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: Grafische Benutzeroberfläche (ca. 24 Punkte)

Es soll eine Dialog-Applikation mit den folgenden Funktionen erstellt werden:

a. Die Applikation soll die Zeit messen, wie lange es dauert, einen std::vector<double> zu sortieren.

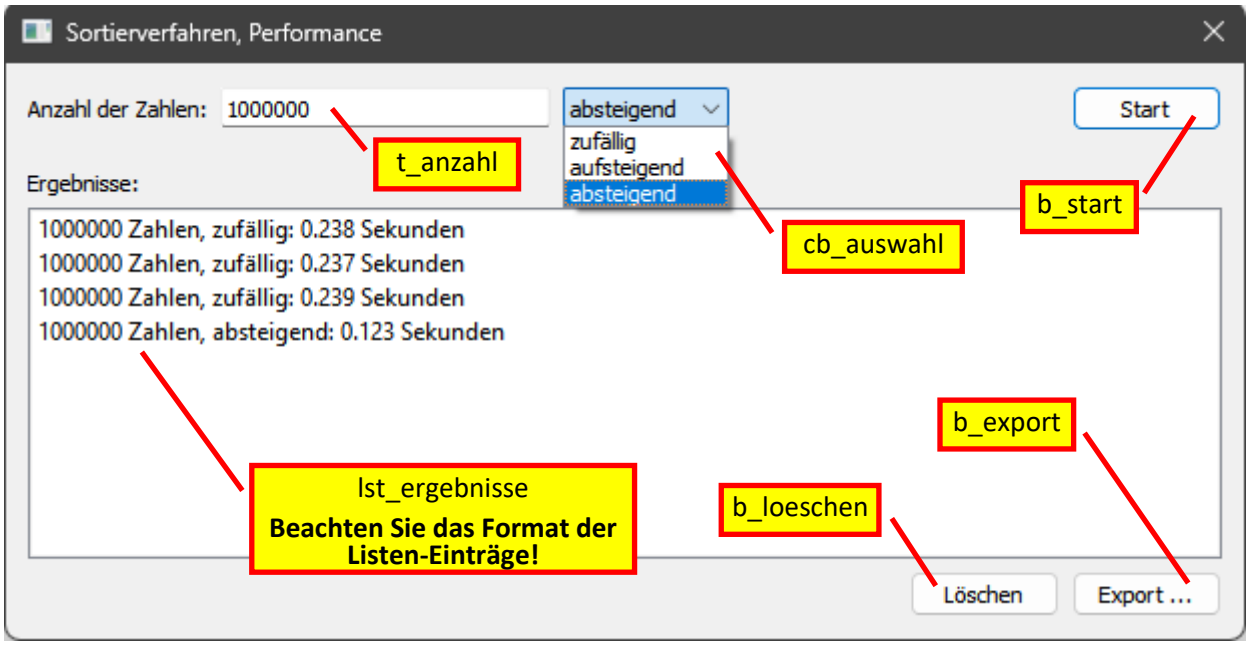
b. Im Textfeld t_anzahl wird eingegeben, wie viele Zahlen sortiert werden sollen.

c. Es ist zu erwarten, dass die Dauer des Sortiervorgangs davon abhängt, ob der Vektor mit zufälligen Werten gefüllt ist oder ob die Werte im Vektor beim Aufruf der Sortierfunktion bereits sortiert sind.

d. In der ComboBox cb_auswahl kann daher ausgewählt werden, ob der Vektor …
   (0) mit zufälligen double-Werten im Bereich -10.0 … +10.0 gefüllt werden soll oder
   (1) ob der Vektor mit der aufsteigenden Zahlenfolge 1.0, 2.0, 3.0 … gefüllt werden soll oder
   (2) ob der Vektor mit der absteigenden Zahlenfolge -1.0, -2.0, -3.0 … gefüllt werden soll.

e. Nach Betätigung der Schaltfläche b_start wird der Vektor mit Zahlen gefüllt, die Sortierfunktion wird aufgerufen. Die ermittelte Zeit wird zur Ergebnisliste hinzugefügt (als neuer Eintrag am Ende der Ergebnisliste).

f. Nach Betätigung der Schaltfläche b_export werden alle Einträge, die sich aktuell in der Ergebnis-Liste befinden, in einer Textdatei abgespeichert. Der Name dieser Textdatei wird über einen QFileDialog ermittelt.

