Programmiersprachen
Sommersemester 2019

Aufgabensammlung 2

Die Lösungen der Aufgaben werden in den Übungen **am 6. Mai** bewertet. Schwerpunkte dieser Aufgabensammlung sind Klassendesign, das Verständnis von Übergabe- und Rückgabemechanismen, das Einhalten von const-correctness und die testgetriebene Softwareentwicklung. Legen Sie für jeden Datentypen eine *hpp* und eine *cpp* im Ordner `source` an und erweitern Sie entsprechend die Datei `source/CMakeLists.txt`.

Achten Sie immer auf const correctness bei der Parameterübergabe, der Rückgabe und bei Methodendeklarationen! Entwickeln Sie alle Klassen und Funktionen mit TDD. Ihre Tests schreiben Sie in die Datei `source/tests.cpp`. Testen Sie auch immer Randfälle!

Achten Sie inbesondere darauf, dass Sie anhand des Style-Guides https://moodle.uni-weimar.de/pluginfile.php/154525/course/section/102508/styleguide_2019.pdf programmieren. Nichteinhaltung des Style-Guides resultiert in Punktabzügen. Nutzen Sie neben dem Vorlesungsskript ausschließlich aktuelle Fachliteratur oder Online-Referenzen, z.B.

- http://en.cppreference.com/

Bei Fragen und Anmerkungen schreiben Sie bitte eine Email an adrian.kreskowski@uni-weimar.de .

Aufgabe 2.1

Auf https://github.com/vrsys/programmiersprachen-aufgabenblatt-2 finden Sie ein Framework, welches es Ihnen ermöglicht, ein Fenster zu öffnen und darin Punkte, Liniensegmente und Text in beliebiger Farbe zu zeichnen sowie die aktuelle Position des Mauszeigers abzufragen. *Forken* Sie das Repository auf `github`.
Unter Ubuntu und ähnlichen Linux-Systemen benötigen Sie:

- cmake
- xorg-dev
- libglu1-mesa-dev
- freeglut3-dev

- libxi-dev
- libxrandr-dev

Unter Windows verwenden Sie *cmake* und eine IDE wie *Visual Studio Community Edition 2019* inklusive der C++-Packages. Das Framework wird mit `C++17` kompiliert. Sprechen Sie uns bitte in der Übung an, falls Sie beim Bauen des Frameworks Probleme haben.
Konfigurieren Sie das Projekt mit `cmake` und bauen Sie es anschließend. Dies kompiliert die Beispielprogramme `example` und `tests`.

Aufgabe 2.2

Erstellen Sie im Ordner `source/` die Dateien `vec2.hpp` und `vec2.cpp`. Die Datei `vec2.cpp` benötigen wir ab Aufgabe 2.3.

Erklären Sie den Zweck der sogenannten *Include Guards* im teilweise vorgegebenen Header `vec2.hpp`. Was passiert, wenn Sie keine Include Guards verwenden und die Date vec2.hpp versuchen zweimal zu inkludieren? Warum ist das so?

Definieren Sie einen Datentyp `Vec2` als `struct`. Dieser Datentyp soll einen Vektor im $\mathbb{R}^2$ repräsentieren. Wir wollen den Datentypen maßgeblich als Datentransfer-Objekt nutzen und werden daher später nur einige Methoden definieren, die uns erlauben Standard-Operationen mit den Vektoren durchzuführen.

```
#ifndef VEC2_HPP
#define VEC2_HPP

// Vec2 data type definition
struct Vec2
{
  /* TODO add member variables with
     default member initialisation */
};

#endif // VEC2_HPP
```

Erweitern Sie die soweit vorgegebene `struct`-Definition, sodass zwei floating point Koordinaten namens `x` und `y` gespeichert werden können. Sorgen Sie mittels default-Memberinitialisierung dafür, dass eine Instanz des Datentyps `Vec2`, welche nicht explizit initialisiert wurde, ihre Membervariablen mit dem Wert 0.0f initialisiert.

Testen Sie die korrekte Initialisierung von Instanzen des Datentyps `Vec2`, jeweils ohne und mit Aggregatinitialisierung mittels Catch.

```
Vec2 a; //requires that 0.0f == a.x and 0.0f == a.y
Vec2 b{5.1f, -9.3f}; /*requires that 5.1f == Approx(b.x)
                                and -9.3f == Approx(b.y)*/
```

[5 Punkte]

Aufgabe 2.3

Erweitern Sie die Klasse `Vec2` testgetrieben. Passen Sie dazu die Datei `source/tests.cpp` an. Fügen Sie nach und nach die im folgenden Quellcode-Fragment angegebenen Operationen zur Klasse hinzu. Entwickeln Sie zu jeder Funktion mehrere Tests.

```
// Vec2 definition
struct Vec2
{
  // ...
  Vec2& operator+=(Vec2 const& v);
  Vec2& operator-=(Vec2 const& v);
  Vec2& operator*=(float s);
  Vec2& operator/=(float s);
  // ...
};
```

