Karlsruher Institut für Technologie
Institut für Technik der Informationsverarbeitung

Aufgaben zum C++ Tutorium

Dipl.-Ing. Adnene Gharbi, Dipl.-Ing. Christoph Roth,
Dipl.-Ing.(FH) Tobias Schwalb, Dipl.-Ing. Michael Tansella
Dipl.-Inform. Timo Sandmann
cand. B.Sc. Andreas Kleff, cand. B.Sc. Felix Mauch

KIT - Universität des Landes Baden Württemberg und nationales
Forschungszentrum in der Helmholtz-Gemeinschaft

Vorwort

Diese Aufgaben sollen Ihnen helfen, Ihre erworbenen Kenntnisse aus Vorlesung und Übung während dem C++ Tutorium am Rechenzentrum praktisch umzusetzen.

Zu jeder Aufgabe ist angegeben, welche Vorkenntnisse Sie brauchen, um diese Aufgabe lösen zu können. Lesen Sie sich also die entsprechenden Abschnitte im Kompendium und in den Richtlinien durch, bevor Sie mit der Aufgabe beginnen.

Vorwissen:
Kompendium: Kapitel X.Y
Richtlinien: Kapitel Z

Zu Ihrer Hilfe sind während dem Tutorium auch Tutoren anwesend. Also zögern Sie nicht, diese um Hilfe zu bitten, wenn Ihnen das Kompendium oder weitere Literaturquellen nicht mehr weiterhelfen. Erwarten Sie allerdings keine Komplettlösungen der Tutoren.

Verbesserungsvorschläge, sowie jede Art von Kritik zu diesem Dokument, sind sehr erwünscht. Bitte senden Sie diese an *harald.bucher@kit.edu*.

4. Auflage, SS2014

Aufgabe 1 Wo liegt die Zahl?

Vorwissen:
Kompendium: Kapitel 3.1 und Kapitel 3.2
Richtlinien: Kapitel 2

Schreiben Sie ein Programm, das die Größe einer eingegebenen Zahl schätzt. Das Programm soll unterscheiden können, ob die Zahl kleiner als 0 ist, genau 0 ist, zwischen 0 und 100 liegt, genau 100 ist oder größer als 100 ist. Liegt die Zahl zwischen 0 und 100, soll der Benutzer gefragt werden, ob die Zahl noch genauer eingeordnet werden soll. Falls die Zahl noch weiter eingeordnet werden soll, soll unterschieden werden, ob die Zahl kleiner als 50 oder größer gleich 50 ist.

Beispielausgabe:

```
Bitte eine ganze Zahl eingeben: 34
Die Zahl liegt zwischen 0 und 100.
Weiter eingrenzen? [j/n] j
Die Zahl ist kleiner als 50
```

Aufgabe 2 Wo liegt die Zahl genau?

Vorwissen:
Kompendium: Kapitel 3.4
Richtlinien: Kapitel 2

Erweitern Sie Ihr Programm aus Aufgabe 1. Liegt die Zahl zwischen 0 und 100, soll sie genau bestimmt werden. Überlegen Sie sich hierzu vorher einen geeigneten Algorithmus zur Bestimmung der Zahl.

Beispielausgabe:

```
Bitte eine ganze Zahl eingeben: 77
Die Zahl liegt zwischen 0 und 100.
Weiter eingrenzen? [j/n] j
Die Zahl ist groesser als 50
Die Zahl ist groesser als 75
Die Zahl ist kleiner als 87
Die Zahl ist kleiner als 81
Die Zahl ist kleiner als 78
Die Zahl ist groesser als 76
Die eingegebene Zahl ist gleich 77
```

Aufgabe 3 Mathematikrechner

Vorwissen:
Kompendium: Kapitel 3.1, 3.2 und 3.4
Richtlinien: Kapitel 2.1 - 2.4

Schreiben Sie einen kleinen Mathematikrechner der nacheinander die erste Zahl, das Rechenzeichen und dann die zweite Zahl einliest. Dabei sollen auch (negative) Gleitpunktzahlen zugelassen sein.

