Fakultät Verkehrswissenschaften „Friedrich List“
Institut für Wirtschaft und Verkehr
Professur für Verkehrsbetriebslehre und Logistik

Klausur
Grundlagen der Programmierung - Gruppe D
Wintersemester 2017/2018
Dresden, 28.02.2018

Prof. Dr. Jörn Schönberger

Aufgabe	1	2	3	4	Bonus	$\sum$
Teilaufgaben	(a)-(h)	(a)-(e)		(a)-(n)		
erreichbar	20	20	10	40		90
erreicht						

Name: _____

Matrikelnummer: _____

Studiengang: ☐ VWI-Bachelor ☐ WiWi ☐ Sonstiges:

Hinweise:

a) Die Bearbeitungszeit beträgt 90 Minuten. Insgesamt können 90 Punkte erreicht werden.

b) Die Klausur besteht aus **4** Aufgaben. Überprüfen Sie die Klausur auf Vollständigkeit!

c) Bitte benutzen Sie die Vorder- und Rückseiten der Lösungsblätter. Benutzen Sie ausschließlich das ausgehändigte Schreibpapier, anderes abgegebenes Papier macht die Klausur ungültig.

d) Es sind alle Aufgaben zu bearbeiten.

e) Bitte versehen Sie jeden Lösungsbogen Ihrer Klausur mit Ihrem Namen und Ihrer Matrikelnummer. Die Aufgabenblätter sind Bestandteil der Klausur und sind mit abzugeben. Schreiben Sie zudem Ihren Namen und Ihre Matrikelnummer an den Beginn jeder abzugebenen Datei.

f) Zugelassene Hilfsmittel sind Schreibgeräte, ausgedruckte Unterlagen der Studierenden sowie Bücher (auch mit eigenen Notizen), ein Übersetzungswörterbuch ohne handschriftliche Notizen und ein nicht-programmierbarer Taschenrechner. Bleistift und Rotstift dürfen nicht verwendet werden. Elektronische Geräte mit Kommunikationsmöglichkeiten sind nicht als Hilfsmittel erlaubt.

Ich habe die Hinweise zur Kenntnis genommen:

Unterschrift: _____ Viel Erfolg!

Aufgabe 1 (Grundlagen und Theorie) **20 Punkte**

a) Die folgenden Bezeichner dürfen nicht als Variablennamen in C++ verwendet werden. Begründen Sie jeweils kurz weshalb. (3 / 20 Punkten)

- auto
- 2easy
- C++

b) Erläutern Sie eine Verwendung des Präprozessors. (2 / 20 Punkten)

c) Nennen Sie die drei Sprachelemente von C++. (3 / 20 Punkten)

d) Nennen Sie zwei Fälle, in denen der Compiler die Implementierung bestimmter Anweisungen übernimmt, sofern sie nicht explizit durch den Programmierer selbst getätigt wurden. (2 / 20 Punkten)

e) Skizzieren Sie jeweils das Syntax-Diagramm für eine `for`-Schleife und die Definition einer Methode. (4 / 20 Punkten)

f) Nennen und erläutern Sie den einzigen Unterschied zwischen der Verwendung von `class` und `struct` bei der Definition eines Benutzertyps. (2 / 20 Punkten)

g) Begründen Sie, warum die Anweisung `5.0 % 3` einen Compiler-Fehler verursacht. Berichtigen Sie den Fehler, sodass die Auswertung des Ausdrucks `2` ergibt. (2 / 20 Punkten)
Hinweis: Ihnen stehen mehrere Möglichkeiten zur Verfügung, die bewirken, dass die Auswertung des Ausdruckes den Wert `2` annimmt

h) Erläutern Sie den Unterschied zwischen einer Zeigervariable und einer Referenz. (2 / 20 Punkten)

Aufgabe 2 (Verständnis von C++ - Quelltext) 20 Punkte

a) In Quelltext 1 sehen Sie die Funktion `summe_matrix()`, welche die Summe aller Elemente einer Matrix berechnet. In diesem Codestück haben sich **fünf** Syntaxfehler versteckt! Markieren Sie die Fehler innerhalb des Quelltextes. Berichtigen Sie die Fehler. (5 / 20 Punkten)

Quelltext 1: Syntaxfehler

```
1   int summe_matrix(int ** matrix, int zeilen; int spalten)
2   {
3     int summe = 0;
4     for (int i = 0; i < zeilen; i+) {
5       for (int j == 0; j < spalten; j++)
6       {
7           summe = summe + matrix(i)(j);
8       }
9     }
10    return summe
11  }
```

b) Nach wie vielen Durchläufen wird die Schleife in Quelltext 2 verlassen? Geben Sie die Werte an, welche von den Variablen `x` und `y` nach dem Verlassen der Schleife auf der Konsole ausgegeben werden. (3 / 20 Punkten)

