Praktikumsaufgabe 3 zur Vorlesung Grundlagen der Programmierung 2

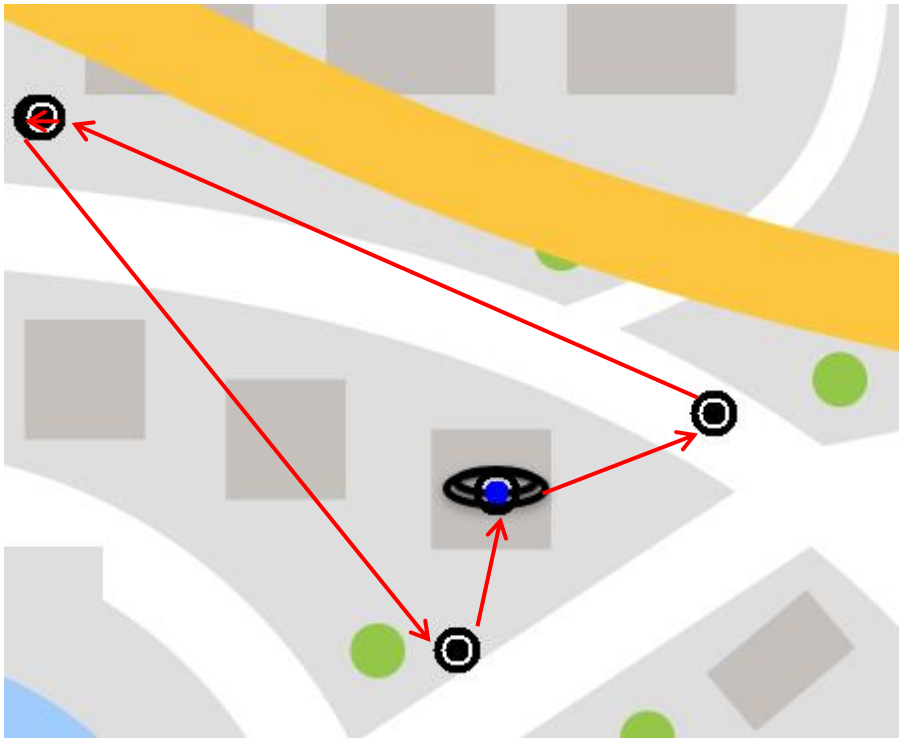
Prof. Dr. Robert Gold

Technische Hochschule Ingolstadt
Sommersemester 2025

Bevor Sie mit der dritten Aufgabe beginnen, sollten Sie die Dateien *vertical.h* und *vertical.cpp* der zweiten Aufgabe am besten unter anderen Namen z.B. *pa2_vertical.h*, *pa2_vertical.cpp* kopieren. Die Dateien *ballistic.h*, *ballistic.cpp*, *ufo.h* und *ufo.cpp* werden in dieser Praktikumsaufgabe nicht verändert. Für den Unit-Test der dritten Aufgabe bitte die neue Datei *pa3_utest.cpp* verwenden.

In dieser Aufgabe sollen vom Startpunkt (0.0, 0.0, 0.0) mehrere Ziele angeflogen werden können. Danach soll das Ufo zum Ausgangspunkt (0.0, 0.0, 0.0) wieder zurückkehren.

Da es mehrere Ziele gibt, können diese auch in unterschiedlichen Reihenfolgen angeflogen werden. Wenn wir n Ziele haben, gibt es $n!$ verschiedene Routen. Die Abbildung zeigt eine Beispielroute zum Anfliegen der Ziele (55.0, 20.0), (-116.5, 95.0), (-10.0, -40.0), (-115.0, 95.0) in der Reihenfolge (55.0, 20.0), (-115.0, 95.0), (-116.5, 95.0), (-10.0, -40.0). Da sich die Ziele immer auf der Höhe 0.0 befinden, sind die z-Koordinaten weggelassen.



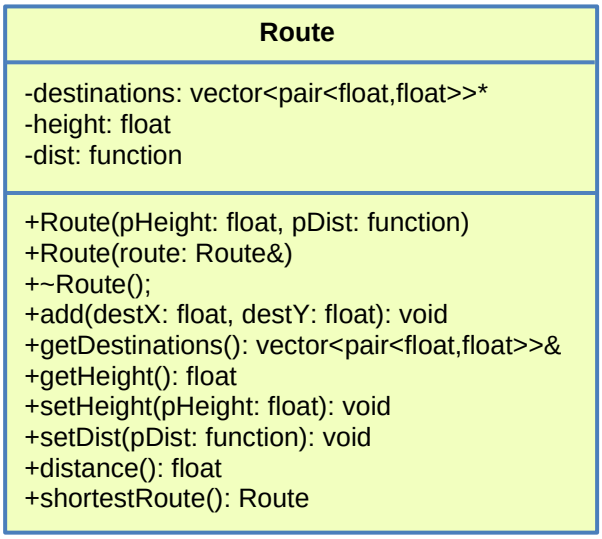
Zur Vereinfachung der Praktikumsaufgabe fliegen wir Routen nur mit Vertical-Ufos.