Anschließend soll die eingegebene Aufgabe vom Programm berechnet und das Ergebnis auf dem Bildschirm ausgegeben werden. Zusätzlich soll das Programm in einer Schleife laufen, sodass der Benutzer mehrere Aufgaben hintereinander rechnen kann, ohne das Programm immer wieder neu starten zu müssen. Eine Möglichkeit zum Beenden soll allerdings trotzdem gegeben sein (z.B. mit einer zusätzlichen Abfrage am Ende jeder Rechenaufgabe).

Beispielausgabe:

```
Geben Sie bitte die erste Gleitpunktzahl ein: 2.34
Geben Sie das Rechenzeichen ein: +
Geben Sie bitte die zweite Gleitpunktzahl ein: 1.11
Ergebnis: 2.34 + 1.11 = 3.45

Moechten Sie das Programm nochmal ausfuehren (j/n)? n
```

Aufgabe 4 Reihendarstellung von versch. Funktionen

Vorwissen:
Kompendium: Kapitel 3.3
Richtlinien: Kapitel 2.5

Erstellen Sie ein Programm, das die Reihendarstellung der e-Funktion, die des Sinus, die des Kosinus, sowie die geometrische Reihe ausgeben kann.
Geben Sie dem Benutzer zunächst ein Menü vor, in dem er zwischen den vier Reihen wählen kann. Danach soll der Benutzer gefragt werden, wie viele Elemente ausgegeben werden sollen.
Falls der Benutzer eine ungültige Wahl trifft, sollen Glieder der geometrischen Reihe ausgegeben werden.
Benutzen Sie für das Menü Switch-Case-Anweisungen. Eine Beispielausgabe finden Sie bei Aufgabe 5.
Hinweis:

$$e^x = \sum_{n=0}^{\infty} \frac{1}{n!} x^n \qquad \sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \qquad \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \qquad \frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$$

Aufgabe 5 Reihendarstellung von Funktionen mit Funktionen

Vorwissen:
Kompendium: Kapitel 4.1 - 4.5
Richtlinien: Kapitel 3.1 - 3.4

Ändern Sie das Programm der letzten Aufgabe ab, sodass Sie für die Ausgabe von Sinus, Kosinus, e-Funktion und geometrischer Reihe jeweils eine Funktion schreiben, die sich um die Einzelheiten kümmert. Innerhalb des Switch-Case Blocks in Ihrem Hauptprogramm müssen Sie dann nun nur noch die entsprechende Funktion aufrufen.

1. Schreiben Sie Ihre Funktionen zunächst so, dass immer die ersten fünf Reihenglieder ausgegeben werden. Implementieren Sie diese Methoden und testen Sie Ihr Programm.

2. Ändern Sie Ihre Funktionen nun so, dass Sie beim Aufruf der Funktion festlegen können, wie viele Reihenglieder ausgegeben werden können.

Beispielausgabe:

```
Welche Funktion soll ausgegeben werden?
(1) e-Funktion
(2) Sinusfunktion
(3) Kosinusfunktion
(4) Geometrische Reihe
Funktion: 3
Wie viele Reihenglieder sollen ausgegeben werden? 3
Es werden die ersten 3 Reihenglieder der Kosinusfunktion ausgegeben.
cos(x) = 1 - (1/(2!))*x^2 + (1/(4!))*x^4
```

Aufgabe 6 Fakultät berechnen, Fehler abfangen

Vorwissen:
Kompendium: Kapitel 4.1 - 4.5.1
Richtlinien: Kapitel 3

Schreiben Sie eine Funktion, welche die Fakultät einer Ganzzahl mit Hilfe einer Schleife berechnet und zurück gibt. Achten Sie dabei auch darauf, dass die Fakultät von Null gleich Eins ist.
Schreiben Sie nun ein Programm, welches

