Übungen OOSWE/Progr. 2 (C++) – KE 9

Der C/C++-Coding Styleguide ist einzuhalten.
Folgende Einstellungen sind für Debug und Release (All Configurations) vorzunehmen:

Einstellung	Wert
Solution Platform	x86
Properties->Conf. Properties->C/C++->General->Warning Level	Level4 (/W4)
Properties->Conf. Properties->C/C++->General->Treat Warnings As Errors	Yes (/WX)
Properties->Conf. Properties->C/C++->General->SDL checks	Yes (/sdl)
Properties->Conf. Properties->C/C++->Code Generation->Basic Runtime Checks	Default
Properties->Conf. Properties->C/C++->Code Generation->Security Checks	Enable Security Checks (/GS)
Properties->Conf. Properties->C/C++->Language->C++ Language Standard	ISO C++ 20 Standard

Legen Sie sich eine Solution an, die alle Aufgaben als Projekte enthält.

Aufgabe 1: Laufzeitanalysen, Speichern der Resulte in Textdatei, Weiterverarbeitung in Excel: Hinweis: Hier nur mit Release, x64 als Target und Optimization = Disabled

1.1 Legen Sie ein neues Projekt an, welches nur aus einer main.cpp-Datei besteht.
In einer globalen Lookuptabelle (ggf. anfänglich kleinere Werte, z.B. mit Faktor 0.1) soll die Anzahl einzufügender Werte stehen:

```
const uint32_t cau32Lookup[] = { 1000000U, 2000000U, 3000000U, 4000000U, 5000000U,
                                 6000000U, 7000000U, 8000000U, 9000000U, 10000000U};
```

Iterieren Sie mittels auto ranged loop in main durch die Lookuptabelle. Innerhalb dieser Schleife sind vier Collections zu instanziieren.

```
std::list<uint32_t> u32list;
std::vector<uint32_t> u32vector1;
std::vector<uint32_t> u32vector2;
u32vector2.reserve(ait); //auto iterator of loop, Element of cau32Lookup
std::set<uint32_t> u32set;
```

Fügen Sie die entsprechende Anzahl von Elementen mittels einer Schleife in die vier Collections (aktueller Wert des Schleifenzählers) ein. Bei den Sequenzen ist push_back und beim Set ist insert zu verwenden.

Bei v32vector2 wird oben noch die Anzahl der Elemente angegeben, d.h. u32vector2 reserviert sich schon den Speicher, während u32vector1 bei Bedarf wachsen muss.

Für jedes Befüllen einer Collection (4 Collections, jeweils zehn unterschiedliche Anzahlen an Elementen) soll die Zeit im Microsekunden gemessen werden. Dazu ist <chrono> zu inkludieren. Die Zeitmessung sieht wie folgt aus:

```
auto astart_time = std::chrono::steady_clock::now();

// Do some work

auto aend_time = std::chrono::steady_clock::now();
std::cout << std::chrono::duration_cast<std::chrono::microseconds>
(aend_time - astart_time).count() << std::endl;
```

1.2 Nachdem die vier Collections in einem Durchlauf (äußere Schleife) befüllt wurden, soll nach dem größten Element in der Collection mittels des find-Algorithmus gesucht werden. Für das Set verwenden Sie zusätzlich noch die find-Methode des Sets. Führen jeweils eine Zeitmessung durch (1 x list, 2 x vector, 2 x set).

1.3 Die gemessenen Zeiten für das Einfügen und für das Suchen sollen in einer txt-Datei gespeichert werden. Die einzelnen Werte sollen durch Tabs getrennt werden. Dadurch ist eine Weiterverarbeitung mit Excel möglich. Es ist dazu nur noch <fstream> (f für Filestream) zu inkludieren. Das folgende Beispiel ist ohne Exception-Handling. Das Schreiben in die Datei geschieht wie bei std::cout.

