Masterstudiengänge der FK03

Programmierung von CAx-Systemen

Name	Vorname	Studiengang

Aufgabe 1	Aufgabe 2	Aufgabe 3	Aufgabe 4	Summe	

Aufgabensteller: **Küpper**

Hilfsmittel: **Schriftliche Hilfsmittel sind erlaubt,**
Bearbeitung mit Bleistift ist erlaubt.

Viel Erfolg!!!

Aufgabe 1: Objektorientierte Programmierung, Optimierung von Programmen (ca. 24 Punkte)

Die im abgebildeten C++-Quelltext gezeigte Klasse „matrix“ ermöglicht Berechnungen mit Matrizen.

1.1. Vervollständigen Sie den vorbereiteten Quelltext:
- Programmieren Sie den operator<< zur Ausgabe einer Matrix auf einem Stream. Die einzelnen Elemente der Matrix sollen mit einer Feldbreite von 6 Zeichen und mit 2 Nachkommastellen ausgegeben werden. Tipp: Im Hauptprogramm main() finden Sie ein Beispiel zur formatierten Ausgabe von Kommazahlen …
- Programmieren Sie den operator+ zur Addition von zwei Matrizen. Sie dürfen ohne weitere Überprüfung davon ausgehen, dass die beiden zu addierenden Matrizen dieselbe Dimension/Größe haben.

1.2. Die Methode matrix::at() hat einen Rückgabewert des Typs „double&“. Wozu dient hier das Und-Zeichen (&), warum würde ein Rückgabewert des Typs „double“ nicht funktionieren?

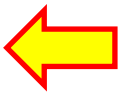
1.3. Die Methode matrix::fill_random() belegt alle Elemente der Matrix mit zufälligen Werten. In welchem Bereich liegen diese zufälligen Werte, was ist der kleinste bzw. was ist der größte mögliche Wert?

1.4. Die Funktionen mul1(), mul2() und mul3() führen Matrix-Matrix-Multiplikationen durch. Bei größeren Matrizen, zum Beispiel mit 1000 x 1000 Elementen, ist mul2() deutlich schneller fertig als mul1(). Erläutern Sie, warum die Funktion mul2() schneller abläuft als mul1().

