Masterstudiengang Technische Berechnung und Simulation

Programmierung von CAx-Systemen – Teil 1

Name	Vorname	Matrikelnummer

Aufgabe 1	Aufgabe 2	Aufgabe 3	Summe	

Aufgabensteller: **Dr. Reichl, Dr. Küpper**

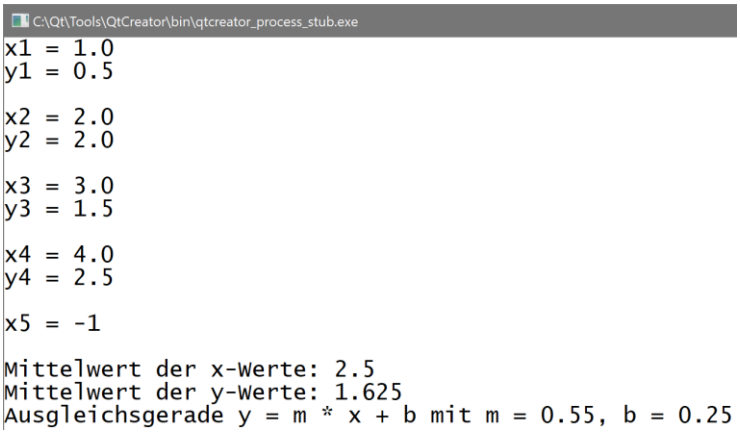
Hilfsmittel: **Taschenrechner nicht zugelassen,**
PC/Notebook nicht zugelassen,
sonstige eigene Hilfsmittel sind erlaubt,
Bearbeitung mit Bleistift ist erlaubt.

Viel Erfolg!!!

Aufgabe 1: Programmierung mit der C++-Standardbibliothek (ca. 18 Punkte)

Erstellen Sie ein C++-Programm, welches für n Messwertpaare x_i, y_i (mit $i = 1 \dots n$) die dazugehörige Ausgleichsgerade $y = m \cdot x + b$ berechnet.

- Nach dem Programmstart wird der Anwender zur Eingabe der einzelnen x- und y-Werte aufgefordert (Datentyp **double**, siehe Bildschirmfoto).
- Die Messwerte werden in zwei Vektoren **x_vec** und **y_vec** vom Typ **vector<double>** gespeichert. Die Eingabe wird beendet, wenn der Anwender einen negativen x-Wert eingibt.
- Wenn weniger als zwei Messwertpaare eingegeben wurden ($n < 2$), wird das Programm mit einer Fehlermeldung abgebrochen. (Dies ist in der nebenstehenden Abbildung nicht sichtbar.)



- Die Mittelwerte der x-Werte (im Folgenden als $\bar{x}$ bezeichnet) und der y-Werte (im Folgenden als $\bar{y}$ bezeichnet) werden berechnet und auf dem Bildschirm ausgegeben. Programmieren Sie zur Berechnung der Mittelwerte eine Funktion **mittel()** und rufen Sie diese Funktion aus Ihrem Hauptprogramm **main()** heraus auf.
- Schließlich berechnet das Programm die Koeffizienten m und b der Ausgleichsgeraden und gibt beide Werte ebenfalls auf dem Bildschirm aus. (Alle Eingaben in Ihrem Programm sollen von **cin**, alle Ausgaben sollen auf **cout** erfolgen. Sie sollen so realisiert werden, wie es im Bildschirmfoto gezeigt ist.)

Für den Koeffizienten *m* gilt:

$$m = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Für den Koeffizienten *b* gilt:

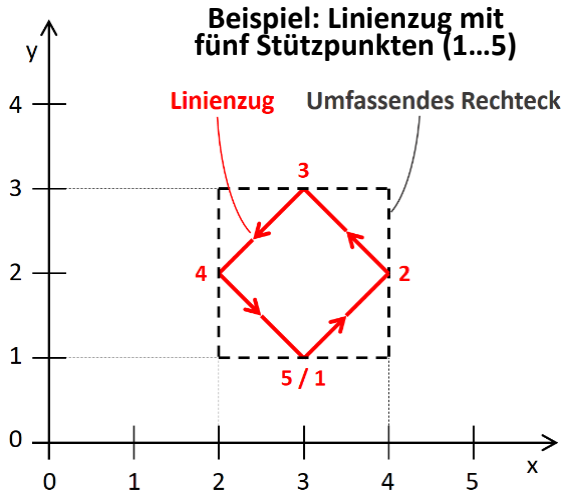
$$b = \bar{y} - m \cdot \bar{x}$$

```
// ------------------------------------------------------------------------
// C++-Programm zur Berechnung von Ausgleichsgeraden
// ------------------------------------------------------------------------
#include <iostream>
#include <vector>

using namespace std;
typedef vector<double> d_vector;

// ------------------------------------------------------------------------
// Gibt den Mittelwert der Elemente im Vektor vec zurück;
// falls der Vektor leer ist, wird 0 (null) zurückgegeben.
// ------------------------------------------------------------------------
double mittel(d_vector vec)
{









}
```

```
// ------------------------------------------------------------------------
// Hauptprogramm
// ------------------------------------------------------------------------
int main(void)
{
    d_vector x_vec, y_vec;









    return 0;
}
```

Aufgabe 2: Objektorientierte Programmierung (ca. 20 Punkte)

Schreiben Sie ein C++-Programm zur Berechnung von Linienzügen, die aus einer Serie von Stützpunkten bestehen. Die Koordinaten der Stützpunkte werden in Instanzen der Klasse **Punkt** gespeichert, ein kompletter Linienzug wird durch die Klasse **Linienzug** repräsentiert.



