Praktikumstermin Nr. 10, INF: Klassen, Mocking beim Testen

Abgabe im GIP-INF Praktikum der Woche 13.12.-17.12.2021.

(Pflicht-) Aufgabe INF-10.01: Klasse `MyRectangle`

Legen Sie in Visual Studio ein neues leeres Projekt an und innerhalb dieses Projekts dann leere Dateien `MyRectangle.h`, `MyRectangle.cpp`, `main.cpp`.

Programmieren Sie in den entsprechenden Dateien die Klasse `MyRectangle` gemäß den folgenden Anforderungen:

Jedes Objekt der Klasse `MyRectangle` besitze zwei `int` Attribute `x1` und `y1` für die linke obere Ecke des Rechtecks und zwei weitere `int` Attribute `x2`, `y2` für die rechte untere Ecke. Alle diese Attribute sollen gegen Zugriff von außen geschützt werden; stellen Sie dies sicher und programmieren Sie Getter und Setter für jedes dieser Attribute. Programmieren Sie ferner einen weiteren Setter `set()`, der vier Parameter nimmt und damit alle vier Attribute gleichzeitig setzt.

Der `MyRectangle` Konstruktor soll vier `int` Parameter übernehmen und die vier Attribute entsprechend setzen. Es soll ferner möglich sein, ein `MyRectangle` Objekt ohne Angabe von Parametern anzulegen. Dann soll die linke obere Ecke auf `(0,0)` und die rechte untere Ecke auf `(20,20)` gesetzt werden.

Hauptprogramm, Datei `main.cpp`:

Sie finden in Ilias die Headerdatei `CImgGIP06.h`. Legen Sie innerhalb Ihres Projekts eine leere Headerdatei unter diesem Namen an und copy-pasten Sie den Inhalt der Datei aus Ilias in diese Datei.

Kopieren Sie folgenden Quelltext in Ihre Datei `main.cpp`:

```
#include <iostream>
using namespace std;

#define CIMGGIP_MAIN
#include "CImgGIP06.h"

#include "MyRectangle.h"
```

```
int main()
{
    while (gip_window_not_closed())
    {
        int x1_1 = gip_random(0, gip_win_size_x - 1);
        int y1_1 = gip_random(0, gip_win_size_y - 1);
        int x2_1 = gip_random(x1_1, gip_win_size_x - 1);
        int y2_1 = gip_random(y1_1, gip_win_size_y - 1);
        MyRectangle r1(x1_1, y1_1, x2_1, y2_1);

        int x1_2 = gip_random(0, gip_win_size_x - 1);
        int y1_2 = gip_random(0, gip_win_size_y - 1);
        int x2_2 = gip_random(x1_2, gip_win_size_x - 1);
        int y2_2 = gip_random(y1_2, gip_win_size_y - 1);
        MyRectangle r2(x1_2, y1_2, x2_2, y2_2);

        // Alles neu zeichnen ...
        gip_white_background();
        r1.draw();
        r2.draw();

        // Pausieren ...
        gip_sleep(4);
    }
    return 0;
}
```

Erweitern Sie Ihre Klasse `MyRectangle` um eine *konstante* parameterlose Methode `void draw() const`, welche das `MyRectangle` Objekt mittels des Funktionsaufrufs `gip_draw_rectangle(x1, y1, x2, y2, blue);` zeichnet.

Führen Sie Ihr Programm aus, um zu testen, ob es funktioniert. Es sollten im 4-Sekunden-Takt immer wieder zwei neue blaue Rechtecke erscheinen.

(Pflicht-) Aufgabe INF-10.02: Kollisionserkennung

Erweitern Sie Ihr Hauptprogramm hinter den beiden `draw()` Zeilen um die Zeilen …

```
        if (r1.does_not_collide_with(r2))
            gip_draw_text(10, 10, "Keine Kollision.");
        else
            gip_draw_text(10, 10, "Kollision!");
```

Erweitern Sie nun Ihre Klasse `MyRectangle` um eine *konstante* Methode `bool does_not_collide_with(const MyRectangle& other) const;` zur Erkennung einer Kollision zwischen den beiden Rechtecken `*this` und `other`.