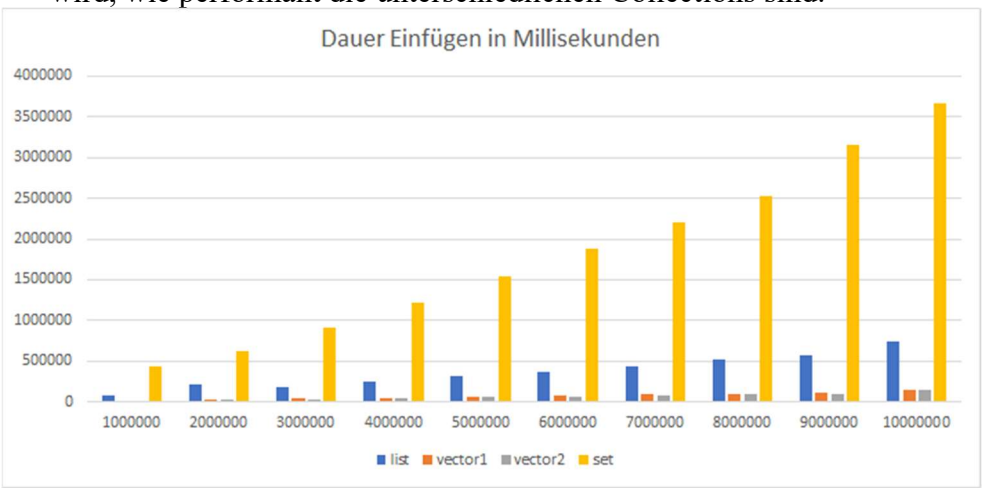
```
std::ofstream OutputFile;
OutputFile.open("PerformanceContainer.txt");

OutputFile << ait;
OutputFile << "\t" << std::chrono::duration_cast<std::chrono::microseconds>
(aend_time - astart_time).count();

OutputFile.close();
```

elements	list	vector1	vector2	set	fl_list	fl_vector1	fl_vector2	fl_set1	fl_set2
1000000	79597	12770	11158	433330	14120	1416	2205	28122	3
2000000	208177	30320	23080	630467	15871	1761	1595	31872	2
3000000	187032	37150	31749	912278	22741	2982	2530	46112	2
4000000	247942	52185	45640	1221950	32344	3598	3969	60889	2
5000000	322563	61621	53501	1542386	39969	4856	4069	74940	2
6000000	368735	77205	64522	1876759	46742	5387	5126	98079	2
7000000	438064	90426	78520	2201190	54289	5913	5932	110185	2
8000000	513654	102752	88683	2519415	65479	6780	8079	122452	2
9000000	577809	117199	101894	3163576	86319	7514	7162	171423	4
10000000	740338	151465	143617	3667334	98027	8054	8082	205927	5

1.4 Mittels Excel können Sie PerformanceContainer.txt öffnen. Excel erkennt das Tab als Trennzeichen. Sie können sich verschiedene Diagramme erzeugen, woran ersichtlich wird, wie performant die unterschiedlichen Collections sind.



