Ludwig-Maximilians-Universität München
LFE Medieninformatik
Prof. Dr. Andreas Butz
Dipl.-Medieninf. Sebastian Boring
Dipl. Inf. Arnd Vitzthum

Wintersemester 2006/2007
Übungsblatt 1
23. Oktober 2006

Übung "Augmented Reality"

Abgabetermin:
Die Lösung zu diesem Übungsblatt ist bis zum 30. Oktober 2006 abzugeben.

Inhalt:
Dieses Übungsblatt dient zum Kennenlernen der Programmiersprache *C++* mit ihren Unterschieden zu Java. Die auf diesem Blatt geforderten Programme sollen als Übung dienen, damit Sie den Umgang mit *C++* erlernen können. Außerdem erlernen Sie die Verwendung von Funktionen, Header-Dateien und Klassen. Ferner werden das Schlüsselwort `virtual` sowie Mehrfachvererbung näher betrachtet. Achten Sie auch in diesem Blatt darauf, dass Ihr Code lesbar und gut dokumentiert ist.

Aufgabe 0 (H) Compiler und Makefiles

Zu Beginn sollen Sie sich mit dem verwendeten Compiler `gcc` vertraut machen. Einen Überblick der verschiedenen Optionen des Compilers können Sie sich mit `man gcc` auf der Kommandozeile ausgeben lassen. Die Eingabe von Hand ist bei vielen Quelldateien sehr umfangreich und damit langwierig. Hierfür gibt es jedoch die sogenannten *Makefiles*, die durch einen Aufruf von `make` auf der Kommandozeile die gewünschten Operationen durchführen.

a) Machen Sie sich mit der Funktionsweise des Compilers und des Linkers vertraut.

b) Verschaffen Sie sich einen Überblick über *Makefiles*. Als ersten Einstieg genügt hier folgende Webseite: `http://www.ijon.de/comp/tutorials/makefile.html`

c) Laden Sie sich das auf der „Ubungsseite bereitgestellte *Makefile* herunter.

Aufgabe 1 (P) Ein erstes C++ Programm

In dieser Aufgabe schreiben Sie ein erstes *C++* Programm. Hierbei wird zunächst der Fokus auf die Ausgabe (sowohl unformatiert als auch formatiert) gelegt. Abschließend wird vom Benutzer ein Tastendruck für das Beenden des Programms verlangt.

a) Erstellen Sie eine *C++* Datei namens *aufgabe_1.cpp*, die zunächst nur die Methode `main` enthält. Fügen Sie ferner die benötigten `include` Anweisungn in Ihren Code ein, die es Ihnen erlauben, die Standardein- und Standardausgabe sowie die Konsolendeklarationen zu verwenden.

b) Verwenden Sie nun die Standardausgabe (also `std::cout`) um `Hello World!` oder einen beliebigen anderen String auszugeben. Erweitern Sie dies durch die Ausgabe einer ganzen Zahl und eines Gleitkommawerts. Trennen Sie dabei jede Zeile mit `std::endl`.

c) Die Ausgaben mit `std::cout` sind nicht formatiert und folgen den Default-Einstellungen, d.h. dass z.B. bei einer Gleitkommazahl genau sechs Nachkommastellen angezeigt werden. Dies führt nicht immer zum gewünschten Ergebnis. Für die formatierte Ausgabe wird die Funktion `printf` verwendet. Geben Sie nun folgendes in jeweils einer neuen Zeile aus:

- Ausgabe von `'a'` und `65` jeweils als Zeichen (d.h. als `character`)
- Ausgabe zweier ganzen Zahlen (d.h. als `int`)
- Ausgabe einer Zahl mit `10` führenden Leerzeichen
- Ausgabe einer Zahl mit `10` führenden Nullen
- Ausgabe einer Zahl in den Zahlensystemen `Dezimal`, `Hexadezimal` und `Octal`. Für die letzten beiden Zahlensysteme geben Sie dies bitte mit und ohne führendes Zeichen (d.h. `0` bzw. `0x`) an
- Ausgabe einer Gleitkommazahl mit vier Vorkommastellen sowie zwei Nachkommastellen. Zusätzlich soll die Zahl hier auch in der e-Notation dargestellt werden
- Ausgabe einer Zahl mit einer definierten Anzahl (als Parameter übergeben) von Leerzeichen direkt vor dieser
- Ausgabe eines beliebigen Strings

