C/C++ Programmierung
Übungsblatt 4

Florian Oetke

Entwicklungsumgebung
Für dieses Übungsblatt müssen Sie Code in mehreren Dateien schreiben und mit CMake-Projekten und IDEs arbeiten. Hierzu finden Sie auf Panopto entsprechende Videos zur Verwendung von Microsoft Visual Studio und CLion. Weiterführende Informationen können Sie u.a. der offiziellen Dokumentation für CMake, Visual Studio und CLion entnehmen

1 Das Schlüsselwort static

Mit internal Linkage aus der letzten Vorlesung haben wir nun insgesamt drei unterschiedliche Stellen gesehen, an denen das Schlüsselwort `static` in C++ stehen kann. Geben Sie für jede davon ein kurzes Beispiel an und beschreiben Sie, was `static` in dieser Situation bewirkt.

2 Caesar-Verschlüsselung mit Hilfsfunktion

In dieser Aufgabe soll die Caesar-Verschlüsselung von Übungsblatt 2, wie in der Vorlesung gezeigt, auf mehrere Dateien aufgeteilt werden.

Zur Erinnerung: Bei dem Verfahren wird jeder Buchstabe in der Eingabe um einen bestimmten Offset zyklisch nach rechts verschoben. Das heißt, wenn wir z.B. `'Y'` um drei Zeichen verschieben, erhalten wir entsprechend `'B'`. Andere Zeichen (Zahlen, Leerzeichen, Umlaute, Sonderzeichen, ...) sollen unverändert zurückgegeben werden.

Im Handout finden Sie bereits eine `CMakeLists.txt` sowie einige, teilweise noch leere, Header- und Quellcode-Dateien.

Deklarieren Sie die Funktionen `encrypt` und `decrypt` in der entsprechenden Header-Datei und implementieren Sie diese in der zugehörigen CPP-Datei. Die Funktionen sollen sich im Namespace `caesar` befinden, neben dem Eingabe-String den Offset als Parameter erhalten, um den die Eingabe verschoben werden soll, und die kodierte bzw. dekodierte Ausgabe mit `return` zurückgeben.

Definieren Sie mindestens eine Hilfsfunktion in `caesar.cpp`, die nur innerhalb dieser Datei zugänglich ist, um die Implementierung einfacher zu gestalten bzw. um die Wiederholungen im Code zu reduzieren.

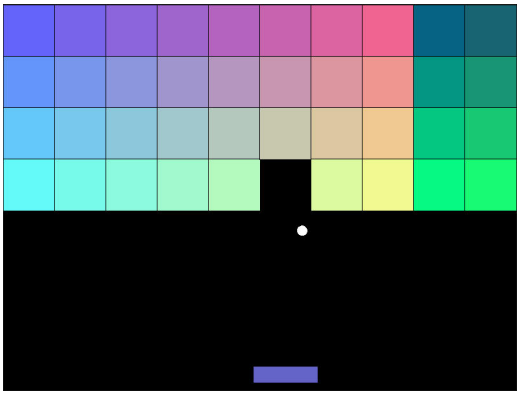
3 Fragen zum Arbeiten mit mehreren Dateien

Beantworten Sie die folgenden Fragen kurz in eigenen Worten:

1. Wozu dient das Schlüsselwort `inline` und was genau bewirkt es bei der Definition von Funktionen?
2. Was sind Include Guards und warum sind sie (oder `#pragma once`) bei Header-Dateien notwendig?
3. Welche drei Arten von Linkage haben wir kennengelernt und was bewirken sie jeweils? Geben Sie für jede ein kurzes Beispiel an, in dem eine Variable mit entsprechender Linkage definiert wird.

4 Breakout

In dieser Aufgabe wollen wir nun zum ersten Mal ein etwas komplexeres Programm schreiben, eine einfachere Version des Spielklassikers Breakout.



Hierzu finden Sie im Handout bereits ein vorbereitetes CMake-Projekt, bei dem nur noch die zwei unten beschriebenen Klassen `Ball` und `Box` fehlen.