- eine Ganzzahl von der Tastatur einliest
- wenn diese negativ ist eine Fehlermeldung ausgibt
- ansonsten die eben geschriebene Funktion zur Fakultätsberechnung aufruft
- und das Ergebnis der Berechnung auf dem Bildschirm ausgibt

Hinweis: Für alle natürlichen Zahlen n ist $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$
Zusatz: Die Fakultätsfunktion lässt sich auch sehr effizient als rekursive Funktion implementieren.
Beispielausgabe:

```
Bitte eine Ganzzahl eingeben: 5
5! = 120
```

Aufgabe 7 Quadrieren mit Zeigerübergabe

Vorwissen:
Kompendium: Kapitel 4.5.3
Richtlinien: Kapitel 3.5

Erstellen Sie eine Funktion, welche eine als Zeiger übergebene Variable quadriert. Schreiben Sie anschließend ein kleines Testprogramm, welches eine Zahl von der Tastatur einliest und diese anschließend quadriert. Rufen Sie hierzu die von Ihnen erstellte Quadradfunktion auf.
Beispielausgabe:

```
Bitte geben Sie eine Zahl ein: 4
4^2 ergibt 16
```

Aufgabe 8 Quadrieren mit Übergabe als Referenz

Vorwissen:
Kompendium: Kapitel 4.5.2
Richtlinien: Kapitel 3.5

Schreiben Sie nun ihre Funktion aus Aufgabe 7 so um, dass ihr eine Referenz als Parameter übergeben werden kann. Rufen Sie in Ihrem Hauptprogramm die neue Funktion auf.

Aufgabe 9 Multiplikation von komplexen Zahlen

Vorwissen:
Kompendium: Kapitel 2.11, Kapitel 4
Richtlinien: Kapitel 2, 3

In dieser Aufgabe soll eine Funktion zur Multiplikation zweier komplexer Zahlen implementiert werden. Stellen Sie hierzu eine komplexe Zahl mit einem Array aus zwei Integer-Zahlen dar.

$$z = 2 + j3 \Rightarrow arr[2] = \{2, 3\}$$

Schreiben Sie nun eine Funktion, welche drei komplexe Zahlen als Parameter akzeptiert. Die ersten beiden Zahlen sollen miteinander multipliziert werden und das Ergebnis in der dritten gespeichert werden.
Erstellen Sie anschließend ein kleines Testprogramm, das zwei komplexe Zahlen vom Benutzer abfragt und diese mit der eben von Ihnen erstellten Funktion miteinander multipliziert.
Beispielausgabe:

```
Bitte geben Sie den Realteil der ersten Zahl ein: -1
Bitte geben Sie den Imaginaerteil der ersten Zahl ein: 3
Bitte geben Sie den Realteil der zweiten Zahl ein: 2
Bitte geben Sie den Imaginaerteil der zweiten Zahl ein: -1
zahl1 * zahl2 = 1 + j7
```

Aufgabe 10 Multiplikation von 3×3-Matrizen

Vorwissen:
Kompendium: Kapitel 2.11, Kapitel 4
Richtlinien: Kapitel 2, 3

Nachdem Sie in Aufgabe 9 die Multiplikation komplexer Zahlen umgesetzt haben, erstellen Sie nun eine Funktion zum Multiplizieren von 3×3-Matrizen nach dem Falkschen Schema.
Die Multiplikationsfunktion soll wieder drei Argumente erhalten: die beiden eingegebenen Matrizen, sowie die leere Matrix für das Ergebnis.
Tipp: Schreiben Sie für die Ein- und Ausgabe ebenfalls eigene Funktionen, so müssen Sie diese nur einmal schreiben und können Sie für beide Matrizen ausführen.