Hinweis: Eine Referenz zur Funktion `printf` und formatierter Ausgabe finden Sie unter: `http://www.cplusplus.com/ref/cstdio/printf.html`

d) Nachdem alle Anfragen an den Benutzer durchgeführt wurden, soll eine Meldung angezeigt werden, die dem Benutzer mitteilt, dass er *die Enter-Taste* drücken soll, um das Programm zu beenden. Dieser Schritt soll nun (sofern nicht ausdrücklich anders verlangt) in allen Programmen am Schluss verwendet werden.

Hinweis: Verwenden Sie die Funktion `getchar()`, um die Ausführung des Programms solange zu unterbrechen, bis eine Taste gedrückt wird.

Aufgabe 2 (P) Benutzereingaben

In dieser Aufgabe lernen Sie die Standardeingabe kennen. Aus den vom Benutzer eingegebenen Werten sollen bestimmte Ausgaben generiert werden. Dazu wird eine neue Datei erstellt (*aufgabe_2.cpp*), die alle nachfolgend genannten Punkte erfüllen muss.

a) Verwenden Sie zunächst die `using`-Direktive, um die Verwendung von der Ein- und Ausgabestreams zu vereinfachen. Verwenden Sie hierzu den Namensraum in dem diese Streams deklariert sind.

b) Fordern Sie den Benutzer nun auf eine ganze Zahl zwischen 1000 und 9999 einzugeben. Wiederholen Sie diese Abfrage solange bis der Benutzer eine vierstellige Zahl angegeben hat.

c) Unter den vierstelligen Zahlen gibt es einige Zahlen XXYY, die folgendes erfüllen:

$$XX^2 + YY^2 = XXYY;$$

Erweitern Sie Ihr Programm um die Berechnung, ob die vom Benutzer eingegebene Zahl eine solche besondere Zahl ist. Ist dies der Fall geben Sie die korrekte Gleichung aus, z.B. $12^2 + 33^2 == 1233$. Andernfalls soll dem Benutzer die Ungleichheit angezeigt werden, z.B. $12^2 + 34^2 ! = 1234$.

Aufgabe 3 (P) Schleifen

In dieser Aufgabe sollen Sie die drei verschiedenen Schleifen-Typen kennenlernen. Dies werden später wichtige Elemente sein, die Sie für OpenGL benötigen werden. Sie sind jedoch analog zu Java verwendbar. Erstellen Sie hierfür eine neue Datei namens *aufgabe_3.cpp*, die die nachfolgend genannten Schlafen enthalten soll.

a) Erstellen Sie eine `for`-Schleife, die anhand der ASCII-Codes die Zahlen 0 bis 9 ausgibt, d.h. diese Schleife soll vom ASCII-Wert für '0' bis zum ASCII-Wert von '9' laufen. Für die Umwandlung eines Parameters `x`, der als `int` vorliegt, in `char` können Sie z.B. den Ausdruck `static_cast<char>(x)` verwenden.

b) Fügen Sie eine `while`-Schleife hinzu, die alle Großbuchstaben nach dem Schema der vorhergehenden Aufgabe ausgibt.

c) Fügen Sie eine `do-while`-Schleife hinzu, die alle kleinen Buchstaben nach dem Schema der vorhergehenden Aufgaben ausgibt.

