HOCHSCHULE RUHR WEST
UNIVERSITY OF APPLIED SCIENCES
INSTITUT INFORMATIK

23.09.2025

Klausur

MMI und GUI-Programmierung

Sommersemester 2025

Studiengänge: AI, MTI, WI

Prüfer: Prof. Dr. Malte Weiß

Persönliche Angaben

Nachname	Vorname	Matrikelnummer

Bewertung

Frage	Punkte	Erreicht
1	9	
2	4	
3	4	
4	6	
5	3	
6	8	
7	10	

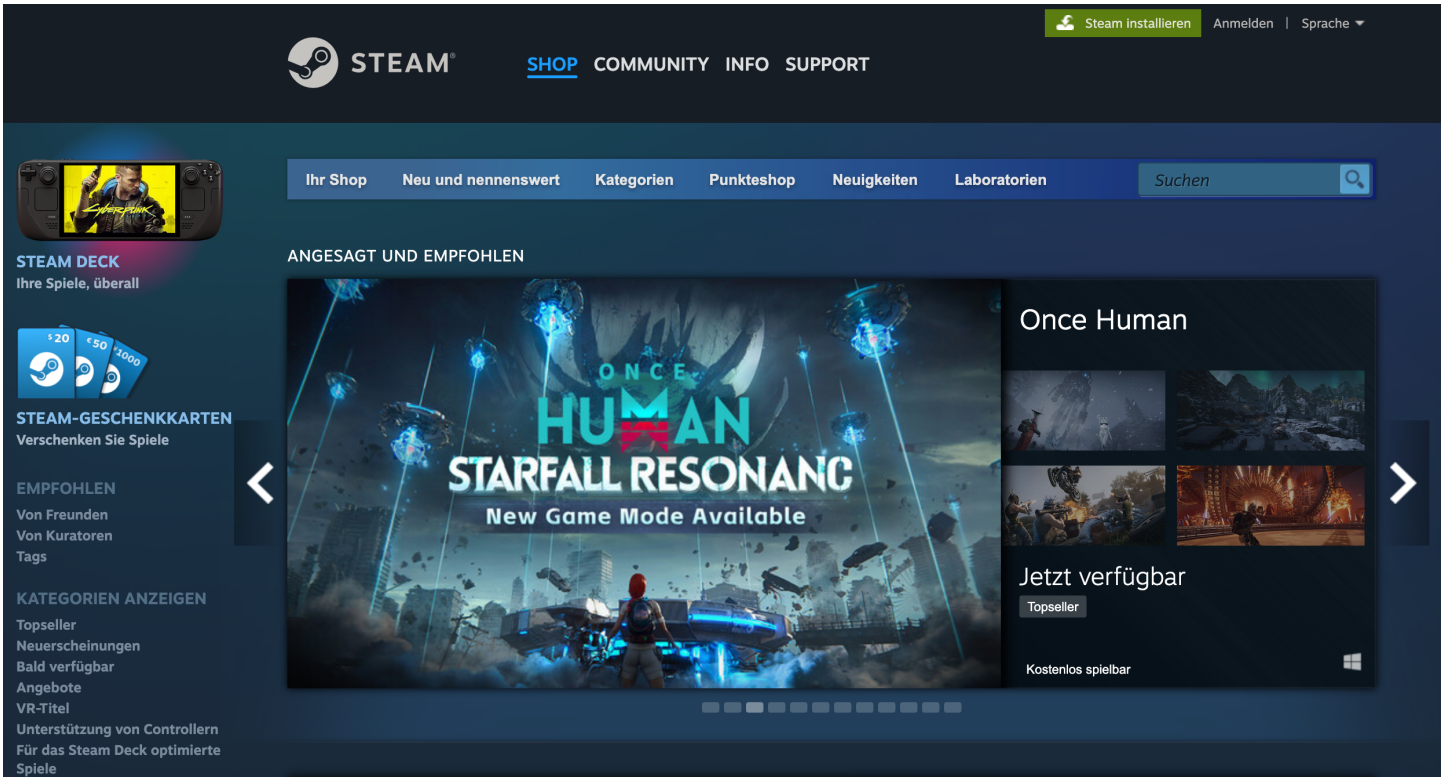
Frage	Punkte	Erreicht
8	5	
9	12	
10	4	
11	4	
12	26	
13	5	
Gesamt:	100	

