

Praktikumsaufgabe 2 zur Vorlesung Grundlagen der Programmierung 2

Prof. Dr. Robert Gold

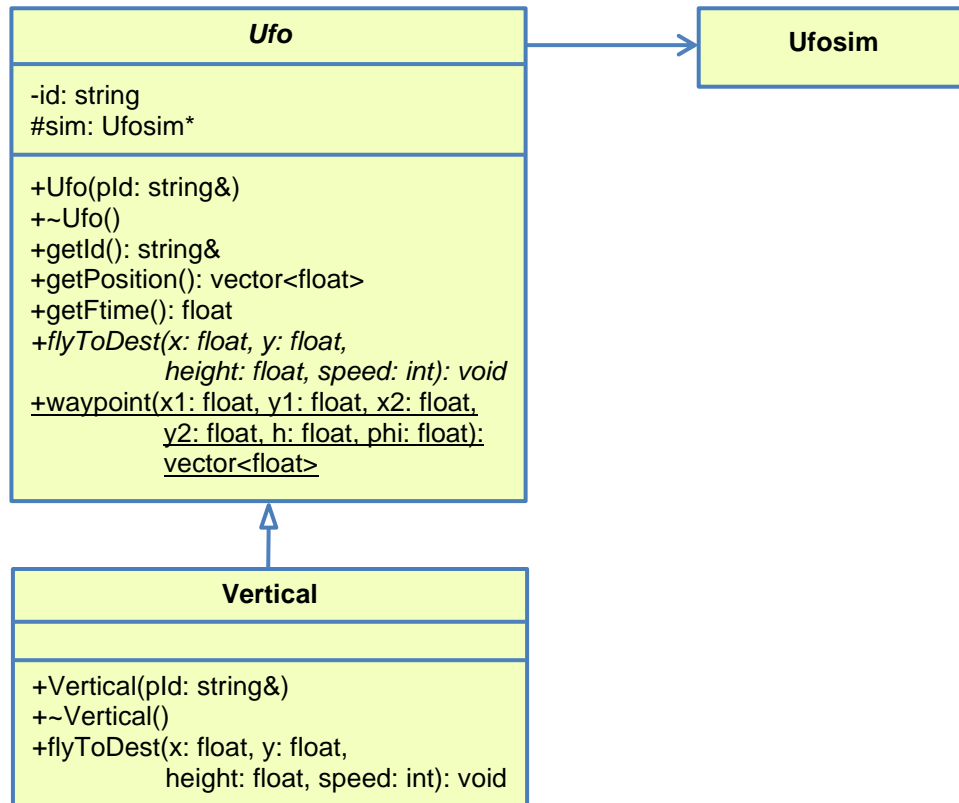
Technische Hochschule Ingolstadt
Sommersemester 2025

Bevor Sie mit der zweiten Aufgabe beginnen, sollten Sie die Dateien *vertical.h* und *vertical.cpp* der ersten Aufgabe am besten unter anderen Namen z.B. *pa1_vertical.h*, *pa1_vertical.cpp* kopieren. Für den Unit-Test der zweiten Aufgabe bitte die neue Datei *pa2_utest.cpp* verwenden.

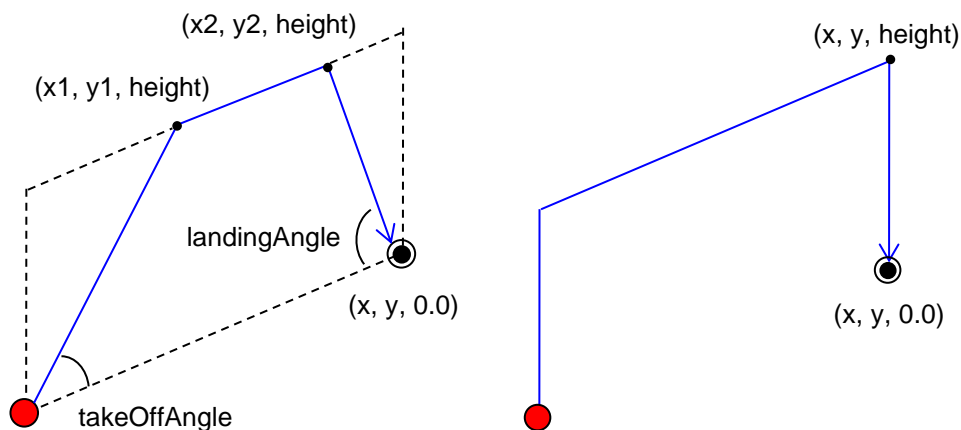
- a) Zuerst soll eine abstrakte Klasse `Ufo` in Dateien *ufo.h*, *ufo.cpp* erstellt werden. In diese Klasse werden alle Attribute und Methoden der Klasse `Vertical` verschoben. Das Attribut `type` und den Getter dafür können Sie löschen, sie werden nicht mehr gebraucht. Die Methode `flyToDest` soll rein virtuell sein. Die Implementierung dieser Methode wird gelöscht. Der Destruktor soll virtuell sein.
- b) Von der Klasse `Ufo` wird die Klasse `Vertical` abgeleitet. Wir brauchen in der Klasse `Vertical` lediglich einen Konstruktor, einen Destruktor und die Methode `flyToDest`. Da diese Methode die gleichnamige Methode in der Basisklasse überschreibt, soll das Schlüsselwort `override` angegeben werden.