2.1. Programmieren Sie den **Operator <<** zur Ausgabe eines Stützpunkts auf einem Ausgabe-Stream.

2.2. Programmieren Sie die Funktion **Abstand()**, welche die Entfernung (den euklidischen Abstand) zweier Stützpunkte berechnet.

2.3. Programmieren Sie die Methode **Linienzug::Add()**, welche einen weiteren Stützpunkt zum Linienzug hinzufügt.

2.4. Programmieren Sie die Methode **Linienzug::Anzahl()**, welche die aktuelle Anzahl der Stützpunkte im Linienzug zurückgibt.

2.5. Programmieren Sie die Methode **Linienzug::Gesamtlaenge()**, welche die Gesamtlänge des Linienzugs berechnet, also die Summe der Teilstrecken vom ersten bis zum letzten Stützpunkt.

2.6. Vervollständigen Sie die Methode **Linienzug::Umfassendes-Rechteck()**, welche die Koordinaten des umfassenden Rechtecks ermittelt (siehe nebenstehende Abbildung). Bei einem „leeren" Linienzug ohne Stützpunkte sollen alle Koordinaten als 0 (null) zurückgegeben werden.

2.7. Programmieren Sie die Methode **Linienzug::Ausgeben()**, welche die Koordinaten aller Stützpunkte untereinander auf dem Bildschirm ausgibt (siehe Bildschirmfoto am Ende dieser Aufgabe für ein Beispiel). Zur Ausgabe der Koordinaten verwenden Sie bitte den in Unterpunkt 2.1 definierten **Operator <<**.

```
// ------------------------------------------------------------------------
// Berechnungen von Linienzügen
// ------------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;

// ------------------------------------------------------------------------
// Klasse zum Abspeichern eines Stützpunkts (mit x- und y-Koordinaten)
// ------------------------------------------------------------------------
class Punkt
{
private:
    double x, y;  // Koordinaten des Stützpunkts

public:
    Punkt() { x = 0; y = 0; }
    Punkt(double my_x, double my_y) { x = my_x; y = my_y; }
    double GetX() { return x; }
    double GetY() { return y; }
};

// Koordinaten des Stützpunkts im Format (12.345;2.123) ausgeben
ostream& operator<< (ostream& strm, Punkt p)  // AUFGABE 2.1
{




}
```

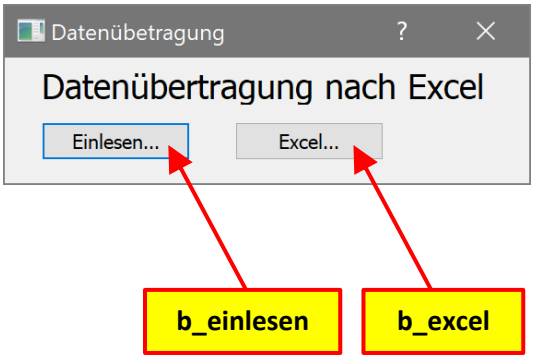
```
// Abstand der Stützpunkte a und b berechnen und zurückgeben
double Abstand(Punkt a, Punkt b)  // AUFGABE 2.2
{




}

// ------------------------------------------------------------------------
// Klasse zum Abspeichern eines Rechtecks. Das Rechteck wird durch seine
// beiden Eckpunkte P1(xmin, ymin) sowie P2(xmax, ymax) beschrieben.
// ------------------------------------------------------------------------
class Rechteck
{
public:
    double xmin, ymin;  // Eckpunkt P1
    double xmax, ymax;  // Eckpunkt P2
};

// ------------------------------------------------------------------------
// Klasse zum Arbeiten mit Linienzügen. Die einzelnen Punkte des
// Linienzugs werden über die Methode Add() hinzugefügt. Anschließend
// können verschiedene Berechnungen durchgeführt werden.
// ------------------------------------------------------------------------
class Linienzug
{
private:
    // Vektor mit Stützpunkten
    vector<Punkt> data;

public:
    // Einen Punkt zum Linienzug hinzufügen
    void Add(Punkt p)  // AUFGABE 2.3
    {




    }

    // Anzahl der Punkte im Linienzug ermitteln
    int Anzahl(void)  // AUFGABE 2.4
    {




    }

    // Gesamtlänge des Linienzugs berechnen
    double Gesamtlaenge(void)  // AUFGABE 2.5
    {









    }
```

```
    // Umfassendes Rechteck ermitteln
    Rechteck UmfassendesRechteck(void)  // AUFGABE 2.6
    {
        Rechteck b;
        if(data.size() == 0)
        {
            b.xmin = b.xmax = b.ymin = b.ymax = 0;
            return b;
        }









    }

    // Alle Punkte des Linienzugs ausgeben
    void Ausgeben(void)  // AUFGABE 2.7
    {









    }
};

// ----------------------------------------------------------------------------
// Hauptprogramm
// ----------------------------------------------------------------------------
int main(void)
{
    Linienzug z;
    z.Add(Punkt(3, 1)); z.Add(Punkt(4, 2));
    z.Add(Punkt(3, 3)); z.Add(Punkt(2, 2));
    z.Add(Punkt(3, 1));

    cout << "Laenge: " << z.Gesamtlaenge() << endl;
    cout << "Anzahl: " << z.Anzahl() << endl << endl;
    z.Ausgeben();

    Rechteck b = z.UmfassendesRechteck();
    cout << endl << "Umfassendes Rechteck: " << endl;
    cout <<   "x_min = " << b.xmin;
    cout << ", y_min = " << b.ymin << endl;
    cout <<   "x_max = " << b.xmax;
    cout << ", y_max = " << b.ymax << endl;
    return 0;
}
```

```
C:\Qt\Tools\QtCreator\bin\qtcreator_proces
Laenge: 5.65685
Anzahl: 5

P1: (3;1)
P2: (4;2)
P3: (3;3)
P4: (2;2)
P5: (3;1)

Umfassendes Rechteck:
x_min = 2, y_min = 1
x_max = 4, y_max = 3
```