1.5. Erläutern Sie, warum die Funktion mul3() sogar noch schneller abläuft als die Funktion mul2().

```
#include <iomanip>    // fixed, setw, setprecision
#include <stdlib.h>   // rand, abort
#include <chrono>     // time_since_epoch(), erst ab c++11
#include <vector>
#include <iostream>
using namespace std;
using namespace std::chrono;

// Matrix-Klasse
class matrix
{
private:
    int my_dim;
    vector<double> my_data;
```

```
public:
    matrix(int dim)  // Matrix mit (dim x dim) Null-Elementen erzeugen
    {
        my_dim = dim;
        my_data.resize(dim * dim, 0.0);
    }

    int dim()  // Dimension der Matrix abfragen
    {
        return my_dim;
    }

    double& at(int row, int col)  // Zugriff auf Element in Zeile row und Spalte col
    {
        return my_data[row * my_dim + col];
    }

    void fill_random()  // Matrix mit Zufallszahlen füllen
    {
        for(int idx = 0; idx < my_dim * my_dim; ++idx)
            my_data[idx] = 2.0 * rand() / RAND_MAX - 1.0;
    }

    void fill_zeros()  // Alle Matrix-Elemente auf null setzen
    {
        for(int idx = 0; idx < my_dim * my_dim; ++idx)
            my_data[idx] = 0.0;
    }

    void transpose()  // Matrix transponieren
    {
        for(int row = 0; row < my_dim; ++row)
        {
            for(int col = 0; col < row; ++col)
            {
                double tmp = at(row, col);
                at(row, col) = at(col, row);
                at(col, row) = tmp;
            }
        }
    }
};

// Matrix auf Stream ausgeben

               operator<<

{









}
```



```
// Matrizen addieren; Sie dürfen davon ausgehen, dass m1 und m2 dieselbe Größe haben
                operator+
{









}

// Unterschiedliche Varianten der Matrix-Matrix-Multiplikation
matrix mul1(matrix m1, matrix m2)
{
    int dim = m1.dim();
    matrix result(dim);

    for(int row = 0; row < dim; ++row)
    {
        for(int col = 0; col < dim; ++col)
        {
            double sum = 0;
            for(int idx = 0; idx < dim; ++idx)
                sum += m1.at(row, idx) * m2.at(idx, col);
            result.at(row, col) = sum;
        }
    }
    return result;
}

// Variante 2
matrix mul2(matrix m1, matrix m2)
{
    int dim = m1.dim();
    matrix result(dim);
    m2.transpose();

    for(int row = 0; row < dim; ++row)
    {
        for(int col = 0; col < dim; ++col)
        {
            double sum = 0;
            for(int idx = 0; idx < dim; ++idx)
                sum += m1.at(row, idx) * m2.at(col, idx);
            result.at(row, col) = sum;
        }
    }
    return result;
}
```

```
// Variante 3
matrix mul3(matrix m1, matrix m2)
{
    int dim = m1.dim();
    matrix result(dim);
    m2.transpose();

#pragma omp parallel for
    for(int row = 0; row < dim; ++row)
    {
        for(int col = 0; col < dim; ++col)
        {
            double sum = 0;
            for(int idx = 0; idx < dim; ++idx)
                sum += m1.at(row, idx) * m2.at(col, idx);
            result.at(row, col) = sum;
        }
    }
    return result;
}

// Aktuellen Zeitpunkt bestimmen: Millisekunden seit 1970
double millis()
{
    milliseconds ms = duration_cast<milliseconds>(
        system_clock::now().time_since_epoch());
    return (double)ms.count();
}

// Hauptprogramm
int main(void)
{
    int dim = 1000;
    matrix m1(dim), m2(dim), r1(dim), r2(dim), r3(dim);
    m1.fill_random(); m2.fill_random();

    // Matrix-Matrix-Multiplikation testen
    auto t0 = millis(); r1 = mul1(m1, m2);
    auto t1 = millis(); r2 = mul2(m1, m2);
    auto t2 = millis(); r3 = mul3(m1, m2); auto t3 = millis();

    cout << "dim = " << setw(5) << dim;
    cout << " ; " << fixed << setw(7) << setprecision(3) << (t1 - t0) / 1000;
    cout << " ; " << fixed << setw(7) << setprecision(3) << (t2 - t1) / 1000;
    cout << " ; " << fixed << setw(7) << setprecision(3) << (t3 - t2) / 1000;

    // Weitere Matrix-Methoden testen
    matrix a(3), b(3);
    a.fill_random(); b.fill_random();
    cout << endl << endl << "Addition:" << endl << (a + b) << endl;
}
```

```
C:\Qt\Qt5.12.12\Tools\QtCreat
dim =  1000 ;   1.707 ;   0.330 ;   0.078

Addition:
 -0.93;   1.18;  -0.56
  0.44;  -0.46;  -0.37
 -0.00;   0.59;   1.05
```

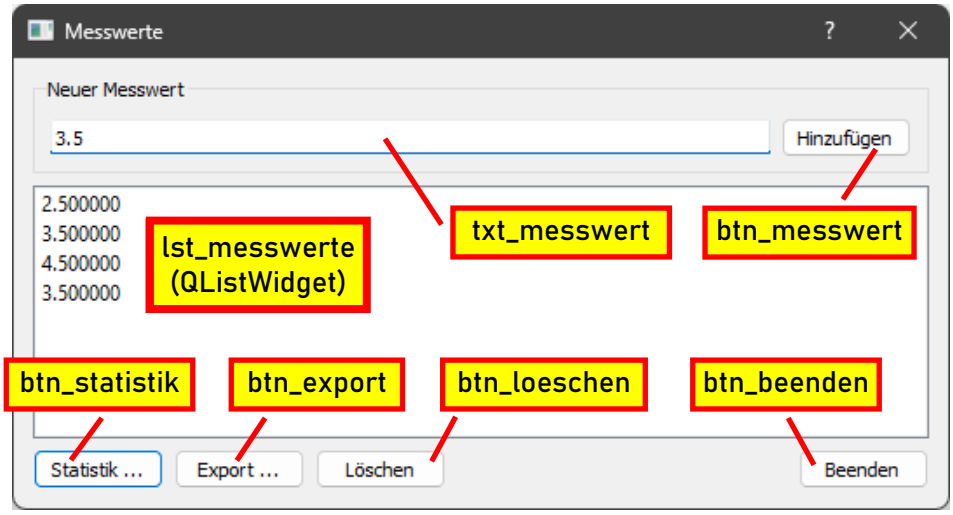
Aufgabe 2: Grafische Benutzeroberfläche, C++-Standardbibliothek (ca. 29 Punkte)

