ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert
Moritz Carmesin

29. November 2019
Blatt 6

Systemnahe Software I (WS 2019/2020)

Abgabe bis zum 6. Dezember 2019, 14:00 Uhr

Lernziele:

- Aufbau und Verwaltung dynamischer Datenstrukturen

Aufgabe 8: Long chain of short trips

In dieser Aufgabe machen wir eine kleine Reise durch Deutschland. Sie starten in Ulm und sollen einen Weg mit vielen möglichst nicht weit voneinander entfernten Zwischenstationen bis nach Kiel finden. In der Textdatei *gemeinden.txt* stellen wir die notwendigen Daten für alle Gemeinden in Deutschland bereit. Jede Gemeinde ist hier durch eine Zeile repräsentiert, bei der zu Beginn der Name angegeben ist und durch Doppelpunkte getrennt die geographischen Koordinaten (Breitengrad und Längengrad). Die Daten wurden aus der deutschsprachigen Wikipedia extrahiert. So sehen zwei ausgewählte Datensätze in der Datei aus:

```
theon$ grep Ulm: gemeinden.txt
Neu-Ulm:48.39417:9.99889
Ulm:48.40083:9.98722
theon$
```

Das Ziel des Spiels ist es, von Ulm nach Kiel zu gelangen und dabei Zwischenaufenthalte in Gemeinden und Städten so auszuwählen, dass die maximale Distanz minimiert wird, die zwischen zwei aufeinanderfolgenden Orten zurückzulegen ist. Hier ist ein beispielhafter Spielverlauf:

```
theon$ distances
*** Long chain of short trips ***
Your objective is to travel from Ulm to Kiel
through a long chain of intermediate towns where
the maximal distance between two consecutive towns
of your journey is to be minimized.
You are currently located in Ulm.
```

```
Next town? Feuchtwangen
Distance from Ulm to Feuchtwangen: 91.5 km
This is the new maximal distance!
You are currently located in Feuchtwangen.
Next town? Würzburg
Distance from Feuchtwangen to Würzburg: 81.0 km
You are currently located in Würzburg.
Next town? Fulda
Distance from Würzburg to Fulda: 87.9 km
You are currently located in Fulda.
Next town? Kassel
Distance from Fulda to Kassel: 85.9 km
You are currently located in Kassel.
Next town? Göttingen
Distance from Kassel to Göttingen: 53.7 km
You are currently located in Göttingen.
Next town? Hildesheim
Distance from Göttingen to Hildesheim: 67.6 km
You are currently located in Hildesheim.
Next town? Hannover
Distance from Hildesheim to Hannover: 34.0 km
You are currently located in Hannover.
Next town? Walsrode
Distance from Hannover to Walsrode: 56.7 km
You are currently located in Walsrode.
Next town? Hamburg
Distance from Walsrode to Hamburg: 87.7 km
You are currently located in Hamburg.
Next town? Neumünster
Distance from Hamburg to Neumünster: 57.1 km
You are currently located in Neumünster.
Next town? Kiel
Distance from Neumünster to Kiel: 32.2 km
Welcome to Kiel!
Your maximal intermediate distance was 91.5 km
theon$
```

Um für einen eingegebenen Ortsnamen rasch die zugehörigen geographischen Koordinaten zu ermitteln, wird eine Hash-Tabelle benötigt, mit deren Hilfe Ortsnamen in Zeiger in die entsprechende Datenstruktur für eine Gemeinde abgebildet werden. Die Hash-Tabelle soll aus einer sogenannten Bucket-Tabelle bestehen, bei der alle Gemeinden mit dem gleichen Hash-Wert in einer linearen Liste geführt werden. Der dafür notwendige Speicher ist dynamisch per *malloc* oder *calloc* zu belegen. Ihr Programm muss dabei überprüfen, ob das Belegen von Speicher jeweils erfolgreich war. Wenn ein Nullzeiger zurückgeliefert wird, sollte Ihr Programm die Ausführung mit einer Fehlermeldung beenden.

Hinweise:

Für das Einlesen von Zeilen dürfen Sie gerne ein festdimensioniertes **char**-Array verwenden – 128 Bytes sind hierfür ausreichend. Als Einlesefunktion empfiehlt sich hier die Verwendung von *fgets*, das allerdings den Zeilentrenner am Ende belässt (falls der noch hineingepasst hat). Wenn Sie im Einlese-Array die Doppelpunkte bzw. den Zeilentrenner durch Nullbytes ersetzen, haben Sie drei Zeichenketten für den Namen der Gemeinde, die Breite und die Länge, ohne diese zunächst umkopieren zu müssen. Spätestens aber, wenn Sie in der dynamischen Datenstruktur einen neuen Datensatz für eine Gemeinde anlegen, müssen Sie den Namen dann in den dynamisch belegten Speicher umkopieren. Hierzu empfiehlt sich die Funktion *strdup* aus **#include** <string.h>. So wird sichergestellt, dass in der Datenstruktur der Hash-Tabelle keine Zeiger auf lokale Arrays verbleiben. Für die Konvertierung der Koordinaten in entsprechende **double**-Werte empfiehlt sich die Verwendung der Funktion *sscanf*() oder *atof()*.

Für die Distanzen zwischen zwei Reisestationen soll auf dem WGS84-Ellipsoid der Abstand berechnet werden. Die hierzu verwendbare Formel findet sich in der Wikipedia. Nach **#include** <math.h> sind die hierfür benötigten trigonometrischen Funktionen wie *sin*, *cos* und *atan* benutzbar. Unter Solaris sollte dann noch beim Zusammenbau des Programms die Option „-lm“ angegeben werden. Für die Konstante π empfielt es sich, diese selbst direkt mit **#define** PI 3.14159265358979323846 zu vereinbaren. Gelegentlich wird π unter dem Namen *M_PI* in <*math.h*> zur Verfügung gestellt – dies sichert der C-Standard jedoch nicht zu.

Reichen Sie bitte Ihre Lösung mit folgendem Kommando ein:

```
theon$ submit ss1 8 distances.c
```

Das Programm wird dabei einigen Tests unterzogen, bei der die Ausgabe verglichen wird. Diese Tests sind naturgemäß nur dann erfolgreich, wenn Sie die Ausgabetexte der Vorlage wortwörtlich übernehmen. Wenn die Eingabe beendet wird, sollte "Bye!\n" ausgegeben werden, gefolgt von einem Aufruf von *exit*(1). Die Vorlage rechnet mit **double** und gibt die Distanzen mit dem *printf*-Format "%.1lf" aus. Wenn Sie möchten, können Sie gerne die Vorlage verwenden, bei der alle *printf*-Aufrufe bereits enthalten sind.

Viel Erfolg!