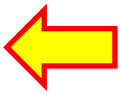


1.1. Markieren Sie im C++-Quelltext den Konstruktor und den Destruktor der Klasse „Dialog“.

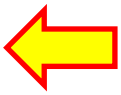
1.2. Vervollständigen Sie den Quelltext der Datei Dialog.cpp. Beachten Sie dabei die Kommentare und die Hinweise zur Aufgabenstellung im Quelltext.

```
#include "dialog.h"
#include "ui_dialog.h"
#include <time.h>      // clock()
#include <algorithm>   // sort()
#include <fstream>     // ofstream
#include <sstream>     // ostringstream
#include <vector>
#include <QFileDialog>
#include <QMessageBox>
using namespace std;

Dialog::Dialog(QWidget *parent) : QDialog(parent), ui(new Ui::Dialog)
{
       this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint);
       ui->setupUi(this);
}
```

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

void Dialog::on_b_start_clicked()
{
	// Welcher Wert wurde in der ComboBox ausgewählt?
	int auswahl = ui->cb_zufall->currentIndex();

	// auswahl = 0: zufällige double-Werte im Bereich -10.0 ... +10.0
	//         = 1: aufsteigende Zahlenfolge 1.0, 2.0, 3.0 ... (beginnend bei +1)
	//         = 2: absteigende Zahlenfolge -1.0, -2.0, -3.0 ... (beginnend bei -1)

	// Der im Textfeld t_anzahl eingegebene Wert wird in die int-Variable n übertragen.
	// (Bei einer ungültigen Eingabe oder bei n<10 wird eine QMessageBox mit einer Fehlermeldung
	// angezeigt und die Methode Dialog::on_b_start_clicked() wird direkt wieder verlassen.)
	int n;
```

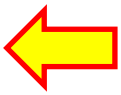


```
	// Vektor mit "n" Zahlen füllen. Je nach "auswahl" handelt es sich dabei entweder
	// um zufällige Zahlen, um eine aufsteigende oder um eine absteigende Zahlenfolge.
	vector<double> vec;
```

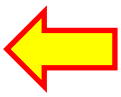


```
	// Jetzt wird ermittelt, wie lange das Sortieren des Vektors dauert (in Sekunden).
	auto c1 = clock();
	sort(vec.begin(), vec.end());  // Alle Elemente des Vektors sortieren
	auto c2 = clock();
	double sekunden = double(c2 - c1) / CLOCKS_PER_SEC;
```

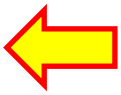
```
        // Es wird eine weitere Zeile zur Ergebnis-Liste "lst_ergebnisse" hinzugefügt.
        // (Zur Formatierung der Ergebnisse: siehe Bildschirmfoto!)
        string txt;
```



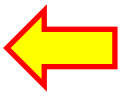
```
        ui->lst_ergebnisse->addItem(txt.c_str());
}

void Dialog::on_b_export_clicked()
{
        // Alle Einträge in der Ergebnis-Liste werden in einer Textdatei gespeichert.
        // Zunächst wird der Dateiname ermittelt.
        string filename = QFileDialog::getSaveFileName().toStdString();
        if(filename.size() == 0) { return; }

        // Nun wird die Textdatei zum Schreiben geöffnet.
```



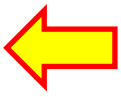
```
        // Schließlich werden die einzelnen Zeilen der Ergebnis-Liste der
        // Reihe nach durchlaufen und in der Textdatei gespeichert.
        int n = ui->lst_ergebnisse->count();
        for(int i = 0; i < n; ++i)
        {
                string zeile = ui->lst_ergebnisse->item(i)->text().toStdString();
```



```
        }

        // Meldungsfenster (QMessageBox) anzeigen: "Die Ergebnisse wurden gespeichert."
```



```
}

void Dialog::on_b_loeschen_clicked()
{
        // Alle Einträge aus der Ergebnis-Liste im Dialogfenster löschen
```