Ablauf der Prüfung

- **Zeit für die Lösung dieses Aufgabenblatts**: 120 Minuten.
- **Erlaubte Hilfsmittel**: Cheat Sheet am Ende der Klausur. Handgeschriebener, ein- oder beidseitig beschriebener Notizzettel im DIN-A4-Format.
- **Nicht erlaubte Hilfsmittel**: Mobiltelefon, Kommunikation (E-Mail, Web, SMS, etc.). Die Verwendung eines nicht erlaubten Hilfsmittels wird als Täuschungsversuch gewertet.
- **Stift und Papier**: Schreiben Sie ausschließlich auf dem Papier der Aufgabenblätter. Sollten die Zwischenräume nicht ausreichen, können Sie zusätzliches Papier von der Aufsicht erhalten. Beschriften Sie dieses sofort mit ihrem Namen und Ihrer Matrikelnummer. Eigenes Papier ist nicht zulässig. Schreiben Sie ausschließlich mit dokumentenechten schwarzen oder blauen Stiften.
- **Vollständigkeit des Aufgabenblatts**: Die Klausur besteht aus den Aufgabenseiten 5 – 23 und einer Funktionsreferenz auf Seite R1. Die Aufgabenblätter sind doppelseitig bedruckt. Prüfen Sie, ob alle Blätter vorhanden sind.
- **Fragen**: Melden Sie sich, wenn Sie eine Frage haben. Die Aufsicht kommt zu Ihnen, verlassen Sie nicht unaufgefordert Ihren Platz.
- **Zwischenschritte**: Geben Sie Zwischenschritte an. So können Sie selbst bei falschem Endergebnis einen Teil der Punkte erreichen.

Mensch-Maschine-Interaktion

1. Betrachten Sie den folgenden Screenshot einer Website: (9)



Nennen Sie **drei** Gestaltgesetze, die hier eingesetzt werden, und beschreiben Sie, welchen Effekt sie bei der Wahrnehmung der Website haben.

(Bitte lösen Sie die Aufgabe auf der nächsten Seite.)

2. Wieso sollte man einem Kunden erst ein Wireframe zeigen, statt ihm direkt ein fertiges UI-Design vorzulegen? Begründen Sie Ihre Antwort. (4)

C++

3. Nennen Sie für die folgenden Sachverhalte den richtigen Begriff.

   (a) Durch welches Schlüsselwort wird eine Methode in C++ als polymorph markiert? (1)

   (b) Implementiert man mehrere Funktionen mit dem gleichen Namen aber anderen Parameterlisten, bezeichnet man die Funktionen als... (1)

   (c) Ein Smart-Pointer, der sicherstellt, dass es immer nur genau einen Besitzer für ein Objekt gibt, heißt... (1)

   (d) Um zu zeigen, dass ein Funktion-Member beim Aufruf ein Objekt *nicht* verändert, hängt man folgendes Schlüsselwort an. (1)

Zeiger und Referenzen

4. Betrachten Sie die folgenden Codeausschnitte und geben Sie an, was ausgegeben wird:

   (a) Code: (2)

```
int x = 5;
int& y = x;
y++;
printf("%d\n", x);
```

   Ausgabe:

   (b) Code: (2)

```
int a = 2, b = 3;
int& x = a;
x = b;
printf("%d\n", a);
```

   Ausgabe:

   (c) Code: (2)

```
const char* p = "Hallo";
const char*& q = p;
q++;
printf("%s\n", p);
```

   Ausgabe:

5. Wofür setzt man in C++ das Schlüsselwort `friend` ein? (3)

6. Schreiben Sie eine generische C++-Funktion `countOccurence`. Sie soll für ein Array und ein Element zurückgeben, wie oft das Element im Array vorkommt. Der Typ der Element(e) soll dabei generisch sein. Tipp: `template` (8)

7. Betrachten Sie die folgende Funktion `createFactorialSequence`.

```
int* createFactorialSequence(size_t numElements) {
    int* array = new int[numElements];

    int current = 1;
    for(size_t i = 1; i <= numElements; i++) {
        current *= i;
        array[i - 1] = current;
    }

    return array;
}
```

Die Funktion erhält eine Zahl n und gibt ein dynamisch reserviertes Array mit n Elementen zurück, das die Folge von Fakultäten beinhaltet (1!, 2!, 3! usw.).

