Praktikumstermin Nr. 09, INF:
HTML Datei aus Vorlage & Daten, Klassen, Mocking beim Testen

Hinweis: Erschrecken Sie nicht, die Lösungen der Aufgaben sind nicht so aufwändig wie der textuelle Umfang dieser Praktikumsanleitung vielleicht befürchten lässt … Insbesondere bei den Aufgaben 09-03 bis 09-05 müssen eigentlich nur die (wenigen) in der Anleitung vorgegebenen Schritte "abgearbeitet" werden …

(Pflicht-) Aufgabe INF-09.01: Webseitendatei erstellen aus Vorlagendatei und Datendatei (Dateioperationen, Stringoperationen)

Hinweis:
Sie brauchen im Praktikum nur das resultierende Programm vorzuzeigen, welches alle Anforderungen der kompletten Aufgabe löst. Einzelne Zwischenversionen (z.B. gemäß den unten angegebenen Arbeitsschritten, falls Sie sich an diesen orientieren wollen) brauchen Sie nicht dauerhaft zu speichern bzw. im Praktikum vorzuzeigen.

Legen Sie in Visual Studio ein neues leeres C++ Projekt für diese Praktikumsaufgabe an. Den Namen des Projekts können Sie frei wählen. Legen Sie dann innerhalb des Projekts eine neue C++ Quelltextdatei (`.cpp` Datei) an. Auch hier können Sie den Namen der Datei frei wählen (Endung `.cpp`).

Öffnen Sie nun den Ordner mit den Dateien dieses Projekts im Windows Explorer: Klicken Sie dazu mit der rechten Maustaste in Visual Studio auf den Projektnamen, dann *"Ordner in Datei-Explorer öffnen"* (zweiter Eintrag von unten in dem Kontextmenü des Projekts) auswählen.

Laden Sie folgende Dateien aus dem GIP Praktikumsordner in Ilias herunter (erst einmal in einen beliebigen Zielordner):

`webseite.html.tmpl`, die Webseiten-Vorlagendatei (*„Vorlage" = „Template"*)
Achtung: Diese Datei heißt nach dem Herunterladen aus Ilias aus Sicherheitsgründen leider `webseitehtmltmpl.sec`. Sie müssen die heruntergeladene Datei also umbenennen in `webseite.html.tmpl`.

`personendaten.txt`, die Datendatei.

Kopieren oder verschieben Sie diese Dateien in den Ordner mit den Dateien Ihres Projekts.

Gesamtaufgabe:
Schreiben Sie ein C++ Programm, welches die beiden Dateien einliest und mittels der Daten aus den beiden Dateien eine neue Ausgabedatei `webseite.html` schreibt, gemäß den unten beschriebenen Regeln.

