Übungen OOSWE/Progr. 2 (C++) – KE 10

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: Intelligente Zeiger

1.1 Erstellen Sie ein neues Projekt. Deklarieren und implementieren Sie eine Funktion void TestCreateObjects (void), welche von main aufgerufen wird.

```
void TestCreateObjects (void);

int main(void)
{
   TestCreateObjects();

   return 0;
}
```

In TestCreateObjects soll ein Objekt der Klasse Complex (KE08_AG2) dynamisch allokiert werden (pComplex1). Allerdings soll auf die Freigabe des Speichers am Ende von TestCreateObjects verzichtet werden. Rufen Sie einige public-Methoden von Complex auf und verifizieren Sie die Rückgabewerte. Realisieren Sie in main die bereits mehrfach eingesetzte Überwachung von Memory Leaks. Falls noch nicht vorhanden: Fügen Sie in den Kon- und Destruktor von Complex noch unter Debug Testausgaben hinzu.

1.2 new-Operatoren sind nun durch Smart-Pointer zu ersetzen. Erstellen Sie in TestCreateObjects zwei intelligente Zeiger vom Typ std::unique_ptr für die Klasse Complex. Rufen Sie über den Pointer einige public-Methoden auf und verifizieren Sie die Rückgabewerte.

- Erstellung durch Konstruktor
- Erstellung durch std::make_unique
- Nutzen Sie das Keyword „auto“ zur Vereinfachung des Codes

Eine Freigabe des Speichers ist hier nicht notwendig.
Wann werden die beiden durch die Smart Pointer erzeugten Objekt wieder freigegeben?

1.3 new-Operatoren sind nun durch Smart-Pointer zu ersetzen. Erstellen Sie in TestCreateObjects zwei intelligente Zeiger vom Typ std::shared_ptr für die Klasse Complex

- Erstellung durch Konstruktor
- Erstellung durch std::make_shared
- Nutzen Sie das Keyword „auto“ zur Vereinfachung des Codes

1.4 Implementieren Sie zwei weitere Funktionen void vTestSharedObjects(***) mit den Übergabeparametern der Klasse Complex als

- unique_ptr
- shared_ptr

Wieso kann ein std::unique_ptr nicht an andere Funktionen übergeben werden?
Lösen Sie das Problem durch eine geringfügige Anpassung am Übergabeparameter. Überlegen Sie hierzu welche Methoden durch den Compiler aufgerufen werden müssten, und wie Sie diese Aufrufe vermeiden können.

1.5 Implementieren Sie die Funktion void vTestMoveObject().

- Erstellen Sie einen std::unique_ptr auf ein Objekt der Klasse Complex.
- Erstellen Sie einen std::shared_ptr und übernehmen Sie das Objekt des unique_ptr
- Nutzen Sie std::move

1.6 Erweitern Sie Laboraufgabe KE06-1 (CAD Drawing) durch ein Einsatz von std::shared_ptr Objekten. Kopieren Sie Ihr Projekt und fügen Sie die std::shared_ptr so ein, dass keine Memory Leaks entstehen und dennoch keine manuelle Löschung von Objekten vorgenommen werden muss. Achten Sie darauf, dass der Einsatz von Polymorphismus erhalten bleibt.

Übungen OOSWE/Progr. 2 (C++) – KE 10

Aufgabe 2: Lambda-Expressions

2.1 Implementieren Sie die Funktion TestLambda. Erstellen Sie ein einfaches Lambda-Objekt namens lambda1, mit dem Keyword „auto“, welches „Hello Lambda“ auf der Konsole ausgibt. Führen Sie Ihre Lambda-Anweisung aus.

2.2 Erstellen Sie in TestLambda ein einfaches Lambda-Objekt namens lambda2, welches zwei Integer als Übergabeparameter erwartet und die Summe der beiden Parameter zurückgibt. Geben Sie das Ergebnis, nicht direkt im Lambda, auf der Konsole aus.

2.3 Erstellen Sie in TestLambda ein einfaches Lambda-Objekt namens lambda3, welches zwei generische Übergabeparameter erwartet und das Ergebnis von operator+ der beiden Parameter zurückgibt. Geben Sie das Ergebnis, nicht direkt im Lambda, auf der Konsole aus.

Testen Sie lambda3 mit folgenden Übergabeparametern und geben Sie das Ergebnis auf der Konsole aus:

- int, int
- double, double,
- int, double
- std::string, std::string

2.4 Implementieren Sie die Funktion TestContainerLambda sowie die Struktur Circle:

```
struct Circle
{
    double dX;
    double dY;
    double dRadius;
};
```

Erstellen Sie einen std::vector für den Typ Circle und initialisieren Sie diesen mit
{0,0,0}, {10,10,10}, {-10,-10,10}, {0, 10, 20}.

2.4.1 Suchen Sie im std::vector nach dem Circle Objekt mit dem Radius > 10 durch Verwendung von std::find_if und einem passenden Lambda-Ausdruck. Achten Sie auf die Performance in Bezug auf Call-By-Value/Reference.

2.4.2 Sortieren Sie std::vector nach der Position in X aufsteigend, durch Verwendung von std::sort und einem passenden Lambda-Ausdruck. Achten Sie auf die Performance in Bezug auf Call-By-Value/Reference.

2.4.3 Definieren Sie einen weiteren std::vector<double>, welcher die X-Koordinaten der Kreise enthalten soll. Nutzen Sie zur Befüllung des Vektors die Funktion std::for_each in Kombination mit einem geeigneten Lambda-Ausdruck. Das Ergebnis ist ein Positionsvektor, welcher lediglich die X-Komponente aller Kreise enthält.

2.4.4 Geben Sie den Inhalt des Positionsvektors aus 2.4.3 auf der Konsole aus, in dem Sie std::for_each mit einem geeigneten Lambda-Ausdruck verwenden.

2.5 Implementieren Sie die Funktion void ExecuteSort, die einen Lambda-Ausdruck übergeben bekommt und diesen ausführt. Keine weiteren Übergabeparameter oder Rückgabewerte erlaubt.

Definieren Sie den Lambda-Ausdruck in TestContainerLambda wie folgt:
- Sortieren Sie den std::vector<Circle> nach Radius aufsteigend.
- Hängen Sie den zu sortierenden Vektor an Ihren Lambda-Ausdruck an.
- Daten und Code werden somit als Lambda-Ausdruck gemeinsam übergeben.

Definieren Sie den Lambda-Ausdruck direkt im Funktionsaufruf von ExecuteSort. Es wird kein namentlich benanntes Lambda-Objekt benötigt.

Geben Sie mittels std::for_each alle sortierten Kreise in TestContainerLambda auf der Konsole aus. Vergewissern Sie sich, dass die Sortierung erfolgreich war und der Vektor nicht bereits vorher sortiert gewesen ist.

2.6 Implementieren Sie die Funktion void SortParallel(void). Kopieren Sie Ihren std::vector<Circle> aus 2.5 und fügen diesen ein. Sortieren Sie den Vektor erneut durch Einsatz des anonymen Lambdas aus 2.5. Anstelle der Funktion ExecuteSort wird allerdings ein neuer Thread erzeugt, welcher das Lambda parallel ausführt.

```
// run thread (automatic run by creation of thread object)
std::thread t( LAMBDA );

// await thread return / finish
if (t.joinable())
    t.join();
```

Geben Sie das sortierte Array erneut aus um zu überprüfen, ob die Sortierung erfolgreich war. Herzlichen Glückwunsch, Sie haben soeben eine Aufgabe in einem parallelen Thread ausgeführt, ohne großen Aufwand.