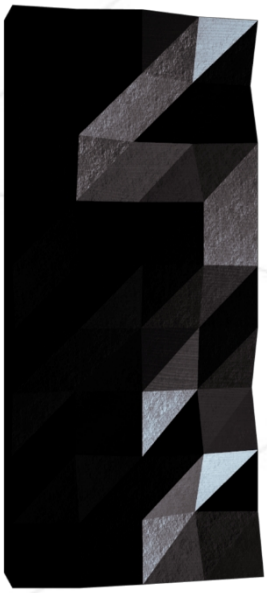

# C++ - Modul 03

## Vererbung

*Zusammenfassung:*
*Dieses Dokument enthält die Übungen des Moduls 03 aus den C++ Modulen.*

*Fassung: 7*

# Inhalt

# Kapitel I

# Einleitung

*C++ ist eine allgemeine Programmiersprache, die von Bjarne Stroustrup als Weiterentwicklung der Programmiersprache C oder "C with Classes" (Quelle: Wikipedia) entwickelt wurde.*

Das Ziel dieser Module ist es, Sie in die **objektorientierte Programmierung** einzuführen. Dies wird der Ausgangspunkt für Ihre C++-Reise sein. Viele Sprachen werden empfohlen, um OOP zu lernen. Wir haben uns für C++ entschieden, da es von Ihrem alten Freund C abgeleitet ist. Da es sich um eine komplexe Sprache handelt, und um die Dinge einfach zu halten, wird Ihr Code dem Standard C++98 entsprechen.

Wir sind uns bewusst, dass modernes C++ in vielerlei Hinsicht anders ist. Wenn Sie also ein kompetenter C++-Entwickler werden wollen, liegt es an Ihnen, nach den 42 Common Core weiterzugehen!

# Kapitel II

# Allgemeine Vorschriften

**Kompilieren**

- Kompilieren Sie Ihren Code mit C++ und den Flags -Wall -Wextra -Werror
- Ihr Code sollte sich trotzdem kompilieren lassen, wenn Sie das Flag -std=c++98 hinzufügen

**Formatierungs- und Benennungskonventionen**

- Die Übungsverzeichnisse werden folgendermaßen benannt: ex00, ex01, ... , exn
- Benennen Sie Ihre Dateien, Klassen, Funktionen, Mitgliedsfunktionen und Attribute wie in den Richtlinien gefordert.
- Schreiben Sie Klassennamen im Format **UpperCamelCase**. Dateien, die Klassencode enthalten, werden immer nach dem Klassennamen benannt. Zum Beispiel: Klassenname.hpp/Klassenname.h, Klassenname.cpp, oder Klassenname.tpp. Wenn Sie also eine Header-Datei haben, die die Definition einer Klasse "BrickWall" enthält, die für eine Ziegelmauer steht, lautet ihr Name BrickWall.hpp.
- Wenn nicht anders angegeben, müssen alle Ausgabemeldungen mit einem Zeilenumbruch abgeschlossen und auf der Standardausgabe ausgegeben werden.
- *Auf Wiedersehen Norminette!* In den C++-Modulen ist kein Kodierungsstil vorgeschrieben. Sie können Ihrem Lieblingsstil folgen. Aber denken Sie daran, dass ein Code, den Ihre Mitbewerber nicht verstehen, auch nicht benotet werden kann. Geben Sie Ihr Bestes, um einen sauberen und lesbaren Code zu schreiben.

**Erlaubt/Verboten**

Sie programmieren nicht mehr in C. Zeit für C++! Deshalb:

- Sie dürfen fast alles aus der Standardbibliothek verwenden. Anstatt sich also an das zu halten, was Sie bereits kennen, wäre es klug, so viel wie möglich die C++-ähnlichen Versionen der C-Funktionen zu verwenden, an die Sie gewöhnt sind.
- Sie können jedoch keine andere externe Bibliothek verwenden. Das bedeutet, dass C++11 (und abgeleitete Formen) und Boost-Bibliotheken verboten sind. Die

folgenden Funktionen sind ebenfalls verboten: *printf(), *alloc() und free(). Wenn Sie sie verwenden, wird Ihre Note 0 sein und das war's.

- Beachten Sie, dass, sofern nicht ausdrücklich anders angegeben, der using-Namensraum <ns_name> und
  Freundschaftswörter sind verboten. Andernfalls wird Ihre Note -42 sein.
- **Du darfst die STL nur in den Modulen 08 und 09 verwenden.** Das bedeutet: keine **Container** (Vektor/Liste/Map/usw.) und keine **Algorithmen** (alles, was den <algorithm>-Header erfordert) bis dahin. Andernfalls wird Ihre Note -42 sein.