(a) Schreiben Sie die Funktion so um, dass Sie mit einem Shared-Pointer statt mit `new` arbeitet und diesen zurückgibt. (6)

(b) Welchen Vorteil hat die Nutzung des Shared-Pointers gegenüber `new`? (4)

8. Betrachten Sie die folgenden Struktur, die eine Person beschreibt: (5)

```
struct Person {
    char name[128];
    int age;
};
```

Implementieren Sie den <<-Operator, so dass sich eine Person menschenlesbar über `cout` ausgeben lässt, z.B. so:

```
Person p = { "Bob", 30 };
std::cout << p << std::endl;    // Ausgabe: Bob (30)
```

9. Implementieren Sie Klasse `PositiveList`. Sie speichert eine geordnete Liste von `int`-Werten (Tipp: `std::vector` als Attribut verwenden) und bietet folgende Operatoren an. (12)

   - `operator +=` fügt einen einzelnen Wert hinzu. Ist der Wert aber 0 oder negativ, bleibt der Operator wirkungslos.
   - `operator` $\sim$ gibt die Anzahl der Elemente zurück.
   - `operator +` addiert zwei Objekte vom Typ PositiveList miteinander, so dass eine neue PositiveList entsteht, die alle Werte von beiden Listen enthält.

   Der folgende Code sollte demnach funktionieren:

```
PositiveList list1;
list1 += 3;
list1 += 10;
list1 += -7;   // keine Wirkung, negative Zahl wird nicht hinzugefügt
list1 += 3;
std::cout << ~list1 << std::endl;    // Ausgabe: 3

PositiveList list2;
list2 += 3;
list2 += 10;
std::cout << ~list2 << std::endl;    // Ausgabe: 2

std::cout << ~(list1 + list2) << std::endl;     // Ausgabe: 5
```

10. Nehmen wir an, Sie implementieren eine Klasse `MyVector`, die eine Liste implementiert. Sie wollen erlauben, über den `[]`-Operator auf ein Element der Liste zuzugreifen. (4)

    Erklären Sie, wieso man üblicherweise **zwei** Varianten des Operators implementiert:

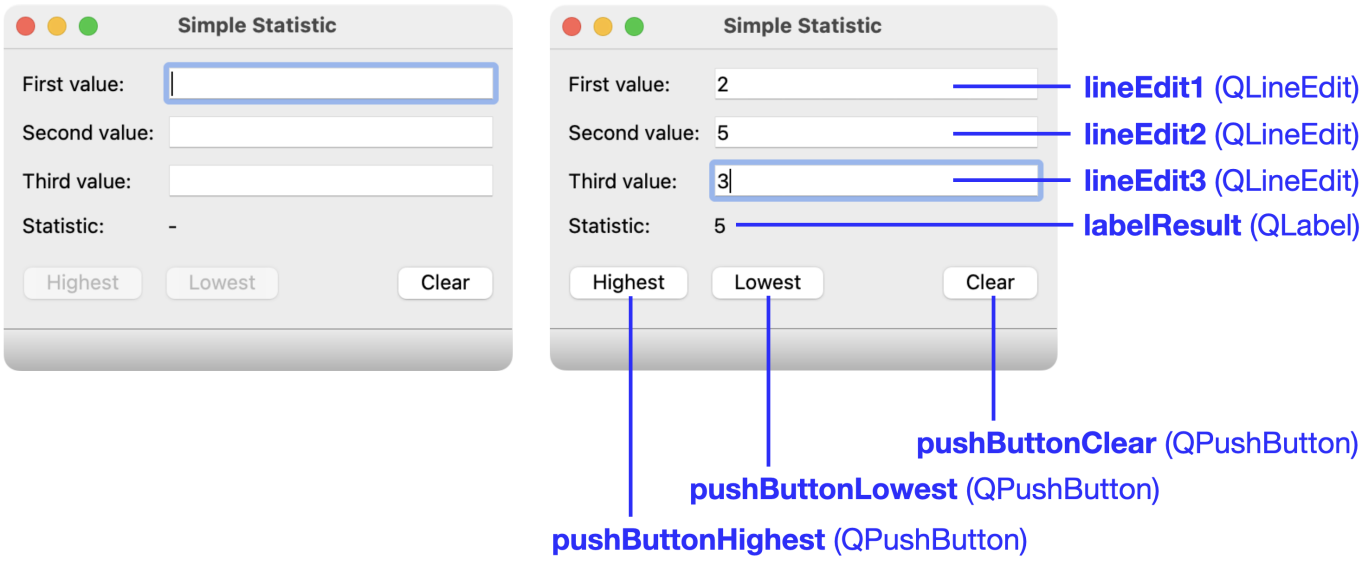
```
int& MyVector::operator[](const size_t& index);

const int& MyVector::operator[](const size_t& index) const;
```

