Aufgabe	1	2	3	∑
Punkte	6	23	0	29
Bonus	0	0	2	2
Erreicht				

123456

Max Mustermann

AuP (Wintersemester 22/23)

Nicht öffnen, warten Sie auf Anweisungen!

PRÜFUNG

ALGORITHMEN UND PROGRAMMIERUNG

Willkommen zur Klausur **Algorithmen und Programmierung**! Beachten Sie bitte die folgenden Regeln:

- Sie haben zur Beantwortung aller Fragen 300 Minuten Zeit.
- Mit der Beantwortung des Take-Home-Examens erklären Sie, dass Sie **gesundheitlich in der Lage** sind, eine Prüfung abzulegen.
- Bitte stellen Sie Ihre Lösung lesbar und sauber dar. Soweit möglich verwenden Sie bitte digitale Programme zur Darstellung der Lösung. Achten Sie bei Fotos oder Scans auf Lesbarkeit. Sollten Sie Annahmen treffen, stellen Sie diese ebenfalls dar.
- Quellcode muss in C-Version C99 geschrieben sein (keine Erweiterungen, z.B. gnu/IBM). Jede Quellcodedatei muss eigenständig übersetzbar (gcc –std=c99 -Wall -c <file.c>) sein und mit der Compileroption -Wall keine Warnungen oder Fehler werfen. Quellcodedateien oder Headerdateien müssen im ASCII-Format vorliegen (.c oder .h Dateien). Laden Sie keine Projektdateien, Makefiles, Objektdateien oder ausführbare Dateien hoch.
- Es dürfen (außer selbstdefinierten) ausschließlich die nach dem C99-Standard notwendig zur Standardbibliothek gehörenden Funktionen genutzt werden[a].
- Die Nutzung von *goto*-Anweisungen, globalen und/oder statischen Variablen ist **nicht** erlaubt.
- Es ist auf korrekte Speichernutzung zu achten (keine Speicherzugriffsfehler, Freigabe von dynamisch angelegten Variablen etc.)
- Laden Sie alle Lösungen in OPAL hoch. Sie können hierbei mehrere Dateien verwenden. Wenn Sie Grafiken oder Scans hochladen, verwenden Sie ausschließlich .pdf, .jpg und .png Dateien. Texte können in UTF8- oder ASCII-Plaintext, oder als .pdf hochgeladen werden. Verwenden Sie keine Wordprozessorformate wie *.doc, *.dox *.rtf etc. Quellcode muss als .c-Datei hochgeladen werden.
- Die Arbeit ist selbstständig anzufertigen. Es dürfen zur Erstellung die üblichen Tools zur Programmerstellung (Compiler, Editoren, Manpages,...) genutzt werden.
- Plagiieren führt zum vollständigen Punktverlust bei allen Beteiligten (also auch ggf. beim Plagiatsgeber). Während der Korrektur der Prüfung wird eine Plagiatsprüfung durchgeführt.
- Bei unerlaubten Handlungen wird die Prüfung **aller involvierten Studierenden** mit Note 5 bewertet.
- Wenn Sie eine Frage oder technische Schwierigkeiten haben, melden Sie sich bitte bei den Betreuern im privaten Chat im BigBlueButton-Raum unter `https://webroom.hrz.tu-chemnitz.de/gl/mar-7ss-4nr-en5`.

Zum Bestehen der Prüfung müssen Sie mindestens 15 Punkte erreichen. Viel Erfolg!

[a] Siehe z.B. `http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf`

Komplexaufgabe - Rundreise

Piraten in der Karibik sind allgemein für ihre Beutezüge bekannt. Doch was machen sie eigentlich, wenn sie einmal nicht unterwegs sind, um die nächste Prise einzufahren? Ganz klar, sie spielen Dame um ihre Schiffe, ihr wichtigstes Hab und Gut. Wer hat denn schon einen Piraten ohne Schiff gesehen? Das wäre ja kein Pirat mehr. Nein, das wäre ein einfacher Dieb und kein Pirat, der etwas auf sich hält, lässt sich gerne als Dieb abstempeln.

Dummerweise hat Captain Jack letzten Abend sein eigenes Schiff verspielt. Doch er kann sich diesem Verlust nicht einfach hingeben. Thatch, an den Jack sein Schiff verloren hat, gibt ihm eine letzte Chance, sein Schiff zurückzukaufen. Wenn Jack 100 Dukaten auftreiben kann, bevor das Schiff in 4,5 Stunden versteigert wird, kann er es zurückkaufen.