Aufgabe 3: Grafische Benutzeroberflächen, COM-Schnittstellen (ca. 12 Punkte)

Die abgebildete Benutzeroberfläche wurde mit QtWidgets erstellt. Das Programm liest eine Liste von Punkten, angegeben durch x- und y-Koordinaten, aus einer Textdatei und speichert sie in einem Vektor „data“, der zuvor geleert wird (**Schaltfläche „b_einlesen“**). Danach werden die Koordinaten aus dem Vektor „data“ in eine Excel-Tabelle übertragen (**Schaltfläche „b_excel“**).



- In jeder Zeile der Textdatei steht eine x- und eine y-Koordinate. Zwischen den Werten befindet sich ein Leerzeichen, zum Beispiel:

```
0.223 0.135
0.256 0.121
0.299 0.130
0.303 0.132
```

(Beispiel-Koordinaten für Aufgabe 3.2)

- Das Programm soll alle Koordinaten aus der Textdatei einlesen, bis das Ende der Datei erreicht ist.
- Die Klasse „Punkt“ und der Vektor „data“ sind wie folgt deklariert:

```
class Punkt
{
public:
    double x, y;
};

class Dialog : public QDialog
{
private:
    std::vector<Punkt> data;
    // Es folgen weitere Attribute, Methoden, Slots usw...
};
```

3.1. Vervollständigen Sie den Quelltext zur Reaktion auf die Schaltfläche „b_einlesen“:

```
void Dialog::on_b_einlesen_clicked()
{
    auto filename = QFileDialog::getOpenFileName(this, "", "", "");
    if(filename.length() == 0) return;

    // Vektor "data" leeren; ausgewählte Textdatei öffnen;
    // Punkte aus Datei einlesen und in "data" speichern...









}
```

3.2. In der Textdatei stehen genau diejenigen vier Punkte (= acht Koordinaten), die im Beispiel oben auf der vorherigen Seite abgedruckt sind. Die Koordinaten der Punkte wurden bereits eingelesen und im Vektor „data“ gespeichert. Wie sieht das Excel-Tabellenblatt nach Ausführung der Methode „on_b_excel_clicked()“ aus?

```
void Dialog::on_b_excel_clicked()
{
    excel.setControl("Excel.Application");
    excel.setProperty("Visible", true);

    QAxObject *active = excel.querySubObject("ActiveSheet");
    if(!active)
    {
        QAxObject *workbooks = excel.querySubObject("Workbooks");
        workbooks->dynamicCall("Add(void)");
        active = excel.querySubObject("ActiveSheet");
    }

    QAxObject *cells = active->querySubObject("Cells(int,int)", 1, 2);
    cells->setProperty("Value", "Anzahl:");

    cells = active->querySubObject("Cells(int,int)", 1, 3);
    cells->setProperty("Value", data.size());

    int col = 1;
    for(auto p : data)
    {
        cells = active->querySubObject("Cells(int,int)", 3, col);
        cells->setProperty("Value", p.x);

        cells = active->querySubObject("Cells(int,int)", 4, col);
        cells->setProperty("Value", p.y);

        ++col;
    }
}
```