Qt

11. Beschreiben Sie zwei Aspekte, die eine QObject-Instanz von einer normalen C++-Klasse unterscheiden. (4)

12. Betrachten Sie das folgende Wireframe einer GUI, mit der sich einfache Berechnungen durchführen lassen.



Die GUI wurde bereits im Qt-Designer umgesetzt. Es gilt die folgende Anwendungslogik:

- Ein Nutzer kann Zahlen in die drei Eingabefelder eintragen.
- Klick ein Nutzer auf die Highest-Schaltfläche, wird der größte eingegebene Wert im Ergebnisfeld (resultLabel) ausgegeben.
- Klick ein Nutzer auf die Lowest-Schaltfläche, wird der kleinste eingegebene Wert im Ergebnisfeld (resultLabel) ausgegeben.
- Die Schaltflächen Lowest und Highest sind genau dann **sichtbar**, wenn sich Werte in allen Feldern befinden.
- Die Schaltfläche Clear kann immer angeklickt werden.
- Initial sind alle Eingabefelder leer und das Ergebnisfeld enthält einen Strich. Bei Klick auf Clear wird dieser Initialzustand wiederhergestellt.

Tipps:

- Es bietet sich an, eine zentrale Methode zu schreiben, die die GUI-Dymnamik umsetzt, z.B. das (De)Aktivieren oder (Un)sichtbar-Machen von Elementen.
- Die Referenz im Anhang enthält eine Übersicht über sinnvolle Funktionen, insbesondere für die Combo-Box.

(a) Beschreiben Sie, welche Layout-Manager und Spacer eingesetzt werden, um die gezeigte GUI umzusetzen. Beachten Sie dabei, dass die Fenstergröße geändert werden kann. Sie dürfen die Aufgabe über eine Grafik, einen Text oder beides lösen. (6)

(b) Vervollständigen Sie den Code auf den folgenden Seiten, um die beschriebene Logik zu implementieren. Trennen Sie den Quelltext in Header- und Source-Datei sinnvoll auf. (20)

Headerdatei mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

class MainWindow : public QMainWindow {
   Q_OBJECT

public:
   MainWindow(QWidget *parent = nullptr);

