C++ Hinweise zum Übungsblatt 02

Für diese Veranstaltung solltest Du, sofern Du bislang keine C++ Erfahrung hattest, wenigstens das *C++ Propädeutikum* besucht haben, welches i.d.R. jedes Jahr Ende September vor Beginn des Wintersemesters an der Universität Bremen stattfindet. An einer Universität obliegt es Dir, dass Du an entsprechenden Vorkursen teilnimmst oder Dir für ein Modul vorausgesetztes Wissen und Fertigkeiten selbstständig aneignest.

Dennoch findest Du nachfolgend eine kurze Übersicht, was bei der Arbeit mit C++ für dieses Übungsblatt wichtig und anders im Vergleich zu Java (oder Processing) ist. Die Erklärungen geben die Unterschiede zwischen Java und C++ nicht vollständig wieder und sollen Dir nur einen groben Überblick und die nötigen Keywords geben, um nach tiefergehenden Informationen suchen zu können.

Header- und Source-Files

In C++ teilt man Klassen aus verschiedenen Gründen häufig in Header- und Source-File auf, wobei die Header-Datei (`.h`) lediglich die Deklaration der Klasse und der Funktionen/Methoden enthält und die Source-Datei (`.cpp`) lediglich die Implementierung der vorher deklarierten Funktionen.

Im Framework nutzen wir dies auch: Die `Vec4f`-Klasse ist in `Vec4f.h` und `Vec4f.cpp` aufgeteilt und die `Mat4f`-Klasse ist in `Mat4f.h`.

Erstellen von Objekten

In Java kann man Objekte lediglich mit dem Schlüsselwort `new` anlegen, die im Speicher auf dem *Heap* abgelegt werden, z.B. via:

```
Object myObj = new Object(); // Java Style
```

Diese Objekte werden in Java automatisch mithilfe des Garbage Collectors aufgeräumt und der Speicher wieder freigegeben, sobald sie nicht mehr benutzt werden.

In C++ kannst Du ein Objekt ebenfalls auf dem Heap anlegen, also schreiben:

```
Vec4f* myVec = new Vec4f(); // C++ Style
```

Da in C++ beim Schlüsselwort `new` immer ein Pointer zurückgegeben wird, musst man ein Sternchen-Symbol hinter `Vec4f` schreiben, also `Vec4f*`. So gibt man in C++ an, dass nur ein Pointer auf das Objekt und nicht das Objekt selbst in die Variable gespeichert werden soll. C++ selbst erkennt allerdings nicht automatisch, dass Objekte nicht mehr benutzt werden und löscht diese nicht selbstständig - das musst Du mittels `delete`-Schlüsselwort machen.

Im Vergleich zu Java ist es allerdings auch möglich, Objekte direkt auf dem *Stack* zu erzeugen, auf dem einzelne Funktionsaufrufe während der Programmausführung gespeichert werden. Dazu kannst Du in einer Funktion einfach schreiben:

```
Vec4f vectorA = Vec4f();
Vec4f vectorB = Vec4f(1.0f, 0.0f, 0.0f);
```

Da diese Objekte auf dem *Stack* gespeichert werden, werden diese selbstständig gelöscht, sobald die Funktion vollständig ausgeführt wurde, also der Funktionsaufruf zuende ist.

C++ bietet dafür auch eine Kurzform:

```
Vec4f vectorA;
Vec4f vectorB(1.0f, 0.0f, 0.0f);
```

Damit wir uns nicht um die Speicherverwaltung kümmern müssen, werden wir in den Übungsaufgaben Objekte – wann immer möglich und sinnvoll – auf dem Stack speichern, also **kein** `new` verwenden.

Für die Interessierten: C++ bietet natürlich auch Möglichkeiten, Objekte einfach auf dem Heap anzulegen, ohne dass man sich um das Freigeben des Speichers kümmern muss, siehe Smart Pointer (z.B: `std::shared_ptr`), aber dazu später bei Bedarf mehr.

Operatoren überladen

In Java kann man nur mit primitiven Datentypen wie `float`, `int`, usw. mithilfe von Operatoren wie + oder − rechnen, z.B.:

```
float a;
float b;
float c = a + b;
```

Wenn Du mit Objekten selbst erstellter Klassen auf diese Art rechnen möchtest, klappt das in Java nicht.

In C++ dagegen ist es möglich, eine Vielzahl von Operatoren zu "überladen" (z.B. +,−,⋆,/, aber auch == oder !=) – d.h. Du entscheidest selbst, was diese Operatoren zurückgeben, indem Du eine Funktion dafür implementierst. In der Matrix-Klasse `Mat4f` ist zum Beispiel die nachfolgende Funktion für den ⋆ Operator deklariert:

```
Vec4f operator*(const Vec4f vector) const {
   // do and return something:
   [...]
}
```

Nachdem Du die zugehörige Implementierung fertig gestellt hast, kannst Du mit Variablen von Objekten einfach rechnen:

```
Mat4f m;
Vec4f v;
Vec4f result = m * v;
```

Der Ausdruck `m * v` gibt schließlich den Wert zurück, den die selbst definierte Funktion `operator*` zurückgibt.

Namespaces

In C++ wirst Du häufiger etwas wie `std::vector` lesen – in diesem Falle ist `std` ein Namespace, in welchem die Klasse `vector` implementiert ist. Namespaces sind ein wenig mit Packages in Java vergleichbar, aber Da Du sie im Rahmen der Aufgabe nicht wirklich benutzen musst, verzichten wir hier auf den Vergleich zu Java-Packages.

Weitere Infos gibt es z.B. unter https://de.wikipedia.org/wiki/Namensraum

Die Klasse std::vector

Zugegeben: Der Name `std::vector` ist im Kontext der Computergrafik etwas irreführend, da es sich dabei nicht um einen Vektor im algebraischen Sinne handelt. Die Klasse `std::vector` ist vergleichbar mit der ArrayList in Java - es ist also eine Liste, die beliebig lang werden kann und in welcher Objekte gespeichert werden können.

Bei der Erstellung eines `std::vector` wird wie bei der Erstellung einer ArrayList auch der Typ der Objekte in spitzen Klammern angegeben:

```
std::vector<Vec4f> myListOfVectors;
```

Die Methode `push_back` ist vergleichbar mit der Methode `add` der ArrayList, so kann man wie nachfolgend gezeigt Objekte hinzufügen über:

```
myListOfVectors.push_back(Vec4f());
```

Da man in C++ auch den `[]`-Operator überladen kann (was in der `std::vector`-Klasse gemacht wurde), kann man so wie bei Arrays auf die Elemente zugreifen, z.B.:

```
// Speichert den X-Wert unseres oben zur Liste hinzugefuegten algebraischen Vektors Vec4f
    in eine Variable:
float x = myListOfVectors[0].x;
```

Bei Interesse findest Du Weiteres in der C++ Dokumentation: https://en.cppreference.com/w/cpp/container/vector.

Mathe-Funktionen

Um Funktionen wie die Quadratwurzel oder die Exponentialfunktion nutzen zu können, verwenden wir `cmath` – diese wird bereits im vorgegebenen Programmcode an entsprechenden Stellen mit eingebunden, wodurch Du u.a. folgende Funktionen verwenden kannst:

- die Quadratwurzel `sqrt(float value)`
- die Exponentialfunktion `pow(float base, float exp)`
- der Betrag `abs(float value)`

Siehe Dir bei Bedarf die C++ Dokumentation an: https://en.cppreference.com/w/cpp/header/cmath.