Jack hat von Schätzen an verschiedenen Orten gehört, die zusammengenommen genug Dukaten liefern, um das Schiff zurückzukaufen und das Ganze mit einer Flasche Rum zu feiern. Da er kein Schiff mehr besitzt, muss Jack diese Orte mit den öffentlichen Fähren aufsuchen. Diese Fähren sind zwar nicht sonderlich schnell, fahren aber immerhin immer genau dann, wenn man sie braucht (keine Wartezeit auf die nächste Fähre). Helfen Sie Jack eine Route von Ort zu Ort zu finden, welche jeden Ort (außer den Ausgangspunkt) genau einmal besucht. Das Ende der Route soll ihr Ausgangspunkt sein. Zusätzlich muss die Route schnell genug sein, sodass Jack rechtzeitig vor der Versteigerung wieder zurück ist.
Ein beispielhafter Aufruf Ihres Programmes könnte wie folgt aussehen:

```
reise 3 Havanna Santiago Kingston Havanna Santiago 60 Havanna Kingston 90
   Santiago Kingston 120
```

Der erste Parameter nach dem Programmnamen enthält die Information über die Anzahl der Orte, die Jack besuchen muss. Darauf folgen die Namen dieser Orte. Zuletzt bekommen Sie die Fährverbindungen übergeben. Fährverbindungen bestehen zwischen zwei Orten und benötigen eine Reisezeit (in Minuten). Sie sind symmetrisch. Das heißt: Wenn eine Verbindung von A nach B existiert, dann existiert auch eine Verbindung von B nach A mit derselben Dauer für die Überfahrt.

Zur Verwaltung der Daten bekommen Sie in der Datei `reise.h` zwei Datenstrukturen zur Verfügung gestellt. Zusätzlich wird der Datentyp `minutes_t` für Zeitwerte in Minuten definiert.

Listing 1: Definition des Datentypes `stadt_t`

```
typedef struct {
    size_t id;
    char name[30];
} stadt_t;
```

Typ `stadt_t` ist eine `struct`, welche für einen Ort einen eindeutigen Namen und eine eindeutige ID verwaltet. Der Name einer Stadt darf maximal 29 Zeichen lang sein. Die Definition des Datentyps wird in Listing 1 gezeigt.

Listing 2: Definition des Datentypes `routen_t`

```
typedef struct {
    size_t anzahlStaedte;
    stadt_t* staedte;
    minuten_t** verbindungen;
} routen_t;
```

Der zweite Typ `routen_t` dient der Verwaltung der Fährverbindungen. Seine Definition wird in Listing 2 präsentiert. Der Verbundtyp besteht aus drei Variablen. Variable `anzahlStaedte` enthält die Anzahl aller Orte. Der Pointer `staedte` zeigt auf eine Liste von Orten. Variable `verbindungen` verweist auf ein dynamisch angelegtes Array von Pointern. Diese verweisen wiederum je auf ein Array des Typs `minutes_t`. In diesen Arrays wird jeweils die Dauer der Fährfahrten von einem Ort zu allen anderen Orten verwaltet. Eine Variable des Typs `routen_t` kann für den gegebenen Beispiel-Aufruf wie folgt aussehen (kein C-Code):

```
anzahlStaedte = 3
staedte = {{0, Havanna}, {1, Santiago}, {2, Kingston}}
verbindungen = {
    [0] -> { 0,  60,  90},
    [1] -> {60,   0, 120},
    [2] -> {90, 120,   0}
}
```

Ein gültiger Rundweg für dieses Beispiel wäre:

```
{{0, Havanna}, {2, Kingston}, {1, Santiago}, {0, Havanna}}
```

Eine Eingabe zum Testen Ihres Programms stellen wir Ihnen in der Datei `input.dat` zur Verfügung. Für den Aufruf ihres Programms mit diesen Eingaben verwenden Sie den folgenden Befehl:

```
cat input.dat | xargs ./reise
```

Dadurch werden die Daten als Parameter dem Programmaufruf übergeben.

Nutzen Sie die Datei `reise.c` für die Implementation von Aufgabe 2. Die darin enthaltene `main`-Funktion dürfen Sie anpassen, solange Sie dabei die Funktionssignaturen und -aufrufe der in der Aufgabenstellung vorgegebenen Funktionen nicht verändern. Sie dürfen jederzeit Hilfsfunktionen implementieren, wenn Sie es für notwendig erachten. Die definierten Datenstrukturen und Typdefinitionen dürfen Sie nicht verändern.

Aufgabe 1 ..... (6)

Die folgenden Aufgaben sollen Ihnen bei der Modellierung Ihres Programmierproblems helfen.