Fügen Sie ihre Dateien und Änderungen zu git hinzu:

```
git add source/vec2.hpp source/vec2.cpp
git commit -m "Add Vec2 definition"
git add source/tests.cpp source/CMakeLists.txt
git commit -m "Add tests for Vec2
```

[8 Punkte]

Aufgabe 2.4

Implementieren Sie nun folgende freie Funktionen testgetrieben. Achtung, diese Operatoren sind keine Memberfunktionen! Entwickeln Sie zu jeder Funktion **mehrere** Tests.

Machen Sie sich die bereits implementierten Memberfunktionen Ihres Datentypen aus der vorherigen Aufgabe zu Nutze, indem Sie in der Definition Ihrer freien Funktionen Instanzen von `Vec2` erstellen. Duplizieren Sie keine Implementierung!

```
Vec2 operator+(Vec2 const& u, Vec2 const& v);
Vec2 operator-(Vec2 const& u, Vec2 const& v);
Vec2 operator*(Vec2 const& v, float s);
```

```
Vec2 operator/(Vec2 const& v, float s);
Vec2 operator*(float s, Vec2 const& v);
```

Committen Sie Ihre Änderungen. **[10 Punkte]**

Aufgabe 2.5

In der folgenden Aufgabe sollen sie 2×2-Matrizen implementieren.
Erstellen Sie im Ordner `source/` die Dateien `mat2.hpp` und `mat2.cpp`. Die Elemente der Matrix werden bereits so default-memberinitialisiert, dass ohne explizite Aggregatinitialisierung eine Einheitsmatrix erzeugt wird. Implementieren Sie nun den vorgegebenen Operator zur Matrizenmultiplikation als Memberfunktion und als freie Funktion.

Entwickeln Sie zu jeder Funktion **mehrere** Tests und **commiten** Sie Ihre Änderungen.

```
// Mat2 definition

#include <array>

struct Mat2 {

  // matrix layout:
  //    e_00  e_10
  //    e_01  e_11

  float e_00 = 1.0f;
  float e_10 = 0.0f;
  float e_01 = 0.0f;
  float e_11 = 1.0f;

  //TODO (in mat2.cpp) Definition v. operator*=
  Mat2& operator*=(Mat2 const& m);

};

//  TODO (in mat2.cpp) Definition v. operator*
Mat2 operator*(Mat2 const& m1, Mat2 const& m2);
```

[5 Punkte]

Aufgabe 2.6

Ermöglichen Sie nun die Multiplikation einer Matrix mit einem Vektor. Implementieren Sie eine Methode zur Determinantenberechnung und freie Funktionen zur Berechnung der Inversen und der Transponierten einer 2×2-Matrix. Definieren Sie außerdem eine freie Funktion, die für einen im Bogenmaß gegebenen Winkel eine Rotationsmatrix erstellt.

Testen Sie alle Funktionen und Methoden und **committen** Sie Ihre Änderungen.

```
struct Mat2
{
  //...
  float det() const;
  //...
};

Vec2 operator*(Mat2 const& m, Vec2 const& v);
Vec2 operator*(Vec2 const& v, Mat2 const& m);
Mat2 inverse(Mat2 const& m);
Mat2 transpose(Mat2 const& m);
Mat2 make_rotation_mat2(float phi);
```

[12 Punkte]

Aufgabe 2.7

Erklären Sie den Unterschied zwischen `class` und `struct`. Was ist ein Datentransferobjekt (DTO)? Erstellen Sie im Ordner `source/` eine Datei `color.hpp` und definieren Sie einen Datenyp `Color`, welcher drei Farbintensitäten `r, g` und `b` als Gleitkommazahl-Attribute ($r, g, b \in [0, 1]$) beinhaltet.
Sorgen Sie dafür, dass Instanzen des Datentyps `Color` per Default den internen Zustand für einen mittleren Grauwert annehmen. Schreiben Sie **keine** Konstruktoren!
Testen und **Commiten** Sie Ihre Änderungen.

[3 Punkte]

Aufgabe 2.8

In dieser Aufgabe entwerfen Sie Ihre ersten Klassen. Entwickeln Sie die Klassen `Circle` und `Rectangle` testgetrieben. Verwenden Sie hier das Schlüsselwort `class`. Überlegen Sie, welche Eigenschaften einen Kreis bzw. ein achsenparalleles

Rechteck auszeichnen und statten Sie die Klassen mit gehen, dass Ihre eeigneten **Attributen** und **Konstruktoren** aus.

