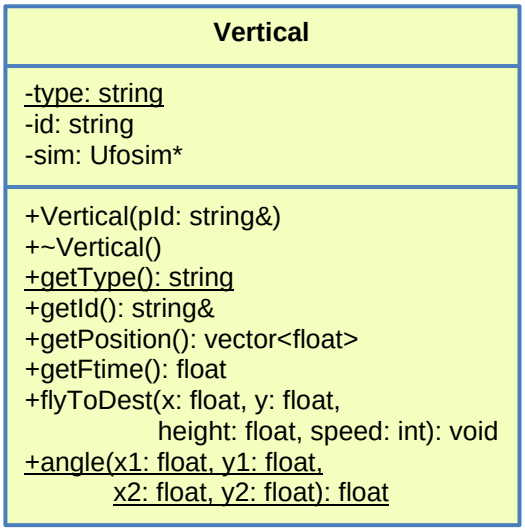
Grundlagen der Programmierung 2: Praktikumsaufgabe 1

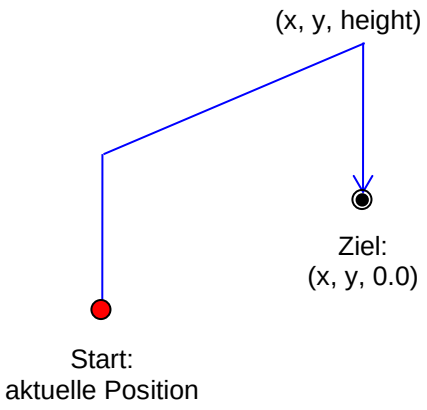
Prof. Dr. Robert Gold

Technische Hochschule Ingolstadt
Sommersemester 2024

a) Erstellen Sie eine Klasse `Vertical` mit den in der folgenden Abbildung dargestellten Attributen und Methoden in zwei Dateien *vertical.h* und *vertical.cpp*.



Ein Objekt dieser Klasse repräsentiert eine Drohne, die senkrecht von der aktuellen Position bis zur Flughöhe `height` aufsteigt, auf dieser Flughöhe zum Punkt über dem Zielpunkt fliegt und dann senkrecht am Ziel (x, y, 0.0) landet.



Alle Parameter sollen `const` sein. Methoden, die die Attribute nicht ändern, sollen ebenfalls `const` sein. Das Attribut `type` und die Methoden `getType` und `angle` sind statisch. Statische Elemente werden im Klassendiagramm durch Unterstreichung gekennzeichnet. Das Attribut `type` soll `const` sein.

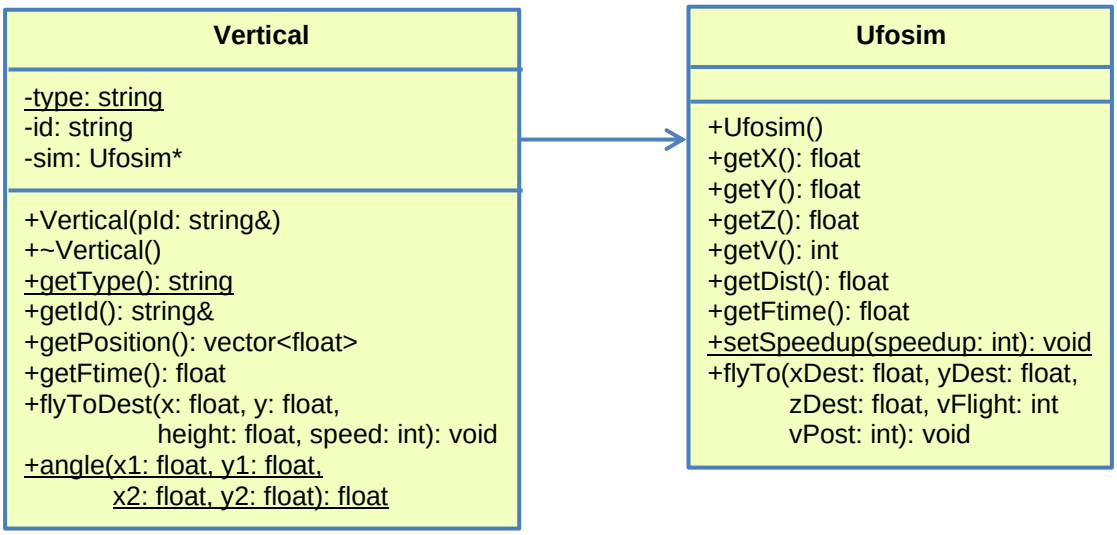
Das Attribut `sim` noch nicht einfügen. Das kommt erst später.