(a) Erstellen Sie eine EBNF für die Syntax des Programmaufrufs. Namen von Städten bestehen dabei immer aus einem Großbuchstaben, gefolgt von mehreren Kleinbuchstaben (die Grenze von 29 Zeichen muss hier nicht berücksichtigt werden). Zeiten sind immer in ganzen Minuten, ohne führende Nullen, gegeben. Die Anzahl der Städte ist stets eine natürliche Zahl. Auch hier gibt es keine führenden Nullen. Den Namen des Programms müssen Sie in der EBNF nicht beachten. Weiterhin müssen Sie nicht auf die semantische Korrektheit der von Ihren Pruduktionsregeln erzeugten Worte achten. Das heißt, dass Sie nicht darauf achten müssen, dass die gegebene Anzahl von Städten mit der Anzahl gelesener Stadtnamen übereinstimmt. (2)

(b) Beschreiben Sie kurz das allgemeine Vorgehen beim Backtracking. (1)

(c) Übertragen Sie Backtracking auf die Funktion `findeWeg` (2d) und geben Sie diese in Pseudocode an. (3)

Aufgabe 2 ..... (23)

Schreiben Sie ein C-Programm `reise.c`, mit dessen Hilfe Jack eine geeignete Route für seine Schatzsuche finden kann. Bearbeiten Sie dazu die nachfolgenden Aufgaben:

(a) Implementieren Sie eine Funktion `erzeugeRouten`, die aus den übergebenen Parametern die gewünschte Datenstruktur erzeugt und entsprechenden Speicher alloziert. Der Funktion werden die Kommandozeilenparameter (ohne Programmname) aus der main-Funktion zusammen mit der Parameteranzahl übergeben. Die ID der ersten übergebenen Stadt soll 0 sein. (5)

```
routen_t erzeugeRouten(size_t anzahlParameter, const char**
    parameter);
```

(b) Schreiben Sie eine Funktion freigeben, die allozierten Speicher für Variablen des Typs `routen_t` freigibt. (3)

```
void freigeben(routen_t routen);
```

(c) Schreiben Sie eine Funktion `sortiereStaedte`, welche die Liste von Städten innerhalb der übergebenen Datenstruktur `routen` alphabetisch (beginnend bei a) nach ihrem Namen sortiert (zwischen Groß- und Kleinschreibung wird nicht unterschieden). Die Zuordnung der IDs zur Stadt soll dabei nicht verändert werden. Die sortierte Liste muss durch eine aufrufende Funktion nutzbar sein. (5)

```
void sortiereStaedte(routen_t routen);
```

(d) Schreiben Sie eine Funktion `findeWeg`, die mittels Backtracking einen geeigneten Rundweg für die verbleibende Zeit findet. Das heißt, dass jeder Ort außer dem Ausgangsort genau einmal besucht werden muss. Der Ausgangsort ist der Ort mit der ID 0. (10)

Die Funktion bekommt die `routen` sowie die verbleibende Zeit `maxDauer` übergeben. Sie können davon ausgehen, dass `routen` mit `erzeugeRouten` initialisiert wurde. Allerdings soll die Funktion `findeWeg` sowohl mit der sortierten, als auch mit der unsortierten Liste von Städten funktionieren. Variable `maxDauer` enthält die maximal verbleibende Zeit bis zur Versteigerung in Minuten. Zusätzlich wird ein Pointer auf einen Pointer übergeben, in dem nach dem Funktionsruf das dynamische Feld mit dem gefundenen Rundweg gespeichert wird (`weg`). Dieser Weg beginnt und endet mit dem Ausgangspunkt der Rundreise. Pointer `weg` zeigt initial auf einen `NULL`-Pointer. Die Funktion gibt die Anzahl der Städte im gefundenen Rundweg zurück. Kann kein Weg gefunden werden, so soll 0 zurückgegeben werden. Beachten Sie, dass Sie mehrere Hilfsfunktionen nutzen dürfen.

```
size_t findeWeg(routen_t routen, stadt_t** weg, minuten_t maxDauer);
```

Aufgabe 3 (Bonus) ..... (+2)

Analysieren Sie die Laufzeit-Komplexität Ihrer Lösung aus Aufgabe 2c in Abhängigkeit der Länge der Liste von Städten in Variable `routen`. Nehmen Sie dabei für alle C-Befehle (außer Schleifen und selbst geschriebene Funktionen) eine Laufzeit von 1 an. Welche Komplexitätsklasse ergibt sich?