

Übungen OOSWE/Progr. 2 (C++) – KE 2

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:

Implementieren Sie ein Programm, welches eine Klasse Complex realisiert. Ein Objekt dieser Klasse soll eine komplexe Zahl z repräsentieren. Legen Sie sich hierzu ein neues Projekt (KE02_AG1) mit den folgenden Dateien an:

- Main.cpp (enthält nur die main-Funktion)
- Complex.h (enthält die Deklaration der Klasse Complex – im Namespace „MyMath“)
- Complex.cpp (enthält die Implementierung der Klasse Complex – using namespace MyMath sollte vereinfachend vor der Implementierung erfolgen)

Die Klasse Complex soll zwei Membervariablen (Attribute) haben: `f64_t f64Real_` und `f64_t f64Img_`. Auf beide Attribute soll von außen nicht zugegriffen werden können. Implementieren Sie die folgenden Getter und Setter-Funktionen (public), welche die Membervariablen beschreiben und lesen können: `f64_t` ist dabei ein eigener Typedef (C/C++ Coding Styleguide, DV2 ES).

- `f64_t f64Getf64Real(void)`
- `void vSetf64Real(f64_t f64Realnew)`
- `f64_t f64Getf64Img(void)`
- `void vSetf64Img(f64_t f64Imgnew)`

Ebenso sollen zwei weitere Getter-Funktionen realisiert werden, die den Betrag und den Winkel einer komplexen Zahl zurückgeben. Verwenden Sie hierzu die `<cmath>` Bibliothek (Vergessen Sie nicht `_USE_MATH_DEFINES` vor dem include von `<cmath>`).

- `f64_t f64GetAbs(void)` -> gibt den Betrag zurück
- `f64_t f64GetArg(void)` -> gibt den Winkel der komplexen Zahl in Grad zurück.

Eine weitere Funktion in Complex ist zu implementieren, welche den Wert der komplexen Zahl auf der Konsole ausgibt (z.B. „Cartesian Coordinates: $3 + j(-4)$ | Polar Coordinates: $5 * \exp(j(-53.13))$ “). Nutzen Sie hierzu `cout` aus `<iostream>`. Verwenden Sie **nicht**: `using namespace std`;

- `void vPrintComplexNumber(void)`

Es soll auch ein eigener Konstruktor in Complex zur Verfügung gestellt werden, der zwei Initialwerte für `f64Real_` und `f64Img_` übergeben bekommt.

Allokieren Sie sich statisch in main eine Instanz der Klasse Complex unter Nutzung des Namespaces.

`MyMath::Complex Z1(3.,4.);`

- Geben Sie mit `vPrintComplexNumber` die komplexe Zahl aus.
- Rufen Sie alle Getter-Funktionen auf und verifizieren Sie den richtigen Rückgabewert.
- Beschreiben Sie Membervariablen mit den Setter-Funktionen mit den vier Testfällen und geben Sie danach mit `vPrintComplexNumber` die komplexe Zahl aus.

Testfall 1: `f64Real_ = -1.0 f64Img_ = -1.0` Testfall 2: `f64Real_ = 2.0 f64Img_ = -2.0`

Testfall 3: `f64Real_ = 0.0 f64Img_ = -2.0` Testfall 4: `f64Real_ = 0.0 f64Img_ = 0.0`

Übungen OOSWE/Progr. 2 (C++) – KE 2

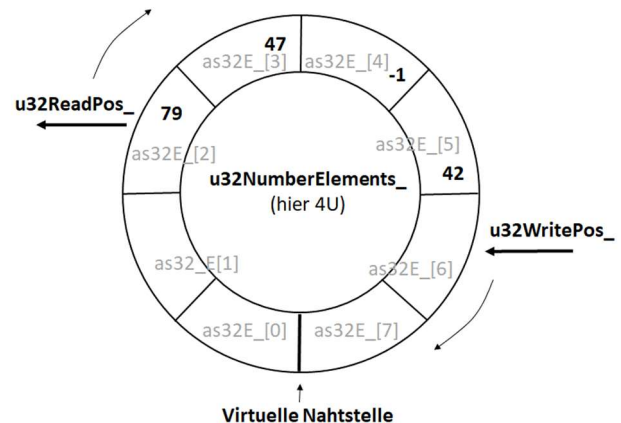
Aufgabe 2:

Implementieren Sie ein Programm, welches eine Klasse Ringbuffer implementiert. Ein Objekt dieser Klasse soll die Datenstruktur eines Ringbuffers repräsentieren. Legen Sie sich hierzu ein neues Projekt (KE02_AG2) mit den folgenden Dateien an:

- Main.cpp (enthält nur die main-Funktion)
- Ringbuffer.h (enthält die Deklaration der Klasse Ringbuffer)
- Ringbuffer.cpp (enthält die Implementierung der Klasse Ringbuffer – using namespace MyDataStructures)

Die Klasse Ringbuffer soll sich im Namespace „MyDataStructures“ befinden.