Um die Aufgabe etwas einfacher zu gestalten, gibt es in dem Projekt die Datei `vector2.hpp`, welche einen einfachen zweidimensionalen Vektor implementiert, sowie die Datei `helper.hpp`, welche einige für Sie nützliche Hilfsfunktionen bereitstellt:

- `resolve_collision()` erhält als Parameter den Normalen-Vektor der Kollision (erster Parameter von `Ball::on_collision`) und eine Referenz auf die aktuelle Geschwindigkeit und passt die Geschwindigkeit entsprechend dem Kollisionsverhalten an.
- `create_rectangle_shape()` Konstruiert ein `sf::RectangleShape` mit der als Parameter übergebenen Größe und Farbe.
- `create_circle_shape()` Konstruiert ein `sf::CircleShape` mit dem als Parameter übergebenen Radius und Farbe.

Die mit den oben genannten Hilfsfunktionen erzeugten geometrischen Formen, können anschließend mit `shape.setPosition(x, y);` verschoben und z.B. wie folgt, mittels einer Referenz auf das `sf::RenderWindow`, gezeichnet werden:

```
1 void draw(sf::RenderWindow& window) {
2     window.draw(shape_);
3 }
```

Hinweis
Das Projekt lädt beim ersten Configure-Schritt von CMake die SFML-Bibliothek herunter, was einige Zeit in Anspruch nehmen kann. Darüber hinaus müssen unter Linux ggf. weitere Abhängigkeiten installiert werden.

Beim Kompilieren werden ihnen u.U. einige Warnungen von CMake und ihrem C++-Compiler auffallen, die von dieser Bibliothek verursacht werden. Diese Warnungen brauchen Sie beim Bearbeiten der Aufgabe nicht zu berücksichtigen.

Die beiden noch zu schreibenden Klassen sollten entsprechend der folgenden Definition aufgebaut sein:

- Box: Die Klasse modelliert sowohl den Schläger des Spielers, als auch die Boxen am oberen Bildschirmrand und sollte in einer Datei `box.hpp` definiert werden.

Box
- shape_ : sf::RectangleShape - position_ : Vector2 - size_ : Vector2
+ Box(position : Vector2, size : Vector2, color : sf::Color) + move(movement : Vector2) + draw(window : sf::RenderWindow&)const + position()const : Vector2 + size()const : Vector2

- Ball: Die Klasse modelliert den Spielball und sollte in einer Datei `ball.hpp` definiert werden.

Ball
- max_velocity_ : static const float = 600.f - shape_ : sf::CircleShape - position_ : Vector2 - velocity_ : Vector2 - radius_ : float
+ Ball(position : Vector2, velocity : Vector2, radius : float, color : sf::Color) + update(delta_time : float) + on_collision(normal : Vector2, penetration : Vector2 = Vector2{0,0}) + draw(window : sf::RenderWindow&)const + position()const : Vector2 + radius()const : float + speed_up(factor : float)

  - `update()`: Aktualisiert die aktuelle Position, indem sie um die Geschwindigkeit, multipliziert mit der verstrichenen Zeit (`delta_time`), erhöht wird.
  - `on_collision()`: Wird aufgerufen, wenn es zu einer Kollision mit einer Box gekommen ist und ändert als Reaktion die aktuelle Geschwindigkeit (`resolve_collision()` Hilfsfunktion). Da der Ball evtl. bereits in die Box eingedrungen ist, muss er außerdem ggf. zurückbewegt werden, indem die aktuelle Position um `penetration` reduziert wird.
  - `speed_up()`: Wird aufgerufen, wenn der Ball eine von den Boxen am oberen Rand getroffen hat, und kann optional implementiert werden, um die Geschwindigkeit des Balls zu erhöhen, indem sie mit dem übergebenen Faktor multipliziert wird. Die Geschwindigkeit sollte danach allerdings nicht größer als `max_velocity_` sein, d.h. die Länge des Vektors darf maximal so groß werden.