Aufgabe 4 (P) Arrays und Pointer

In dieser Aufgabe lernen Sie grundlegende Arrays kennen. Außerdem wird das Konzept der Pointer betrachtet, die unter anderem als Iterator auf einem solchen Array dienen können. Erstellen Sie eine Datei namens *aufgabe_4.cpp*, die die Teilaufgabe *b)* umsetzen wird.

a) Betrachten Sie zunächst folgende Beispielsequenz:

```
int arr[] = { 0, 10, 20, 30, 40, 50 };
int val, *p;
p = arr;
val = *p;
val = *p++;
val = *++p;
val = (*p)++;
val = *(p++);
val = ++*p;
val = ++(*p);
```

Welchen Wert nimmt `val` nach Ausführung dieser Sequenz an und welche Gestalt hat `arr`? Geben Sie zusätzlich die Werte von `val`, `p` und `arr` für jede Zeile an und begründen Sie das Verhalten.

b) Bei einem Aufruf von `main` werden zwei Parameter übergeben. Der erste - `argc` - gibt an wieviele Argumente beim Programmstart mitgeliefert wurden, der zweite - `argv` - beinhaltet diese. Beachten Sie, dass (im Gegensatz zu Java) der Programmaufruf selbst auch ein Argument ist, d.h. `argc` hat mindestens den Wert 1 und `argv` ist nicht leer.

Geben Sie nun die Anzahl der übergebenen Argumente aus. Danach sollen alle Argumente ausgegeben werden, wobei für ein Argument zusätzlich jedes einzelne Zeichen zeilenweise dargestellt werden soll. Für die Implementierung der Einzelzeichenausgabe sollen Pointer verwendet werden.

Beispiel: Der Aufruf `aufgabe_4 argument` liefert folgendes Ergebnis:

```
Anzahl uebergebener Argumente: 2
Uebergebene Argumente:
        Argument#0: aufgabe_4
                Teil #1: a
                Teil #2: u
                Teil #3: f
                ...
                Teil #9: 4
        Argument#1: argument
                Teil #1: a
                Teil #2: r
                ...
                Teil #8: t
```

Aufgabe 5 (P) Funktionen und Funktionsaufrufe

In dieser Aufgabe soll die Erstellung von Funktionen sowie deren Aufruf während des Programms erlernt werden. Als Beispiel soll die Potenzfunktion implementiert werden.

a) Erstellen Sie eine Datei namens *aufgabe_5.cpp*, die die Methode `int main(int argc, char** argv)` sowie die Methode `int potenz(int x, int y)` enthält. Verlangen Sie vom Benutzer zwei Eingaben. Die erste Eingabe ist die Basis, die zweite die `positive, ganzzahlige` Potenz. Lassen Sie die Eingabe solange wiederholen, bis die Potenz positiv (d.h. größer oder gleich 0) ist. Verwenden Sie zur Berechnung eine Rekursion.

b) Drehen Sie nun die Definition der beiden Funktionen um, d.h. `main` soll vor `potenz` in der Datei stehen. Was geschieht beim kompilieren? Wie kann dies verhindert werden?

c) Erstellen Sie nun zwei Dateien namens `potenz.h` und `potenz.cpp`. Fügen Sie einen Funktionsprototypen in die Header-Datei mit der Signatur `int potenzNichtRekursiv(int, int)` ein, und implementieren Sie diese Funktion in der *C++*-Datei. Fügen Sie ferner das Statement `#include "potenz.h"` im Hauptprogramm und in der *C++*-Datei ein, und lassen Sie die Potenz nun auf beide Arten berechnen.

d) Die Ergebnisse sollen nun folgendermaßen angezeigt werden (bspw. bei *3 hoch 4*):
`3 ^ 4 = 81`.

