C/C++ Programmierung
Übungsblatt 3

Florian Oetke

1 Fehlerbehandlung in Dreieck-Klasse

Implementieren Sie eine Datenstruktur `Triangle`, die ein Dreieck basierend auf den Längen seiner drei Seiten modelliert.

Ihre Klasse sollte einen Konstruktor haben, der die Längen der drei Seiten als `float` entgegennimmt und in privaten Member-Variablen speichert, sowie Methoden zur Berechnung des Umfangs und der Fläche[1].

Im Konstruktor sollten Sie sicherstellen, dass die Parameter gültig sind. Prüfen Sie dazu mit `assert`, ob die Seitenlängen alle `> 0` sind. Außerdem sollten Sie kontrollieren, ob die übergebenen Seitenlängen ein gültiges Dreieck bilden[2] und andernfalls eine `std::invalid_argument` Exception werfen.

Testen Sie Ihre Lösung mit der folgenden `main` Funktion. Die Ausgabe sollte dabei so aussehen:

```
circumference: 9
area:          2.90474
Triangle validation works
```

```cpp
#include <iostream>
#include <cmath>
#include <cassert>
#include <exception>

// TODO

int main() {
    auto triangle = Triangle{2.f, 4.f, 3.f};
    std::cout << "circumference: " << triangle.circumference() << "\n";
    std::cout << "area:          " << triangle.area() << "\n";

    try {
        Triangle{2.f, 4.f, 1.f};
    } catch(const std::invalid_argument&) {
        std::cout << "Triangle validation works\n";
    }

    // Sollte das Programm abbrechen
    // Triangle{2.f, 4.f, 0.f};
}
```

[1] Die Fläche des Dreiecks können Sie mit dem *Satz des Heron* berechnen. Die Quadratwurzel, die hierzu notwendig ist, können Sie in C++ mit der Funktion `std::sqrt()` berechnen (aus `#include <cmath>`).

[2] Ein Dreieck mit Seitenlängen a, b und c ist genau dann gültig, wenn $a + b > c$, $a + c > b$ und $b + c > a$ gilt.

2 Vergleichsoperatoren

Ergänzen Sie die unten aufgeführte `Contact` Datenstruktur um die im Beispiel verwendeten Vergleichsoperatoren. Überlegen Sie ggf., wie Sie sich die Aufgabe möglichst einfach machen können. Die Operatoren sollten jeweils zuerst den Nachnamen und nur bei identischen Nachnamen auch die Vornamen vergleichen.

Bei einer korrekten Implementierung sollte für alle Vergleiche in der `main` `1` ausgegeben werden.

```cpp
#include <iostream>
#include <string>
#include <tuple>

struct Contact {
    std::string last_name;
    std::string first_name;

    // TODO
};

int main() {
    const auto mycroft  = Contact{"Holmes", "Mycroft"};
    const auto sherlock = Contact{"Holmes", "Sherlock"};
    const auto james    = Contact{"Moriarty", "James"};

    std::cout << "A: " << (sherlock == sherlock) << "\n";
    std::cout << "B: " << (sherlock != mycroft ) << "\n";
    std::cout << "C: " << (sherlock  > mycroft ) << "\n";
    std::cout << "D: " << (sherlock >= mycroft ) << "\n";
    std::cout << "E: " << (sherlock <= sherlock) << "\n";
    std::cout << "F: " << (sherlock >= sherlock) << "\n";
    std::cout << "G: " << (sherlock  < james   ) << "\n";
}
```

3 Vektor3 Operatoren

In dieser Aufgabe soll die `Vector3` Datenstruktur aus der Vorlesung erweitert werden, sodass die folgenden Operatoren unterstützt werden:

- `Vector3 + Vector3`
- `Vector3 * float`
- `float * Vector3`
- `Vector3 *= float`

Achten Sie dabei besonders auf den korrekten Rückgabetyp, ob die Funktion `const` sein sollte und ob Sie eine freie oder eine Member-Funktion verwenden müssen.

