Programmiersprachen
Sommersemester 2017

Aufgabensammlung 2

Die Lösungen der Aufgaben werden in den Übungen **am 8.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 jede Klasse 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!
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 andreas.bernstein@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 16.04 benötigen Sie:

- cmake
- xorg-dev
- libglu1-mesa-dev
- freeglut3-dev
- libxi-dev
- libxrandr-dev

Unter Windows verwenden Sie *cmake* und *Visual Studio* oder http://landinghub.visualstudio.com/visual-cpp-build-tools mit *Visual Studio Code*. Das Framework

wird mit `C++14` kompiliert. Sprechen Sie uns bitte in der Übung an, falls Sie beim Bauen 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`. Erklären Sie den Zweck der sogenannten *include guards* im Header `vec2.hpp`.
Definieren Sie eine Klasse `Vec2`. Diese Klasse repräsentiert einen Vektor im $\mathbb{R}^2$.

```
#ifndef VEC2_HPP
#define VEC2_HPP

// Vec2 class definition
struct Vec2
{
  // TODO Constructors

  float x;
  float y;
};

#endif // VEC2_HPP
```

Erweitern Sie die Klasse `Vec2` mit einem Standardkonstruktor der die Member mit 0 initialisiert, und einem Konstruktor, welcher die x- und y-Koordinate des Punktes übergeben bekommt. Schreiben Sie für jeden der Konstruktoren einen Test. Die Datei `vec2.cpp` müssen Sie noch in der Datei `source/CMakeLists.txt` als Abhängigkeit zu den Programmen `example` und `tests` hinzufügen.
Erklären Sie den Begriff Destruktor. Müssen Sie für die Klasse `Vec2` einen Destruktor implementieren? **[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"
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.

```
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);
```

Commiten 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`. Statten Sie die Klasse mit einem Standardkonstruktor und einem User-Konstruktor aus. Der Standardkonstruktor erzeugt eine Einheitsmatrix. Implementieren Sie außerdem die vorgegebenen Operatoren zur Matrizenmultiplikation. Entwickeln Sie zu jeder Funktion **mehrere** Tests und **commiten** Sie ihre Änderungen.

```
// Mat2 definition
struct Mat2
{
  // ...
  Mat2& operator*=(Mat2 const& m);
```

```
  // ...
};

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 Funktionen zur Berechnung der Inversen und der Transponierten einer 2×2-Matrix. Definieren Sie außerdem eine Funktion, die für einen im Bogenmaß gegebenen Winkel eine Rotationsmatrix erstellt. Testen Sie alle Funktionen und Methoden und commiten 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 Datentransferobject (DTO)? Erstellen Sie im Ordner `source/` eine Datei `color.hpp` und definieren Sie ein `struct Color`, welches drei Farbintensitäten `r, g` und `b` als Gleitkommazahl-Attribute ($r, g, b \in [0, 1]$) hat. Statten Sie `Color` mit Konstruktoren aus, sodass folgende Initialisierungen möglich sind:

```
Color black{0.0}; // sets r=g=b=0.0
Color red{1.0,0.0,0.0};
```

Commiten Sie ihre Änderungen. **[3 Punkte]**

Aufgabe 2.8

Entwerfen 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 geeigneten Attributen, Konstruktoren sowie *get*-Methoden aus. Nutzen Sie ihre Klasse `Vec2`. Ein achsenparalleles Rechteck kann durch durch zwei Punkte aufgespannt werden. Die linke untere Ecke bezeichnen Sie mit `min_` und die rechte obere mit `max_`. Implementieren Sie passende Tests in der Datei `source/tests.cpp`. Commiten Sie ihre Änderungen.

Hinweis: Beachten Sie die Hinweise zur Parameterübergabe per `const&` und initialisieren Sie immer alle Attribute in der Memberinitialisierungsliste. **[8 Punkte]**

Aufgabe 2.9

Erweitern Sie die Tests für beide Klassen mit der Methode `circumference` und implementieren Sie diese. Sollte diese Methode als `const` deklariert werden? Was ist der Unterschied zwischen einer Methode und einer Funktion ? **Commiten** 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. Commiten 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. Commiten Sie ihre Änderungen.

Hinweis: Approximieren Sie die Darstellung eines Kreises durch eine feste Anzahl von Liniensegmenten. **[10 Punkte]**

Aufgabe 2.12

Überladen Sie die `draw`-Methoden beider Klassen, sodass die Farbe, mit der das Objekt gezeichnet werden soll, explizit übergeben werden kann. Nutzen Sie

diese Methoden in `example.cpp`. Commiten 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 (Punkte innerhalb und außerhalb).
Erweitern Sie das Beispielprogramm , sodass Objekte in denen sich der Mauszeiger befindet blau gezeichnet werden, alle anderen Objekte in ihrer eigenen Farbe. Die Mauszeigerposition erhalten mit der Methode `mouse_position` vom `Window`. Verwalten Sie die Objekte in einem `std::vector<Circle>` und einen `std::vector<Rectangle>`. Commiten Sie ihre Änderungen. **[10 Punkte]**

Aufgabe 2.14

Schreiben Sie ein Program, 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. Commiten Sie ihre Änderungen. **[10 Punkte]**

Programmiersprachen
Sommersemester 2017

Aufgabensammlung 2

Die Lösungen der Aufgaben werden in den Übungen **am 8.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 jede Klasse 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!
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 andreas.bernstein@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 16.04 benötigen Sie:

- cmake
- xorg-dev
- libglu1-mesa-dev
- freeglut3-dev
- libxi-dev
- libxrandr-dev

Unter Windows verwenden Sie *cmake* und *Visual Studio* oder http://landinghub.visualstudio.com/visual-cpp-build-tools mit *Visual Studio Code*. Das Framework

wird mit `C++14` kompiliert. Sprechen Sie uns bitte in der Übung an, falls Sie beim Bauen 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`. Erklären Sie den Zweck der sogenannten *include guards* im Header `vec2.hpp`.
Definieren Sie eine Klasse `Vec2`. Diese Klasse repräsentiert einen Vektor im $\mathbb{R}^2$.

```
#ifndef VEC2_HPP
#define VEC2_HPP

// Vec2 class definition
struct Vec2
{
  // TODO Constructors

  float x;
  float y;
};

#endif // VEC2_HPP
```

Erweitern Sie die Klasse `Vec2` mit einem Standardkonstruktor der die Member mit 0 initialisiert, und einem Konstruktor, welcher die x- und y-Koordinate des Punktes übergeben bekommt. Schreiben Sie für jeden der Konstruktoren einen Test. Die Datei `vec2.cpp` müssen Sie noch in der Datei `source/CMakeLists.txt` als Abhängigkeit zu den Programmen `example` und `tests` hinzufügen.
Erklären Sie den Begriff Destruktor. Müssen Sie für die Klasse `Vec2` einen Destruktor implementieren? **[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"
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.

```
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);
```

Commiten 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`. Statten Sie die Klasse mit einem Standardkonstruktor und einem User-Konstruktor aus. Der Standardkonstruktor erzeugt eine Einheitsmatrix. Implementieren Sie außerdem die vorgegebenen Operatoren zur Matrizenmultiplikation. Entwickeln Sie zu jeder Funktion **mehrere** Tests und **commiten** Sie ihre Änderungen.

```
// Mat2 definition
struct Mat2
{
  // ...
  Mat2& operator*=(Mat2 const& m);
```

```
  // ...
};

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 Funktionen zur Berechnung der Inversen und der Transponierten einer 2×2-Matrix. Definieren Sie außerdem eine Funktion, die für einen im Bogenmaß gegebenen Winkel eine Rotationsmatrix erstellt. Testen Sie alle Funktionen und Methoden und commiten 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 Datentransferobject (DTO)? Erstellen Sie im Ordner `source/` eine Datei `color.hpp` und definieren Sie ein `struct Color`, welches drei Farbintensitäten `r, g` und `b` als Gleitkommazahl-Attribute ($r, g, b \in [0, 1]$) hat. Statten Sie `Color` mit Konstruktoren aus, sodass folgende Initialisierungen möglich sind:

```
Color black{0.0}; // sets r=g=b=0.0
Color red{1.0,0.0,0.0};
```

Commiten Sie ihre Änderungen. **[3 Punkte]**

Aufgabe 2.8

Entwerfen 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 geeigneten Attributen, Konstruktoren sowie *get*-Methoden aus. Nutzen Sie ihre Klasse `Vec2`. Ein achsenparalleles Rechteck kann durch durch zwei Punkte aufgespannt werden. Die linke untere Ecke bezeichnen Sie mit `min_` und die rechte obere mit `max_`. Implementieren Sie passende Tests in der Datei `source/tests.cpp`. Commiten Sie ihre Änderungen.

