Übungen OOSWE/Progr. 2 (C++) – KE 8

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: Erstellung von einem Klassentemplate für ein Array

1.1 Legen Sie ein neues Projekt an, welches ein Template **MyArray** für ein Array (Implementierung nur in einer Headerdatei TemplateArray.h) im Namespace MyTemplates enthält. An dieses Klassentemplate werden die folgenden Anforderungen gestellt:

- Private Membervariable für die maximale Größe (siehe Script)
- Privater Zeiger auf den Inhalt (siehe Script)
- Konstruktor, welcher die Arraygröße erhält und den notwendigen Speicher allokiert (siehe Script)
- Destruktor (siehe Script)
- Ändern der Größe mittels einer Methode Resize – Elemente bleiben ggf. erhalten! Beachten Sie das Vergrößern und Verkleinern der Anzahl der Elemente. Exception Handling optional.
- Beschreiben und Auslesen der Elemente mit Operatorfunktion [](siehe Script) – Werfen Sie noch eine Exception, falls der übergebene Index nicht korrekt ist:
  `throw invalid_argument("Index out of range");`
- Methode, welche die aktuelle Arraygröße zurückgibt.
- Überschreiben Sie auch den <<-Operator für cout. Dazu sollten Sie vor der Templateklasse MyArray (im namespace MyTemplates) die Templatemethode definieren, welche von MyArray als friend angesehen wird.

```
//forward declaration of template class
template<class T, uint32_t u32Size>
class MyArray;

//forware declaration of friend method
template <class T, uint32_t u32Size>
std::ostream& operator<<(std::ostream& ostr, const MyArray<T, u32Size>& crT);
```

In MyArray ist dann noch am Ende

```
friend std::ostream& operator<<<T, u32Size>(std::ostream& ostr, const
MyArray<T, u32Size>& cref);
```

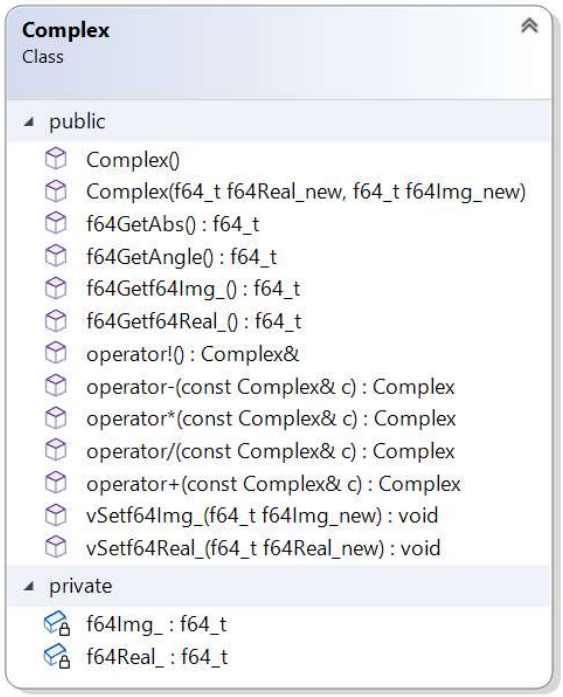
hinzuzufügen.

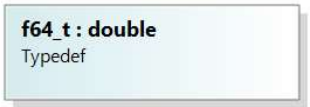
1.2 Wenden Sie dieses Template für Integer an (in TestCreateObjects).

- Anlegen eines Objektes der Templateklasse für int32_t (5 Elemente). Name des Objektes sei MyIntArray.
- Befüllen des Arrays mit der Operatorfunktion []
- Ausgabe des Objektes mit z.B. `std::cout << MyIntArray << std::endl;`
- Vergrößern des Objektes mit Resize (Größe 7), überschreiben der Elemente mit dem Index 5-6 und Ausgabe des Objektes.
- Verkleinern des Objektes myIntArray mit Resize (Größe 3) und Ausgabe des Objektes

1.3 Wenden Sie dieses Template für Complex an (in TestCreateObjects). Verwenden Sie Complex aus KE07_AG2. Die Klasse Complex muss noch angepasst werden, damit diese für MyArray angewendet werden kann. Der Compiler gibt Ihnen einen Hinweis ;-)

- Anlegen eines Objektes der Templateklasse für Complex (5 Elemente). Name des Objektes sei MyComplexArray
- Befüllen des Arrays mit der Operatorfunktion []
- Ausgabe des Objektes mit z.B. `std::cout << MyComplexArray << std::endl;`
- Vergrößern des Objektes mit Resize (Größe 7), überschreiben der Elemente mit dem Index 5-6 und Ausgabe des Objektes.
- Verkleinern des Objektes MyComplexArray mit Resize (Größe 3) und Ausgabe des Objektes