Beispielausgabe:

```
Matrix A eingeben:
Element a11 eingeben: 1
Element a12 eingeben: -2
Element a13 eingeben: 0
Element a21 eingeben: 2
Element a22 eingeben: -3
Element a23 eingeben: 0
Element a31 eingeben: 2
Element a32 eingeben: 1
Element a33 eingeben: -2
Eingegebene Matrix:
1 -2 0
2 -3 0
2 1 -2

Matrix B eingeben:
Element b11 eingeben: 0
Element b12 eingeben: 1
Element b13 eingeben: -2
Element b21 eingeben: 3
Element b22 eingeben: 3
Element b23 eingeben: -3
Element b31 eingeben: 1
Element b32 eingeben: -2
Element b33 eingeben: 4
Eingegebene Matrix:
0 1 -2
3 3 -3
1 -2 4

A * B:
-6 -5 4
-9 -7 5
1 9 -15
```

Aufgabe 11 Eine kleine Autoklasse

Vorwissen:
Kompendium: Kapitel 5.1 - 5.3
Richtlinien: Kapitel 6

Schreiben Sie eine Klasse für ein Auto. Das Auto soll folgende privaten Attribute besitzen:

- Marke
- Baujahr
- Beschleunigung von 0 auf 100 km/h

- Höchstgeschwindigkeit

Erstellen Sie außerdem die folgenden Methoden:

- Get- & Set-Methoden für sämtliche Attribute
- `beschleunigen`: Erstellen Sie die Ausgabe wie in der Beispielausgabe zu sehen und geben Sie nach dem Beschleunigungsvorgang die erreichte Höchstgeschwindigkeit aus.
- `hupen`: Erzeugen Sie eine Ausgabe für das Hupen.

Schreiben Sie anschließend ein kleines Testprogramm für Ihre Autoklasse, welches ein Auto Objekt erzeugt, dieses beschleunigt und anschließend hupt.
Verwenden Sie für die Automarke einen eigenen Datentyp, der mindestens die folgenden Werte annehmen kann: *Mercedes, Audi, Volkswagen, BMW.*
Zusatz: Ihr Auto soll Bescheid sagen, wenn es erzeugt wird und wenn es zerstört wird.
Beispielausgabe:

```
Ein neues Auto wird erstellt!
Beschleunige bis zur Maximalgeschwindigkeit... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
183 km/h erreicht.
Tuutuuuuut!
Ein Auto wird zerstoert
```

Aufgabe 12 String-Behandlung

Vorwissen:
Kompendium: Kapitel 5.4
Richtlinien: Kapitel 6

Erzeugen Sie einen String mit dem Inhalt "*Ich studiere gerne an der Universitaet Karlsruhe.*".

1. Manipulieren Sie den String nun, sodass er nach der Manipulation lautet: "*Ich studiere Elektro- und Informationstechnik am Karlsruher Institut fuer Technologie. :)*"
2. Fügen Sie nun noch in den Satz ein, wie lange Sie bereits am KIT studieren.
3. Entfernen Sie jetzt wieder das Smiley am Ende des Satzes.

Beispielausgabe:

```
Ich studiere gerne an der Universitaet Karlsruhe.
Etwas Manipulation...
Ich studiere Elektro- und Informationstechnik am Karlsruher Institut fuer
Technologie. :)
Nochmal etwas Manipulation...
Ich studiere seit 2 Semestern Elektro- und Informationstechnik am Karlsruher Institut
fuer Technologie. :)
Das Smiley wirkt etwas unprofessionell...
Ich studiere seit 2 Semestern Elektro- und Informationstechnik am Karlsruher Institut
fuer Technologie.
```

Aufgabe 13 Ein paar Fahrzeuge

Vorwissen:
Kompendium: Kapitel 6, 7, 8
Richtlinien: Kapitel 6

Erzeugen Sie zunächst eine Fahrzeugsklasse nach dem Beispiel der Autoklasse in Aufgabe 9. Erweitern Sie die Klasse um die Methode `steckbrief()`, welche Marke, Baujahr, Beschleunigung, sowie Höchstgeschwindigkeit des Fahrzeugs ausgibt.
Leiten Sie nun von dieser Basisklasse die Klassen LKW, Auto und Motorrad ab.
Die abgeleiteten Klassen sollen sich in der Ausgabe der Methode `hupen()` unterscheiden:

Fahrzeug "Ich hupe!"