```
}
```

Aufgabe 2: SQL-Datenbanken (ca. 20 Punkte)

Gegeben sind die folgenden Tabellen zum Abspeichern von Messwerten in einer SQL-Datenbank:

Tabelle: MeasurementData

Id	RecordNo	XPosition	Voltage	DeviceId	ChannelId
1	1	-1,9200E-07	0,08	1	1
2	1	-1,9160E-07	0	1	1
3	1	-1,9120E-07	0	1	1
4	1	-1,9080E-07	0	1	1
5	1	-1,9040E-07	0	1	1
6	1	-1,9000E-07	0	1	1
7	1	-1,8960E-07	0	1	1
8	1	-1,8920E-07	0,08	1	1

Tabelle: Device

Id	Model	Firmware
1	MDO3024	1,20

Tabelle: Channel

Id	Name	OffsetV	AttnV
1	CH1	0,00	1,00
2	CH2	0,00	1,00

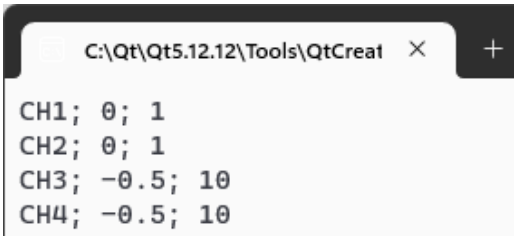
2.1 Formulieren Sie eine SELECT-Abfrage, um alle Messwerte mit dem Attribut RecordNo = 1 aus der Datenbank abzurufen. Zu jedem Messwert sollen (nur!) die Attribute XPosition und Voltage aus der Datenbank gelesen werden. Die Messwerte sollen nach der XPosition aufsteigend sortiert werden.

2.2 Formulieren Sie eine CREATE TABLE-Anweisung, um die Tabelle Device in der Datenbank anzulegen. (Hinweise: Das ganzzahlige Attribut „Id“ ist der Primärschlüssel. Als „Model“ kann ein Text von maximal 25 Zeichen angegeben werden. Die Angabe der „Firmware“ erfolgt als Kommazahl.)

2.3 Formulieren Sie eine SQL-Anweisung, um alle Messwerte aus der Tabelle MeasurementData zu löschen, bei denen der Wert von RecordNo kleiner als 10 ist.

2.4 Was ist ein Primärschlüssel, was ist ein Fremdschlüssel? (Bitte mit einigen Stichwörtern beantworten - oder mit einem konkreten Beispiel.)

2.5 Der folgende Quelltext liest alle Datensätze aus der Tabelle „Channel“. Vervollständigen Sie den Quelltext so, dass die Datensätze auf std::cout wie im Bildschirmfoto gezeigt ausgegeben werden.




```
// Zugriff auf SQLite-Datenbank
#include <iostream>
#include <stdlib.h>  // EXIT_FAILURE
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>
#include <QVariant>

int main()
{
	using namespace std;

	// Datenbank-Verbindung öffnen, ggf. neue SQLite-Datei anlegen
	QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
	db.setDatabaseName("c:/temp/oscope1.db");
	if(db.open() == false) { exit(EXIT_FAILURE); }

	// Datensätze aus Tabelle auslesen (Name -> Text, OffsetV und AttnV -> Kommazahlen)
	QSqlQuery query(db);
	query.exec("SELECT Name, OffsetV, AttnV FROM Channel ORDER BY Name");

	// QSqlDatabase-Verbindung wieder schließen

}
```

Aufgabe 3: Erstellen von Klassen (ca. 23 Punkte)

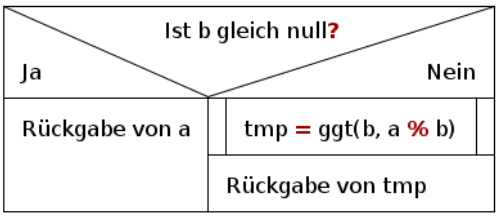
In dem folgenden C++-Quelltext wird eine Klasse zum Rechnen mit Brüchen (rationalen Zahlen) programmiert. Jeder Bruch besteht aus einem ganzzahligen Zähler und einem ganzzahligen Nenner. Im Hauptprogramm main() wird die folgende Berechnung durchgeführt:

$$\frac{1}{3}\cdot\frac{1}{2}+\frac{1}{3}:\frac{1}{2}=\frac{5}{6}$$

Hinweise:

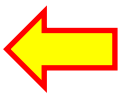
- Es gibt drei (!) Konstruktoren. Beachten Sie dazu die Kommentare im Hauptprogramm main().
- Die Funktion ggt() berechnet den größten gemeinsamen Teiler von a und b mithilfe eines rekursiven Verfahrens. Programmieren Sie die Funktion ggt() so, dass der Ablauf dem Struktogramm entspricht.
- Alle arithmetischen Operatoren (+ - * /) sorgen dafür, dass der Bruch nach der Berechnung gekürzt wird.
- Der Programmablauf soll dem Bildschirmfoto entsprechen, siehe nächste Seite.

int ggt(int a, int b)

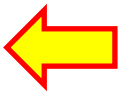


Vervollständigen Sie den vorbereiteten Quelltext.

