Programmieren in C++ Versuch 2

Prof. Dr. R. Manzke, Sommersemester 2022

Zielsetzungen

- OOP mit C++
- Projektstruktur mit mehreren Dateien aufsetzen
- CMake nutzen, um ein Hauptprogramm und eine Bibliothek zu erzeugen
- Namensräume sinnvoll einsetzen
- Klassenhierarchie mit einfacher Vererbung aufsetzen
- Konstruktoren und Destruktoren implementieren
- Copy und Move Constructor, Bedeutung von “r-value” als Tempvalue
- Abstrakte Klassen einsetzen, Pur-Virtuelle und Virtuelle Funktionen einsetzen
- Überladung von Funktionen und Operatoren
- Smart Pointer einsetzen
- Iteratoren einsetzen
- Stack / Heap Unterschiede
- C++ Umsetzung von Design Patterns, hier Factory

Zeitrahmen

- Es besteht Anwesenheitspflicht zu den Laboren.
- Versuch 2 soll zum 3. Versuchstermin abgenommen werden. Es stehen insofern 2 Wochen Bearbeitungszeit zur Verfügung.
- Die Abnahme erfolgt durch das Laborpersonal.
- Eine erfolgreiche Abnahme während der Laborzeit ist notwendig für die Erteilung des Labortestats.
- Der fertige Quellcode soll im Moodle abgegeben werden.
- Die Aufgabe kann im Team von max. 2 Personen oder alleine bearbeitet werden.

Aufgabenstellung

1. Nutzen sie eine Compilerumgebung ihrer Wahl und erzeugen sie ein neues C++ Projekt in einer IDE ihrer Wahl mit folgenden Dateien bzw. Anforderungen:
    - main.cpp, inklusive der Einsprungsfunktion ***int main(int argc, char *argv[])***
    - CMakeLists.txt, als Projektbeschreibungsdatei für das CMake Build-System
    - Pro Klasse je eine .[c|h]pp Datei, gleiche Namensgebung des Dateinamens wie Klassenname
2. Schreiben sie ausführbaren C++ Programmcode nach folgenden Anforderungen

Teil A.1:

Erstellen sie eine neue Klasse A mit folgenden Anforderungen:

- Privates Attribut “_nptr” als Pointer auf einen “int”, initialisiert mit “nullptr”
- Privates Attribut “_aptr” als Pointer auf ein “int” array, initialisiert mit “nullptr”
- Expliziter Konstruktor (c’tor)
    - Mit “int” als Argument
    - Alloziert “int” auf dem Heap
        - Lässt “_nptr” auf den neu allozierten “int” zeigen
    - Alloziert “int array” auf dem Heap mit 5 Elementen
        - Lässt “_aptr” auf das 1. Elements des “int” arrays zeigen
    - Initialisiert den Wert des neu allozierten “int”, auf den “_nptr” zeigt, mit dem Konstruktorargument
    - Initialisiert alle Elemente des “int” arrays, auf den “_aptr” zeigt, mit 42
    - Gibt mit “cout” aus, mit welchem Wert initialisiert worden ist:
        - “class A constructor: “ Wert
- Destruktor (d’tor)
    - Gibt mit “cout” den “int” Wert aus, auf den _nptr zeigt
        - “class A destructor: “ Wert
    - Gibt “_nptr” auf dem Heap frei
    - Gibt “_aptr” auf dem Heap frei
    - Gibt mit “cout” “nullptr” aus wenn _nptr auf keinen int zeigt (nutzen sie die “Yoda Notation” bei ihrer if-Abfrage!)
        - “class A destructor: nullptr”

Klasse A soll bei folgendem Programmcode

```
A a1{3};
```

folgendes ausgeben

```
class A constructor: 3
class A destructor: 3
```

Teil A.2:

- Erweitern sie die Klasse A um einen Kopierkonstruktor, welcher bei Aufruf mit “cout” “class A copy constructor: “ Wert ausgibt
- Der Kopierkonstruktor soll eine tiefe Kopie des Objekts erstellen

Klasse A soll bei folgendem Programmcode

```
A a1{3};
A a2{a1};
A a3{A{4}};
```

folgendes ausgeben

```
class A constructor: 3
class A copy constructor: 3
class A constructor: 4
class A destructor: 4
class A destructor: 3
class A destructor: 3
```

Teil A.3:

- Erweitern sie die Klasse A um einen Verschiebekonstruktor, welcher bei Aufruf mit “cout” “class A move constructor: “ Wert ausgibt
- Der Verschiebekonstruktor soll den Zeiger “_nptr”, sowie “_aptr” mit std::swap im move-Objekt austauschen

Klasse A soll bei folgendem Programmcode

```
vector<A> v;
v.push_back(A{5});
```

folgendes ausgeben

```
class A constructor: 5
class A move constructor: 5
class A destructor: nullptr
class A destructor: 5
```

Teil A.4:

- Erweitern sie die Klasse A um einen Kopierzuweisungsoperator, welcher bei Aufruf mit “cout” “class A copy assignment operator: “ Wert ausgibt
- Der Kopierzuweisungsoperator soll auch testen, ob das Objekt sich selbst zugewiesen wird (“a5 = a5”) und dann sich selbst über “*this” zurückgeben, ohne neuen Speicher für den “int” zu allozieren
- Der Kopierzuweisungsoperator erzeugt eine tiefe Kopie

Klasse A soll bei folgendem Programmcode

```
A a5{7};
A a6{8};
a6 = a5;
```

folgendes ausgeben

```
class A constructor: 7
class A constructor: 8
class A copy assignment operator: 7
class A destructor: 7
class A destructor: 7
```

Teil A.5:

- Erweitern sie die Klasse A um einen Verschiebezuweisungsoperator, welcher bei Aufruf mit “cout” “class A move assignment operator: “ Wert ausgibt
- Hier soll erneut mit std::swap der Inhalt von “_nptr”, sowie “_aptr” im move-Objekt ausgetauscht werden

Klasse A soll bei folgendem Programmcode

```
A a7{9};
a7 = A{10};
```

folgendes ausgeben

```
class A constructor: 9
class A constructor: 10
class A move assignment operator: 10
class A destructor: 9
class A destructor: 10
```

Teil A.6:

- Nutzen sie nun die Verschiebefunktion “std::move” zur Erzwingung von “move semantics” beim Erzeugen einer neuen Instanz der Klasse A

Klasse A soll bei folgendem Programmcode

```
A a8{11};
A a9{move(a8)};
```

folgendes ausgeben

```
class A constructor: 11
class A move constructor: 11
class A destructor: 11
class A destructor: nullptr
```

Fragen zu Teil A:

- Warum wird bei "`A a3{A{4}};`" in Teil A.2 kein Kopierkonstruktor aufgerufen?
- Warum wird in Teil A.3 "`class A destructor: nullptr`" ausgegeben?
- Warum wird in Teil A.5 "`class A destructor: 9`" ausgegeben?
- Inwiefern verhält sich Teil A.6 anders als Teil A.2?

Teil B:

Implementieren sie in C++ nach dem Entwurfsmuster "Factory" eine Fabrik für Gerichte. Erfüllen sie dabei die folgenden Anforderungen:

Anforderung	Beschreibung	Erfüllt
1	Erzeugen sie ihre Factory in einer separaten Bibliothek "dishlib", erstellen sie dazu ein Unterverzeichnis und die entsprechende CMakeLists.txt Datei, legen sie in dem Unterverzeichnis alle [c|h]pp Dateien der Bibliothek ab.	
2	Binden sie ihre Bibliothek im Hauptprogramm ein, ändern sie dazu entsprechend die CMakeLists.txt Datei des Hauptprograms.	
3	Kapseln sie alle Klassen der Dishlib in dem Namensraum "dishlib".	
4	Implementieren sie eine abstrakte Klasse "AbstractDish" in der DishLib mit den folgenden öffentlichen Attributen und Methoden: - "Prepare()" soll später von abgeleiteten Klassen implementiert werden (d.h. pur-virtuell) und ein Gericht zubereiten, - "GetDishName()" gibt den Namen des Gerichts zurück, - "GetIngredients()" gibt alle Zutaten als Vektor zurück, - "GetNumberIngredients()" gibt die Anzahl der Zutaten zurück. Und folgenden geschützten Attributen und Methoden, entscheiden sie hier sinnvoll, ob diese "protected oder private" sein sollen: - "AddIngredients()" fügt Zutaten zum Gericht hinzu, - "vIngredients" Vektor mit Zutaten, - "dishName" Name des Gerichts. Löschen sie den default Konstruktor. Setzen sie den Destruktor auf default (muss dieser auch virtuell sein?). Erzeugen sie einen expliziten Konstruktor mit Namen des Gerichts als Parameter. Beim Aufruf dieses Konstruktors soll der Name des	