Erläuterung der Methoden:

- Der Konstruktor verwendet den Parameter zur Initialisierung des Attributs `id`. Das Attribut `type` soll außerhalb des Konstruktors mit dem String "vertical" initialisiert werden.
- Der Destruktor ist erstmal leer. Er wird später ergänzt.
- Die Methoden `getType` und `getId` sind Getter der gleichnamigen Attribute.
- Die Methode `getPosition` gibt einen Vektor mit drei Elementen (0.0, 0.0, 0.0) zurück. Das werden wir später noch ändern.
- Die Methode `getFtime` gibt 0.0 zurück. Auch das werden wir später noch ändern.
- Die Methode `flyToDest` macht erstmal gar nichts. Diese Methode wird später ergänzt.
- Die Methode `angle` gibt 0.0 zurück. Auch diese Methode wird später ausprogrammiert.

Für diese Klasse gibt es einen Unit-Test in *pa1_utest.cpp*. Der Unit-Test kann bereits ausgeführt werden, aber da wir noch nicht fertig sind, wird nur der erste Testfall `initially` fehlerfrei laufen.

b) Gegeben ist eine einfache Simulationsklasse `Ufosim`, die den Flug von verschiedenen Fluggeräten, genannt Ufo, simuliert. Die folgende Abbildung zeigt die öffentlichen Methoden dieser Klasse. Die Dateien dazu sind *ufosim.h* und *ufosim.cpp*.



Wir ergänzen nun die Klasse `Vertical` um das Attribut `sim`. Dieses Attribut ist ein Pointer auf ein `Ufosim`-Objekt. Außerdem werden die Methoden der Klasse `Vertical` geändert bzw. ergänzt.

Erweiterung bzw. Ergänzung der Methoden:

- Im Konstruktor soll die Methode `setSpeedup` der Klasse `Ufosim` mit dem Parameter 4 aufgerufen werden. Da diese Methode statisch ist, lautet der Aufruf

```
Ufosim::setSpeedup(4);
```

Außerdem soll ein neues `Ufosim`-Objekt angelegt werden:

```
sim = new Ufosim();
```

- Im Destruktor wird der Speicherplatz für `sim` freigegeben:

```
delete sim;
```

- Die Methode `getPosition` ruft die Methoden `getX`, `getY`, `getZ` der Klasse `Ufosim` mit dem Objekt `sim` auf und gibt die Werte als Vektor mit drei Elementen zurück. Die Aufrufe von `getX`, `getY`, `getZ` lauten `sim->getX()`, `sim->getY()`, `sim->getZ()`
- Die Methode `getFtime` ruft die gleichnamige Methode der Klasse `Ufosim` mit dem Objekt `sim` auf und gibt den Wert zurück. Der Aufruf lautet `sim->getFtime()`.
- Die Methode `flyToDest` fliegt die Drohne wie oben beschrieben zum Ziel (x, y, 0.0). Dazu wird die Methode `flyTo` der Klasse `Ufosim` mit dem Objekt `sim` verwendet.

  Die Methode `flyTo` der Klasse `Ufosim` realisiert den Flug von der aktuellen Position der Drohne in gerader Linie zum Punkt `xDest`, `yDest`, `zDest`. Die Fluggeschwindigkeit ist `vFlight`. Nach dem Erreichen des Ziels fliegt die Drohne mit der Geschwindigkeit `vPost`. Wenn beispielsweise `vPost` gleich 0 ist, bleibt die Drohne stehen. Im Falle `vPost` gleich `vFlight` fliegt die Drohne mit gleicher Geschwindigkeit weiter.

  Um das Ziel (x, y, 0.0) zu erreichen, muss `flyTo` dreimal aufgerufen werden, ein Aufruf für jeden der drei oben beschriebenen Flugabschnitte.