private:
   Ui::MainWindow *ui;
```

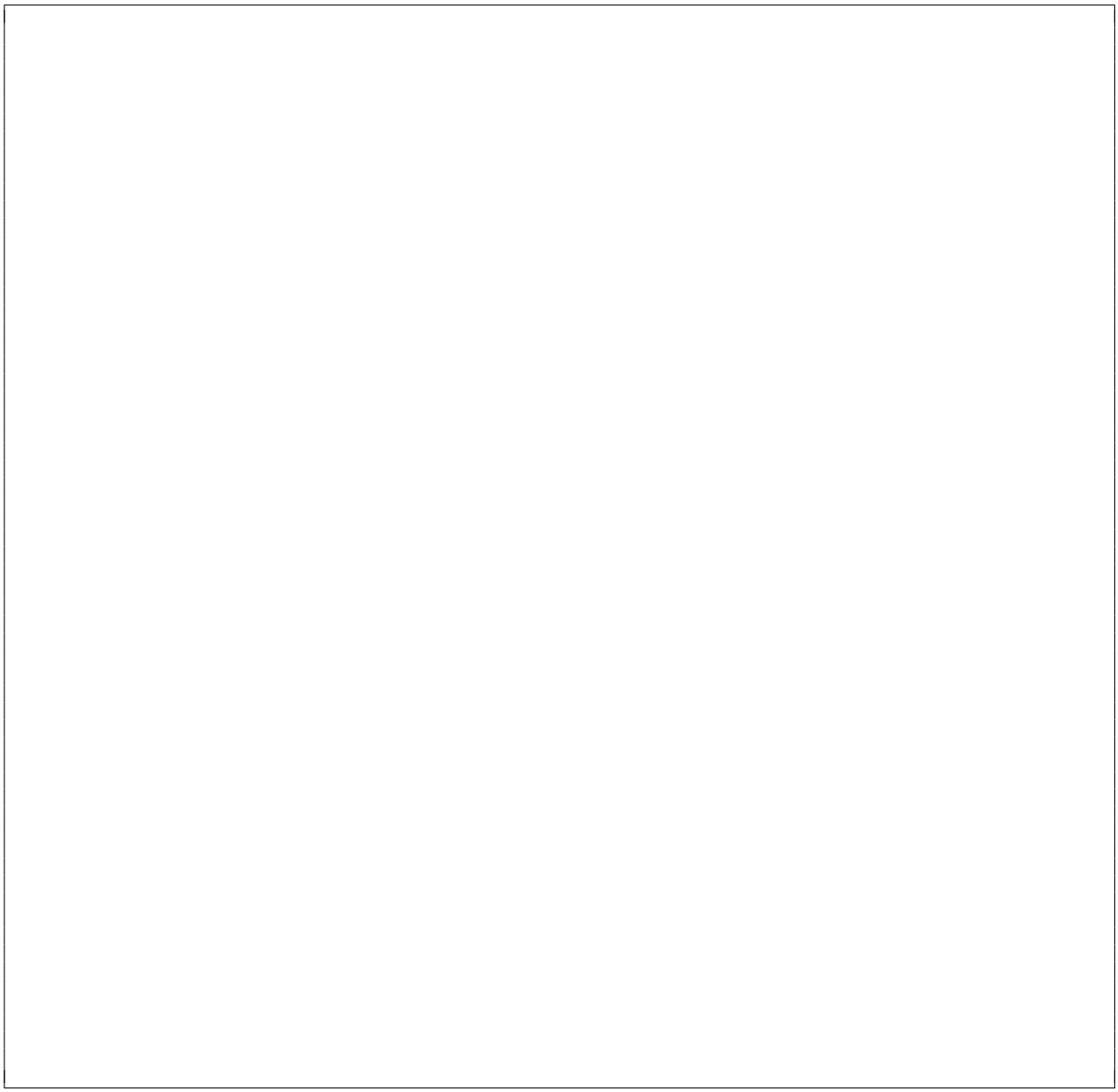
```
};

#endif // MAINWINDOW_H
```

Sourcedatei mainwindow.cpp (Konstruktor)

```
#include "mainwindow.h"
#include <QMessageBox>

MainWindow::MainWindow(QWidget * parent): QMainWindow(parent) {
```



```
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

Sourcedatei mainwindow.cpp (weitere Methoden)

Nebenläufigkeit

13. Das folgende Beispiel verwendet einen Mutex, um zu verhindern, dass der Code in der Funktion parallel ausgeführt werden kann. (5)

```
QMutex mutex;

int gcd(int a, int b) {
    mutex.lock();
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    mutex.unlock();
    return a;
}
```

Wie kann man testen, dass der Mutex tatsächlich funktioniert? Der Code darf dafür verändert werden.

Qt Cheat Sheet

Signale

QPushButton / QAbstractButton

void clicked(bool checked = false)
void toggled(bool checked)
void pressed()
void released()

QComboBox

void currentIndexChanged(int index)

Methoden

QLabel

Text auslesen
QString text() const

Wert setzen
void setText(const QString &)
void setNum(int)

QComboBox

Index des ausgewählten Eintrags auslesen
int currentIndex() const

Eintrag mit Index auswählen
void setCurrentIndex(int)

QString

String mit Zahl erstellen
QString::number(int)
String in Zahl umwandeln
toInt()
toFloat()

Slots

QWidget

bool close()
void hide()
void setDisabled(bool disable)
void setEnabled(bool)
void setFocus()
void setHidden(bool hidden)
void setVisible(bool visible)
void setWindowModified(bool)
void setWindowTitle(const QString &)
void show()
void update()

QAbstractButton

void animateClick(int msec = 100)
void click()
void setChecked(bool)
void setIconSize(const QSize &size)
void toggle()

QPushButton *(erbt von QAbstractButton)*

void showMenu()

Layout-Manager

QHBoxLayout
QVBoxLayout
QFormLayout
QBoxLayout
QGridLayout
QStackedLayout

Thread-Synchronisation

QMutex
QReadWriteLock
QSemaphore