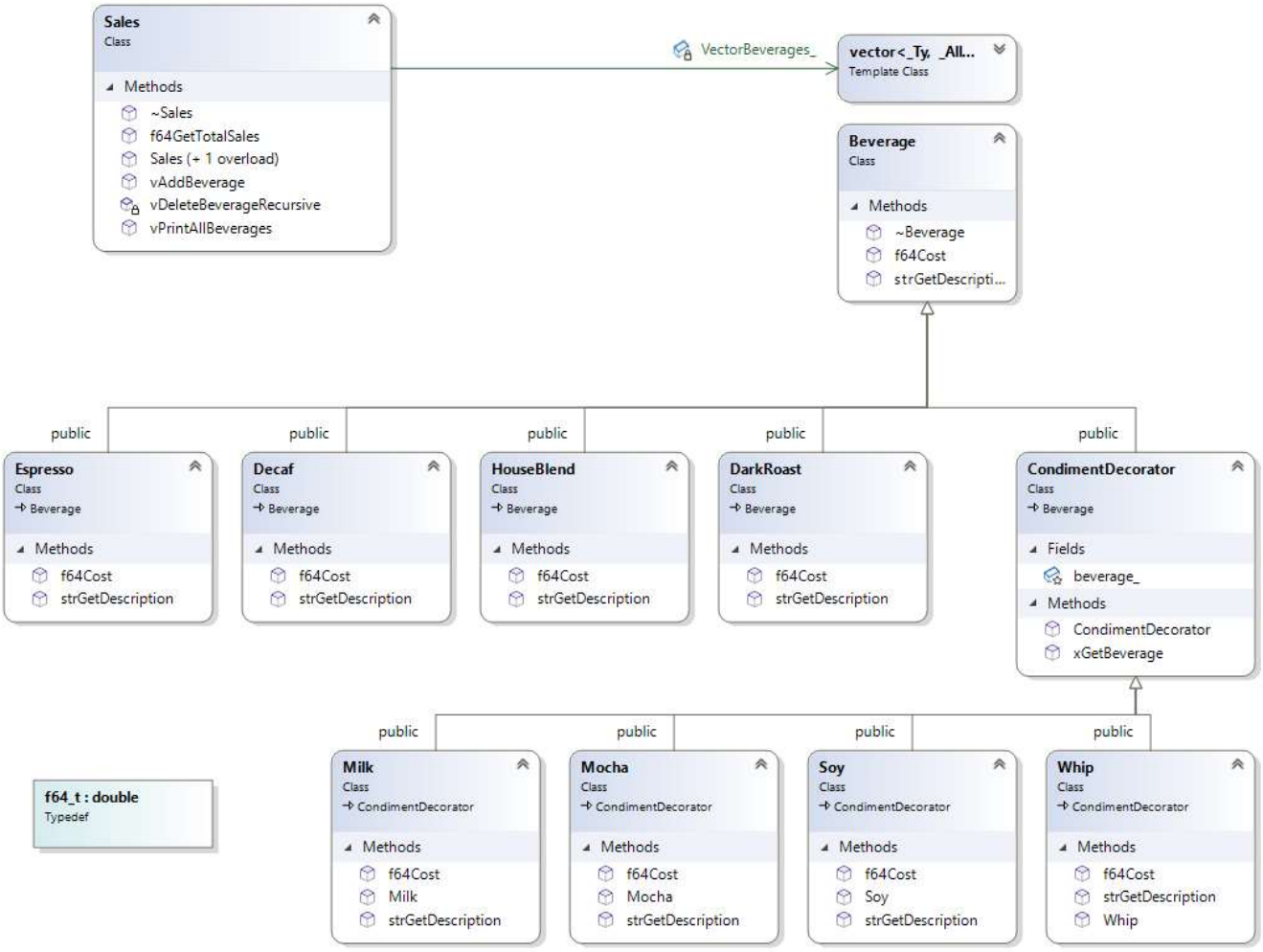
1.5 Freiwillig: Stellen Sie nun wieder auf Optimization = /Ox und vergleichen Sie die Unterschiede!

Übungen OOSWE/Progr. 2 (C++) – KE 9

Aufgabe 2: Decorator Pattern

Arbeiten Sie sich in die Thematik Decorator Pattern anhand von Kapitel 3 des Buches *Freeman, Eric; Robson, Elisabeth: Head First Design Patterns, O'Reilly, 2021* ein. Das Buch steht als E-Book in der HS-Bibliothek zur Verfügung.

Das zu erstellende C++ Programm sieht wie folgt aus:



Übungen OOSWE/Progr. 2 (C++) – KE 9

2.1 Legen Sie ein neues Projekt an. Deklarieren und implementieren Sie eine Funktion `static void vClientCode (void)`, welche von main aufgerufen wird.

```
#include <iostream>
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#include "Decorator.h"
#include "Sales.h"

static void vClientCode(void);

int main(void)
{
   vClientCode();
   _CrtDumpMemoryLeaks();

   return 0;
}

static void vClientCode(void)
{
   //Sales MySales;
   //…
}
```

2.2 Das komplette Decorator Pattern (Klasse Beverage und alle Klassen davon unterhalb im Klassendiagramm) ist nur in einer h-Datei zu realisieren (Decorator.h). Dies erleichtert den Einstieg deutlich – in der Praxis würde dieses Pattern allerdings in verschiedene cpp- und h-Dateien aufgeteilt werden. Alle Klassen des Decorator Pattern befinden sich im namespace MyDesignPattern.