Hinweis: Beachten Sie die Hinweise zur Parameterübergabe per `const&` und initialisieren Sie immer alle Attribute in der Memberinitialisierungsliste. **[8 Punkte]**

Aufgabe 2.9

Erweitern Sie die Tests für beide Klassen mit der Methode `circumference` und implementieren Sie diese. Sollte diese Methode als `const` deklariert werden? Was ist der Unterschied zwischen einer Methode und einer Funktion ? **Commiten** 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. Commiten 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. Commiten Sie ihre Änderungen.

Hinweis: Approximieren Sie die Darstellung eines Kreises durch eine feste Anzahl von Liniensegmenten. **[10 Punkte]**

Aufgabe 2.12

Überladen Sie die `draw`-Methoden beider Klassen, sodass die Farbe, mit der das Objekt gezeichnet werden soll, explizit übergeben werden kann. Nutzen Sie

diese Methoden in `example.cpp`. Commiten 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 (Punkte innerhalb und außerhalb).
Erweitern Sie das Beispielprogramm , sodass Objekte in denen sich der Mauszeiger befindet blau gezeichnet werden, alle anderen Objekte in ihrer eigenen Farbe. Die Mauszeigerposition erhalten mit der Methode `mouse_position` vom `Window`. Verwalten Sie die Objekte in einem `std::vector<Circle>` und einen `std::vector<Rectangle>`. Commiten Sie ihre Änderungen. **[10 Punkte]**

Aufgabe 2.14

Schreiben Sie ein Program, 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. Commiten Sie ihre Änderungen. **[10 Punkte]**

Programmiersprachen
Sommersemester 2017

Aufgabensammlung 2

Die Lösungen der Aufgaben werden in den Übungen **am 8.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 jede Klasse 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!
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 andreas.bernstein@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 16.04 benötigen Sie:

- cmake
- xorg-dev
- libglu1-mesa-dev
- freeglut3-dev
- libxi-dev
- libxrandr-dev

Unter Windows verwenden Sie *cmake* und *Visual Studio* oder http://landinghub.visualstudio.com/visual-cpp-build-tools mit *Visual Studio Code*. Das Framework

wird mit `C++14` kompiliert. Sprechen Sie uns bitte in der Übung an, falls Sie beim Bauen 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`. Erklären Sie den Zweck der sogenannten *include guards* im Header `vec2.hpp`.
Definieren Sie eine Klasse `Vec2`. Diese Klasse repräsentiert einen Vektor im $\mathbb{R}^2$.

```
#ifndef VEC2_HPP
#define VEC2_HPP

// Vec2 class definition
struct Vec2
{
  // TODO Constructors

  float x;
  float y;
};

#endif // VEC2_HPP
```

Erweitern Sie die Klasse `Vec2` mit einem Standardkonstruktor der die Member mit 0 initialisiert, und einem Konstruktor, welcher die x- und y-Koordinate des Punktes übergeben bekommt. Schreiben Sie für jeden der Konstruktoren einen Test. Die Datei `vec2.cpp` müssen Sie noch in der Datei `source/CMakeLists.txt` als Abhängigkeit zu den Programmen `example` und `tests` hinzufügen.
Erklären Sie den Begriff Destruktor. Müssen Sie für die Klasse `Vec2` einen Destruktor implementieren? **[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"
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.

```
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);
```

Commiten 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`. Statten Sie die Klasse mit einem Standardkonstruktor und einem User-Konstruktor aus. Der Standardkonstruktor erzeugt eine Einheitsmatrix. Implementieren Sie außerdem die vorgegebenen Operatoren zur Matrizenmultiplikation. Entwickeln Sie zu jeder Funktion **mehrere** Tests und **commiten** Sie ihre Änderungen.

```
// Mat2 definition
struct Mat2
{
  // ...
  Mat2& operator*=(Mat2 const& m);
```

```
  // ...
};

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 Funktionen zur Berechnung der Inversen und der Transponierten einer 2×2-Matrix. Definieren Sie außerdem eine Funktion, die für einen im Bogenmaß gegebenen Winkel eine Rotationsmatrix erstellt. Testen Sie alle Funktionen und Methoden und commiten 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 Datentransferobject (DTO)? Erstellen Sie im Ordner `source/` eine Datei `color.hpp` und definieren Sie ein `struct Color`, welches drei Farbintensitäten `r, g` und `b` als Gleitkommazahl-Attribute ($r, g, b \in [0, 1]$) hat. Statten Sie `Color` mit Konstruktoren aus, sodass folgende Initialisierungen möglich sind:

```
Color black{0.0}; // sets r=g=b=0.0
Color red{1.0,0.0,0.0};
```

Commiten Sie ihre Änderungen. **[3 Punkte]**

Aufgabe 2.8

Entwerfen 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 geeigneten Attributen, Konstruktoren sowie *get*-Methoden aus. Nutzen Sie ihre Klasse `Vec2`. Ein achsenparalleles Rechteck kann durch durch zwei Punkte aufgespannt werden. Die linke untere Ecke bezeichnen Sie mit `min_` und die rechte obere mit `max_`. Implementieren Sie passende Tests in der Datei `source/tests.cpp`. Commiten Sie ihre Änderungen.

Hinweis: Beachten Sie die Hinweise zur Parameterübergabe per `const&` und initialisieren Sie immer alle Attribute in der Memberinitialisierungsliste. **[8 Punkte]**

Aufgabe 2.9

Erweitern Sie die Tests für beide Klassen mit der Methode `circumference` und implementieren Sie diese. Sollte diese Methode als `const` deklariert werden? Was ist der Unterschied zwischen einer Methode und einer Funktion ? **Commiten** 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. Commiten 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. Commiten Sie ihre Änderungen.

Hinweis: Approximieren Sie die Darstellung eines Kreises durch eine feste Anzahl von Liniensegmenten. **[10 Punkte]**

Aufgabe 2.12

Überladen Sie die `draw`-Methoden beider Klassen, sodass die Farbe, mit der das Objekt gezeichnet werden soll, explizit übergeben werden kann. Nutzen Sie

diese Methoden in `example.cpp`. Commiten 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 (Punkte innerhalb und außerhalb).
Erweitern Sie das Beispielprogramm , sodass Objekte in denen sich der Mauszeiger befindet blau gezeichnet werden, alle anderen Objekte in ihrer eigenen Farbe. Die Mauszeigerposition erhalten mit der Methode `mouse_position` vom `Window`. Verwalten Sie die Objekte in einem `std::vector<Circle>` und einen `std::vector<Rectangle>`. Commiten Sie ihre Änderungen. **[10 Punkte]**

Aufgabe 2.14

Schreiben Sie ein Program, 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. Commiten Sie ihre Änderungen. **[10 Punkte]**

Programmiersprachen
Sommersemester 2017

Aufgabensammlung 2

Die Lösungen der Aufgaben werden in den Übungen **am 8.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 jede Klasse 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!
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 andreas.bernstein@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 16.04 benötigen Sie:

- cmake
- xorg-dev
- libglu1-mesa-dev
- freeglut3-dev
- libxi-dev
- libxrandr-dev

Unter Windows verwenden Sie *cmake* und *Visual Studio* oder http://landinghub.visualstudio.com/visual-cpp-build-tools mit *Visual Studio Code*. Das Framework

wird mit `C++14` kompiliert. Sprechen Sie uns bitte in der Übung an, falls Sie beim Bauen 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`. Erklären Sie den Zweck der sogenannten *include guards* im Header `vec2.hpp`.
Definieren Sie eine Klasse `Vec2`. Diese Klasse repräsentiert einen Vektor im $\mathbb{R}^2$.

```
#ifndef VEC2_HPP
#define VEC2_HPP

// Vec2 class definition
struct Vec2
{
  // TODO Constructors

  float x;
  float y;
};

#endif // VEC2_HPP
```

Erweitern Sie die Klasse `Vec2` mit einem Standardkonstruktor der die Member mit 0 initialisiert, und einem Konstruktor, welcher die x- und y-Koordinate des Punktes übergeben bekommt. Schreiben Sie für jeden der Konstruktoren einen Test. Die Datei `vec2.cpp` müssen Sie noch in der Datei `source/CMakeLists.txt` als Abhängigkeit zu den Programmen `example` und `tests` hinzufügen.
Erklären Sie den Begriff Destruktor. Müssen Sie für die Klasse `Vec2` einen Destruktor implementieren? **[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"
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.