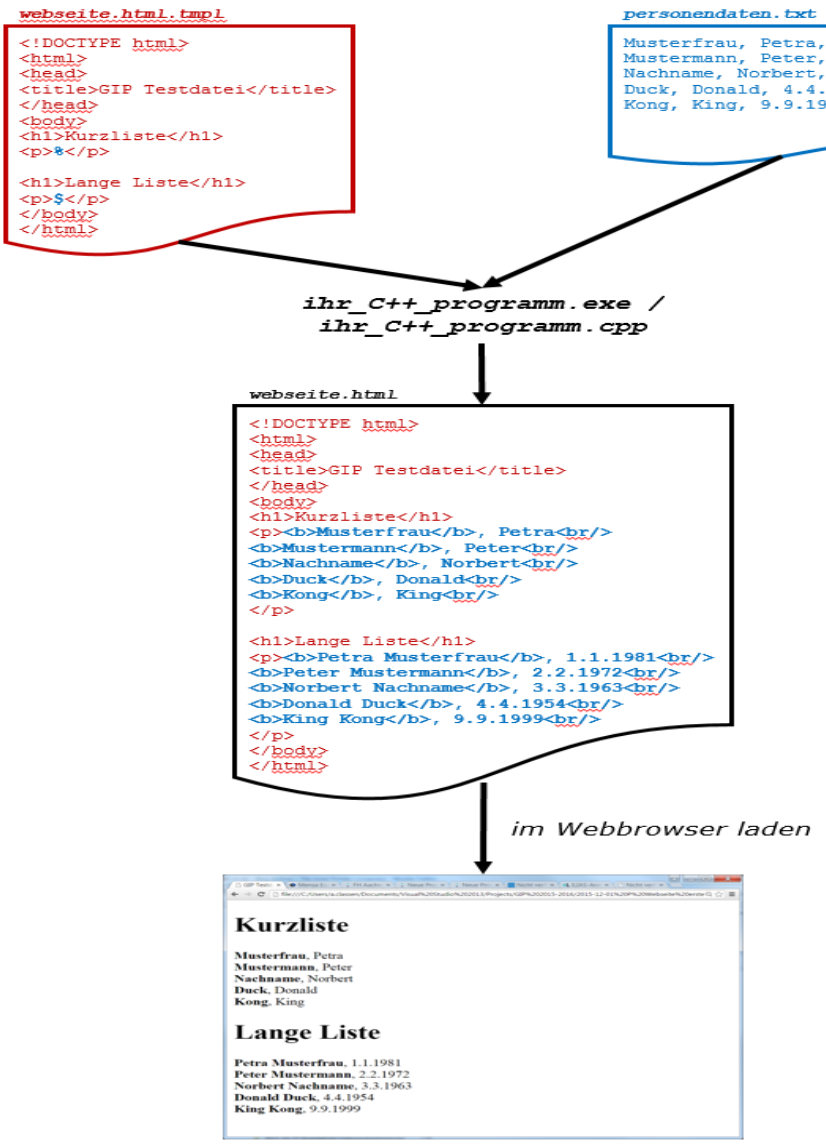
Beim Inhalt der Ausgabedatei `webseite.html` wird es sich um eine Webseite im HTML Format handeln. Öffnen Sie zur Überprüfung, ob Ihr Programm richtig gearbeitet hat, die Ausgabedatei in einem Webbrowser (z.B. in Firefox per Tastendruck `Strg-O`), um sie entsprechend aufbereitet anzeigen zu lassen.
Sollten Sie Ihr C++ Programm erneut starten, so können Sie den aktualisierten Inhalt der Datei mittels "*Reload*" im Webbrowser neu anzeigen lassen.

Das Vorgehen Ihres C++ Programms bei der Erzeugung der Ausgabedatei sei dabei wie folgt:

- Im Prinzip wird der Inhalt der Vorlagendatei gelesen und meist unverändert in die Ausgabedatei geschrieben.

- Kommt in einer Zeile der Vorlagendatei das Zeichen `%` vor, so soll dieses einzelne Zeichen ersetzt werden durch folgenden längeren Text:
  Den Text für die Kurz-Liste der Personen. Jeder „kurze" Personeneintrag der Liste besteht aus dem Nachnamen (aus der Datendatei) in Fettschrift, gefolgt von einem Komma und einem Leerzeichen, gefolgt vom Vornamen in normaler Schrift.
  Wie dieser Text für den Inhalt der Liste genau gebildet wird, ist weiter unten beschrieben.

- Kommt in einer Zeile der Vorlagendatei das Zeichen **`$`** vor, so soll dieses Zeichen ersetzt werden durch folgenden Text:
  Den Text für die „Langform-Liste". Jeder „lange Personeneintrag" besteht aus dem Vornamen gefolgt vom Nachnamen, beides in Fettschrift, gefolgt von einem Komma und einem Leerzeichen, gefolgt vom Geburtsdatum. Außer dem Vor- und dem Nachnamen sei der ganze Text in normaler Schrift.
  Wie der Text für diese Liste genau gebildet wird, ist weiter unten beschrieben.



