

✍ Übungen OOSWE/Progr. 2 (C++) – KE 3

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	Both
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.

Die geforderten Kommentare sind in Englisch zu hinterlegen.

Aufgabe 1:

Implementieren Sie ein Programm, welches die Klasse Ringbuffer aus KE 2 wiederverwendet. Legen Sie sich dazu das Projekt KE03_AG1 an.

- Main.cpp
- Ringbuffer.h (Kopieren Sie diese Datei aus KE 2 in das neue Projektverzeichnis)
- Ringbuffer.cpp (Kopieren Sie diese Datei aus KE 2 in das neue Projektverzeichnis)

Fügen Sie die beiden Ringbuffer-Dateien mittels *Add* und *Existing Item..* dem Projekt hinzu.

1.1 Fügen Sie an den passenden Stellen `const` und `noexcept` bei den Funktionen ein. `const` muss vor `noexcept` stehen.

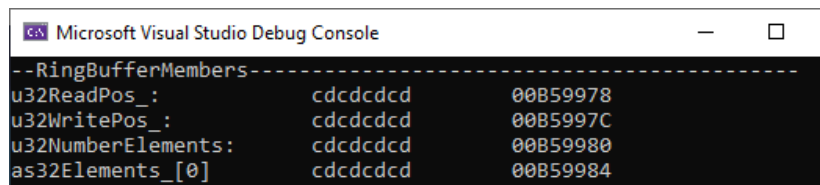
1.2 Kommentieren Sie den leeren Konstruktor aus. Jetzt wird der default-Konstruktor bei der Instanziierung des Objektes verwendet. Um dies zusätzlich kenntlich zu machen, sollten Sie bei der Deklaration der Klasse folgende Ergänzung durchführen:

```
Ringbuffer(void) = default;
```

Implementieren Sie eine weitere public Memberfunktion:

```
void vPrintRingbufferMembersValueAndAddress(void) const noexcept;
```

welche von allen Membervariablen (Attribute) den Wert und die Adresse auf der Konsole ausgibt. Beim Array `as32Elements_` hat dies nur für `as32Elements_[0]` zu erfolgen. Die Werte in der nachfolgenden Abbildung können je nach RTC-Einstellungen abweichen (cccccccc).



```
Microsoft Visual Studio Debug Console
--RingBufferMembers-----
u32ReadPos_:             cdcdcdcd      00B59978
u32WritePos_:            cdcdcdcd      00B5997C
u32NumberElements:       cdcdcdcd      00B59980
as32Elements_[0]         cdcdcdcd      00B59984
```

Damit die Wert hexadezimal ausgegeben werden, ist dies am Funktionsanfang mittels `std::cout << std::hex;` zu setzen und am Funktionsende wieder auf dezimal zurückzusetzen.

```
void Ringbuffer::vPrintRingbufferMembersValueAndAddress(void) const noexcept
{
    std::cout << std::hex;    /* more code */
    std::cout << std::dec;
}
```

Übungen OOSWE/Progr. 2 (C++) – KE 3

1.3 Deklarieren und implementieren Sie eine Funktion `void vTestCreateObjectsRingbuffer(void)`, welche von `main` aufgerufen wird.

```
void vTestCreateObjectsRingbuffer(void);

int main(void)
{
    vTestCreateObjectsRingbuffer();

    return 0;
}
```

Allokieren Sie jeweils zwei Objekte der Klasse `Ringbuffer` **statisch** oberhalb von `main` (global) und zwei Objekte innerhalb von `TestCreateObjectsRingbuffer` (lokal). Der einzige Unterschied soll dabei sein, dass einmal ein `()` hinter dem Objektnamen steht und das andere Mal nicht (*Brace yourself!* soll hier bewusst **nicht** beachtet werden, um den Most Vexing Parse zu sehen).

Allokieren Sie zwei Objekte der Klasse `Ringbuffer` **dynamisch** in `vTestCreateObjectsRingbuffer`. Hier ist ebenso einmal ein `()` hinter den Klassennamen einzufügen und das andere Mal nicht (auch kein `{ }`).

