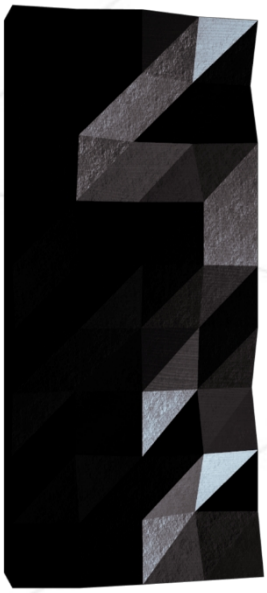

C++ - Modul 01

Speicherzuweisung, Zeiger auf Mitglieder, Referenzen, Switch-Anweisung

Zusammenfassung:
Dieses Dokument enthält die Übungen des Moduls 01 aus den C++ Modulen.

Fassung: 10

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: BraiiiiiiinnnzzzZ

	Übung : 00
BraiiiiiiinnnzzzZ	
Turn-in-Verzeichnis : *ex00/*	
Einzureichende Dateien: `Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp`	
Verbotene Funktionen : `Keine`	

Implementieren Sie zunächst eine Zombie-Klasse. Sie hat ein privates String-Attribut name.
Fügen Sie der Klasse Zombie eine Memberfunktion void announce(void); hinzu. Zombies kündigen sich wie folgt an:

<Name>: BraiiiiiiinnnzzzZ...

Drucken Sie die spitzen Klammern (< und >) nicht aus. Für einen Zombie namens Foo würde die Meldung lauten:

Foo: BraiiiiiiinnnzzzZ...

Dann implementieren Sie die beiden folgenden Funktionen:

- Zombie* newZombie(std::string name);
  Sie erstellt einen Zombie, benennt ihn und gibt ihn zurück, damit Sie ihn außerhalb des Funktionsbereichs verwenden können.
- void randomChump(std::string name);
  Sie erschafft einen Zombie, gibt ihm einen Namen, und der Zombie kündigt sich selbst an.

Was ist nun der eigentliche Sinn der Übung? Sie müssen herausfinden, in welchem Fall es besser ist, die Zombies auf dem Stack oder Heap zu allozieren.

Zombies müssen zerstört werden, wenn man sie nicht mehr braucht. Der Destruktor muss zu Debugging-Zwecken eine Meldung mit dem Namen des Zombies ausgeben.

Kapitel IV

Übung 01: Mehr Grips!

	Übung : 01
Mehr Hirn!	
Turn-in-Verzeichnis : *ex01/*	
Einzureichende Dateien: `Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, zombieHorde.cpp`	
Verbotene Funktionen : `Keine`	

Zeit, eine **Horde Zombies** zu erschaffen!

Implementieren Sie die folgende Funktion in der entsprechenden Datei:

```
Zombie*     zombieHorde( int N, std::string name );
```

Es muss N Zombie-Objekte in einer einzigen Zuweisung zuordnen. Dann muss sie die Zombies initialisieren und jedem von ihnen den als Parameter übergebenen Namen geben. Die Funktion gibt einen Zeiger auf den ersten Zombie zurück.

Implementieren Sie Ihre eigenen Tests, um sicherzustellen, dass die Funktion zombieHorde() wie erwartet funktioniert.

Versuchen Sie, announce() für jeden der Zombies aufzurufen.

Vergessen Sie nicht, alle Zombies zu löschen und nach **Speicherlecks zu** suchen.

Kapitel V

Übung 02: HI THIS IS BRAIN

	Übung : 02
HALLO, HIER IST BRAIN	
Turn-in-Verzeichnis : *ex02/*	
Einzuliefernde Dateien: `Makefile, main.cpp`	
Verbotene Funktionen : `Keine`	

Schreiben Sie ein Programm, das Folgendes enthält:

- Eine String-Variable, initialisiert mit "HI THIS IS BRAIN".
- stringPTR: Ein Zeiger auf die Zeichenkette.
- stringREF: Ein Verweis auf die Zeichenkette.

Ihr Programm muss drucken:

- Die Speicheradresse der String-Variablen.
- Die von stringPTR gehaltene Speicheradresse.
- Die Speicheradresse, die von stringREF

gehalten wird. Und dann:

- Der Wert der String-Variable.
- Der Wert, auf den stringPTR zeigt.
- Der Wert, auf den stringREF zeigt.