Die Listentexte (siehe blaue Sektionen in `webseite.html` im Diagramm) ergeben sich wie folgt:

- Jede Listenzeile entspricht einer Datenzeile der Datendatei.
- Jede Listenzeile wird abgeschlossen mit dem Text `<br/>`.
- Der Text der Listenzeile ergibt sich aus den Zeilen der Datendatei, die wie in der Aufgabenstellung oben beschrieben umgewandelt werden.
- Ein Text wird in Fettdruck ausgegeben, wenn er innerhalb die Zeichen `<b>...</b>` steht.

Testlauf:

Keine sichtbaren Eingaben und Ausgaben des Programms. Reines Lesen und Schreiben der Dateien.

Arbeitsschritte (Sie müssen diese Schritte nicht unbedingt befolgen, dies ist nur ein "Angebot"; ihr Programm muss auch nicht unbedingt den hier gemachten Vorschlägen entsprechen):

1. Schreiben Sie ein C++ Programm, welches die Datendatei `personendaten.txt` zeilenweise einliest und jede eingelesene Zeile wieder auf den Bildschirm ausgibt. *Pseudo-Code:*

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

int main()
{
    string eingabezeile;
    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;
    solange(eingabezeile aus Datendatei lesen) {
        eingabezeile ausgeben;
    }
    Datendatei schließen;
    return 0;
}
```

2a. Erweitern Sie dieses C++ Programm:

- Definieren Sie in einer separaten Headerdatei `person.h` einen strukturierten Datentyp für Personen.

- Definieren Sie in einer separaten Implementierungsdatei `person.cpp` eine Funktion …

`Person extrahiere_person(string eingabezeile)`

… welche aus einer Eingabezeile die Personendaten extrahiert:
Die Eingabezeile soll am ersten Komma aufgespalten werden und der Teil bis zum Komma als Nachname verwendet werden.

Der zweite Teil der Aufspaltung soll wieder am ersten Komma aufgespalten werden. Der Teil bis zum Komma soll als Vorname benutzt werden, der Teil dahinter als Geburtsdatum.

Benutzen Sie zum Aufspalten die Funktion `spalte_ab_erstem()` aus den Offline-Aufgaben. Platzieren Sie diese in separaten Dateien `texte.h` und `texte.cpp`.

Die Funktion `extrahiere_person()` soll dann den so erstellten `Person`-Wert zurückgeben.

Lassen Sie von Ihrem C++Hauptprogramm die Daten der erhaltenen Person ausgeben. *Pseudo-Code:*

```
// In person.h: ////////////////////////
struct Person { ... };

// In person.cpp: ////////////////////////
Person extrahiere_person(string eingabezeile)
{
    Person p;
    string rest;
    spalte_ab_erstem(eingabezeile, ',', p.nachname, rest);
    spalte_ab_erstem(rest, ',', p.vorname, p.geburtsdatum);
    return p;
}

// Datei main.cpp: ////////////////////////
#include <iostream>
#include <string>
#include <fstream>
// ... weitere include ...
using namespace std;

int main()
{
    string eingabezeile = "";
    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;
    solange(eingabezeile aus Datendatei lesen) {
        Person person = extrahiere_person(eingabezeile);
        // Dann person.vorname, person.nachname,
        //      person.geburtsdatum ausgeben
    }
    Datendatei schließen;
    return 0;
}
```

2b. Fällt Ihnen auf, dass die einzelnen Personendaten noch unnötige Leerzeichen am Anfang und Ende der Einzelwerte der Person enthalten können?

Falls ja: Erweitern Sie die Funktion `extrahiere_person()`, so dass bei der Rückgabe des Werts `p` die Komponentenwerte `vorname`, `nachname`, `geburtsdatum` "gesäubert" werden. Benutzen Sie dazu die `string trimme(string s)` Funktion aus den Offline-Aufgaben (Code der Funktion so ändern, dass statt Pluszeichen jetzt Leerzeichen entfernt werden), dir Sie ebenfalls in `texte.h` bzw. `texte.cpp` platzieren:

```
Person extrahiere_person(string eingabezeile)
{
    Person p; string rest = "";
    spalte_ab_erstem(eingabezeile, ',', p.nachname, rest);
    spalte_ab_erstem(rest, ',', p.vorname, p.geburtsdatum);
    p.nachname = trimme(p.nachname);
    p.vorname = trimme(p.vorname);
    p.geburtsdatum = trimme(p.geburtsdatum);
    return p;
}
```

3. Erweitern Sie nun Ihr Hauptprogramm, um den Kurztext gemäß den Vorgaben erstellen zu lassen. Die bisherige Ausgabe der Personenwerte kann wieder weggelassen werden.

```
int main()
{
    string eingabezeile = "", kurztext = "", langtext = "";
    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;
    solange(eingabezeile aus Datendatei lesen) {
        Person person = extrahiere_person(eingabezeile);
        // kurztext ausgeben lassen ...
        kurztext += br(
                        b(person.nachname) + ", " + person.vorname
                    ) + "\n";
    }
    Datendatei schließen;
    return 0;
}
```

Es werden dabei zwei Hilfsfunktionen benötigt:
`string br(string s)` hängt an den String `s` den Text `<br/>` an.
`string b(string s)` hängt vor den String `s` den Text `<b>` und dahinter den Text `</b>`. Diese beiden Funktionen können Sie in der `main.cpp` platzieren.

4. Erweitern Sie nun Ihr Hauptprogramm, um den Langtext gemäß den Vorgaben erstellen zu lassen.

```
int main()
{
    string eingabezeile = "", kurztext = "", langtext = "";
    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;
    solange(eingabezeile aus Datendatei lesen) {
        Person person = extrahiere_person(eingabezeile);
        // kurztext erstellen ...
        kurztext += br(
                        b(person.nachname) + ", " + person.vorname
                    ) + "\n";
        // langtext erstellen ...
        langtext += br(
                        b(person.vorname + " " + person.nachname) +
                        ", " +
                        person.geburtsdatum
                    ) + "\n";
    }
    Datendatei schließen;
    return 0;
}
```

5. Erweitern Sie Ihr Hauptprogramm um das Lesen der Vorlagendatei und das Schreiben der Ausgabedatei. Benutzen Sie dabei die `ersetze()` Funktion aus den Offline-Aufgaben.

```
int main()
{
    string eingabezeile = "", kurztext = "", langtext = "";
    Datendatei "personendaten.txt" als textuelle Eingabedatei öffnen;
    solange(eingabezeile aus Datendatei lesen) {
        Person person = extrahiere_person(eingabezeile);
        // kurztext erstellen ...
        kurztext += br(
                        b(person.nachname) + ", " + person.vorname
                    ) + "\n";
        // langtext erstellen ...
        langtext += br(
                        b(person.vorname + " " + person.nachname) +
                        ", " +
                        person.geburtsdatum
                    ) + "\n";
    }
    Datendatei schließen;

    Templatedatei "webseite.html.tmpl" als textuelle Eingabedatei öffnen;
    Ausgabedatei "webseite.html" als textuelle Datei zum Schreiben öffnen;
    solange(eingabezeile aus Templatedatei lesen) {
        eingabezeile = ersetze(eingabezeile, '%', kurztext);
        eingabezeile = ersetze(eingabezeile, '$', langtext);

        Schreibe eingabezeile + "\n" in die Ausgabedatei;
    }
    Templatedatei schließen;
    Ausgabedatei schließen;

    return 0;
}
```

(Pflicht-) Aufgabe INF-09.02: 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 `CImgGIP05.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 "CImgGIP05.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 parameterlose Methode `void draw()`, 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-09.03: 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-09.04:
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 `CImgGIP05.h`

Fügen Sie in `MyRectangle.cpp` direkt vor der Zeile `#include "CImgGIP05.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(int argc, char* const argv[])
{
	int result = Catch::Session().run(argc, argv);
	cout << "Resultatwert: " << result << endl;

	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-09.05: 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 "CImgGIP05.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 `CImgGIP05Mock.h` (die ursprüngliche Headerdatei `CImgGIP05.h` können Sie belassen, die werden wir jetzt „einfach ignorieren") mit folgendem Inhalt:

```
// Datei: CImgGIP05Mock.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 "CImgGIP05Mock.h"` (dort, wo vorher `#include "CImgGIP05.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.