Es soll eine QtWidgets-Dialogapplikation zur Auswertung von Messwerten erstellt werden:

- Die Messwerte werden in txt_messwert eingegeben und können durch Betätigung der Schaltfläche „Hinzufügen“ zur Liste hinzugefügt werden. Gültige Messwerte sind entweder positiv oder gleich null.
- Wenn ein ungültiger Messwert eingegeben wird (Messwert < 0 oder ungültige Zahl in txt_messwert), dann wird der Hinweis „Ungültiger Messwert!“ in einem Meldungsfenster (QMessageBox) angezeigt.
- Nach Betätigung der Schaltfläche „Statistik …“ wird ein Meldungsfenster mit einer einfachen statistischen Auswertung aller Messwerte angezeigt (Anzahl, Mittelwert, Standardabweichung). Die Standardabweichung der Messwerte wird folgendermaßen berechnet (N = Anzahl, $\bar{x}$ = Mittelwert der Messwerte):

$$\sigma = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \bar{x})^2}$$

- Über die Schaltfläche „Export …“ können alle Messwerte in einer Textdatei gespeichert werden.

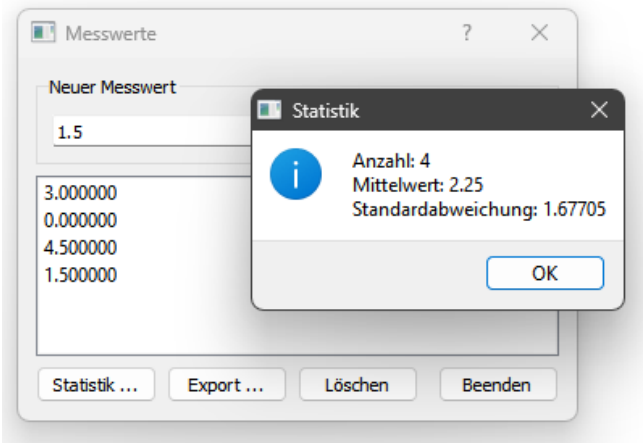



```
// ***** Datei Dialog.h *****
#include <QDialog>
#include <vector>
typedef std::vector<double> d_vect;

QT_BEGIN_NAMESPACE
namespace Ui { class Dialog; }
QT_END_NAMESPACE

class Dialog : public QDialog
{
    Q_OBJECT

public:
    Dialog(QWidget *parent = nullptr);
    ~Dialog();

private slots:
    void on_btn_messwert_clicked();
    void on_btn_export_clicked();
    void on_btn_loeschen_clicked();
    void on_btn_beenden_clicked();
    void on_btn_statistik_clicked();

private:
    Ui::Dialog *ui;
    d_vect data;  // Die Messwerte werden in lst_messwerte und in diesem C++-Vektor gespeichert!
    double mittelwert();
    double standardabweichung();
};
```

Aufbau des Dialogfensters:

Objekt	Klasse
Dialog	QDialog
btn_beenden	QPushButton
btn_export	QPushButton
btn_loeschen	QPushButton
btn_statistik	QPushButton
grp_messwert	QGroupBox
btn_messwert	QPushButton
txt_messwert	QLineEdit
horizontalSpacer	Spacer
lst_messwerte	QListWidget

```cpp
// ***** Datei Dialog.cpp *****
#include "dialog.h"
#include "ui_dialog.h"
#include <string>
#include <sstream>
#include <fstream>
#include <algorithm>
#include <math.h>
#include <QMessageBox>
#include <QFileDialog>
using namespace std;

Dialog::Dialog(QWidget *parent) :
    QDialog(parent), ui(new Ui::Dialog)
{
    ui->setupUi(this);
}

Dialog::~Dialog()
{
    delete ui;
}

void Dialog::on_btn_messwert_clicked()
{
    // TODO: Messwert aus txt_messwert an die Messwerte-Liste lst_messwerte anhängen. Alle
    // Messwerte werden zusätzlich auch im C++-Vektor data gespeichert. Nach der Eingabe eines
    // ungültigen Messwerts (Messwert < 0 oder ungültige Zahl in txt_messwert eingegeben) wird
    // die Meldung "Ungültiger Messwert!" in einer QMessageBox angezeigt.  (Aufgabe 2.3)









}

void Dialog::on_btn_statistik_clicked()
{
    // TODO: Anzahl, Mittelwert und Standardabweichung der Messwerte in einer QMessageBox
    // anzeigen (siehe Bildschirmfoto!). Falls gar keine Messwerte vorhanden sind, wird
    // als Mittelwert und als Standardabweichung jeweils 0.0 angezeigt.    (Aufgabe 2.4)









}

void Dialog::on_btn_export_clicked()
{
    QString fname = QFileDialog::getSaveFileName(this, "Export");  // Dateinamen festlegen
    if(!fname.size()) { return; }

    // TODO: Alle Messwerte werden untereinander in einer Textdatei gespeichert. Anschließend
    // wird die Meldung "Datei wurde erstellt." in einer QMessageBox angezeigt.  (Aufgabe 2.5)
```



```
void Dialog::on_btn_loeschen_clicked()
{
    // TODO: C++-Vektor data und Bildschirmliste lst_messwerte löschen.  (Aufgabe 2.6)



}

void Dialog::on_btn_beenden_clicked()
{
    this->close();
}

double Dialog::mittelwert()
{
    // TODO: Mittelwert der Messwerte berechnen; Rückgabe == 0, falls Anzahl == 0. (Aufgabe 2.1)









}

double Dialog::standardabweichung()
{
    // TODO: Standardabweichung berechnen (Rückgabe == 0, falls Anzahl == 0. (Aufgabe 2.2)









}
```

2.1. Programmieren Sie die Methode Dialog::mittelwert() zur Berechnung des Mittelwerts der Messwerte.

2.2. Programmieren Sie die Methode Dialog::standardabweichung() zur Berechnung der Standardabweichung.

2.3. Programmieren Sie die Methode Dialog::on_btn_messwert_clicked() zum Speichern eines neuen Messwerts in der Bildschirmliste lst_messwerte sowie im C++-Vektor data.

2.4. Programmieren Sie Dialog::on_btn_statistik_clicked() zur Ausgabe der Messwerte-Anzahl, des Mittelwerts und der Standardabweichung. (Hinweis: Zur Berechnung verwenden Sie die Methoden aus Aufgabe 2.1 und 2.2.)

2.5. Programmieren Sie die Methode Dialog::on_btn_export_clicked() zum Speichern aller Messwerte in einer Textdatei. Der Name der Textdatei wird per QFileDialog ausgesucht.

2.6. Programmieren Sie die Methode Dialog::on_btn_loeschen_clicked() zum Löschen von lst_messwerte und data.

2.7. Markieren Sie im abgebildeten Quelltext den Destruktor der Klasse „Dialog“.

2.8. Markieren Sie im abgebildeten Quelltext den Konstruktor der Klasse „Dialog“.

2.9. Was ist ein „Destruktor“, wozu dient er? (Bitte kurze Erläuterung in einigen Stichworten!)

Aufgabe 3: SQL-Datenbanken (ca. 21 Punkte)

Gegeben ist die folgende SQL-Datenbank zur Projektverwaltung in einer kleinen Ingenieurgesellschaft. Die Liste der Angestellten findet sich in der Tabelle „EmployeeData“, alle laufenden Projekte in „ProjectData“. In der Tabelle „ProjectAssignment“ ist angegeben, welche Mitarbeiterinnen und Mitarbeiter an welchen Projekten arbeiten.