  Die Drohne soll also von der aktuellen Position nach oben auf die Höhe `height` fliegen, danach nach (x, y, height) und zum Schluss nach (x, y, 0.0). Die Fluggeschwindigkeit `speed` wird bei den Aufrufen einfach als Parameter `vFlight` eingesetzt. Nach jedem Flugabschnitt soll die Drohne stehenbleiben. Deshalb wird `vPost` gleich 0 gewählt.

  Der erste Aufruf lautet z.B.

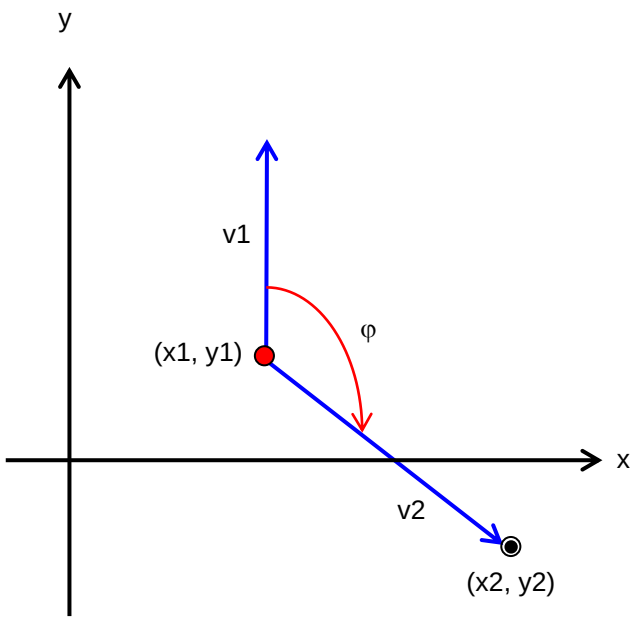
```
sim->flyTo(sim->getX(), sim->getY(), height, speed, 0);
```

- Die Methode `angle` programmieren wir in der nächsten Teilaufgabe aus.

Bis auf den letzten Unit-Test `angle` sollten jetzt alle Unit-Tests in *pa1_utest.cpp* fehlerfrei laufen!

c) Jetzt wird die noch fehlende Methode `angle` fertiggestellt.

Die Methode `angle` berechnet den Winkel φ zwischen den Vektoren v1 und v2 (siehe folgende Abbildung), wobei der Vektor v1 parallel zur y-Achse ist. Der Winkel wird in Grad im Uhrzeigersinn von der Nordrichtung (positive y-Achse) gemessen. In der Abbildung beträgt der Winkel φ ca. 135 Grad. Der zulässige Bereich ist [0.0, 360.0[. Wenn die Parameter x1, y1 mit x2, y2 übereinstimmen, soll das Ergebnis 0.0 sein. Achten Sie beim Dividieren darauf, dass der Nenner nicht Null ist. Der Winkel kann als Flugrichtung interpretiert werden.



d) Die Unit-Tests in *pa1_utest.cpp* sollten nun fehlerfrei laufen. Sie können aber auch ein Hauptprogramm dazu schreiben, um Flüge von Vertical-Ufos zu beobachten. Beispiel eines Hauptprogramms:

```
#include <iostream>
#include "vertical.h"

int main()
{
    Vertical vert("r2d2");
    // fly from (0.0, 0.0, 0.0) to (5.0, -1.5, 0.0)
    // at altitude 4.0 with 10 km/h
    vert.flyToDest(5.0, -1.5, 4.0, 10);
    // fly from (5.0, -1.5, 0.0) to (-3.0, 0.0, 0.0)
    // at altitude 8.0 with 5 km/h
    vert.flyToDest(-3.0, 0.0, 8.0, 5);

    return 0;
}
```

e) Die Abgabe besteht aus den Dateien *vertical.h* und *vertical.cpp*.

Alle Parameter, alle Referenzrückgaben und alle Methoden sollten, soweit möglich, `const` sein. In *vertical.h* und in *vertical.cpp* sollte das Schlüsselwort `const` deshalb jeweils 15 mal vorkommen.

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

```
g++ -std=c++20 -I"C:\Program Files\boost_1_84_0" vertical.cpp ufosim.cpp
pa1_utest.cpp -o pa1.exe
```

Alle Unit-Tests in *pa1_utest.cpp* müssen fehlerfrei laufen. Starten Sie dazu das Programm mit dem Befehl

```
pa1
```

Das Ergebnis muss

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

sein.