S. Kurtz, T. Schwabe,
G. Gonnella, D. Willrodt

Universität Hamburg
Zentrum für Bioinformatik

Programmierung für Naturwissenschaften
Sommersemester 2015
Übungen zur Vorlesung: Ausgabe am 30.04.2015

Punkteverteilung: Aufgabe 5.1: 5 Punkte, Aufgabe 5.2: 5 Punkte

Abgabe bis zum 06.05.2015, 10:00 Uhr.

Aufgabe 5.1 Implementieren Sie eine Funktion
`void tokenizer(FILE *fp, TokenHandlerFunc tokenhandler, void *data);` die aus einem Datei-Zeiger `fp` zeichenweise liest. Die Folge der Eingabezeichen soll zu sogenannten Worten (Token) zusammengefaßt werden. Ein Wort besteht aus einer zusammenhängenden Folge von alphanumerischen Zeichen und dem Unterstrich (_) ohne Leerzeichen oder Sonderzeichen. Das aktuelle Wort soll jeweils in einem dynamisch allokierten Array über dem Basistyp `char` gespeichert werden. Achten Sie darauf, dass diese Strings mit `'\0'` terminiert sind.

Die an `tokenizer` übergebene Funktion `tokenhandler` wird dann jeweils auf das aktuelle Wort angewendet. Falls die Funktion `tokenhandler` weitere Daten braucht, um ein Token zu bearbeiten, so sind diese Daten über den dritten Parameter `data` zu übergeben. Dieser Pointer wird als zweiter Parameter an `tokenhandler` übergeben, wenn die Funktion aufgerufen wird. Sollte `tokenhandler` keine weiteren Daten benötigen, so wird `NULL` an `tokenizer` übergeben. Die Deklaration des Typsynonyms `TokenHandlerFunc` sieht also wie folgt aus:

```
typedef void (*TokenHandlerFunc)(const char *, void *);
```

Schreiben Sie diese beiden Deklarationen in eine Datei `tokenizer.h` und implementieren Sie `tokenizer` in einer Datei `tokenizer.c`. Implementieren Sie nun ein Hauptprogramm `enumwords.c`, dass einen Dateinamen übergeben bekommt. Diese Datei wird mit `fopen` zum Lesen geöffnet und der entsprechende `FILE`-Zeiger an die Funktion `tokenizer` übergeben. Außerdem bekommt letztere Funktion noch als zweiten Parameter eine Funktion, die den Inhalt des \0-terminierten Stringpuffers in `tokenizer` zeilenweise auf die Standard-Ausgabe schreibt. Wenn man beispielsweise `enumwords.x` mit dem Namen der Datei, die die Beschreibung dieser Übungsaufgabe enthält, aufruft, dann sehen die ersten vier Zeilen der Ausgabe so aus:

```
Implementieren
Sie
eine
Funktion
```

Verwenden Sie wie gewohnt das Makefile aus den Materialien zu diesem Übungsblatt und rufen `make test` auf. Ihr Programm muss diesen Test erfolgreich absolvieren. Der Test verwendet die ebenfalls in den Materialien vorhandenen Dateien `text.txt` und `test.out`.

Aufgabe 5.2 In der fiktiven Welt Simul lebt die fiktive Bakterienart *Enpe completii*. Das Genom des Bakterium enthält ein Gen *dolly*, welches in zwei Varianten (Allelen) *dolly-A* und *dolly-B* vorkommen kann.

Auf Simul läuft die Zeit in diskreten Zeiteinheiten. Wir sprechen daher von Generationen. In jeder Generation hat jedes *Enpe completii* Bakterium eine gewisse Wahrscheinlichkeit, sich zu vermehren, d.h. sich in zwei identische Individuen zu teilen. Diese Wahrscheinlichkeit p_A bzw. p_B ist vom *dolly*-Allel abhängig (p_A für *dolly-A*, p_B für *dolly-B*).

Da die Ressourcen auf Simul stark beschränkt sind, kann die Populationsgröße nicht wachsen. Falls ein *Enpe completii* sich vermehrt, stirbt daher unmittelbar nach der Teilung ein zufälliges Individuum der Population. Dieses darf auch einer der zwei neu aus der Zellteilung entstandenen Individuen sein. Die Populationsgröße ist also in allen Generationen konstant.

Schreiben Sie ein C Programm `simul_evolution.c` welches durch eine Simulation ermittelt, nach wie vielen Generationen die gesamte Population von *Enpe completii* entweder nur noch das Allel *dolly-A* oder nur noch das Allel *dolly-B* enthält, d.h. diese Variante des Gens *dolly* ist fixiert.

Das C Programm hat 5 Kommandozeilenparameter:

1. n_A = Anzahl von *dolly-A*-Individuen am Anfang
2. p_A
3. n_B = Anzahl von *dolly-B*-Individuen am Anfang
4. p_B
5. `maxsteps` = Anzahl der maximal zu simulierenden Zeiteinheiten

Sobald ein Generation entsteht, in der eine der beiden Varianten von *dolly* fixiert ist, endet das Programm und liefert eine Ausgabe in folgendem Format:

```
fixed:<A|B><TAB>steps:<nsteps>
```

Falls nach `maxsteps` Schritten keine solche Generation entstand, endet das Programm und liefert eine Ausgabe in folgendem Format:

```
simulation stopped after <nsteps> steps (A:<nA>,B:<nB>)
```

In einem zusätzlichen optionalen Kommandozeilenparameter kann der Name einer Ausgabedatei angegeben werden. Falls ein solcher Dateiname angegeben wird, werden nach der Kopfzeile `step<TAB>nA<TAB>nB` für jede Generation die aktuellen Werte für n_A und n_B zeilenweise in folgendem Format in die entsprechende neu zu erstellende Datei geschrieben:

```
nstep<TAB>nA<TAB>nB
```

z.B.:

```
0<TAB>100<TAB>100
1<TAB>99<TAB>101
2<TAB>98<TAB>102
...
```

In STiNE finden Sie ein Makefile, mit dem Sie Ihr Programm kompilieren und testen können.

Die Lösungen zu diesen Aufgaben werden am 07.05.2015 besprochen.