Das ist alles, keine Tricks. Das Ziel dieser Übung ist es, Referenzen zu entmystifizieren, die völlig neu erscheinen können. Obwohl es einige kleine Unterschiede gibt, ist dies eine andere Syntax für etwas, das Sie bereits tun: Adressmanipulation.

Kapitel VI

Übung 03: Unnötige Gewalt

	Übung : 03
Unnötige Gewalt	
Turn-in-Verzeichnis : *ex03/*	
Einzureichende Dateien: `Makefile, main.cpp, Weapon.{h, hpp}, Weapon.cpp, HumanA.{h, hpp}, HumanA.cpp, HumanB.{h, hpp}, HumanB.cpp`	
Verbotene Funktionen : `Keine`	

Implementieren Sie eine Waffenklasse, die hat:

- Ein privater Attributtyp, der eine Zeichenkette ist.
- Eine getType()-Mitgliedsfunktion, die einen konstanten Verweis auf type zurückgibt.
- Eine setType()-Mitgliedsfunktion, die den Typ unter Verwendung des als Parameter übergebenen neuen Typs festlegt.

Erstellen Sie nun zwei Klassen: **MenschA** und **MenschB**. Sie haben beide eine Waffe und einen Namen. Sie haben auch eine Mitgliedsfunktion attack(), die anzeigt (natürlich ohne die spitzen Klammern):

<Name> greift mit seinem <Waffentyp> an

HumanA und HumanB sind bis auf diese beiden winzigen Details fast identisch:

- Während HumanA die Waffe in seinem Konstruktor übernimmt, tut dies HumanB nicht.
- MenschB hat vielleicht **nicht immer** eine Waffe, während MenschA **immer** bewaffnet sein wird.

Wenn Ihre Implementierung korrekt ist, wird die Ausführung des folgenden Codes für beide Testfälle einen Angriff mit einer "groben Stachelkeule" und einen zweiten Angriff mit "einer anderen Art von Keule" ausgeben:

```
int main()
{
  {
    Weapon club = Weapon("grobe Stachelkeule");

    HumanA bob("Bob", club);
    bob.attack();
    club.setType("eine andere Art von Club");
    bob.attack();
  }
  {
    Weapon club = Weapon("grobe Stachelkeule");

    HumanB jim("Jim");
    jim.setWeapon(club);
    jim.attack();
    club.setType("eine andere Art von Club");
    jim.attack();
  }

  0 zurückgeben;
}
```

Vergessen Sie nicht, auf **Speicherlecks** zu achten.

In welchem Fall denken Sie, dass es am besten wäre, einen Zeiger auf Weapon zu verwenden? Und einen Verweis auf Weapon? Und warum? Denken Sie darüber nach, bevor Sie mit dieser Übung beginnen.

Kapitel VII

Übung 04: Sed ist für Verlierer

	Übung : 04
Sed ist für Verlierer	
Turn-in-Verzeichnis : *ex04/*	
Einzureichende Dateien: `Makefile, main.cpp, *.cpp, *.{h, hpp}`	
Verbotene Funktionen : `std::string::replace`	

Erstellen Sie ein Programm, das drei Parameter in der folgenden Reihenfolge benötigt: einen Dateinamen und zwei Zeichenketten, s1 und s2.

Es öffnet die Datei <Dateiname> und kopiert ihren Inhalt in eine neue Datei <Dateiname>.replace, wobei jedes Vorkommen von s1 durch s2 ersetzt wird.

Die Verwendung von C-Dateimanipulationsfunktionen ist verboten und wird als Betrug gewertet. Alle Funktionen der Klasse std::string sind erlaubt, außer replace. Verwenden Sie sie mit Bedacht!

Natürlich müssen Sie auch mit unerwarteten Eingaben und Fehlern umgehen. Sie müssen Ihre eigenen Tests erstellen und einreichen, um sicherzustellen, dass Ihr Programm wie erwartet funktioniert.