Hinweis:
Zwei Rechtecke kollidieren nicht, …
… wenn die rechte Kante des ersten Rechtecks schon links von der linken Kante des zweiten Rechtecks liegt (welche Beziehung bedeutet dies für die x-Koordinaten x1, x2 der beiden Rechtecke?), oder …
… wenn die rechte Kante des zweiten Rechtecks schon links von der linken Kante des ersten Rechtecks liegt (welche Beziehung bedeutet dies für die x-Koordinaten x1, x2 der beiden Rechtecke?), oder …
… wenn die obere Kante des zweiten Rechtecks schon unterhalb der unteren Kante des ersten Rechtecks liegt (welche Beziehung bedeutet dies für die y-Koordinaten y1, y2 der beiden Rechtecke?), oder …
… wenn die obere Kante des ersten Rechtecks schon unterhalb der unteren Kante des zweiten Rechtecks liegt (welche Beziehung bedeutet dies für die y-Koordinaten y1, y2 der beiden Rechtecke?).

Testen Sie die `does_not_collide_with()` Methode, indem Sie das Programm laufen lassen und die Ausgaben beobachten und visuell verifizieren.

(Pflicht-) Aufgabe INF-10.03: Unit Testing

Legen Sie in Visual Studio ein neues leeres Projekt an und kopieren Sie per copy-paste (nicht: „Hinzufügen einer existierenden Datei“) die beiden Dateien `MyRectangle.h` und `MyRectangle.cpp` dorthin (d.h. leere Dateien unter diesen Namen anlegen, dann Inhalt per copy-paste übertragen). Ebenso die Datei `CImgGIP06.h`

Fügen Sie in `MyRectangle.cpp` direkt vor der Zeile `#include "CImgGIP06.h"` noch die Zeile `#define CIMGGIP_MAIN` ein.

Nun wollen wir die Methode `does_not_collide_with()` unit-testen.

Legen Sie dazu eine Datei `main.cpp` mit folgendem Inhalt an:

```
// Datei: main.cpp

#include <iostream>
using namespace std;

#define CATCH_CONFIG_RUNNER
#include "catch.h"
```

```
int main()
{
    Catch::Session().run();

    system("PAUSE");
    return result;
}
```

Legen Sie ferner eine Datei `collision_test.cpp` folgenden Inhalts an:

```
// Datei: collision_test.cpp

#include "catch.h"
#include "MyRectangle.h"

TEST_CASE("Pruefung der Methode MyRectangle::does_not_collide_with()") {
    REQUIRE(MyRectangle(200,200,300,300).does_not_collide_with(MyRectangle(100,200,150,300)) == true);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(400, 200, 500, 300)) == true);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(200, 0, 300, 100)) == true);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(200, 400, 300, 500)) == true);

    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(100, 200, 200, 300)) == false);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(100, 200, 250, 300)) == false);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(300, 200, 400, 300)) == false);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(250, 200, 500, 300)) == false);

    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(200, 100, 300, 200)) == false);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(200, 100, 300, 250)) == false);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(200, 300, 300, 400)) == false);
    REQUIRE(MyRectangle(200, 200, 300, 300).does_not_collide_with(MyRectangle(200, 250, 300, 400)) == false);

}
```

Legen Sie jetzt noch in Ihrem Projekt eine leere Datei `catch.h` an und copy-pasten Sie den Inhalt der gleichnamigen Datei aus Ilias in diese Datei.

Starten Sie das Programm. Mal schauen, ob Ihre Methode `does_not_collide_with()` alle Unit Tests besteht…

(Pflicht-) Aufgabe INF-10.04: Unit Testing mit Mocken der CImg Library

Wie Sie sicher sehen, geht im Kontext der Tests immer ein leeres `CImg` Fenster auf, obwohl die graphische Ausgabe im Kontext der Unit Tests nicht benötigt wird und nur unnötigen „Ballast" erzeugt.

Die einzige Datei des aktuellen Projekts, in der `CImg` noch benutzt wird, ist `MyRectangle.cpp`. Was können wir also tun?

Versuch: Löschen Sie die Zeile `#include "CImgGIP06.h"` in der Datei `MyRectangle.cpp`. Ergebnis: Das Projekt dürfte nicht compilieren, da in

der `draw()` Methode immer noch `CImg` Funktionalität benutzt wird, obwohl die `draw()` Methode im Kontext unserer Unit Tests von `does_not_collide_with()` gar keine Relevanz hat.