Quelltext 2: Do-While Quelltext

```
1   int x = 1, y = 10000;
2   do
3   {
4     y = y / 10;
5     x++;
6   } while (y >= 10);
7   cout << "x ist " << x << "\ty ist " << y << endl;
```

c) Was ergibt die Auswertung der folgenden Ausdrücke? Nennen Sie zudem den jeweiligen Datentyp. (3 / 20 Punkten)

- `1 / 2 * 90;`
- `5 > 10;`
- `0 ? 42.0 : 5.0;`

d) Was ist der Zweck der in Quelltext 3 dargestellten Funktion `doSmth`. Wie nennt sich die hier verwendete Programmiertechnik, bei der sich eine Funktion selbst aufruft? (3 / 20 Punkten)

Quelltext 3: Interpretation

```
1   int doSmth(unsigned int n)
2   {
3     return (n == 0) ? 1 : n * doSmth(n - 1);
4   }
```

e) Das folgende Programm in Quellcode 4 enthält **drei** logische Fehler, ist aber syntaktisch korrekt und verursacht auch keinen Laufzeitfehler. Das eigentliche Ziel des übernächtigten Programmierers war es, den Wert und den Index des kleinsten Elementes in einem Array, dessen Länge bekannt ist, zu ermitteln. Allerdings erzeugt ein Probelauf des Programms mit einem Array mit 5 Elementen die Ausgabe «`Kleinster Wert 0 bei Index 5`». Wie kommt es zu dieser Ausgabe? Markieren Sie die Fehler innerhalb des Quelltextes und berichtigen Sie die Fehler. Die Beseitigung aller Fehler führt zu der gewünschten Ausgabe «`Kleinster Wert 10 bei Index 0`». (6 / 20 Punkten)

Quelltext 4: Logische Fehler

```
1   #include <iostream>
2   using namespace std;
3   int main()
4   {
5     int arr[6] = { 10, 20, 30, 40, 50 };  // Array initilisieren
6     int valueLowest = 1000000;            // absurd hoher Startwert
7     int indexLowest = -1;                 // Startindex fuer Schleife
8     int i = 1;                            // Startwert-Schleifenvariable
9     for (; i < 5; i++);
10    {
11      if (arr[i] < valueLowest)
12      {
13        indexLowest = i;
14        valueLowest = arr[i];
15      }
16    }
17    cout << "Kleinster Wert " << valueLowest << " bei Index " << indexLowest;
18  }
```

Hinweis: Hier bestand in der Klausur ein Fehler. Die Verwendung von verschiedenen Bezeichnern für `indexLowest` war unbeabsichtigt. Die Aufgabe wurde von der Bewertung ausgeschlossenen, alle darin erreichten Punkte zählten als Bonus.

Aufgabe 3 (Programmentwurf) **10 Punkte**

In seinem Werk „Die Elemente“ beschreibt der Mathematiker **Euklid** einen Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen **a** und **b**. Das sogenannte Verfahren der *Wechselwegnahme* nutzt dabei den Umstand, dass bei der Subtraktion des kleineren Elements vom größeren Element, der größte gemeinsame Teiler unverändert bleibt. Führt man diese Subtraktion solange aus, bis **b** den Wert *null* annimmt, entspricht der Wert von **a** dem größten gemeinsamen Teiler der beiden Ausgangszahlen. Sofern die Ausgangswerte der natürlichen Zahlen **a** und/oder **b** bereits *null* sind, soll zudem der Wert des jeweils anderen Elementes zurückgegeben werden (bzw. *null* sofern beide Elemente den Wert *null* aufweisen).
Skizzieren Sie den Ablauf in einem Flussdiagramm.

Aufgabe 4 (Programmierung) **40 Punkte**

Ihnen ist aus der Vorlesung und der Übung das `MyAirline.de` Projekt bestens bekannt. Um die Stärken und Schwächen des Flugnetzes zu identifizieren, entscheidet das Unternehmen, den Verlauf der Simulation noch genauer aufzuzeichnen. Die Informationen über den Ausgang einer Buchungsanfrage werden dafür in der korrespondierenden Klasse `_BOOKINGREQUEST` hinterlegt. Die Klasse `_SCENARIO` soll zudem alle Buchungsanfragen (`_BOOKINGREQUEST`), die im Verlauf der Simulation zufällig generiert werden, abspeichern. Des Weiteren erhält die Klasse `_SCENARIO` eine neue Methode, die dazu dient, die abgespeicherten Buchungsanfragen gefiltert nach einem Kriterium auf der Konsole auszugeben. Das Kriterium stellt dabei der Ausgang der Buchungsanfrage dar. Die unterschiedlichen Ausgänge für eine Buchungsanfrage sind Tabelle 1 zu entnehmen.