	Gerichts initialisiert werden.	
5	Implementieren sie spezialisierte Klassen “VegetableSoup” und “PizzaMargherita” in der DishLib die von AbstractDish sinnvoll ableiten. Implementieren sie “Prepare()” dabei sollen über “AddIngredients()” Zutaten hinzugefügt werden, sowie die Zubereitungsschritte auf der Konsole ausgegeben werden. Löschen sie jeweils den default Konstruktor. Erzeugen sie einen expliziten Konstruktor mit Namen des Gerichts als Argument.	
6	Implementieren sie eine Klasse “DishFactory” in der DishLib zur Erzeugung von “AbstractDish” Objekten mit folgenden öffentlichen Attributen bzw. Methoden: - “DishType” Enumerator Klasse zur Identifizierung der möglichen Gerichte, - “CreateDish()” die anhand eines Arguments “dishType” vom Typ “DishType” neue Objekte der Gerichte auf dem Heap erzeugt, sie mit “Prepare()” zubereitet und diese als “unique_ptr<AbstractDish>” zurück gibt.	
7	Implementieren sie für ihr Hauptprogramm eine Klasse “Customer” mit folgenden öffentlichen Attributen bzw. Methoden: - “ServeDish()” der als Argument ein “dish” übergeben kann (Referenz auf unique_ptr<AbstractDish>), - “EatDish()” die das servierte “dish” Objekt “verspeist” und damit einen nullptr zurück lässt. Und folgenden geschützten Attributen und Methoden, entscheiden sie hier sinnvoll, ob diese “protected oder private” sein sollen: - “customerDish” unique_ptr auf das servierte Gericht, - “customerName”, Name des Kunden. Löschen sie den default Konstruktor. Erzeugen sie einen expliziten Konstruktor mit Namen des Kunden als Parameter. Beim Aufruf dieses Konstruktors soll der Name des Kunden initialisiert werden.	
8	Implementieren sie in ihrem Hauptprogramm einen Vektor “vDishes” mit Gerichten. Nutzen sie hierfür “unique_ptr<AbstractDish>” als Selbsdefinierten Datentyp “DishType”.	
9	Nutzen sie die Factory, um Gerichte zu erzeugen und fügen sie diese dem Vektor hinzu. Nutzen sie hier Verschiebesemantik!	
10	Sortieren sie die Gerichte im Vektor aufgrund der Anzahl ihrer Zutaten. Nutzen sie hierfür std::sort. Nutzen sie für die Sortierfunktion einen Lambda-Audruck!	
11	Geben sie die nach Menge der Zutaten sortierten Gerichte namentlich mit der Menge der Zutaten aus.	

12	Erzeugen sie einen Vektor mit Kunden. Nutzen sie dazu erneut unique_ptr.	
13	Servieren sie ihre Gerichte den Kunden, nutzen sie dazu die “ServeDish()” Methode. Nutzen sie Verschiebesemantik! Passen sie den Vektor nach dem Verschieben der Elemente an, d.h. entfernen sie das nun ungültige Element.	
14	Lassen sie die Kunden die Gerichte verspeisen. Nutzen sie dazu eine range-based-loop und die “EatDish()” Methode.	
15	Vermeiden sie die Doppelung von Code!	
16	Kapseln sie alle Attribute / Methoden soweit wie möglich!	
17	Nutzen sie die eingebauten Datentypen sinnvoll, beachten sie dabei auch signed / unsigned bei Ganzzahltypen!	
18	Vermeiden sie das Kopieren von Objekten soweit möglich!	
19	Nutzen sie iostream zur Textein und -ausgabe!	
20	Nutzen sie std::string für Zeichenketten!	
21	Setzen sie wann immer sinnvoll “const” und Referenzen ein!	

Anhang

Referenzen

- https://cmake.org/cmake/help/latest/command/add_subdirectory.html
- https://cmake.org/cmake/help/latest/command/include_directories.html
- https://cmake.org/cmake/help/latest/command/target_link_libraries.html
- https://cmake.org/cmake/help/latest/command/add_library.html
- https://en.cppreference.com/w/cpp/language/initialization
- https://en.cppreference.com/w/cpp/language/nullptr
- https://en.cppreference.com/w/cpp/language/explicit
- https://en.cppreference.com/w/cpp/language/function#Deleted_functions
- https://en.cppreference.com/w/cpp/language/new
- https://en.cppreference.com/w/cpp/language/delete
- https://en.cppreference.com/w/cpp/language/constructor
- https://en.cppreference.com/w/cpp/language/destructor
- https://en.cppreference.com/w/cpp/algorithm/copy

- https://en.cppreference.com/w/cpp/algorithm/fill_n
- https://en.cppreference.com/w/cpp/language/default_constructor
- https://en.cppreference.com/w/cpp/language/copy_constructor
- https://en.cppreference.com/w/cpp/language/move_constructor
- https://en.cppreference.com/w/cpp/language/copy_assignment
- https://en.cppreference.com/w/cpp/language/move_assignment
- https://en.cppreference.com/w/cpp/algorithm/move
- https://en.cppreference.com/w/cpp/algorithm/swap
- https://en.cppreference.com/w/cpp/container/vector
- https://en.cppreference.com/w/cpp/keyword/virtual
- https://en.cppreference.com/w/cpp/memory/unique_ptr
- https://en.cppreference.com/w/cpp/container/vector/push_back
- https://en.cppreference.com/w/cpp/container/vector/erase
- https://en.cppreference.com/w/cpp/language/auto
- https://en.cppreference.com/w/cpp/language/enum
- https://en.cppreference.com/w/cpp/language/access
- https://en.cppreference.com/w/cpp/language/abstract_class
- http://www.cplusplus.com/reference/string/string/
- https://en.cppreference.com/w/cpp/language/override
- https://en.cppreference.com/w/c/language/value_category
- https://en.cppreference.com/w/cpp/algorithm/sort
- https://en.cppreference.com/w/cpp/language/lambda
- https://www.oreilly.com/library/view/head-first-design/0596007124/ch04.html
- https://github.com/jmossberg/rvalues-move-semantics-cpp