Wir könnten jetzt natürlich versucht sein, die `draw()` Methode ebenfalls zu löschen. Aber der Code der „Klasse unter Test" darf nicht verändert werden, das ist eine Art „heiliges Gesetz des Testens", denn sonst testen Sie nicht mehr das „Original", sondern eine veränderte Variante der Klasse `MyRectangle` …

Lösung: Wir belassen die Klasse `MyRectangle` so wie sie ist und ersetzen die `CImg` Library durch eine „Attrappe / Fälschung" (engl: „mock"). Die Fälschung zeichnet sich dadurch aus, dass Sie „praktisch nichts tut".

Erstellen Sie also eine neue Headerdatei `CImgGIP06Mock.h` (die ursprüngliche Headerdatei `CImgGIP06.h` können Sie belassen, die werden wir jetzt „einfach ignorieren") mit folgendem Inhalt:

```
// Datei: CImgGIP06Mock.h

#pragma once

const unsigned char white[] = { 255, 255, 255 };
const unsigned char black[] = { 0, 0, 0 };
const unsigned char red[] = { 255, 0, 0 };
const unsigned char green[] = { 0, 255, 0 };
const unsigned char blue[] = { 0, 0, 255 };

const unsigned int gip_win_sizeX = 600;       // Fenstergroesse X
const unsigned int gip_win_sizeY = 600;       // Fenstergroesse Y

inline void gip_draw_rectangle(unsigned int x0, unsigned int y0,
    unsigned int x1, unsigned int y1, const unsigned char *const color = black) {}
```

Setzen Sie nun an den Anfang der Datei `MyRectangle.cpp` die Direktive `#include "CImgGIP06Mock.h"` (dort, wo vorher `#include "CImgGIP06.h"` stand).

Nun sollten Ihre Unit Tests auch wieder funktionieren, ohne dass sich aber ein leeres Fenster öffnet oder sonst irgendwelche graphische Funktionalität benutzt würde.

Solches „Mock:en" wird beim Software-Unit-Testen häufig eingesetzt, z.B. wenn eine Software eigentlich intern auf eine große Datenbank zugreift, man beim Testen aber nicht solch eine schwergewichtige Datenbank installieren und füllen möchte. Dann werden die Methoden zum Datenbankzugriff „ge-mock:ed", d.h. ersetzt durch Methoden, die

stattdessen auf eine leichtgewichtige Datenbank-„Attrappe" zugreifen, die dann Dummy-Daten für die Tests liefert.

(Pflicht-) Aufgabe INF-10.05: Templates

Programmieren Sie einen `struct` Datentyp Tupel, der zwei Komponentenwerte `komponente1` und `komponente2` von beliebigen, ggfs. unterschiedlichen Datentypen speichert. Definieren Sie Tupel als Template-Datentyp.

Programmieren Sie außerdem eine Template-Funktion …
`int vergleiche(… p1 … , … p2 …)`
… die zwei Werte des Tupel-Typs miteinander vergleicht und `-1` zurückgibt, wenn beide Komponentenwerte des ersten Tupels kleiner sind als die jeweiligen Parameterwerte des zweiten Tupels, die `+1` zurückgibt falls beide Komponentenwerte des zweiten Tupels kleiner sind als die jeweiligen Parameterwerte des ersten Tupels und die ansonsten den Wert `0` zurückgibt.
Die beiden Parameterwerte der Funktion seien von einem beliebigen, aber identischen Tupel-Typ (also Typen der Komponenten identisch spezialisiert, siehe Hauptprogramm zum Testlauf).

Für die Definition des Tupel-Typs und der Template-Funktion sollen zwei Dateien `tupel.h` und `tupel.cpp` genutzt werden.