Auto "Tuutuut!"

LKW "Troeoeoeoet!"

Motorrad "Tuetuet!"

Außerdem soll im Steckbrief ausgegeben werden, um welchen Fahrzeugtyp es sich handelt.
Erstellen Sie nun ein Testprogramm, welches ein dynamisches Objekt der Klasse `Fahrzeug` erstellt. Bieten Sie dem Benutzer hierzu zuerst ein Menü, um den Fahrzeugtyp zu wählen. Geben Sie anschließend die Marke ebenfalls über ein Benutzermenü ein. Lesen Sie nun noch die restlichen Fahrzeugdaten ein.
Wenn der Benutzer einen ungültigen Fahrzeugtyp eingibt, soll ein Fahrzeug vom Typ Fahrzeug erstellt werden.
Wird eine ungültige Marke eingegeben, setzen Sie die Marke auf "unbekannt".

Geben Sie nun den Steckbrief des Fahrzeugs aus und hupen Sie ein mal.
Wenn ein Fahrzeug erstellt wird, soll eine entsprechende Meldung über die Erzeugung informieren und den Fahrzeugtyp nennen. Das selbe soll beim Zerstören eines Fahrzeugs geschehen.
*Hinweis: Verwenden Sie für jede Klasse getrennte Header- und *.cpp-Dateien.*
Beispielausgabe:

```
Was fuer ein Fahrzeug soll erstellt werden?
(1) Auto
(2) LKW
(3) Motorrad
Eingabe: 3
Ein neues Fahrzeug wird erstellt!
Ein neues Motorrad wird erstellt!
Marke eingeben:
(1) Mercedes
(2) Audi
(3) Volkswagen
(4) BMW
(5) Andere
Eingabe: 4
Baujahr eingeben: 1987
Hoechstgeschwindigkeit eingeben: 190
Beschleunigung von 0 auf 100 km/h in Sekunden eingeben: 5.4

Steckbrief:
Ich bin ein Motorrad der Marke BMW
Mein Baujahr ist 1987
Ich komme in 5.4 Sekunden von 0 auf 100 km/h
Meine Hoechstgeschwindigkeit ist 190 km/h

Beschleunige bis zur Maximalgeschwindigkeit... ... ... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ... ... ...
190 km/h erreicht.
Tuetuet!

Ein Motorrad wird zerstoert!
Ein Fahrzeug wird zerstoert!
```

Aufgabe 14 Garage mit fstream einlesen

Vorwissen:
Kompendium: Kapitel 8, Kapitel 9, Anhang B.1.1
Richtlinien: Kapitel 5, Kapitel 6

Jetzt wollen wir das Programm aus Aufgabe 11 erweitern. Erstellen Sie eine Klasse namens Garage, welche in einem Vektor als Attribut Fahrzeuge aufnehmen kann. Erstellen Sie außerdem eine Methode, welche eine Textdatei *garage.txt* einlesen kann und daraus die Informationen der geparkten Fahrzeuge sammelt. Ein Eintrag in der Textdatei ist nach dem folgenden Schema aufgebaut:

```
[Fahrzeugtyp]
Marke: automarke
Baujahr: jahreszahl
Geschwindigkeit: maxGeschwindigkeit
Beschleunigung: beschleunigung
```

Kommentarzeilen in der Textdatei beginnen mit einem #-Zeichen.
Erstellen Sie für jeden Eintrag in der Textdatei ein Objekt in Ihrem Fahrzeug-Vektor.
Schreiben Sie nun eine Methode der Klasse Garage, welche den Inhalt der Garage, also des Fahrzeugsvektors ausgibt. Hupen Sie nach jedem ausgegebenen Fahrzeug einmal mit demselben.
Überlegen Sie sich, wie ein Testprogramm für Ihre Garagenklasse aussehen müsste und erstellen Sie dieses.