Die Ausgabe des fertigen Programms sollte so aussehen:

```
0.11, 0.22, 0
```

```cpp
#include <iostream>
#include <cmath>

struct Vector3 {
    float x;
    float y;
    float z = 0.f;

    // TODO
};


int main() {
    auto position   = Vector3{0, 0};
    auto velocity   = Vector3{1, 2};
    auto delta_time = 0.1f;

    position = position + velocity * delta_time;

    velocity *= 0.1f;

    position = position + delta_time * velocity;

    std::cout << position.x << ", "
              << position.y << ", "
              << position.z << "\n";
}
```

4 Ungültige Pointer-Zugriffe finden

Das folgende Programm ist ein Beispiel für unsachgemäße Verwendung von Pointern und enthält einige ungültige Zugriffe auf nicht mehr gültige Objekte.

1. An welchen Stellen wird oder könnte auf nicht mehr gültige Objekte zugegriffen werden und wo wurden diese Objekte jeweils zerstört?
2. Zeichnen Sie ein Speicherabbild vom Zustand am Anfang des letzten Aufrufs von `pool.area()` aus Z.52 und markieren Sie die bereits zerstörten Objekte.

```
 1 #include <iostream>
 2 #include <string>
 3
 4 struct Rect {
 5     float width;
 6     float height;
 7
 8     float area()const {
 9         return width * height;
10     }
11 };
12 struct Circle {
13     float radius;
14
15     float area()const {
16         return radius*radius * 3.141f;
17     }
18 };
19 struct Pool {
20     Rect*   base_area;
21     Circle* basin;
22
23     float area()const {
24         return base_area->area() - basin->area();
25     }
26 };
27
28 const float& max(const float& a, const float& b) {
29     return a >= b ? a : b;
30 }
31 const float& min(const float a, const float& b) {
32     return a <= b ? a : b;
33 }
34 Circle& default_circle() {
35     static auto circle = Circle{1};
36     return circle;
37 }
38
39 int main() {
40     auto rect = Rect{5, 5};
41     auto pool = Pool{&rect, &default_circle()};
42     std::cout << "Pool area: " << pool.area() << "\n";
43
44     pool.base_area->width  = max(pool.base_area->width,  10);
45     pool.base_area->height = min(pool.base_area->height, 3);
46
47     if(pool.basin->radius < 2.f) {
48         auto larger_circle = Circle{2.5f};
49         pool.basin = &larger_circle;
50     }
51
52     std::cout << "New pool area: " << pool.area() << "\n";
53 }
```

5 Implementieren einer Klasse anhand eines UML-Diagramms

Implementieren Sie die Datenstruktur, welche durch das folgende UML-Diagramm beschrieben wird, sodass die unten angegebene main-Funktion korrekt funktioniert.

Einige Stellen im Diagramm, an denen ein `const` stehen könnte, sind mit `???` gekennzeichnet. Entscheiden Sie bei der Implementierung, ob Sie an den entsprechenden Stellen ein `const` verwenden und geben Sie in einem Kommentar stichpunktartig an, warum `const` in diesem Fall notwendig/sinnvoll oder nicht sinnvoll/möglich ist.

Customer
- first_name_ : ??? std::string - last_name_ : ??? std::string - account_balance_ : ??? int = 0
+ Customer(first_name : ??? std::string&, last_name : ??? std::string&) + first_name()??? : std::string + last_name()??? : std::string + account_balance()??? : int + deposit(amount : int)??? + withdraw(amount : int)??? + print()???

```cpp
#include <iostream>
#include <string>

// Customer
// ...

int main() {
    auto customer = Customer{"John", "Dow"};
    customer.deposit(15);
    customer.withdraw(5);
    customer.print();

    customer = Customer{"Jon", "Dow"};
    customer.withdraw(5);
    customer.print();
}
```

Die Ausgabe des Programms sollte so aussehen:

```
Dow, John
  Balance: 10
Dow, Jon
  Balance: -5
```