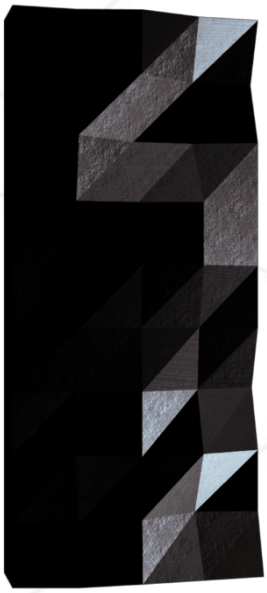

C++ - Modul 09

STL

Zusammenfassung:
Dieses Dokument enthält die Übungen des Moduls 09 aus den C++ Modulen.

Fassung: 2

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 müssen eine Vielzahl von Klassen implementieren. 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

Modulspezifische Regeln

Die Verwendung der Standardcontainer ist für die Durchführung aller Übungen in diesem Modul obligatorisch.

Sobald ein Container verwendet wird, können Sie ihn für den Rest des Moduls nicht mehr verwenden.

Es ist ratsam, das Thema in seiner Gesamtheit zu lesen, bevor Sie die Übungen durchführen.

Für jede Übung muss mindestens ein Behälter verwendet werden, mit Ausnahme der Übung 02, für die zwei Behälter verwendet werden müssen.

Sie müssen für jedes Programm ein Makefile einreichen, das Ihre Quelldateien mit den Flags -Wall, -Wextra und -Werror zur gewünschten Ausgabe kompiliert.

Sie müssen C++ verwenden, und Ihr Makefile darf nicht neu gelinkt werden.

Ihr Makefile muss mindestens die Regeln $(NAME), all, clean, fclean und re enthalten.

Kapitel IV

Übung 00: Bitcoin-Tausch

	Übung : 00
Bitcoin-Börse	
Turn-in-Verzeichnis : *ex00/*	
Einzureichende Dateien: `Makefile, main.cpp, BitcoinExchange.{cpp, hpp}`	
Verbotene Funktionen : `Keine`	

Sie müssen ein Programm erstellen, das den Wert eines bestimmten Bitcoin-Betrags zu einem bestimmten Datum ausgibt.

Dieses Programm muss eine Datenbank im csv-Format verwenden, die den Bitcoin-Preis im Laufe der Zeit darstellt. Diese Datenbank wird mit diesem Thema zur Verfügung gestellt.

Das Programm nimmt als Eingabe eine zweite Datenbank, in der die verschiedenen zu bewertenden Preise/Daten gespeichert sind.

Ihr Programm muss diese Regeln einhalten:

- Der Name des Programms ist btc.
- Ihr Programm muss eine Datei als Argument annehmen.
- Jede Zeile in dieser Datei muss das folgende Format haben: "Datum | Wert".
- Ein gültiges Datum hat immer das folgende Format: Jahr-Monat-Tag.
- Ein gültiger Wert muss entweder ein Float oder eine positive ganze Zahl zwischen 0 und 1000 sein.

`Sie müssen mindestens einen Container in Ihrem Code verwenden, um diese Übung zu validieren. Mögliche Fehler sollten Sie mit einer entsprechenden Fehlermeldung behandeln.`

Hier ist ein Beispiel für eine input.txt-Datei:

```
$> head input.txt
Datum | Wert
2011-01-03 | 3
2011-01-03 | 2
2011-01-03 | 1
2011-01-03 | 1.2
2011-01-09 | 1
2012-01-11 | -1
2001-42-42
2012-01-11 | 1
2012-01-11 | 2147483648
$>
```

Ihr Programm wird den Wert in Ihrer Eingabedatei verwenden.

Ihr Programm sollte auf der Standardausgabe das Ergebnis des Wertes multipliziert mit dem Wechselkurs entsprechend dem in Ihrer Datenbank angegebenen Datum anzeigen.

Wenn das in der Eingabe verwendete Datum nicht in Ihrer DB vorhanden ist, müssen Sie das nächstgelegene in Ihrer DB enthaltene Datum verwenden. Achten Sie darauf, das untere Datum zu verwenden und nicht das obere.

Im Folgenden finden Sie ein Beispiel für die Verwendung des Programms.

```
$> ./btc
Fehler: Datei konnte nicht geöffnet werden.
$> ./btc input.txt
2011-01-03 => 3 = 0.9
2011-01-03 => 2 = 0.6
2011-01-03 => 1 = 0.3
2011-01-03 => 1.2 = 0.36
2011-01-09 => 1 = 0.32
Fehler: keine positive Zahl.
Fehler: falsche Eingabe => 2001-
42-42 2012-01-11 => 1 = 7.1
Fehler: Eine zu große Zahl.
$>
```

Achtung! Der/die Container, die Sie zur Validierung dieser Übung verwenden, sind für den Rest dieses Moduls nicht mehr verwendbar.

Kapitel V

Übung 01: Umgekehrte polnische Notation

	Übung : 01
RPN	
Turn-in-Verzeichnis : *ex01/*	
Einzureichende Dateien: `Makefile, main.cpp, RPN.{cpp, hpp}`	
Verbotene Funktionen : `Keine`	

Sie müssen ein Programm mit diesen Beschränkungen erstellen:

- Der Programmname ist RPN.
- Ihr Programm muss einen invertierten polnischen mathematischen Ausdruck als Argument verwenden.
- Die Zahlen, die in dieser Operation verwendet und als Argumente übergeben werden, sind immer kleiner als 10. Die Berechnung selbst, aber auch das Ergebnis, berücksichtigen diese Regel nicht.
- Ihr Programm muss diesen Ausdruck verarbeiten und das richtige Ergebnis auf der Standardausgabe ausgeben.
- Tritt bei der Ausführung des Programms ein Fehler auf, sollte eine Fehlermeldung auf der Standardausgabe ausgegeben werden.
- Ihr Programm muss in der Lage sein, Operationen mit diesen Token zu verarbeiten: "+ - / *".