Augenscheinlich sollen sechs Objekte instanziiert werden, jedoch sind zwei Instanzierungen falsch. Um welche handelt es sich? Kommentieren Sie deren Instanzierungen aus.

Schreiben Sie über `main` als Kommentar wie der Compiler die vermeintlichen Instanzierungen interpretiert.

- Rufen Sie in `vTestCreateObjectsRingbuffer` die `vPrintRingbufferMembersValueAndAddress`-Methode der vier Objekte auf. Geben vorher mit `std::cout` noch den Namen der Objektvariablen bzw. des Zeigers aus.
- Geben Sie anschließend auf der Konsole die Größe in Bytes der jeweiligen Objekte aus.
- Geben Sie anschließend die Anfangsadresse der Objekte auf dem Bildschirm aus. Anhand der unterschiedlichen Anfangsadressen können Sie darauf schließen, in welchem Speicherbereich die Objekte sich befinden. Beschreiben Sie oberhalb von `main` als Kommentar wo welche Objekte sich im Speicher befinden.

1.4 Jetzt sollen Memory Leaks gefunden werden. Implementieren Sie eine Überwachung wie bereits in Ing.-Inf. beschrieben wurde („Check“ vor `return 0` in `main`). Nach der Programmausführung finden Sie einen „HexDump“ dieser Memory Leaks im Output-Fenster der IDE. Beheben Sie im Code die Memory Leaks.

1.5 Bei Ausführung des Programms werden die Werte der Membervariablen (Attribute) auf der Konsole ausgegeben. Erkennen Sie Unterschiede/Muster? (kein `{ }` bei der Instanzierung verwenden)

Builden Sie jetzt die Anwendung als Release und führen Sie die Anwendung aus.

Beschreiben Sie tabellarisch als Kommentar über `main` die Unterschiede bezüglich der Initialisierung der Variablen.

1.6 Welche Methoden und Operatoren stehen bei einer Klasse am Anfang automatisch durch den Compiler zur Verfügung? Beschreiben Sie oberhalb von `main` als Kommentar Methoden und Operatoren. Siehe: Abbildung 4.8, Josh Lospinoso: C++ Crash Course, no starch press, 2019

In dieser Aufgabe wurde eine Klassenimplementierung realisiert, die nur den leeren Default-Konstruktor verwendet. Dadurch ergeben sich Unterschiede auch im Hinblick auf Debug und Release.

Übungen OOSWE/Progr. 2 (C++) – KE 3

Aufgabe 2:

Implementieren Sie ein Programm, welches die Klasse Ringbuffer aus KE03_AG1 wiederverwendet. Legen Sie sich dazu das Projekt KE03_AG2 an. **Stellen Sie nur für Target **Debug** Properties->Conf. Properties->C/C++->Code Generation->Basic Runtime Checks auf **Default**.**

- Main.cpp
- Ringbuffer.h (Kopieren Sie diese Datei aus KE03_AG1 in das neue Projektverzeichnis)
- Ringbuffer.cpp (Kopieren Sie diese Datei aus KE03_AG1 in das neue Projektverzeichnis)

Fügen Sie die beiden Ringbuffer-Dateien mittels *Add* und *Existing Item..* dem Projekt hinzu.

2.1 Fügen Sie der Klasse ein weiteres Attribut (`u32SizeOfBuffer_`) hinzu. Auch soll das Array jetzt dynamisch mit `new` unter Nutzung von `u32SizeOfBuffer_` initialisiert werden. Dabei soll `u32SizeOfBuffer_` an den Konstruktor übergeben werden. Erstellen Sie dazu ein Konstruktor, welcher alle Werte in einer Initialisierungsliste initialisiert und innerhalb des Konstruktors dann dynamisch das Array mit **new** allokiert (`ps32Buffer_`). Die Elemente des Arrays sollen mit 0 initialisiert sein:

```
ps32Buffer_ = new int32_t[u32SizeOfBuffer_]; // without zeroing - How to zeroing?
```

Passen Sie die bereits existierenden Methoden an. In Ihrer Klasse gibt es jetzt nur noch einen Konstruktor!