Für das Aufgaben-Tutorium am Freitag 15.1.2021, als freiwillige Aufgabe:

Aufgabe INF-09.06: Zeichenkette suchen, mit C-Strings

Schreiben Sie eine Headerdatei `suchen.h` sowie eine `.cpp` Datei `suchen.cpp` für eine C++ Funktion …

```
int zeichenkette_suchen(const char* text, const char* zkette);
```

… welche ermittelt, ob die Zeichenkette `zkette` in dem einzeiligen Text `text` (ggfs. mit Leerzeichen und/oder Satzzeichen) vorkommt.
Die maximale Textlänge von `text` und `zkette` betrage jeweils `20` vom Benutzer eingegebene Zeichen.
Die zu suchende Zeichenkette sei nicht leer, d.h. enthalte außer dem Nullterminator noch mindestens ein Zeichen.
Der (zu durchsuchende) Text darf aber leer sein.

Der Benutzer mache nur korrekte Eingaben, d.h. ihr Programm kann davon ausgehen, dass z.B. der eingegebene Text und die eingegebene Zeichenkette nicht länger als 20 eingegebene Zeichen sind.

Sollte die Zeichenkette nicht in dem Text vorkommen, so soll der Wert `-1` zurückgegeben werden. Anderenfalls wird die Startposition zurückgegeben, ab der das erste (linkeste) Vorkommen von `zkette` in `text` beginnt. Die Zählung der Positionen im Text beginne bei `0`.

Die Funktion darf keinerlei C-String Funktionen aus Libraries wie der `cstring` Library oder C++ `string` benutzen, sondern muss ausschließlich über den Array-Indexoperator auf die einzelnen Zeichen der beiden C-Strings zugreifen. Sollten Sie für ihre Programmierung die Länge eines C-Strings benötigen, so müssen sie auch dafür eine eigene Hilfsfunktion programmieren, die nur den Array-Indexoperator für den Zugriff auf die einzelnen Zeichen des C-Strings benutzt.

Beachten Sie, dass C-Strings immer mit dem Null-Terminator enden und dass durch Vergleich mit `'\0'` das Ende eines C-Strings erkannt werden kann.

Legen Sie im Projekt eine leere Headerdatei `catch.h` an und kopieren Sie den Inhalt der in Ilias liegenden gleichnamigen Datei in diese Headerdatei.