```
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);
```

Commiten 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`. Statten Sie die Klasse mit einem Standardkonstruktor und einem User-Konstruktor aus. Der Standardkonstruktor erzeugt eine Einheitsmatrix. Implementieren Sie außerdem die vorgegebenen Operatoren zur Matrizenmultiplikation. Entwickeln Sie zu jeder Funktion **mehrere** Tests und **commiten** Sie ihre Änderungen.

```
// Mat2 definition
struct Mat2
{
  // ...
  Mat2& operator*=(Mat2 const& m);
```

```
  // ...
};

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 Funktionen zur Berechnung der Inversen und der Transponierten einer 2×2-Matrix. Definieren Sie außerdem eine Funktion, die für einen im Bogenmaß gegebenen Winkel eine Rotationsmatrix erstellt. Testen Sie alle Funktionen und Methoden und commiten 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 Datentransferobject (DTO)? Erstellen Sie im Ordner `source/` eine Datei `color.hpp` und definieren Sie ein `struct Color`, welches drei Farbintensitäten `r, g` und `b` als Gleitkommazahl-Attribute ($r, g, b \in [0, 1]$) hat. Statten Sie `Color` mit Konstruktoren aus, sodass folgende Initialisierungen möglich sind:

```
Color black{0.0}; // sets r=g=b=0.0
Color red{1.0,0.0,0.0};
```

Commiten Sie ihre Änderungen. **[3 Punkte]**

Aufgabe 2.8

Entwerfen 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 geeigneten Attributen, Konstruktoren sowie *get*-Methoden aus. Nutzen Sie ihre Klasse `Vec2`. Ein achsenparalleles Rechteck kann durch durch zwei Punkte aufgespannt werden. Die linke untere Ecke bezeichnen Sie mit `min_` und die rechte obere mit `max_`. Implementieren Sie passende Tests in der Datei `source/tests.cpp`. Commiten Sie ihre Änderungen.

Hinweis: Beachten Sie die Hinweise zur Parameterübergabe per `const&` und initialisieren Sie immer alle Attribute in der Memberinitialisierungsliste. **[8 Punkte]**

Aufgabe 2.9

Erweitern Sie die Tests für beide Klassen mit der Methode `circumference` und implementieren Sie diese. Sollte diese Methode als `const` deklariert werden? Was ist der Unterschied zwischen einer Methode und einer Funktion ? **Commiten** 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. Commiten 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. Commiten Sie ihre Änderungen.

Hinweis: Approximieren Sie die Darstellung eines Kreises durch eine feste Anzahl von Liniensegmenten. **[10 Punkte]**

Aufgabe 2.12

Überladen Sie die `draw`-Methoden beider Klassen, sodass die Farbe, mit der das Objekt gezeichnet werden soll, explizit übergeben werden kann. Nutzen Sie

diese Methoden in `example.cpp`. Commiten 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 (Punkte innerhalb und außerhalb).
Erweitern Sie das Beispielprogramm , sodass Objekte in denen sich der Mauszeiger befindet blau gezeichnet werden, alle anderen Objekte in ihrer eigenen Farbe. Die Mauszeigerposition erhalten mit der Methode `mouse_position` vom `Window`. Verwalten Sie die Objekte in einem `std::vector<Circle>` und einen `std::vector<Rectangle>`. Commiten Sie ihre Änderungen. **[10 Punkte]**

Aufgabe 2.14

Schreiben Sie ein Program, 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. Commiten Sie ihre Änderungen. **[10 Punkte]**

Programmiersprachen
Sommersemester 2017

Aufgabensammlung 2

Die Lösungen der Aufgaben werden in den Übungen **am 8.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 jede Klasse 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!
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 andreas.bernstein@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 16.04 benötigen Sie:

- cmake
- xorg-dev
- libglu1-mesa-dev
- freeglut3-dev
- libxi-dev
- libxrandr-dev

Unter Windows verwenden Sie *cmake* und *Visual Studio* oder http://landinghub.visualstudio.com/visual-cpp-build-tools mit *Visual Studio Code*. Das Framework

wird mit `C++14` kompiliert. Sprechen Sie uns bitte in der Übung an, falls Sie beim Bauen 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`. Erklären Sie den Zweck der sogenannten *include guards* im Header `vec2.hpp`.
Definieren Sie eine Klasse `Vec2`. Diese Klasse repräsentiert einen Vektor im $\mathbb{R}^2$.

```
#ifndef VEC2_HPP
#define VEC2_HPP

// Vec2 class definition
struct Vec2
{
  // TODO Constructors

  float x;
  float y;
};

#endif // VEC2_HPP
```