Kern des Patterns bildet die abstrakte Klasse Beverage.

```
class Beverage
{
public:
   virtual ~Beverage() {}
   virtual f64_t f64Cost() const = 0;
   virtual std::string strGetDescription() const = 0;
};
```

Die konkreten Klassen Decaf, HouseBlend, DarkRoast und Espresso erben von Beverage. Die Funktion f64Cost liefert dabei nur den Preis (hardcodiert) zurück, während strGetDecription den Klassennamen zurückgibt.

```
f64_t f64Cost() const override
{
   return 1.2;
}

std::string strGetDescription() const override
{
   return "HouseBlend";
}
```

Getränk	Preis/Euro
HouseBlend	1,20
DarkRoast	2,00
Espresso	1,50
Decaf	1,60

Die Klasse CondimentDecorator erbt ebenfalls von Beverage. Diese Klasse ist ebenso abstrakt (warum?). Sie enthält nur ein protected Attribut beverage_ (Beverage*), welches im Konstruktur initialisiert wird.

```
class CondimentDecorator : public Beverage
{
public:
   CondimentDecorator(Beverage* beverage) : beverage_(beverage)
   {
   }
protected:
   Beverage* beverage_;
};
```

Alle Zutaten (Soy, Whip, Milk und Mocha) erben von CondimentDecorator. Die Konstruktoren dieser Klassen erhalten einen Zeiger auf Beverage und sollen diesen an den Konstruktor von CondimentDecorator weitergeben.

Zutaten	Preis/Euro
Milk	0,25
Mocha	0,30
Soya (Milk)	0,20
Whip	0,35

Bei einer Bestellung ergibt sich dann das folgenden exemplarische **Objektdiagramm**.

Zuerst wird das Getränk ausgewählt und danach die Zutaten hinzugefügt.

Bei der Berechnung der Kosten (f64Cost()) ruft der Clientcode die f64Cost()-Funktion in diesem Beispiel von Whip auf. Dort muss dann die f64Cost()-Funktion der nächsten Zutat aufgerufen werden. Nur das eigentliche Getränk (steht immer ganz rechts in der Objektkette) liefert in f64Cost() den Wert zurück, ohne f64Cost() des Nachfolgers aufzurufen.

Entsprechend ist mit strGetDescription zu verfahren.

```
f64_t f64Cost() const override
{
   return 0.35 + beverage_->f64Cost();
}

std::string strGetDescription() const override
{
   return "Whip " + beverage_->strGetDescription();
}
```

2.3 Die Klasse Sales ist in einer cpp- und h-Datei zu implementieren.

```
class Sales
{
public:
   Sales() = default;
   Sales(const Sales& r) = delete;
   virtual ~Sales();

   void AddBeverage(MyDesignPattern::Beverage*);
   f64_t f64GetTotalSales();
   void vPrintAllBeverages();

private:
   std::vector<MyDesignPattern::Beverage*> VectorBeverages_;

};
```

Allokieren Sie in der Funktion vClientCode statisch ein Objekt der Klasse Sales und fügen Sie dem Vector VectorBeverages_ die folgenden drei Bestellungen hinzu:

```
MyDesignPattern::Beverage* pb1 = new MyDesignPattern::Espresso();
MyDesignPattern::Beverage* pc1 = new MyDesignPattern::Whip(pb1);
MyDesignPattern::Beverage* pc2 = new MyDesignPattern::Soy(pc1);
MySales.AddBeverage(pc2);

MyDesignPattern::Beverage* pb2 = new MyDesignPattern::HouseBlend();
MyDesignPattern::Beverage* pc3 = new MyDesignPattern::Milk(pb2);
MySales.AddBeverage(pc3);

MyDesignPattern::Beverage* pb3 = new MyDesignPattern::DarkRoast();
MySales.AddBeverage(pb3);
```