```
#include <iostream>
#include <stdlib.h>  // abort()
using namespace std;

// Fehlermeldung ausgeben, Programm abbrechen
void error(string msg)
{
    cout << msg << endl;
    abort();
}

// Berechne den größten gemeinsamen Teiler
int ggt(int a, int b)
{
```



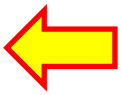
```
}

// Klasse zum Rechnen mit Bruchzahlen
class bruch
{
private:
    int z;  // Zähler
    int n;  // Nenner

public:
    // Konstruktor Nr. 1: Zähler wird auf 0 gesetzt, Nenner wird auf 1 gesetzt
    bruch()
    {
```



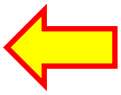
```
    }
```

```
    // Konstruktor Nr. 2: Zähler wird auf den angegebenen Wert gesetzt, Nenner wird auf 1 gesetzt
    bruch(int zaehler)
    {


    }

    // Konstruktor Nr. 3: Falls der Nenner null ist, wird eine Fehlermeldung ausgegeben.
    bruch(int zaehler, int nenner)
    {




    }

    // Aktuellen Wert des Zählers zurückgeben
    int zaehler()
    {


    }

    // Aktuellen Wert des Nenners zurückgeben
    int nenner()
    {


    }

    // Bruch kürzen: Zähler und Nenner durch größten gemeinsamen Teiler teilen
    void kuerzen()
    {
        int teiler = ggt(z, n);
        if(teiler == 0) { error("Fehler: Division durch null"); }
        z /= teiler; n /= teiler;
    }

    // Bruch "ausrechnen" und als double-Wert zurückgeben
    double to_double()
    {
        return (double)z / n;
    }

    // Subtraktion, als Methode innerhalb der Klasse "bruch" definiert
    bruch operator- (bruch b2)
    {




    }

    // Division, als Methode innerhalb der Klasse "bruch" definiert
    bruch operator/ (bruch b2)
    {




    }
};
```

```
// Addition, als freie Funktion definiert
bruch operator+ (bruch b1, bruch b2)
{
    int z_neu = b1.zaehler() * b2.nenner() + b2.zaehler() * b1.nenner();
    int n_neu = b1.nenner() * b2.nenner();

    bruch summe(z_neu, n_neu);
    summe.kuerzen();  // Am Ende jeder Berechnung wird der Bruch gekürzt!
    return summe;
}

// Multiplikation, als freie Funktion definiert
bruch operator* (bruch b1, bruch b2)
{
```



```
}

// Bruch auf C++-Stream ausgeben
ostream& operator<< (ostream& strm, bruch b)
{
```



```
}

// Hauptprogramm
int main()
{
    bruch a(1, 3);  // Konstruktor Nr. 3
    bruch b(1, 2);  // Konstruktor Nr. 3
    bruch c;        // Konstruktor Nr. 1

    c = a * b + a / b;  // (1/3) * (1/2) + (1/3) : (1/2) = (5/6)
    cout << c << " = " << c.to_double() << endl;

    auto my_pi = bruch(22) / bruch(7);  // Konstruktor Nr. 2
    cout << "Naeherung fuer Pi: " << my_pi << " = " << my_pi.to_double() << endl;

    bruch test(10, 0);  // Fehler: Nenner gleich null, Programm wird abgebrochen
    return 0;
}
```

```
C:\Qt\Qt5.12.12\Tools\QtCreat
5/6 = 0.833333
Naeherung fuer Pi: 22/7 = 3.14286
Fehler: Nenner gleich null
```

Aufgabe 4: C++-Standardbibliothek (ca. 13 Punkte)

4.1. Der Wert einer double-Variablen x wird folgendermaßen auf dem Bildschirm ausgegeben:

```
cout << fixed << setw(8) << setprecision(3) << x << endl;
```

Wie lautet ein printf-Aufruf zur Ausgabe der Variablen x mit demselben Format (inkl. Zeilenumbruch)?

4.2. In einem `vector<punkt> v1;` sind mehrere Punkte gespeichert. Vervollständigen Sie die for-Schleife, mit der alle Vektor-Elemente durchlaufen werden. Auf std::cout sollen jeweils die x-Koordinaten der Punkte untereinander ausgegeben werden.

```
class punkt   // So sieht die Definition der Klasse "punkt" aus
{
public:
    double x, y;
};

for(int i = 0; i <                        ; ++i)
```

4.3. Schreiben Sie eine zweite for-Schleife, mit der ebenfalls alle Elemente von v1 durchlaufen werden. Diese zweite Schleife soll ohne Zählvariable programmiert werden („range-based for loop“). Nun sollen jeweils die y-Koordinaten untereinander ausgegeben werden.

```
for(
```

4.4. Wie können alle Elemente des Vektors v1 gelöscht werden? (v1 soll danach ein leerer Vektor sein.)

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

4.6. Was versteht man bei der objektorientierten Programmierung unter einem „Destruktor“?
Nennen Sie ein konkretes Beispiel, wozu ein Destruktor eingesetzt werden könnte.