Erweitern Sie die Klasse `Vec2` mit einem Standardkonstruktor der die Member mit 0 initialisiert, und einem Konstruktor, welcher die x- und y-Koordinate des Punktes übergeben bekommt. Schreiben Sie für jeden der Konstruktoren einen Test. Die Datei `vec2.cpp` müssen Sie noch in der Datei `source/CMakeLists.txt` als Abhängigkeit zu den Programmen `example` und `tests` hinzufügen.
Erklären Sie den Begriff Destruktor. Müssen Sie für die Klasse `Vec2` einen Destruktor implementieren? **[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"
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.

```
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);
```

Commiten 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`. Statten Sie die Klasse mit einem Standardkonstruktor und einem User-Konstruktor aus. Der Standardkonstruktor erzeugt eine Einheitsmatrix. Implementieren Sie außerdem die vorgegebenen Operatoren zur Matrizenmultiplikation. Entwickeln Sie zu jeder Funktion **mehrere** Tests und **commiten** Sie ihre Änderungen.

```
// Mat2 definition
struct Mat2
{
  // ...
  Mat2& operator*=(Mat2 const& m);
```

```
  // ...
};

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 Funktionen zur Berechnung der Inversen und der Transponierten einer 2×2-Matrix. Definieren Sie außerdem eine Funktion, die für einen im Bogenmaß gegebenen Winkel eine Rotationsmatrix erstellt. Testen Sie alle Funktionen und Methoden und commiten 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 Datentransferobject (DTO)? Erstellen Sie im Ordner `source/` eine Datei `color.hpp` und definieren Sie ein `struct Color`, welches drei Farbintensitäten `r, g` und `b` als Gleitkommazahl-Attribute ($r, g, b \in [0, 1]$) hat. Statten Sie `Color` mit Konstruktoren aus, sodass folgende Initialisierungen möglich sind:

```
Color black{0.0}; // sets r=g=b=0.0
Color red{1.0,0.0,0.0};
```

Commiten Sie ihre Änderungen. **[3 Punkte]**

Aufgabe 2.8

Entwerfen 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 geeigneten Attributen, Konstruktoren sowie *get*-Methoden aus. Nutzen Sie ihre Klasse `Vec2`. Ein achsenparalleles Rechteck kann durch durch zwei Punkte aufgespannt werden. Die linke untere Ecke bezeichnen Sie mit `min_` und die rechte obere mit `max_`. Implementieren Sie passende Tests in der Datei `source/tests.cpp`. Commiten Sie ihre Änderungen.

Hinweis: Beachten Sie die Hinweise zur Parameterübergabe per `const&` und initialisieren Sie immer alle Attribute in der Memberinitialisierungsliste. **[8 Punkte]**

Aufgabe 2.9

Erweitern Sie die Tests für beide Klassen mit der Methode `circumference` und implementieren Sie diese. Sollte diese Methode als `const` deklariert werden? Was ist der Unterschied zwischen einer Methode und einer Funktion ? **Commiten** 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. Commiten 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. Commiten Sie ihre Änderungen.

Hinweis: Approximieren Sie die Darstellung eines Kreises durch eine feste Anzahl von Liniensegmenten. **[10 Punkte]**

Aufgabe 2.12

Überladen Sie die `draw`-Methoden beider Klassen, sodass die Farbe, mit der das Objekt gezeichnet werden soll, explizit übergeben werden kann. Nutzen Sie

diese Methoden in `example.cpp`. Commiten 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 (Punkte innerhalb und außerhalb).
Erweitern Sie das Beispielprogramm , sodass Objekte in denen sich der Mauszeiger befindet blau gezeichnet werden, alle anderen Objekte in ihrer eigenen Farbe. Die Mauszeigerposition erhalten mit der Methode `mouse_position` vom `Window`. Verwalten Sie die Objekte in einem `std::vector<Circle>` und einen `std::vector<Rectangle>`. Commiten Sie ihre Änderungen. **[10 Punkte]**

Aufgabe 2.14

Schreiben Sie ein Program, 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. Commiten Sie ihre Änderungen. **[10 Punkte]**