Die Implementierung der Methode `flyToDest` ist unverändert und kann aus *pa1_vertical.cpp* kopiert werden. Sie werden aber feststellen, dass das Attribut `sim` der Basisklasse `Ufo` in `Vertical` nicht sichtbar ist, aber in `flyToDest` benötigt wird. Wie kann man dieses Problem lösen?

Bisher haben wir:



- c) Jetzt wollen wir in einer Klasse **Ballistic** ein Fluggerät modellieren, dessen Flug näherungsweise ein ballistischer Flug ist. Eine Kanonenkugel beschreibt beispielsweise eine solche Flugbahn. Bei uns ist der Flug durch einen Winkel `takeOffAngle`, eine Flughöhe `height` und einen Winkel `landingAngle` gekennzeichnet. Die folgende Abbildung zeigt links die Flugbahn von Fluggeräten der Klasse **Ballistic** und rechts zum Vergleich die Flugbahn der Klasse **Vertical**.



Die Klasse **Ballistic** braucht zwei private float-Attribute `takeOffAngle`, `landingAngle` und zwei public Getter für diese Attribute. Die Attribute werden im Konstruktor initialisiert. Dabei ist darauf zu achten, dass gilt:

$$0 < \text{takeOffAngle} \leq 90 \text{ Grad}, 0 < \text{landingAngle} \leq 90 \text{ Grad}.$$

Wenn eine Bedingung nicht erfüllt ist, soll der Winkel auf den Default-Wert 45 Grad gesetzt werden.

Erstellen Sie auch einen Destruktor, auch wenn darin nichts zu tun ist

- d) Erstellen Sie in der Klasse `Ballistic` zwei public Methoden

```
std::vector<float> firstWaypoint(const float x, const float y,  
                                const float height) const;  
  
std::vector<float> secondWaypoint(const float x, const float y,  
                                  const float height) const;
```

Die erste Methode soll den Punkt x_1, y_1 berechnen und als Vektor von zwei Zahlen zurückgeben. Die zweite Methode gibt entsprechend den Punkt x_2, y_2 zurück. Die Bezeichnungen sind wie in der Skizze auf der linken Seite der vorangegangenen Abbildung.