Fügen Sie Sanity-Checks für `ps32Buffer_` (nullptr) in den Methoden von Ringbuffer ein. Die Methoden `s32WriteElement` und `s32ReadElement` sollen bei einem nullptr (`ps32Buffer_`) den folgenden Wert zurückgeben:

```
#define RINGBUFFER_NULLPTR    -3
```

Deklarieren und implementieren Sie eine Funktion `void vTestCreateObjectsRingbuffer(void)`, welche von main aufgerufen wird.

```
void vTestCreateObjectsRingbuffer(void);
int main(void)
{
    vTestCreateObjectsRingbuffer();

    return 0;
}
```

Legen Sie statisch und dynamisch ein Ringbuffer-Objekt in `TestCreateObjectsRingbuffer` an. Verifizieren Sie die Funktionsweise des Ringbuffers mittels Aufruf der Methoden `s32ReadElement`, `s32WriteElement`, `vPrintRingBuffer`, `vPrintRingbufferMembersValueAndAddress`.

Sind jetzt Memory Leaks vorhanden? Implementieren Sie eine Überwachung wie bereits in Ing.-Inf. beschrieben wurde am Ende von main. Nach der Programmausführung finden Sie einen „HexDump“ dieser Memory Leaks im Output-Fenster der IDE.

Um die Memory Leaks zu beseitigen, muss der im Konstruktor allokierte dynamische Speicher im Heapsegment wieder freigegeben werden. Dies hat im Destruktor zu erfolgen. Implementieren Sie einen Destruktor, der den Speicher wieder freigibt, falls `ps32Buffer_` ungleich nullptr ist.

```
delete[] piBuffer; // Deletes arrays of objects
```

Dieses Pattern wird auch CADRe genannt: Constructor Acquires – Destructor Releases
Alternatives Acronym lautet RAI: Resource Acquisition Is Initialization

Übungen OOSWE/Progr. 2 (C++) – KE 3

Versuchen Sie ein Objekt mit dem leeren Standard-Konstruktor zu instanziiieren! Schreiben Sie Ihre Erkenntnis als Kommentar über main.

Übungen OOSWE/Progr. 2 (C++) – KE 3

2.2 Implementieren Sie den **Copy-** und den **Move-Constructor** für die Klasse Ringbuffer. Der **Copy-Constructor** soll eine **tiefe Kopie** erstellen. Der **Move-Constructor** übergibt alle „Elemente“ an das neue Objekt und setzt die ursprünglichen Werte auf 0 bzw. nullptr zurück. Dieser Move-Constructor ist dann sinnvoll, wenn vermieden werden soll, dass aufwendig initialisierte Objekte wieder gelöscht und an anderer Stelle wieder erzeugt werden sollen.

Copy-Constructor

```
Ringbuffer::Ringbuffer(const Ringbuffer& rotherRingbuffer)
{
    // ToDo
}
```

Move-Constructor

```
Ringbuffer::Ringbuffer(Ringbuffer&& rotherRingbuffer) const noexcept
{
    // ToDo
}
```

Der Copy- und der Move-Constructor werden bei der Erzeugung neuer Objekte aufgerufen:

```
// Copy Constructor is called
MyDataStructures::Ringbuffer RingBuffer2(RingBuffer1);

// Move Constructor is called
MyDataStructures::Ringbuffer RingBuffer3(std::move(RingBuffer1));
```

Verifizieren Sie mit Tests, dass der Copy- und der Move-Constructor funktioniert.

Beschreiben Sie oberhalb von main, warum der Copy-Constructor eine tiefe Copy erstellen muss.

Für Testzwecke und zum tieferen Verständnis der Reihenfolge beim Instanzieren und Löschen von Objekten bietet es sich an, in allen Konstruktoren und Destruktoren auch zukünftig ein `std::cout` einzubauen. Idealerweise kapseln Sie diese Ausgabe mit bedingter Compilierung, indem Sie `_DEBUG` abfragen. Bei `_DEBUG` handelt es sich um eine Definition des Präprozessors, welche bei der Debug Configuration automatisch gesetzt ist. Die Ausgabe soll nur in der Debug Configuration erfolgen.