Legen Sie im Projekt eine Datei `unit_tests.cpp` an mit folgendem Inhalt:

```
// Datei: unit_tests.cpp

#include "catch.h"
#include "suchen.h"

TEST_CASE("Zeichenkette suchen, Text mit Laenge groesser 1, Zeichenkette mit Laenge
groesser 1") {
    REQUIRE(zeichenkette_suchen("abcdabcde", "cda") == 2);
    REQUIRE(zeichenkette_suchen("abcdabcde", "de") == 7);
    REQUIRE(zeichenkette_suchen("abcdabcde", "dex") == -1);
    REQUIRE(zeichenkette_suchen("abcdabcde", "xyz") == -1);
    REQUIRE(zeichenkette_suchen("abcdabcde", "abcdabcd") == 0);
    REQUIRE(zeichenkette_suchen("abcdabcde", "abcdabcdx") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge groesser 1, Zeichenkette mit Laenge
1") {
    REQUIRE(zeichenkette_suchen("abcdabcde", "a") == 0);
    REQUIRE(zeichenkette_suchen("abcdabcde", "c") == 2);
    REQUIRE(zeichenkette_suchen("abcdabcde", "e") == 8);
    REQUIRE(zeichenkette_suchen("abcdabcde", "x") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge 1") {
    REQUIRE(zeichenkette_suchen("a", "a") == 0);
```

```
    REQUIRE(zeichenkette_suchen("a", "c") == -1);
    REQUIRE(zeichenkette_suchen("a", "xy") == -1);
    REQUIRE(zeichenkette_suchen("a", "aa") == -1);
}

TEST_CASE("Zeichenkette suchen, leerer Text") {
    REQUIRE(zeichenkette_suchen("", "") == -1);
    REQUIRE(zeichenkette_suchen("", "a") == -1);
    REQUIRE(zeichenkette_suchen("", "abc") == -1);
}

TEST_CASE("Zeichenkette suchen, Text mit Laenge 20 Zeichen") {
    REQUIRE(zeichenkette_suchen("abcdefghij1234567890", "90") == 18);
    REQUIRE(zeichenkette_suchen("12345678901234567890", "90") == 8);
    REQUIRE(zeichenkette_suchen("abcdefghij1234567890", "9012") == -1);
}

// Vorgegebene Testläufe müssen selbst als Testcases programmiert werden ...
TEST_CASE("Vorgegebene Testlaeufe") {

}
```

Ergänzen Sie die Unit Tests in dieser Datei um die Testfälle, die in den unten angegebenen Testläufen vorgegeben sind.

Schreiben Sie ferner ein C++ Hauptprogramm, welches die Eingaben entgegennimmt, die Funktion aufruft und das Ergebnis ausgibt (siehe Testläufe). Ergänzen Sie dabei den folgenden Programmrahmen:

```
// Datei: main.cpp

#define CATCH_CONFIG_RUNNER
#include "catch.h"

#include <iostream>
using namespace std;

#include "suchen.h"

int main()
{
    if ( Catch::Session().run() ) {
        system("PAUSE"); return 1;
    }

    // Ihr Code ab hier ...


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

Testläufe: *(Benutzereingaben sind zur Verdeutlichung unterstrichen)*

```
===============================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: abcdefg
Bitte geben Sie die zu suchende Zeichenkette ein: bcd99
Die Zeichenkette 'bcd99' ist NICHT in dem Text 'abcdefg' enthalten.
Drücken Sie eine beliebige Taste . . .
```

```
===============================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: abcdefg
Bitte geben Sie die zu suchende Zeichenkette ein: efg
Die Zeichenkette 'efg' ist in dem Text 'abcdefg' enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
===============================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: abc
Bitte geben Sie die zu suchende Zeichenkette ein: abcde
Die Zeichenkette 'abcde' ist NICHT in dem Text 'abc' enthalten.
Drücken Sie eine beliebige Taste . . .
```

```
===============================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: 012 abc abc 89
Bitte geben Sie die zu suchende Zeichenkette ein: abc
Die Zeichenkette 'abc' ist in dem Text '012 abc abc 89' enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
===============================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: xy abc abcdefgh
Bitte geben Sie die zu suchende Zeichenkette ein: abcde
Die Zeichenkette 'abcde' ist in dem Text 'xy abc abcdefgh'
enthalten.
Sie startet ab Zeichen 7 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: xyz 123 456 abc
Bitte geben Sie die zu suchende Zeichenkette ein: 123 4
Die Zeichenkette '123 4' ist in dem Text 'xyz 123 456 abc'
enthalten.
Sie startet ab Zeichen 4 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```

```
================================================================================
All tests passed (27 assertions in 6 test cases)

Bitte geben Sie den Text ein: abc defg
Bitte geben Sie die zu suchende Zeichenkette ein: abc d
Die Zeichenkette 'abc d' ist in dem Text 'abc defg' enthalten.
Sie startet ab Zeichen 0 (bei Zaehlung ab 0).
Drücken Sie eine beliebige Taste . . .
```