In eine Datei `tupel_main.cpp` soll das im Folgenden angegebene Hauptprogramm eingefügt werden. Die Templates sollen die Methodik der expliziten Instanziierung nutzen (damit sollte klar sein, was in die Datei `tupel.h` gehört und was in die Datei `tupel.cpp`).

```
// Datei: tupel_main.cpp

#include <string>
#include <iostream>

#include "tupel.h"

int main()
{
    Tupel<std::string, int> hansi = { "Hansi", 8 };
    Tupel<std::string, int> willi = { "Willi", 77 };

    std::cout << vergleiche<std::string, int>(hansi, willi) << std::endl;

    Tupel<int, int> t1 = { 3 , 4 };
    Tupel<int, int> t2 = { 1 , 2 };

    std::cout << vergleiche<int, int>(t1, t2) << std::endl;

    Tupel<int, int> t3 = { 9 , 1 };
    Tupel<int, int> t4 = { 3 , 5 };

    std::cout << vergleiche<int, int>(t3, t4) << std::endl;

    system("PAUSE");
    return 0;
```

Für das Aufgaben-Tutorium am Freitag 10.12.2021, als freiwillige Aufgabe:

Aufgabe INF-10.06: Dynamische Datenstruktur: Verkettete Liste mit zusätzlichem Pointer auf erstes Element

Gegeben sei eine Datenstruktur für die Speicherung von Listen von Ganzzahlen (Integers) als einfach verkettete Listen auf dem Heap. Zusätzlich verweise jeder Knoten mittels des Pointers `first` auf den aktuell ersten Knoten der Liste:

```
struct TListenKnoten {
    int data;
    TListenKnoten *next;
    TListenKnoten *first;
};
```

Der Anker der Datenstruktur sei eine Variable `TListenKnoten* anker;`

Der `anker` zeige auf das erste Listenelement. Die leere Liste werde durch den Nullpointer als Wert von `anker` repräsentiert.

Der `anker` werde als Parameter an alle Funktionen zur Bearbeitung der Liste übergeben.

Ergänzen Sie das C++-Programm aus der Vorlage um eine neue Funktion `void vorne_anfuegen(…)` welche einen neuen Wert vorne an die Liste anfügt. Überlegen Sie, welche Parameter diese Funktion benötigt.

Außer für den einzufügenden Knoten sollen keine weiteren (unnötigen) `TListenKnoten` alloziert werden.

Achten Sie auch darauf, dass nach dem Einfügen des neuen Elements die Datenstruktur wieder korrekt verkettet ist. Um dies zu kontrollieren, können Sie die Ausgabefunktion aus der Vorlage verwenden, die die Adresse des `first`-Pointers für den jeweiligen Knoten ausgibt (siehe Testlauf).
*Der folgende Testlauf fügt nacheinander die Ganzzahlwerte **5**, dann **4**, dann **3**, dann **2**, dann **1** vorne an die Liste an. Dies geschieht dadurch, dass im Hauptprogramm diese `vorne_anfuegen()` Operationen fest programmiert sind. Es gibt keine Benutzereingaben. Die Adresswerte im Testlauf sind beispielhaft, bei den Testläufen ihres Programms werden dort dann andere Adresswerte stehen. Wichtig ist, dass die `first ptr` Adresswerte immer mit dem des jeweils ersten Knotens übereinstimmen. Alle Einfärbungen im Testlauf sind zur Illustration, ihr Programm soll keine solchen farbigen Ausgaben machen.*

Testlauf (***Adresswerte beispielhaft***, *keine Benutzereingaben*):

```
Leere Liste.

##### Liste Anfang #####
5, this: 00000127213B1680, first ptr: 00000127213B1680
###### Liste Ende #####

##### Liste Anfang #####
4, this: 00000127213B1800, first ptr: 00000127213B1800
5, this: 00000127213B1680, first ptr: 00000127213B1800
###### Liste Ende #####

##### Liste Anfang #####
3, this: 00000127213B13E0, first ptr: 00000127213B13E0
4, this: 00000127213B1800, first ptr: 00000127213B13E0
5, this: 00000127213B1680, first ptr: 00000127213B13E0
###### Liste Ende #####

##### Liste Anfang #####
2, this: 00000127213B16E0, first ptr: 00000127213B16E0
3, this: 00000127213B13E0, first ptr: 00000127213B16E0
4, this: 00000127213B1800, first ptr: 00000127213B16E0
5, this: 00000127213B1680, first ptr: 00000127213B16E0
###### Liste Ende #####

##### Liste Anfang #####
1, this: 00000127213B1740, first ptr: 00000127213B1740
2, this: 00000127213B16E0, first ptr: 00000127213B1740
3, this: 00000127213B13E0, first ptr: 00000127213B1740
4, this: 00000127213B1800, first ptr: 00000127213B1740
5, this: 00000127213B1680, first ptr: 00000127213B1740
###### Liste Ende #####
```