`Sie müssen mindestens einen Container in Ihrem Code verwenden, um diese Übung zu validieren.`

Sie brauchen sich nicht um die Klammern oder Dezimalzahlen zu kümmern.

Hier ist ein Beispiel für eine Standardverwendung:

```
$> ./RPN "8 9 * 9 - 9 - 9 - 4 - 1 +"
42
$> ./RPN "7 7 * 7 -"
42
$> ./RPN "1 2 * 2 / 2 * 2 4 - +"
0
$> ./RPN "(1 + 1)"
Fehler
$>
```

Achtung! Die Container, die Sie in der vorherigen Übung verwendet haben, sind hier verboten. Der/die Container, den/die Sie zur Validierung dieser Übung verwendet haben, ist/sind für den Rest des Moduls nicht mehr verwendbar.

Kapitel VI

Übung 02: PmergeMe

	Übung : 02
PmergeMe	
Turn-in-Verzeichnis : *ex02/*	
Einzureichende Dateien: `Makefile, main.cpp, PmergeMe.{cpp, hpp}`	
Verbotene Funktionen : `Keine`	

Sie müssen ein Programm mit diesen Beschränkungen erstellen:

- Der Name des Programms ist PmergeMe.
- Ihr Programm muss in der Lage sein, eine positive Ganzzahlfolge als Argument zu verwenden.
- Ihr Programm muss den Merge-Insert-Sortieralgorithmus verwenden, um die positive ganzzahlige Folge zu sortieren.

Zur Klarstellung: Ja, Sie müssen den Ford-Johnson-Algorithmus verwenden.

- Wenn während der Programmausführung ein Fehler auftritt, sollte eine Fehlermeldung auf der Standardausgabe ausgegeben werden.

Sie müssen mindestens zwei verschiedene Container in Ihrem Code verwenden, um diese Übung zu validieren. Ihr Programm muss in der Lage sein, mindestens 3000 verschiedene Ganzzahlen zu verarbeiten.

Es wird dringend empfohlen, den Algorithmus für jeden einzelnen Container zu implementieren und somit die Verwendung einer generischen Funktion zu vermeiden.

Im Folgenden finden Sie einige zusätzliche Richtlinien zu den Informationen, die Sie zeilenweise auf der Standardausgabe anzeigen sollten:

- In der ersten Zeile müssen Sie einen expliziten Text anzeigen, gefolgt von der unsortierten positiven Zahlenfolge.
- In der zweiten Zeile müssen Sie einen expliziten Text, gefolgt von der sortierten positiven Ganzzahlfolge, anzeigen.
- In der dritten Zeile müssen Sie einen expliziten Text anzeigen, der die von Ihrem Algorithmus verbrauchte Zeit angibt, indem Sie den ersten Container angeben, der für die Sortierung der positiven ganzzahligen Folge verwendet wird.
- In der letzten Zeile müssen Sie einen expliziten Text anzeigen, der die von Ihrem Algorithmus verbrauchte Zeit angibt, indem Sie den zweiten Container angeben, der zum Sortieren der positiven Ganzzahlfolge verwendet wird.

Das Format für die Anzeige der Zeit, die für die Sortierung verwendet wurde, ist frei, aber die gewählte Genauigkeit muss es ermöglichen, den Unterschied zwischen den beiden verwendeten Behältern deutlich zu erkennen.

Hier ist ein **Beispiel** für eine Standardverwendung:

```
$> ./PmergeMe 3 5 9 7 4
Vorher: 3 5 9 7 4
Nach:       3 4 5 7 9
Zeit für die Verarbeitung eines Bereichs von          5 Elementen mit
std::[..] : 0.00031 us Zeit für die Verarbeitung eines Bereichs von    5
Elementen mit std::[..] : 0.00014 us
$> ./PmergeMe `shuf -i 1-100000 -n 3000 | tr "\n" " "`
Vorher: 141 79 526 321 [...]
Nach:       79 141 321 526 [...]
Zeit zur Verarbeitung eines Bereichs von 3000 Elementen mit std::[..] :
62.14389 us Zeit zur Verarbeitung eines Bereichs von 3000 Elementen mit
std::[..] : 69.27212 us
$> ./PmergeMe "-1" "2"
Fehler
$> # Für OSX USER:
$> ./PmergeMe `jot -r 3000 1 100000 | tr '\n' ' '`
[...]
```

Die Angabe der Zeit ist in diesem Beispiel absichtlich ungewöhnlich. Natürlich müssen Sie die Zeit angeben, die für die Durchführung aller Ihrer Operationen verwendet wird, sowohl für den Sortierteil als auch für den Datenverwaltungsteil.

Achtung! Die Behälter, die Sie in den vorherigen Übungen verwendet haben, sind hier verboten.

Die Behandlung von Fehlern im Zusammenhang mit Duplikaten liegt in Ihrem Ermessen.

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.

16D85ACC441674FBA2DF65190663F33F793984B142405F56715D5225FBAB6E3D6A4F
167020A16827E1B16612137E59ECD492E47AB764CB10B45D979615AC9FC74D521D9
20A778A5E