a) Um die zu fliegende Strecke eines einzelnen Fluges zu berechnen, soll die Klasse `Vertical` durch eine public Methode

```
static float distance(const float x1, const float y1, const float x2,
                      const float y2, const float h);
```

ergänzt werden. Der Rückgabewert ist die Summe der zu fliegenden Distanzen der drei Flugabschnitte, wenn das Ufo von (x1, y1, 0.0) in Flughöhe h nach (x2, y2, 0.0) fliegen soll. Wenn Start und Ziel identisch sind, soll keine Sonderbehandlung durchgeführt werden.

b) Zur Darstellung von Routen soll eine Klasse Route erstellt werden.



Die Klasse hat drei private Attribute

- `destinations`: Pointer auf eine Liste der Ziele. Jedes Ziel ist ein *x*-, *y*-Paar. Die *z*-Koordinate jedes Ziels ist 0.0.
- `height`: Flughöhe
- `dist`: Funktion mit fünf float-Parametern und einem float-Rückgabewert zur Berechnung der zu fliegenden Strecke eines einzelnen Fluges. Verwenden Sie das Klassentemplate `function`.
  Für einen Flug mit einem `Vertical`-Ufo wird die Methode `distance` aus der Klasse `Vertical` verwendet. Aber natürlich sollen auch andere syntaktisch passende Funktionen eingesetzt werden können.

Neben einem Konstruktor, einem Copy-Konstruktor und einem Destruktor gibt es weitere Methoden (siehe Abbildung oben),

Bemerkungen zu den Methoden:

- `add`: Fügt ein Ziel hinten an `destinations` an. Die Parameter sollen const sein.
- `getDestinations`: Getter für `destinations`. Der Rückgabewert soll eine const Referenz sein. Die Methode soll als const deklariert sein.
- `getHeight`, `setHeight`: Getter und Setter für `height`. Der Getter soll als const deklariert sein. Der Parameter des Setters soll const sein.
- `setDist`: Setter für `dist`.

- `distance`: Gibt die gesamte zu fliegende Strecke zurück, wenn von (0.0, 0.0, 0.0) zum ersten Ziel der Liste, danach zum zweiten Ziel usw. und zum Schluss wieder nach (0.0, 0.0, 0.0) zurückgeflogen wird. Verwenden Sie die Funktion `dist`. Wenn die Zieleliste leer ist, soll 0.0 herauskommen. Die Methode soll als const deklariert sein.

  Verwechseln Sie diese Methode nicht mit der gleichnamigen Methode in der Klasse `Vertical`.

Die noch fehlende Methode `shortestRoute` wird in der folgenden Teilaufgabe programmiert.

c) In der letzten Teilaufgabe soll eine kürzeste Flugroute gefunden werden, die startend von (0.0, 0.0, 0.0) alle Ziele in `destinations` anfliegt und wieder nach (0.0, 0.0, 0.0) zurückkehrt. Die Methode `shortestRoute` ordnet die Liste `destinations` entsprechend um.

Die Aufgabe soll durch Ausprobieren aller möglicher Flugrouten gelöst werden. Dazu sollen alle Permutationen betrachtet werden und diejenige, die die kürzeste Flugstrecke hat, bestimmt werden.

Beispiel:

`destinations` = { (10.0, 5.0), (10.0, 0.0), (0.0, 10.0) }
`height` = 3.0
`dist` = die Funktion `distance` der Klasse `Vertical`

Der Flug in der obigen Reihenfolge ist
(0.0, 0.0, 0.0) → (10.0, 5.0, 0.0) → (10.0, 0.0, 0.0) → (0.0, 10.0, 0.0) → (0.0, 0.0, 0.0).

`distance()` gibt 64.3225 zurück.

Nach dem Umordnen der Ziele in (0.0, 10.0), (10.0, 5.0), (10.0, 0.0) ist der Flug
(0.0, 0.0, 0.0) → (0.0, 10.0, 0.0) → (10.0, 5.0, 0.0) → (10.0, 0.0, 0.0) → (0.0, 0.0, 0.0).

`distance()` gibt 60.1803 zurück. Die Flugstrecke ist also kürzer.