Tabelle 1: Möglicher Status einer Buchungsanfrage `_BOOKINGREQUEST`

`int`	**Bedeutung**	`string`
0	unbearbeitete Buchungsanfrage	`unbearbeitet`
1	keine Verbindungen verfügbar	`nichtVerfuegbar`
2	sofortiger Ticketverkauf	`verkaufSofort`
3	Ticketverkauf nach Verhandlung	`verkaufVerhandlung`
4	Ticketverkauf an Preis gescheitert	`abgelehnt`

a) Begeben Sie sich in die Datei `bookingrequest.h`. Fügen Sie in die Definition der Struktur `_BOOKINGREQUEST` ein neues Ganzzahl-Datenelement `ausgang` hinzu. (1 / 40 Punkten)

b) Das Datenelement `ausgang` sollte nun innerhalb der beiden Konstruktoren der Struktur `_BOOKINGREQUEST` initialisiert werden. Da eine Buchungsanfrage in dem Moment ihrer Erstellung noch unbearbeitet ist, weisen Sie dem Datenelement `ausgang` den korrespondierenden Wert `0` zu. Vervollständigen Sie entsprechend **beide** Konstruktoren (In Datei: `bookingrequest.cpp`). (2 / 40 Punkten)

c) Für die Ausgabe einer Buchungsanfrage auf der Konsole, dient die bereits vorbereitete Methode `_BOOKINGREQUEST::print()`, jedoch fehlt noch die Implementierung (In Datei: `bookingrequest.cpp`). Gestalten Sie eine Ausgabe folgender Informationen:

   (a) Start- und Zielflughafen

   (b) Zeitpunkt der Erstellung

   (c) Ausgang der Buchungsanfrage

   Dabei soll anstatt des Zahlen-Werts von `ausgang` die in Tabelle 1 angegebene Zeichenkette auf der Konsole erscheinen. Legen Sie dazu eine `string` Variable `aux` an. Fragen Sie mit `if`-Bedingungen jeden möglichen Status von `ausgang` ab. Bei einer zutreffenden Bedingung belegen Sie `aux` mit der dazugehörigen Zeichenkette. (5 / 40 Punkten)
   Hinweis: Ein Beispiel für die `if`-Bedingung des Status `0` ist in Quelltext 5 dargestellt.

Quelltext 5: Beispiel Status-Abfrage

```
1        // aux muss bereits deklariert sein
2        if(ausgang == 0)
3            aux = "unbearbeitet";
```

d) Für das Speichern aller Buchungsanfragen auch über den Ablauf der Simulation hinaus, benötigt die Klasse `_SCENARIO` ebenfalls ein weiteres Datenelement. Da es sich voraussichtlich um mehr als eine Buchungsanfrage handelt, benötigen Sie ein Array, dessen genaue Länge jedoch nicht bekannt ist. Greifen Sie daher auf eine Zeigervariable zurück, welche die Speicheradresse des ersten Elementes des Arrays enthält. Fügen Sie das entsprechende Datenelement `requestArr` vom Typ Zeiger auf `_BOOKINGREQUEST` in die Klassendefinition von `_SCENARIO` ein (In Datei: `scenario.h`). (2 / 40 Punkten)

e) Aus Teilaufgabe d) folgt unmittelbar, dass der Konstruktor der `_SCENARIO` Klasse das neue Datenelement `requestArr` initialisieren muss (In Datei: `scenario.cpp`). Da vor der Simulation noch keine Buchungsanfragen vorliegen, erhält `requestArr` entsprechend den Wert `NULL`. (2 / 40 Punkten)

f) Für die Freigabe des durch die Klasse `_SCENARIO` (eventuell) belegten Speicherplatzes dient die bereits vorbereitete Methode `clear()`, allerdings fehlt noch die Implementierung (In Datei: `scenario.cpp`). Prüfen Sie zunächst in einer `if`-Bedingung, ob `requestArr` dem Wert `NULL` entspricht. In diesem Fall wurde noch gar kein Speicher reserviert, die Funktion ist dementsprechend mit einer `return`-Anweisung zu verlassen. Geben Sie in jedem anderen Fall mit `delete[] requestArr;` den Speicher wieder frei. Fügen Sie zudem einen Aufruf der Methode `clear()` innerhalb des Destruktors von `_SCENARIO` hinzu. (3 / 40 Punkten)

g) Begeben Sie sich in die `_SCENARIO::ExcuteSim()` Methode (In Datei: `scenario.cpp`). Fügen Sie als erste Anweisungen innerhalb der Methode folgenden Befehle hinzu:

   (a) einen Aufruf der Methode `clear();`

   (b) Speicherreservierung mit Hilfe der Anweisung:
   `requestArr = new _BOOKINGREQUEST[10000];`

   (1 / 40 Punkten)

h) Sofern es keine verfügbare `CONNECTION` für die Buchungsanfrage `BR` gibt, muss der Status von `BR` entsprechend aktualisiert werden. Finden Sie die beiden Stellen (NON-STOP & ONE-STOP) in der Methode `ExcecuteSim()` und setzen Sie das Datenelement `ausgang` auf 1 (In Datei: `scenario.cpp`). (4 / 40 Punkten)

i) Sofern es unmittelbar zu einem Ticketverkauf für die Buchungsanfrage kommt, muss der Status von `BR` entsprechend aktualisiert werden. Finden Sie die beiden Stellen (NON-STOP & ONE-STOP) in der Methode `ExcecuteSim()` und setzen Sie das Datenelement `ausgang` auf 2 (In Datei: `scenario.cpp`). (4 / 40 Punkten)

j) Sofern es nach Preisverhandlungen zu **einem** Ticketverkauf kommt, muss der Status von `BR` aktualisiert werden. Finden Sie die beiden Stellen (NON-STOP & ONE-STOP) in der Methode `ExecuteSim()` und setzen Sie das Datenelement `ausgang` auf 3 (In Datei: `scenario.cpp`). (4 / 40 Punkten)

k) Sofern es nach Preisverhandlungen zu **keinem** Ticketverkauf kommt, muss der Status von `BR` aktualisiert werden. Finden Sie die beiden Stellen (NON-STOP & ONE-STOP) in der Methode `ExecuteSim()` und setzen Sie das Datenelement `ausgang` auf 4 (In Datei: `scenario.cpp`). (4 / 40 Punkten)

l) Speichern Sie nun abschließend die aktualisierte Buchungsanfrage `BR` in `requestArr` am Index `requests` ab. Dies muss am **Ende** der `while`-Schleifen-Iteration der Methode `_SCENARIO::ExecuteSim()` erfolgen! (In Datei: `scenario.cpp`). (2 / 40 Punkten)
Hinweis: Die lokale Variable `request` gibt es bereits, sie zählt die Anzahl der generierten Buchungsanfragen. Daher kann diese Variable auch als aktueller Index im Array `requestArr` genutzen werden.

m) Zur Ausgabe der Daten auf der Konsole wurde bereits die Methode `print()` in der Klasse `_SCENARIO` angelegt, jedoch fehlt noch die Implementierung (In Datei: `scenario.cpp`). Prüfen Sie zunächst in einer `if`-Bedingung, ob `requestArr` dem Wert `NULL` entspricht. In diesem Fall wurde noch gar kein Speicher reserviert, die Funktion muss dementsprechend mit einer `return`-Anweisung verlassen werden. Geben Sie in jedem anderen Fall alle Buchungsanfrage in `requestArr` auf der Konsole aus, sofern das Datenelement `ausgang` der jeweiligen Buchungsanfrage denselben Wert wie das an die Methode übergebene Argument `status` aufweist. Sie benötigen dafür eine `if-Anweisung` innerhalb einer `for`-Schleife für alle `10000` Elemente in `requestArr`. Wenn die Bedingung erfüllt ist, rufen Sie auf dem Objekt der aktuellen Buchungsanfrage die Methode `print()` auf. (5 / 40 Punkten)

n) Rufen Sie innerhalb der `main`-Funktion auf dem `_SCENARIO` Objekt `scenario` die Methode `print()` mit einem Status Ihrer Wahl auf (In Datei: `main.cpp`). (1 / 40 Punkten)
Hinweis: Als Status übergeben Sie eine beliebe Ganzzahl zwischen 0 und 4.

Von Ihnen hochzuladende Dateien:

- Alle von Ihnen modifizierten *.h-Dateien
  a) `bookingrequest.h`
  b) `scenario.h`
- Alle von Ihnen modifizierten *.cpp-Dateien
  a) `bookingrequest.cpp`
  b) `scenario.cpp`
  c) `main.cpp`

Schreiben Sie an den Beginn jeder hochzuladenden Datei Ihren Namen und Ihre Matrikelnummer!