EmployeeData

Id INTEGER, PRIMARY KEY	FirstName VARCHAR(25)	LastName VARCHAR(25)	JobTitle VARCHAR(25)
1	Andrea	Benning	Manager
2	Chris	Dremel	Team Leader
3	Erika	Franke	Software Engineer
4	Gunter	Hauser	Software Engineer
5	Ida	Junge	Software Engineer
6	Klaus	Ludewig	Sales Manager
…	…	…	…

ProjectAssignment

Id INTEGER, PRIMARY KEY	EmployeeId INTEGER	Supervisor INTEGER	ProjectId INTEGER
1	4	1	1
2	2	1	1
3	3	1	2
4	2	1	2
…	…	…	

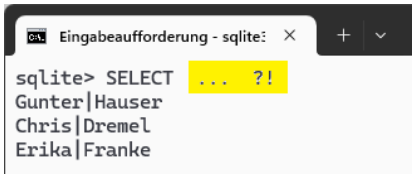
ProjectData

Id INTEGER, PRIMARY KEY	ProjectName VARCHAR(25)	Budget FLOAT
1	Brushless Sim	150000.00
2	Mobile GUI	50000.00
…	…	…

3.1. Formulieren Sie eine CREATE TABLE-Anweisung, um die Tabelle „ProjectData“ anzulegen.

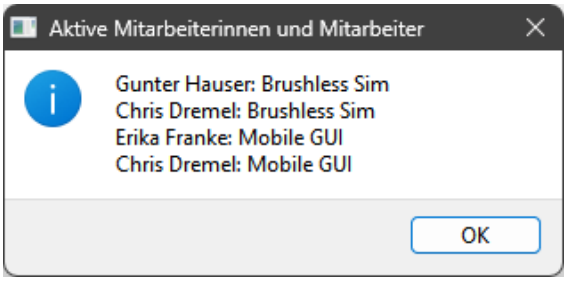
3.2. Formulieren Sie eine CREATE TABLE-Anweisung, um die Tabelle „ProjectAssignment“ anzulegen.
Hinweis: Die Attribute „EmployeeId“, „Supervisor“ und „ProjectId“ sind Fremdschlüssel.

3.3. Formulieren Sie eine SQL-Anweisung, um die Vor- und Nachnamen aller Angestellten aufzulisten, die mindestens einem Projekt zugeordnet sind. Hinweis: keine Sortierung erforderlich!



3.4. Was ist ein Fremdschlüssel? (Bitte mit einigen Stichwörtern beantworten!)

3.5. Der folgende Quelltext stammt aus einer QtWidgets-Dialog-Applikation. Vervollständigen Sie den Quelltext so, dass die Vor- und Nachnamen aller Mitarbeiterinnen und Mitarbeiter zusammen mit ihren Projekten in einer QMessageBox angezeigt werden. Personen, die an mehr als einem Projekt arbeiten, werden mehrfach ausgegeben. Personen, die gerade an keinem Projekt arbeiten, werden nicht ausgegeben.
Hinweis: Die ausgegebenen Daten müssen nicht sortiert werden.




```
// *** Dialog.cpp ***
#include "dialog.h"
#include "ui_dialog.h"
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>
#include <QVariant>
#include <QMessageBox>
#include <sstream>
using namespace std;

void Dialog::on_pushButton_clicked()
{
    // Die Verbindung zur Projekt-Datenbank wurde bereits geöffnet. Über die Variable
    // sqlDatabase (Typ: QSqlDatabase) kann auf die Datenbank zugegriffen werden.
    QSqlQuery query(sqlDatabase);

    // TODO: SELECT-Datenbankabfrage durchführen, Ergebnisse in QMessageBox anzeigen









    // Hinweis: Die Datenbankverbindung wird nicht geschlossen, sie bleibt geöffnet!
}
```

Aufgabe 4: C++-Standardbibliothek (ca. 6 Punkte)

4.1. Erläutern Sie den Unterschied zwischen „Klasse“ und „Instanz“ in der Objektorientierten Programmierung.

4.2. Wie kann die Methode starten() aufgerufen werden?

```
class Fahrzeug {
public:
    void starten() { /* Implementierung */ }
};

class Auto : public Fahrzeug {
public:
    void beschleunigen() { /* Implementierung */ }
};

Auto my_auto;

// TODO: Methode Fahrzeug::starten() für die Instanz my_auto (Typ: Auto) aufrufen ...
```

4.3. Nennen Sie ein Beispiel für eine Klasse, die mittels `#include <complex>` zur Verfügung gestellt wird. Wozu dient diese Klasse?