Um alle Permutationen von `destinations` zu durchlaufen, kann die folgende Schleife verwendet werden:

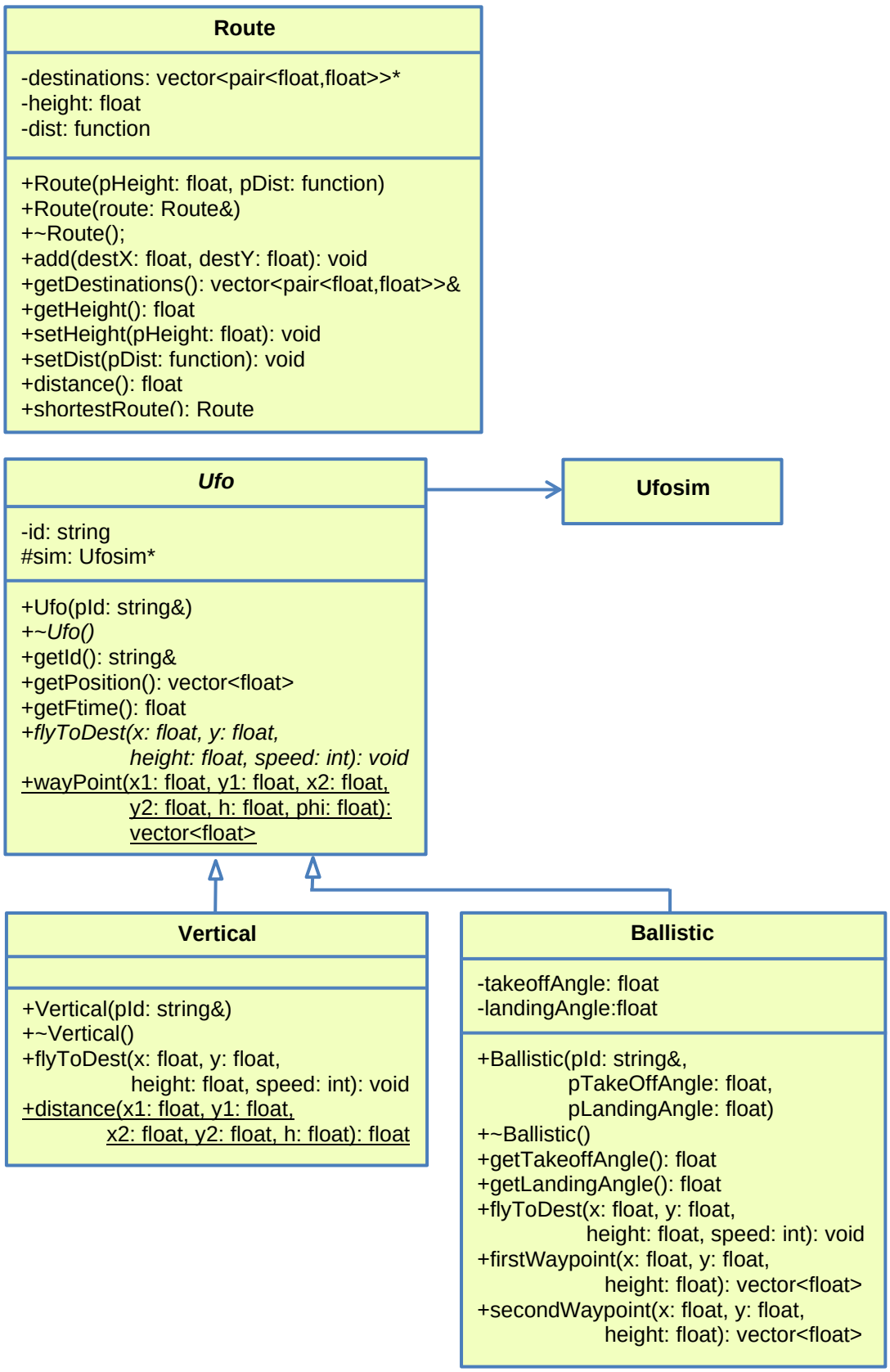
```
while (std::next_permutation(destinations->begin(), destinations->end()))
{
    …
}
```

Im Aufgabenblatt haben wir ein Beispielprogramm dazu erstellt. Damit die Permutationen vollständig durchlaufen werden, muss der Vektor vor der obigen Schleife sortiert werden:

```
std::sort(destinations->begin(), destinations->end());
```

Die Methode `shortestRoute` soll eine Kopie des `Route`-Objekts zurückgeben, dessen `destinations` aber eine kürzeste Flugroute ist.

Insgesamt haben wir jetzt:



d) Die Abgabe besteht aus den Dateien *ballistic.h*, *ballistic.cpp*, *vertical.h*, *vertical.cpp*, *ufo.h*, *ufo.cpp*, *route.h*, *route.cpp*.

Alle Parameter, alle Referenzrückgaben und alle Methoden sollten, soweit möglich, `const` sein.

Bitte überprüfen Sie vor der Abgabe, ob sich das Projekt fehlerfrei erstellen lässt:

```
g++ -Wall -std=c++20 -I"C:\Program Files\boost_1_87_0" ballistic.cpp
vertical.cpp ufo.cpp route.cpp ufosim.cpp pa3_utest.cpp -o pa3.exe
```

Alle Unit-Tests in *pa3_utest.cpp* müssen fehlerfrei laufen. Starten Sie dazu das Programm mit dem Befehl

```
pa3
```

Das Ergebnis muss

```
*** No errors detected
```

sein.