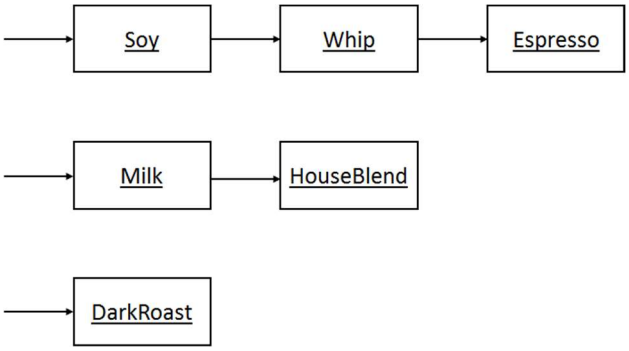
Mit vPrintAllBeverages() sollen alle Getränke nebst Zutaten sowie der Gesamtpreis ausgegeben werden. Die Funktion f64GetTotalSales liefert den gesamten Umsatz zurück. Für beide Funktion bietet sich eine auto ranged loop an.

```
MySales.PrintAllBeverages();
std::cout << "Total Sales: " << MySales.GetTotalSales() << std::endl;
```

2.4 Das Decorator Pattern setzt das **Open-Closed-Prinzip** um. Offen für Erweiterungen und geschlossen für Änderungen. Fügen Sie ein weiteres Getränk (z.B. Cocoa) und eine weitere Zutat (z.B. Sugar) hinzu! Sie werden erkennen, dass Sie die Erweiterung realisieren können, ohne die bestehenden Klassen des Decorator Pattern ändern zu müssen.

Falls Sie, wie anfänglich erwähnt, jede Klasse in einer cpp- und h-Datei realisiert hätten, müssten Sie nur jeweils zwei neue cpp- und h-Dateien hinzufügen (Cocoa.cpp/h, Sugar.cpp/h).

2.5 Nach dem Beenden des Programms sehen Sie Memory Leaks im Output-Fenster. Beheben Sie dies im Destruktor der Klasse Sales.



Ausgehend vom Programmcode in 2.3 sind im Vector VectorBeverages_ nur drei Zeiger gespeichert. Mittels

```
Sales::~Sales()
{
   for (auto it : VectorBeverages)
   {
      delete it;
   }
}
```

würden nur drei Objekte gelöscht werden (Soy, Milk, DarkRoast). Es ist eine private Memberfunktion vDeleteBeveragesRecursive in Sales zu implementieren, welche das Löschen jeder der „Objektkette“ durchführt.

Zuerst muss darin festgestellt werden, ob es sich um eine Zutat handelt. Nur dann ist die Rekursion durchzuführen, ansonsten reicht ein einfaches delete aus. Die Bestimmung, ob es sich um eine Zutat handelt, kann über verschiedene Mechanismen durchgeführt werden:

- Verwendung der typeid aus der RTTI (RunTime Type Information): Hier kann aber nur auf den konkreten Typ und nicht auf die Elternklasse überprüft werden. Alle Zutaten müssten somit abgefragt werden. Das Hinzufügen einer neuen Zutat würde somit auch die Änderung der Sales-Klasse notwendig machen (Verletzung des Open-Closed-Prinzips)
- Durchführung eines dynamic_cast auf CondimentDecorator. Ergibt dieser Cast keinen nullptr, so handelt es sich um eine Kindklasse von CondimentDecorator.

Handelt es sich um eine Zutat, so müssen Sie über eine neue Getterfunktion in CondimentDecorator den beverage_-Zeiger geben lassen. Mit diesem rufen Sie dann DeleteBeveragesRecursive auf. Danach löschen Sie das eigentliche Objekt.

2.6 Freiwillig: Es soll jetzt aber auch möglich sein, dass bei der Bestellung mehr Getränke (z.B. doppelter Espresso – andere Kombinationen sind teilweise auch sinnvoll oder völlig unsinnig) angegeben werden. Passen Sie das Design/Code entsprechend an.

2.7 Freiwillig: Selbstverständlich könnten Sie in ClientCode ein GUI realisieren.