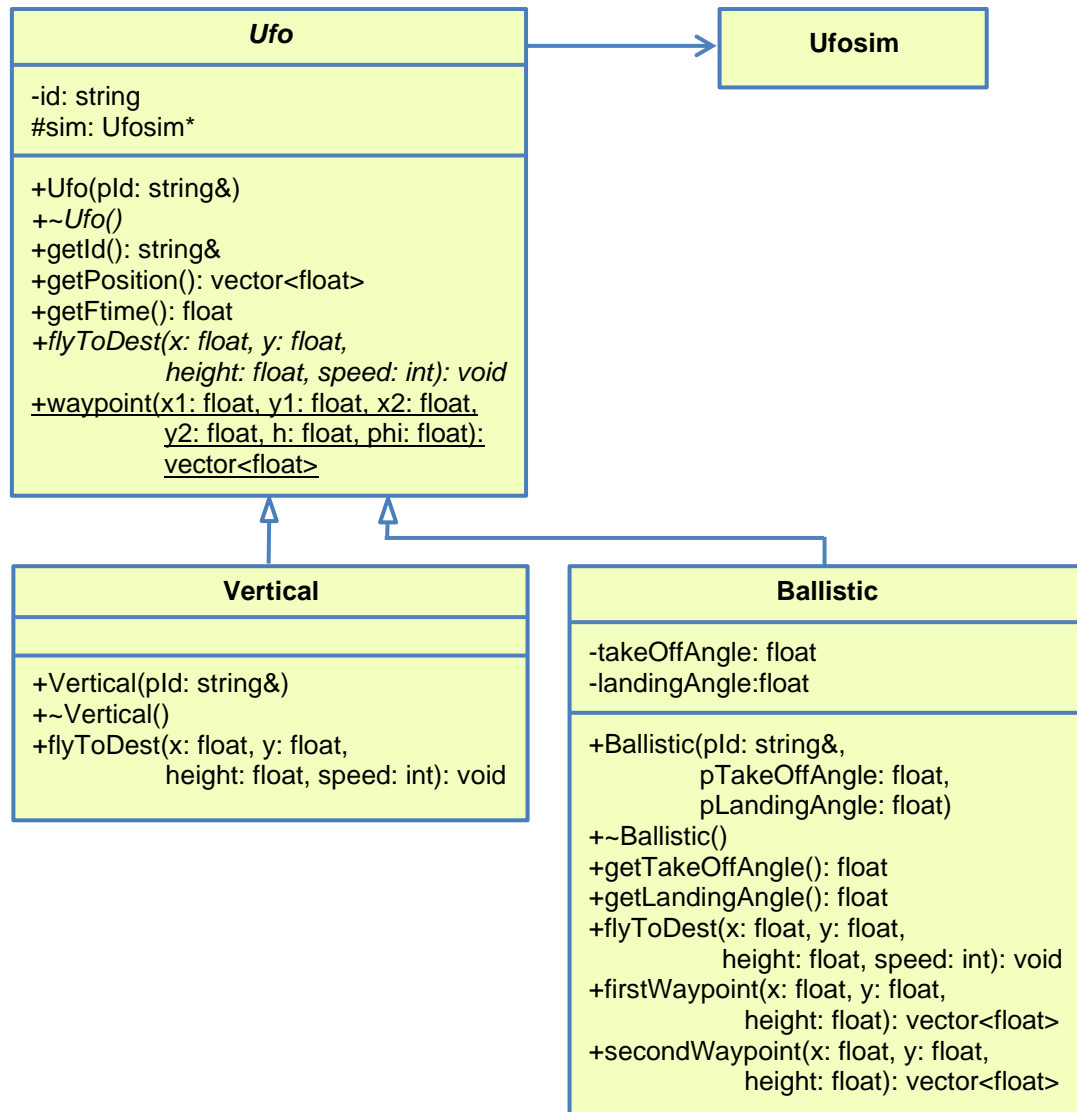
Bitte beachten Sie, dass ein Flug an der aktuellen Position des Ufos beginnt. Diese muss nicht $(0.0, 0.0, 0.0)$ sein!

Verwenden Sie für diese beiden Methoden unbedingt die Methode `waypoint`. Dann bestehen `firstWaypoint` und `secondWaypoint` nur aus jeweils einer Zeile!

- e) Die Klasse `Ballistic` hat ebenfalls eine Methode `flyToDest`, die die Drohne wie oben beschrieben zum Ziel $(x, y, 0.0)$ fliegt. Dazu wird die Methode `flyTo` der Klasse `UfoSim` mit dem Objekt `sim` verwendet. Wieder gibt es drei Flugabschnitte und wieder wird `flyTo` dreimal aufgerufen. Das Fluggerät soll dieses Mal aber nach den ersten beiden Abschnitten nicht stoppen, sondern mit gleicher Geschwindigkeit weiterfliegen.

Der erste Aufruf lautet z.B. `sim->flyTo(x1, y1, height, speed, speed)`. Die Methoden `firstWaypoint` und `secondWaypoint` sollen dabei verwendet werden.

Insgesamt haben wir jetzt:



- f) Die Abgabe besteht aus den Dateien *ballistic.h*, *ballistic.cpp*, *vertical.h*, *vertical.cpp*, *ufo.h*, *ufo.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 ufosim.cpp pa2_utest.cpp -o pa2.exe
```

Alle Unit-Tests in *pa2_utest.cpp* müssen fehlerfrei laufen. Starten Sie dazu das Programm mit dem Befehl

```
pa2
```

Das Ergebnis muss

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

sein.

- g) Zusätzlich soll ein Hauptprogramm in einer Datei *pa2_main.cpp* erstellt werden, mit dem der Flug eines Ufos beobachtet und überprüft werden kann. Diese Datei darf aber nicht abgegeben werden. Sonst würde der Unit-Test fehlschlagen, weil es die Funktion `main` zweimal gibt!

Im Hauptprogramm soll Folgendes implementiert werden:

- `Vertical`-Objekt `vert` mit `new` auf dem Heap angelegt (ID beliebig)
- `Ballistic`-Objekt `ball` mit `new` auf dem Heap angelegt (ID und Winkel beliebig)
- Vektor `ufos` von Pointern auf `Ufo` bestehend aus den Pointern auf `vert` und `ball`
- Konsoleingabe einer Zahl 0 oder 1
- Flug mit dem Ufo in `ufos` mit dem eingegebenen Index (Ziel, Höhe, Geschwindigkeit beliebig)

Das Hauptprogramm soll zusammen mit den Klassen `Ufo`, `Vertical` und `Ballistic` im Praktikum vorgeführt werden.