Ein Ringbuffer ist ein FIFO (First-In-First-Out). Intern hat ein Ringbuffer ein Array, welches die Werte enthält. Das Array `as32Elements_` (kurz `as32E_`) ist sozusagen virtuell zu einem Ring gebogen. Dieser Ringbuffer soll als Elemente `int32_t`-Werte enthalten. Mittels zweier Merker (`u32ReadPos_` und `u32WritePos_`) wird sich die aktuelle Schreib- und Leseposition gemerkt. In einer weiteren Membervariablen `u32NumberElements_` wird die Anzahl der gespeicherten Elemente abgespeichert. Mit jedem Schreib- und Lesevorgang werden der jeweilige Merker (`u32ReadPos_` oder `u32WritePos_`) im Uhrzeigersinn verschoben.



Es soll auch ein eigener Konstruktor in Ringbuffer zur Verfügung gestellt werden, der `u32ReadPos_`, `u32WritePos_` und `u32NumberElements_` mit 0U initialisiert. Ebenso ist `as32Elements_` in einer Schleife mit dem Wert 0 zu befüllen.

Für Rückgabewerte sollen die folgenden symbolischen Konstanten in Ringbuffer.h deklariert werden:

```
#define RINGBUFFER_OKAY      0
#define RINGBUFFER_FULL     -1
#define RINGBUFFER_EMPTY    -2
```

Die Größe von `as32Elements_` soll ebenso mit einer symbolischen Konstanten in Ringbuffer.h festgelegt werden:

```
#define RINGBUFFER_MAXELEMENTS 8
```

Alle Membervariablen (Attribute) sollen privat sein.

Implementieren Sie die Member-Funktionen

- `int32_t s32ReadElement(int32_t& rs32Element)`
- `int32_t s32WriteElement(const int32_t& rs32Element)`
- `void vPrintRingBuffer(void)` (Gibt alle Membervariablen auf dem Bildschirm aus)

Mögliche Rückgabewert sind die obigen symbolischen Konstanten. Referenzen (daher das r vor dem s32) werden dabei übergeben.

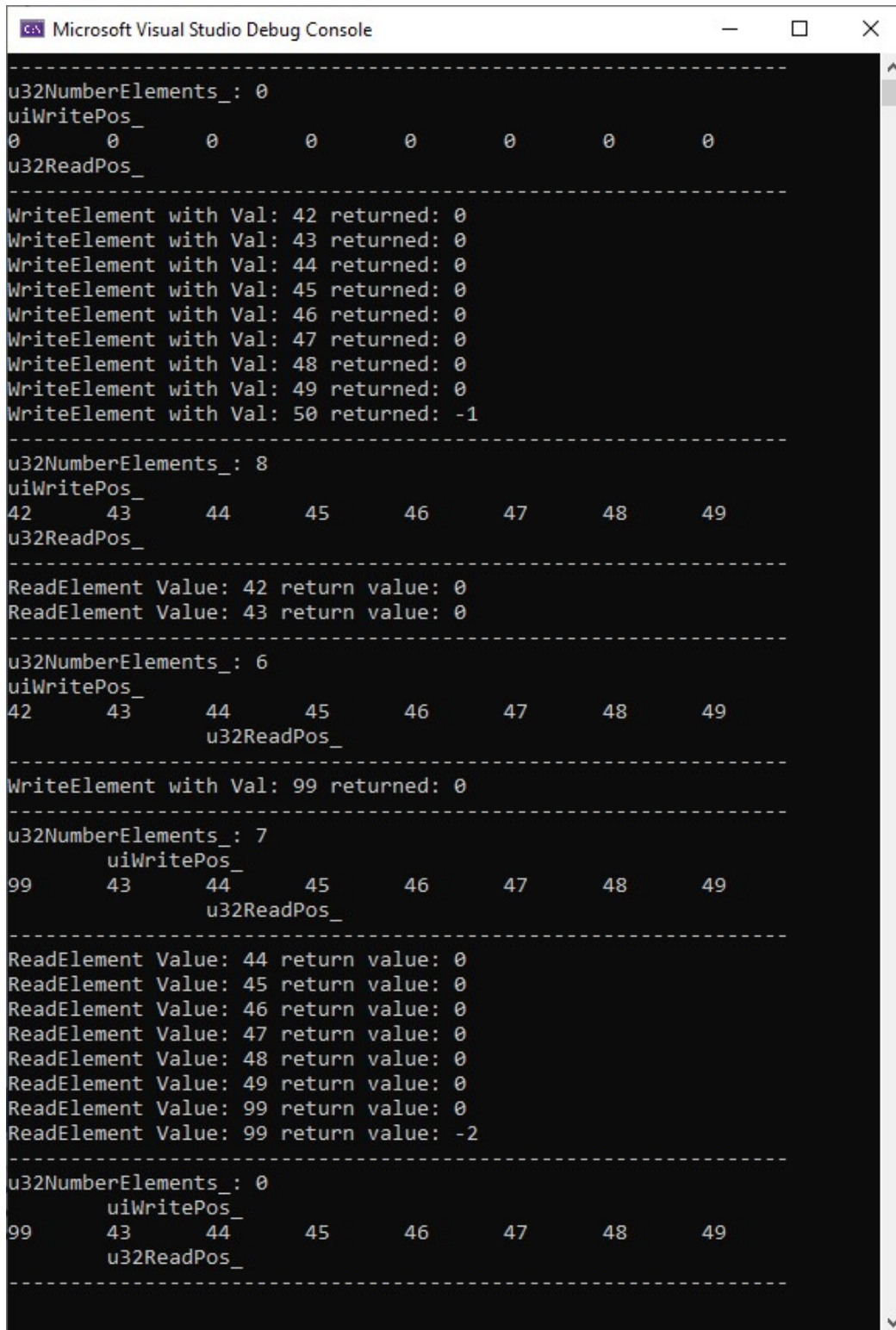
Allokieren Sie sich statisch in main eine Instanz der Klasse Ringbuffer unter Nutzung des Namespaces.