Aufgabe 6 (P) Klassen, Mehrfachvererbung und Ausnahmen

In dieser Aufgabe erlernen Sie den Umgang mit Klassen, Header-Dateien sowie Vererbung und Mehrfachvererbung. Dazu dient ein Beispiel aus der Zoologie. Es sollen Tiere mit ihren unterschiedlichen Klassifikationen modelliert werden.

a) Betrachten Sie die Tiere *Löwe*, *Haifisch* und *Krokodil*. Alle drei haben sowohl gemeinsame als auch unterschiedliche Eigenschaften. Teilen Sie die Tiere in *Landtiere* und *Wassertiere* ein. Bedenken Sie, dass in *C++* auch mehrfache Vererbung möglich ist und berücksichtigen

Sie dies in ihren Überlegungen. Erstellen Sie nun ein UML-Diagramm, das die Zusammenhänge und Unterschiede herausstellt. Die genauen Eigenschaften (wie z.B. *Geschwindigkeit* oder *Alter* sollen zunächst nicht weiter betrachtet werden.

b) Erweitern Sie das UML-Diagramm an geeigneten Stellen um folgende Eigenschaften der Tiere:

- Tiere besitzen ein Alter (Ganzzahlwert)
- *Landtiere* und *Wassertiere* können sich fortbewegen
- Tiere haben eine gewisse Entfernung zurückgelegt (diese kann im Wasser anders sein als am Land)
- Ein *Haifisch* kann eine bestimmte Distanz schwimmen
- Ein *Krokodil* kann eine bestimmte Distanz schwimmen oder laufen
- Ein *Löwe* kann eine bestimmte Distanz laufen

c) Erstellen Sie nun Header-Dateien für jede Klasse, die in Ihrem UML-Diagramm erscheint. Das Diagramm sollte die drei Tiere, die zwei verschiedenen Aufenthaltsklassifikationen (Land oder Wasser) sowie eine allgemeine Klasse für die gemeinsamen Eigenschaften enthalten. Geben Sie Ihren Klassen ferner die Möglichkeit sie darstellen zu lassen. Erstellen Sie zudem alle Klassen im Namensraum `Tiere`.

Beispiel: Mögliche Ausgabe für ein Krokodil:
`Das Krokodil (4 Jahre alt) schwamm 12 sm und lief 10 km.`

d) Implementieren Sie nun diese Klassen. Achten Sie dabei auf eventuell zu überschreibende Funktionen und verwenden Sie das Schlüsselwort `virtual` um den Aufruf der falschen Funktion (also eine der Superklasse) zu verhindern.

e) Erweitern Sie die Klasse *Tiere* um die Funktion `void setAge(int)`, das Alter vorzugeben. Achten Sie jedoch darauf, dass das Alter nie negativ sein darf. Wird dennoch ein negatives Alter übergeben soll eine selbstdefinierte Ausnahme (z.B. `FalschesAlter`) erzeugt werden, die den Fehlertext enthält. Ein Beispiel für einen solchen Fehlertext wäre ”Alter darf nicht negativ sein!”.

f) Schreiben Sie schließlich die Methode `main` in der Sie nacheinander folgendes durchführen sollen:

- Als erstes Statement erzeugen Sie sich einen Zeiger auf ein Tier, z.B. `Tiere::Tier* ein_tier;`. Diesem Pointer werden Sie die drei nachfolgend erzeugten Tiere zuweisen.
- Erzeugen Sie sich einen *Haifisch* der 5 Jahre alt ist, bewegen Sie diesen um 5 Seemeilen und lassen Sie sich das Tier ausgeben
- Erzeugen Sie einen *Löwen* der 3 Jahre alt ist, bewegen Sie diesen erst um 2, dann nochmal um 5 Kilometer und lassen Sie sich das Tier ausgeben
- Erzeugen Sie ein *Krokodil* das 7 Jahre alt ist. Lassen Sie es zunächst 2 Seemeilen schwimmen, dann 5 Kilometer laufen und schließlich wieder 10 Seemeilen schwimmen. Lassen Sie sich das Tier anschließend ausgeben

g) Setzen Sie das Alter des Krokodils auf einen negativen Wert. Ändern Sie dafür außerdem die Methode `main` so ab, dass dieser inkorrekte Fall mittels eines `try-catch`-Blocks abgefangen wird. Geben Sie außerdem die Fehlermeldung (enthalten in der Ausnahme) aus.