Kapitel VIII Übung

05: Harl 2.0

	Übung : 05
Harl 2.0	
Turn-in-Verzeichnis : *ex05/*	
Einzureichende Dateien: `Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp`	
Verbotene Funktionen : `Keine`	

Kennst du Harl? Das tun wir doch alle, oder? Falls nicht, finden Sie unten die Art von Kommentaren, die Harl macht. Sie sind nach Stufen geordnet:

- "**DEBUG**"-Stufe: Debug-Meldungen enthalten kontextbezogene Informationen. Sie werden hauptsächlich zur Problemdiagnose verwendet.
  Beispiel: *"Ich liebe es, für meinen 7XL-Doppel-Käse-Dreifach-Pickle-Spezial-Ketchup-Burger extra Bacon zu haben. Das tue ich wirklich!"*

- Ebene "**INFO**": Diese Meldungen enthalten umfangreiche Informationen. Sie sind hilfreich für die Verfolgung der Programmausführung in einer Produktionsumgebung.
  Beispiel: "*Ich kann nicht glauben, dass zusätzlicher Speck mehr Geld kostet. Sie haben nicht genug Speck in meinen Burger getan! Wenn das so wäre, würde ich nicht mehr verlangen!"*

- Stufe "**WARNUNG**": Warnmeldungen weisen auf ein mögliches Problem im System hin. Sie können jedoch behandelt oder ignoriert werden.
  Beispiel: "*Ich denke, ich habe es verdient, etwas mehr Speck umsonst zu bekommen. Ich komme schon seit Jahren hierher, während du erst seit letztem Monat hier arbeitest."*

- Ebene "**ERROR**": Diese Meldungen weisen darauf hin, dass ein nicht behebbarer Fehler aufgetreten ist. In der Regel handelt es sich um ein kritisches Problem, das ein manuelles Eingreifen erfordert.
  Beispiel: *"Das ist inakzeptabel! Ich möchte jetzt mit dem Manager sprechen."*

Sie werden Harl automatisieren. Das wird nicht schwierig sein, da es immer das G l e i c h e sagt. Sie müssen eine Harl-Klasse mit den folgenden privaten Mitgliedsfunktionen erstellen:

- void debug(void);
- void info(void);
- void warning(void);
- void error(void);

Harl hat auch eine öffentliche Mitgliedsfunktion, die die vier oben genannten Mitgliedsfunktionen in Abhängigkeit von der als Parameter übergebenen Stufe aufruft:

```
void    complain( std::string level );
```

Das Ziel dieser Übung ist die Verwendung von **Zeigern auf Mitgliedsfunktionen**. Dies ist kein Vorschlag. Harl muss sich beschweren, ohne einen Wald von if/else if/else zu verwenden. Es denkt nicht zweimal nach!

Erstellen Sie Tests und reichen Sie diese ein, um zu zeigen, dass Harl sich viel beschwert. Sie können die oben im Betreff aufgeführten Beispiele für Kommentare verwenden oder eigene Kommentare einfügen.

Kapitel IX

Übung 06: Harl-Filter

	Übung : 06
Harl-Filter	
Turn-in-Verzeichnis : *ex06/*	
Einzureichende Dateien: `Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp`	
Verbotene Funktionen : `Keine`	

Manchmal wollen Sie nicht alles hören, was Harl sagt. Führen Sie ein System ein, um zu filtern, was Harl sagt, abhängig von den Protokollebenen, die Sie abhören wollen.

Erstellen Sie ein Programm, das als Parameter eine der vier Ebenen annimmt. Es zeigt alle Meldungen ab dieser Ebene an. Zum Beispiel:

```
$> ./harlFilter "WARNING"
[ WARNING ]
Ich denke, ich habe es verdient, etwas zusätzlichen Speck umsonst zu bekommen.
Ich komme schon seit Jahren hierher, während Sie erst seit letztem Monat hier arbeiten.

(FEHLER)
Das ist inakzeptabel, ich möchte jetzt mit dem Manager sprechen.

$> ./harlFilter "Ich bin nicht sicher, wie müde ich heute
bin..." [Wahrscheinlich beschwert er sich über
unbedeutende Probleme]
```

Es gibt zwar mehrere Möglichkeiten, mit Harl umzugehen, aber eine der effektivsten ist es, ihn auszuschalten.

Geben Sie Ihrer ausführbaren Datei den Namen harlFilter.

In dieser Übung müssen Sie die switch-Anweisung verwenden und vielleicht entdecken.

`Sie können dieses Modul bestehen, ohne die Übung 06 zu bearbeiten.`

Kapitel X

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$$4f1b9de5b5e60c03dcb4e8c7c7e4072c
```