Testen Sie den Ringbuffer, indem Sie mehrfach die obigen Member-Funktionen aufrufen und den Rückgabewert sowie bei `s32ReadElement_` und `s32WriteElement_` noch den Wert der übergebenen Variablen (Referenz) sich auf der Konsole ausgeben. Auch das Verhalten bezüglich der virtuellen Nahtstelle ist zu überprüfen.

Übungen OOSWE/Progr. 2 (C++) – KE 2

Tipp: Beim Verschieben der beiden Marker bietet sich der Modulo-Operator zur Handhabung der virtuellen Nahtstelle an. Diese dürfen nie größer als 7U sein!

Folgende Abbildung zeigt eine mögliche Realisierung der Ausgaben. Die Funktion `vPrintRingbuffer` gibt beispielsweise den Block beginnend und endend mit `-----` aus. Die Position der Texte „`u32WritePos_`“ und „`u32ReadPos_`“ ergibt sich unter Verwendung von Tabs (`\t`) und dem entsprechenden Variablenwert.



```
Microsoft Visual Studio Debug Console

-----
u32NumberElements_: 0
uiWritePos_
0      0      0      0      0      0      0      0
u32ReadPos_
-----

WriteElement with Val: 42 returned: 0
WriteElement with Val: 43 returned: 0
WriteElement with Val: 44 returned: 0
WriteElement with Val: 45 returned: 0
WriteElement with Val: 46 returned: 0
WriteElement with Val: 47 returned: 0
WriteElement with Val: 48 returned: 0
WriteElement with Val: 49 returned: 0
WriteElement with Val: 50 returned: -1
-----

u32NumberElements_: 8
uiWritePos_
42      43      44      45      46      47      48      49
u32ReadPos_
-----

ReadElement Value: 42 return value: 0
ReadElement Value: 43 return value: 0
-----

u32NumberElements_: 6
uiWritePos_
42      43      44      45      46      47      48      49
u32ReadPos_
-----

WriteElement with Val: 99 returned: 0
-----

u32NumberElements_: 7
uiWritePos_
99      43      44      45      46      47      48      49
u32ReadPos_
-----

ReadElement Value: 44 return value: 0
ReadElement Value: 45 return value: 0
ReadElement Value: 46 return value: 0
ReadElement Value: 47 return value: 0
ReadElement Value: 48 return value: 0
ReadElement Value: 49 return value: 0
ReadElement Value: 99 return value: 0
ReadElement Value: 99 return value: -2
-----

u32NumberElements_: 0
uiWritePos_
99      43      44      45      46      47      48      49
u32ReadPos_
-----
```