```
Liefert einen Zeiger auf den Nachfolger des durch `element` referenzierten Elementes. Hat das aktuelle Element keinen Nachfolger, wird `NULL` zurückgegeben.

```
List_element* list_element_prev(const List_element* element);
```
Liefert einen Zeiger auf den Vorgänger des Elementes, referenziert durch `element`. Hat das aktuelle Element keinen Vorgänger, wird `NULL` zurückgegeben.

`void list_append(List* list, unsigned long value);`
Hängt ein neues Element mit dem Wert `value` an das Ende der durch `list` referenzierten Liste. Nutzen sie assertions (`#include <assert.h>` und `assert()`) um sicherzustellen, dass `list` nicht `NULL` ist.

`void list_show(const List* list);`
Gibt alle Elemente der Liste (Index und Wert des Elements) beginnend mit dem ersten Element zeilenweise auf `stdout` aus.

`void list_element_show(const List_element* element);`
`list_element_show` gibt den Wert eines Elements auf `stdout` aus.

`void list_delete_element(List* list, List_element* element);`
Löscht das Listenelement, auf das `element` zeigt, aus der Liste, die durch `list` referenziert wird. Nutzen sie assertions, um verifizieren, dass beide Zeiger nicht `NULL` sind. Nach dem Löschen darf es keine „Lücken“ in der Liste geben.

`void list_delete(List* list);`
Gibt den Speicher für alle Elemente der Liste sowie für die Liste selbst wieder frei.

Testen Sie Ihre Implementation mit den Dateien `linkedlist_test.c` und `linkedlist.dat`, die Sie in Stine finden. Nutzen Sie die dort zu findende Headerdatei `linkedlist.h` und das Makefile `linkedlist_make`.

Um das Makefile zu verwenden benennen Sie es um zu `makefile` oder rufen Sie make wie folgt auf: `make -f linkedlist_make`. Verwenden Sie das make-target `test` um den Test direkt nach dem Kompilieren auszuführen.

Zusatzaufgaben für Fortgeschrittene

Schreiben Sie zusätzlich folgende Funktionen:

`void list_ordered_insert(List* list, unsigned long value);`
Diese Funktion fügt den Wert `value` als neues Element an der richtigen Stelle in die sortierte Liste `list` ein. Nutzen sie assertions.

`bool list_search(const List* list, unsigned long value);`
In der Liste wird ausgehend vom ersten Listenelement nach Elementen mit dem Wert `value` gesucht. Die Funktion geht davon aus, dass die Liste sortiert ist. Wird ein Element mit dem Werte `value` gefunden, liefert die Funktion `true` zurück, sonst `false`.

Um diese zu Testen passen Sie den Programmcode des Tests an.

Aufgabe 8.2 Sie finden in Stine ein Ruby-Tutorial `ruby20minutes.pdf`, das einige der wichtigsten Konzepte von Ruby einführt. Ein Ausdruck des Tutorials wurde in der Vorlesung verteilt. Bearbeiten Sie dieses Tutorial vor allen anderen Ruby-Aufgaben, da diese darauf aufbauen. Verwenden Sie zur Bearbeitung die interaktive Ruby-Shell `irb`, die es erlaubt, Fragmente von Ruby-Programmen auszuprobieren.

Aufgabe 8.3 Machen Sie sich mit dem Ruby Interpreter und seinen Fehlermeldungen vertraut. Verwenden Sie dazu den folgenden fehlerhaften Programmtext

```
#!/usr/bin/env ruby
```

```
dna = 'ACGAATT\tACTTTAGC'
rna = dna.gsub!(/T/, /U/)

print "Here "is the starting DNA:\t"
print "#{dna}\n\n"

print "Here is the result:\n\n"
puts "#{rna)\n"
exit 0
```

den Sie im Material zum Übungsblatt in Stine in der Datei `error10-1.rb` finden.

1. Erstellen Sie eine Kopie hiervon in der Datei `accept10-1.rb`. Modifizieren Sie diese zunächst so, dass der Ruby-Interpreter keine Fehlermeldungen mehr zeigt. Die Anzahl der Änderungen soll dabei möglichst klein sein.
2. Erstellen Sie in der Datei `correct10-1.rb` eine Kopie von `accept10-1.rb`, und modifizieren `correct10-1.rb` so, dass es die folgende Ausgabe liefert:

   ```
   Here is the starting DNA: ACGAATT ACTTTAGC
   Here is the result: ACGAAUU ACUUUAGC
   ```

Dokumentieren Sie Ihre Änderungen. Dazu können Sie das Programm `sdiff -s` verwenden.

Die Lösungen zu diesen Aufgaben werden am 14.12.2015 besprochen.