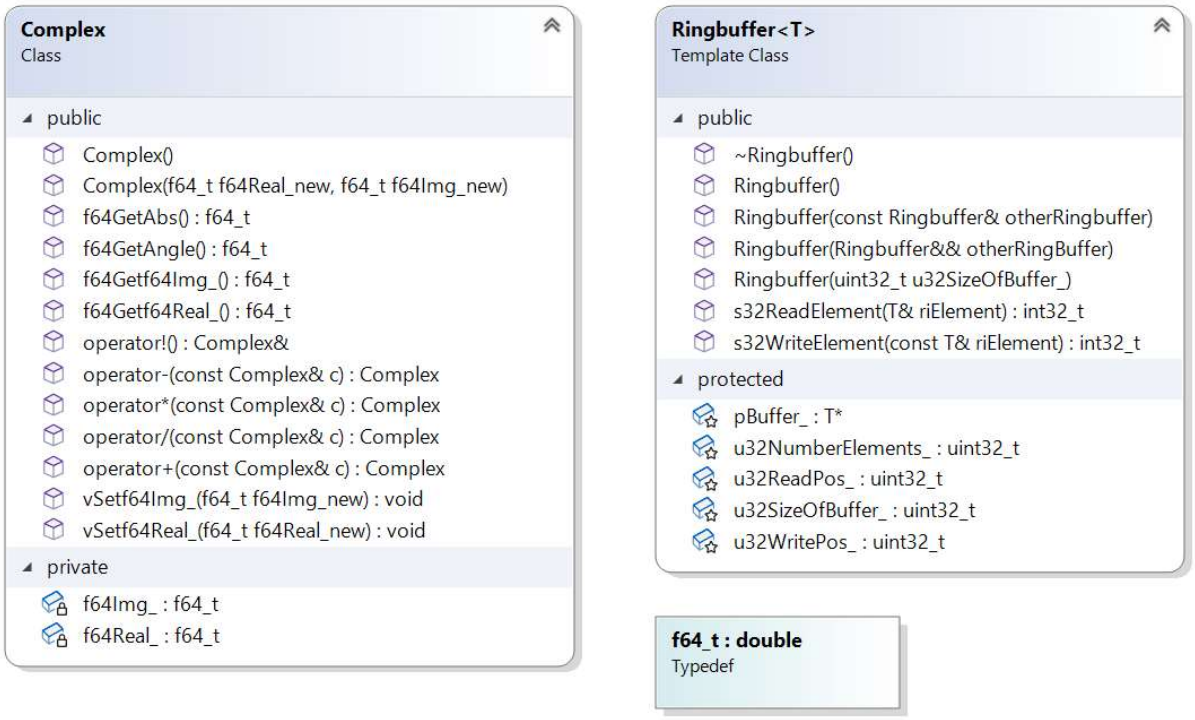




1.4 Überprüfen Sie in main auf Memory Leaks!

Aufgabe 2: Ringbuffer als Template

2.1 Legen Sie sich ein neues Projekt an und kopieren Sie sich die Klasse Complex aus KE08_AG1 hinzu. Legen Sie sich eine neue Headerdatei „RingbufferTemplate.h“ an. Erstellen Sie darin eine Templateklasse für Ringbuffer indem Sie aus KE05_AG3 die Dateien Ringbuffer.cpp und Ringbuffer.h fusionieren. Entfernen Sie darin noch die Deklaration der friend-Funktion.



2.2 Instanziieren Sie einen Ringbuffer für Integer (int32_t) in main. Schreiben Sie zwei Integerzahlen mittels WriteElement in den Ringbuffer und lesen Sie mit ReadElement das älteste Element aus. Geben Sie anschließend dieses Element aus.

2.3 Instanziieren Sie nun einen Ringbuffer für Complex in main (namespace nicht vergesssen). Verwenden Sie die Klasse Complex aus KE08_AG1. Schreiben Sie zwei komplexe Zahlen (Complex) in den Ringbuffer und lesen Sie mit ReadElement das älteste Element aus. Geben Sie anschließend dieses Element aus.

Freiwillig:
2.4 Erweitern Sie das Ringbuffer-Template so, dass Sie den <<-Operator überladen. Hier können Sie auch die bisherigen Features einbauen: 32ReadPos_ und u32WritePos_-Ausgabe beim entsprechenden Element (Tabs etc.). Als Vorlage können Sie KE08_AG1 verwenden.