Aufgabe 15 Lotto-Zahlen Generator

Vorwissen:
Kompendium: Kapitel 8
Richtlinien: Kapitel 6

I n dieser Aufgabe soll ein Zufallszahlengenerator geschrieben werden, welcher im Beispiel dazu verwendet werden kann, um die nächsten Lotto-Zahlen zu generieren. Beim Lotto werden 6 aus 49 Zahlen gezogen, dabei kann keine Zahl doppelt vorkommen (es kann auch keine Null gezogen werden). Das Programm soll dazu in zwei Klassen aufgeteilt werden. Die erste Klasse ist verantwortlich für die Generierung einer Zufallszahl. Die andere Klasse verwendet diese Klasse und generiert die Zahlen für das Lotto.
Schreiben Sie nun im ersten Teil eine Klasse `Random`, welche die privaten Attribute `min` und `max` hat und eine entsprechende öffentliche Methode `init` um diese zu initialisieren. Weiterhin soll eine Methode `initRandom()` in der Klasse enthalten sein. Diese soll den Zufallszahlengenerator mit Hilfe der aktuellen Zeit initialisieren. Dies kann mit dem folgenden Befehl erfolgen:

- `srand(( unsigned int ) time (NULL));`

Auch soll die Klasse eine öffentliche Methode `random()` haben, welche eine Zufallszahl im vorher festgelegten Bereich zurückgibt. *Hinweis: Der folgende Befehl gibt eine ganzzahlige Zufallszahl zwischen 0 und der Konstanten RAND_MAX zurück:*

- `rand();`

Die Klasse `Lotto` soll ein Objekt der Klasse `Random` beinhalten und diese in der öffentlichen Methoden `generateLotto()` dazu verwenden um den Zufallszahlengenerator zu initialisieren und dann sechs entsprechende Zufallszahlen auszugeben.

Aufgabe 16 Paketdienst

Vorwissen:
Kompendium: Kapitel 7,8
Richtlinien: Kapitel 4,6

E s soll eine Datenbank eines Paketdienstes zur Sendungsverfolgung von Paketen objektorientiert programmiert werden. Dabei sollen alle Adressen, welche ein Paket von der Aufgabe bis zur Zustellung durchläuft dokumentiert werden.
Hierzu sollen drei Klassen entstehen. Die erste Klasse speichert eine Adresse mit Namenszeile, Straße, PLZ, Ort und Land in entsprechenden privaten Attributen. Auch soll die Klasse eine Methode enthalten um die Daten vom Benutzer (von der Tastatur) abzufragen und eine Methode um alle Daten auszugeben.
Die zweite Klasse enthält alle Daten zu einem Paket, wie zum Beispiel Sendungsnummer, Größe und Gewicht. Auch soll in dieser Klasse unter Verwendung der ersten Klasse die Absenderadresse und Zieladresse gespeichert werden. Die Zwischenadressen sollen ebenfalls unter Verwendung der ersten Klasse in einem entsprechenden dynamischen Container in der Klasse zum Paket gespeichert werden.
Die letzte Klasse beschreibt den Paketdienst. Die Klasse kann dabei eine beliebige Anzahl an Paketen speichern. Auch bietet die Klasse Methoden zum Hinzufügen eines Paketes und zum Suchen anhand der Sendungsnummer. Nachdem ein Paket über die Suche gefunden wurde, kann ebenfalls auf Wunsch des Users eine Station zu den Zwischenadressen hinzugefügt werden.

Aufgabe 17 Paketdienst 2

Vorwissen:
Kompendium: Kapitel 6,7,8
Richtlinien: Kapitel 4,6

D as Programm zum Paketdienst aus der vorherigen Aufgabe soll so erweitert werden, dass auch Paket-Eintragungen gelöscht werden können. Hierzu soll von der dritten Klasse Paketdienst eine neue Klasse abgeleitet werden, die eine zusätzliche Funktion zum Löschen eines Pakets nach Eingabe der Trackingnummer enthält. Dabei soll doppelter Code vermieden werden, indem entsprechender Code in eine neue (nicht öffentliche) Methode ausgelagert wird.