Schreiben Sie **keine** unnötigen Getter oder Setter-Methoden da diese das Prinzip der Kapselung umgehen würden. Für die korrekte **Funktionsweise der Konstruktoren** müssen Sie demnach **keine** Tests implementieren.

Nutzen Sie Ihren Datentyp `Vec2` innerhalb Ihrer Klassendefinitionen. Ein achsenparalleles Rechteck kann durch zwei Punkte aufgespannt werden. Die linke untere Ecke bezeichnen Sie mit `min_` und die rechte obere mit `max_`.

Committen Sie Ihre Änderungen.

Hinweis: Beachten Sie die Regeln zur Parameterübergabe per `const&` für zusammengesetzte und benutzerdefinierte Datentypen und initialisieren Sie immer alle Instanzvariablen in den Memberinitialisierungsliste der Konstruktoren oder ggf. mit default-Memberinitialisierung wie in der Vorlesung vorgestellt.

[4 Punkte]

Aufgabe 2.9

Erstellen Sie Tests für beide Klassen mit der Methode `circumference` in der Datei `source/tests.cpp`. und implementieren Sie die entsprechende Methode. Sollte diese Methode als `const` deklariert werden? Was ist im Zusammenhang der `const-Correctness` der Unterschied zwischen einer Methode und einer freien Funktion?

Committen Sie Ihre Änderungen.

[5 Punkte]

Aufgabe 2.10

Erweitern Sie die Klassen `Circle` und `Rectangle` um ein Attribut vom Typ `Color` und passen Sie die Konstruktoren entsprechend an.

Committen Sie Ihre Änderungen. **[2 Punkte]**

Aufgabe 2.11

Implementieren Sie für beide Klassen eine `draw`-Methode. Das `Window` in dem das Objekt gezeichnet werden soll wird als Argument übergeben. Nutzen Sie die auf dem `Window` verfügbaren Methoden um einen `Circle` und ein `Rectangle` zu zeichnen. Legen Sie in der Datei `example.cpp` einen `Circle` und ein `Rectangle` und zeichnen Sie beide in der mainloop in deren angegebenen Farben.

Hinweis: Approximieren Sie die Darstellung eines Kreises durch eine feste Anzahl von Liniensegmenten.

Committen Sie ihre Änderungen. **[10 Punkte]**

Aufgabe 2.12

Überladen Sie die `draw`-Methoden beider Klassen, sodass die Liniendicke mit der das Objekt gezeichnet werden soll explizit übergeben werden kann. Nutzen Sie diese Methoden in `example.cpp`. Alternativ können Sie Ihre Klassen auch mit einem zusätzlichen Color-Attribut namens `highlight_color_` ausstatten und die `draw`-Methode so überladen, dass sie einen bool-Parameter übergeben bekommt, der angibt ob ein Objekt in seiner default-Farbe oder in seiner Highlight-Farbe gezeichnet werden soll.

Committen Sie Ihre Änderungen.

Was bedeutet der Begriff Überladen in C++? **[4 Punkte]**

Aufgabe 2.13

Implementieren und testen Sie für beide Klassen eine Methode `is_inside`, mit welcher sich abfragen lässt, ob ein übergebener Punkt (`Vec2`) innerhalb des Objekts liegt.
Testen Sie Ihre Methoden mit Fällen, in denen Punkte innerhalb und außerhalb des Geometrie liegen.
Erweitern Sie das Beispielprogramm, sodass Objekte in denen sich der Mauszeiger befindet doppelt so dick gezeichnet werden wie die restlichen Objekte. Falls Sie in der vorherigen Aufgabe die Highlight-Farbe implementiert haben, können Sie auch alternativ die Objekte, in denen Sich der Mauszeiger befindet in ihrer Highlight-Farbe zeichnen (z.B. blau).

Die Mauszeigerposition erhalten mit der Methode `mouse_position` vom `Window`. Verwalten Sie die Objekte in einem `std::array<Circle>` oder `std::vector<Circle>` bzw. einem `std::array<Rectangle>` oder `std::vector<Rectangle>`.

Hinweis: Die Benutzung der std-Container wird in den Vorlesungsteilen zur STL vorgestellt. Ergänzend können Sie sich schon mit dem Interface der entsprechenden Datentypen in den vorgegebenen Referenzmaterialien vertraut machen. (z.B. http://www.cplusplus.com/reference/vector/vector/?kw=vector)

Commiten Sie Ihre Änderungen. **[10 Punkte]**

Aufgabe 2.14

Schreiben Sie ein Programm, welches eine Analoguhr mit Stunden-, Minuten- und Sekundenzeiger zeichnet. Die Uhr soll die seit dem Programmstart verstrichene Zeit anzeigen. Sie können die Methode `float Window::get_time()` verwenden, um die vergangenen Sekunden abzufragen.

Committen Sie Ihre Änderungen. **[10 Punkte]**

[Gesamt: 88 Punkte]