**Einige Designanforderungen**

- Speicherlecks treten auch in C++ auf. Wenn Sie Speicher zuweisen (mit der Funktion new
  Schlüsselwort), müssen Sie **Speicherlecks** vermeiden.
- Von Modul 02 bis Modul 09 muss der Unterricht in der **orthodoxen kanonischen Form** gestaltet werden**, es sei denn, es ist ausdrücklich etwas anderes angegeben**.
- Jede Funktionsimplementierung in einer Header-Datei (mit Ausnahme von Funktionsschablonen) bedeutet 0 für die Übung.
- Sie sollten in der Lage sein, jeden Ihrer Header unabhängig von anderen zu verwenden. Daher müssen sie alle Abhängigkeiten einschließen, die sie benötigen. Allerdings müssen Sie das Problem der doppelten Einbindung vermeiden, indem Sie **Include-Guards** hinzufügen. Andernfalls wird Ihre Note 0 sein.

**Lies mich**

- Sie können bei Bedarf zusätzliche Dateien hinzufügen (z. B. um Ihren Code aufzuteilen). Da diese Aufgaben nicht von einem Programm überprüft werden, können Sie dies gerne tun, solange Sie die vorgeschriebenen Dateien einreichen.
- Manchmal sehen die Richtlinien einer Übung kurz aus, aber die Beispiele können Anforderungen aufzeigen, die nicht ausdrücklich in den Anweisungen stehen.
- Lesen Sie jedes Modul vollständig durch, bevor Sie beginnen! Wirklich, tun Sie es.
- Bei Odin, bei Thor! Benutze deinen Verstand!!!

Sie werden eine Menge Klassen implementieren müssen. Das kann mühsam sein, es sei denn, Sie können in Ihrem Lieblings-Texteditor Skripte schreiben.

Sie haben einen gewissen Spielraum bei der Durchführung der Übungen. Halten Sie sich jedoch an die vorgeschriebenen Regeln und seien Sie nicht faul. Sie würden eine Menge nützlicher Informationen verpassen! Zögern Sie nicht, sich über theoretische Konzepte zu informieren.

# Kapitel III

# Übung 00: Aaaaund... ÖFFNEN!

|  | Übung : 00 |
|---|---|
| Und... ÖFFNEN! | |
| Turn-in-Verzeichnis : *ex00/* | |
| Einzureichende Dateien: `Makefile, main.cpp, ClapTrap.{h, hpp}, ClapTrap.cpp` | |
| Verbotene Funktionen : `Keine` | |

Zuerst müssen Sie eine Klasse implementieren! Wie originell!

Es wird **ClapTrap** heißen und die folgenden privaten Attribute haben, die mit den in Klammern angegebenen Werten initialisiert werden:

- Name, der als Parameter an einen Konstruktor übergeben wird
- Trefferpunkte (10), stellen die Gesundheit der ClapTrap dar
- Energiepunkte (10)
- Angriffsschaden (0)

Fügen Sie die folgenden öffentlichen Mitgliedsfunktionen hinzu, damit die ClapTrap realistischer aussieht:

- void attack(const std::string& target);
- void takeDamage(unsigned int amount);
- void beRepaired(unsigned int amount);

Wenn ClapTrack angreift, verliert sein Ziel <Angriffsschaden> Trefferpunkte. Wenn ClapTrap sich selbst repariert, erhält es <Anzahl> Trefferpunkte zurück. Angreifen und Reparieren kosten jeweils 1 Energiepunkt. Natürlich kann ClapTrap nichts tun, wenn sie keine Trefferpunkte oder Energiepunkte mehr hat.

In all diesen Mitgliedsfunktionen müssen Sie eine Meldung ausgeben, um zu beschreiben, was passiert ist. Die Funktion attack() könnte zum Beispiel so etwas anzeigen (natürlich ohne die spitzen Klammern):

ClapTrap <Name> greift <Ziel> an und verursacht <Schaden> Punkte Schaden!

Die Konstruktoren und Destruktoren müssen auch eine Meldung anzeigen, damit Ihre Peer-Evaluatoren leicht erkennen können, dass sie aufgerufen wurden.

Implementieren Sie Ihre eigenen Tests und reichen Sie sie ein, um sicherzustellen, dass Ihr Code wie erwartet funktioniert.

# Kapitel IV

# Übung 01: Serena, meine Liebe!

|  | Übung : 01 |
|---|---|
| Serena, meine Liebe! | |
| Turn-in-Verzeichnis : *ex01/* | |
| Einzureichende Dateien: `Dateien der vorherigen Übung + ScavTrap.{h, hpp}, ScavTrap.cpp` | |
| Verbotene Funktionen : `Keine` | |

Da man nie genug ClapTraps haben kann, wirst du nun einen abgeleiteten Roboter erstellen. Er wird **ScavTrap** heißen und die Konstruktoren und den Destruktor von Clap-Trap erben. Allerdings werden seine Konstruktoren, sein Destruktor und attack() andere Meldungen ausgeben. Schließlich sind sich die ClapTraps ihrer Individualität bewusst.

Beachten Sie, dass die richtige Verkettung von Bau und Zerstörung in Ihren Tests gezeigt werden muss. Wenn eine ScavTrap erstellt wird, beginnt das Programm mit dem Bau einer ClapTrap. Die Zerstörung erfolgt in umgekehrter Reihenfolge. Und warum?

**ScavTrap** wird die Attribute von ClapTrap verwenden (ClapTrap folglich aktualisieren) und muss sie initialisieren:

- Name, der als Parameter an einen Konstruktor übergeben wird
- Trefferpunkte (100), stellen die Gesundheit der ClapTrap dar
- Energiepunkte (50)
- Angriffsschaden (20)

ScavTrap wird auch über eine besondere Kapazität verfügen:

```
void guardGate();
```

Diese Mitgliedsfunktion zeigt eine Meldung an, die besagt, dass ScavTrap jetzt im Gate keeper Modus ist.

Vergessen Sie nicht, weitere Tests zu Ihrem Programm hinzuzufügen.

# Kapitel V

# Übung 02: Wiederholte Arbeit

|  | Übung : 02 |
|---|---|
| Sich wiederholende Arbeit | |
| Turn-in-Verzeichnis : *ex02/* | |
| Abzugebende Dateien: `Dateien aus vorherigen Übungen + FragTrap.{h, hpp}, FragTrap.cpp` | |
| Verbotene Funktionen : `Keine` | |

Die Herstellung von ClapTraps geht dir wahrscheinlich langsam auf die Nerven.

Nun implementieren Sie eine FragTrap-Klasse, die von ClapTrap erbt. Sie ist der ScavTrap sehr ähnlich. Allerdings müssen die Nachrichten für den Aufbau und die Zerstörung unterschiedlich sein. Die korrekte Verkettung von Bau und Zerstörung muss in Ihren Tests gezeigt werden. Wenn eine FragTrap erstellt wird, beginnt das Programm mit dem Bau einer ClapTrap. Die Zerstörung erfolgt in umgekehrter Reihenfolge.

Für die Attribute gilt das Gleiche, aber diesmal mit anderen Werten:

- Name, der als Parameter an einen Konstruktor übergeben wird
- Trefferpunkte (100), stellen die Gesundheit der ClapTrap dar
- Energiepunkte (100)
- Angriffsschaden (30)

FragTrap hat auch eine besondere Kapazität:

```
void highFivesGuys(void);
```

Diese Mitgliedsfunktion zeigt eine positive High-Fives-Anfrage auf der Standardausgabe an.

Fügen Sie weitere Tests zu Ihrem Programm hinzu.

# Kapitel VI

# Übung 03: Jetzt wird's schräg!

|  | Übung : 03 |
|---|---|
| Jetzt ist es seltsam! | |
| Turn-in-Verzeichnis : *ex03/* | |
| Abzugebende Dateien: `Dateien aus vorherigen Übungen + DiamondTrap.{h, hpp}, DiamondTrap.cpp` | |
| Verbotene Funktionen : `Keine` | |

In dieser Übung werden Sie ein Monster erstellen: eine ClapTrap, die halb FragTrap, halb ScavTrap ist. Sie wird **DiamondTrap** heißen und sowohl von der FragTrap als auch von der ScavTrap erben. Das ist so riskant!

Die Klasse DiamondTrap wird ein Attribut name private haben. Geben Sie diesem Attribut genau denselben Variablennamen (hier geht es nicht um den Namen des Roboters) wie in der Basisklasse ClapTrap.

Um dies zu verdeutlichen, hier zwei Beispiele.
Wenn die Variable von ClapTrap name ist, geben Sie den Namen name an die von DiamondTrap.
Wenn die Variable von ClapTrap _name ist, geben Sie den Namen _name an die Variable von DiamondTrap.

Ihre Attribute und Mitgliedsfunktionen werden von einer ihrer Elternklassen übernommen:

- Name, der als Parameter an einen Konstruktor übergeben wird
- ClapTrap::name (Parameter des Konstruktors + Suffix "_clap_name")
- Trefferpunkte (FragTrap)
- Energiepunkte (ScavTrap)
- Angriffsschaden (FragTrap)
- attack() (Scavtrap)

Zusätzlich zu den speziellen Funktionen der beiden übergeordneten Klassen verfügt DiamondTrap über eine eigene spezielle Kapazität:

```
void whoAmI();
```

Diese Mitgliedsfunktion zeigt sowohl ihren Namen als auch ihren ClapTrap-Namen an.

Natürlich wird das ClapTrap-Unterobjekt der DiamondTrap nur einmal erstellt. Ja, es gibt einen Trick.

Fügen Sie weitere Tests zu Ihrem Programm hinzu.

Kennen Sie die Compilerflags -Wshadow und -Wno-shadow?

Sie können dieses Modul bestehen, ohne die Übung 03 zu bearbeiten.

# Kapitel VII

# Einreichung und Peer-Evaluierung

Reichen Sie Ihre Arbeit wie gewohnt in Ihrem Git-Repository ein. Nur die Arbeit in Ihrem Repository wird während der Verteidigung bewertet. Zögern Sie nicht, die Namen Ihrer Ordner und Dateien zu überprüfen, um sicherzustellen, dass sie korrekt sind.

```
??????????? XXXXXXXXXX = $3$$cf36316f07f871b4f14926007c37d388
```