Masterstudiengang Technische Berechnung und Simulation

Programmierung von CAx-Systemen – Teil 1

Name	Vorname	Matrikelnummer

Aufgabe 1	Aufgabe 2	Aufgabe 3	Summe	

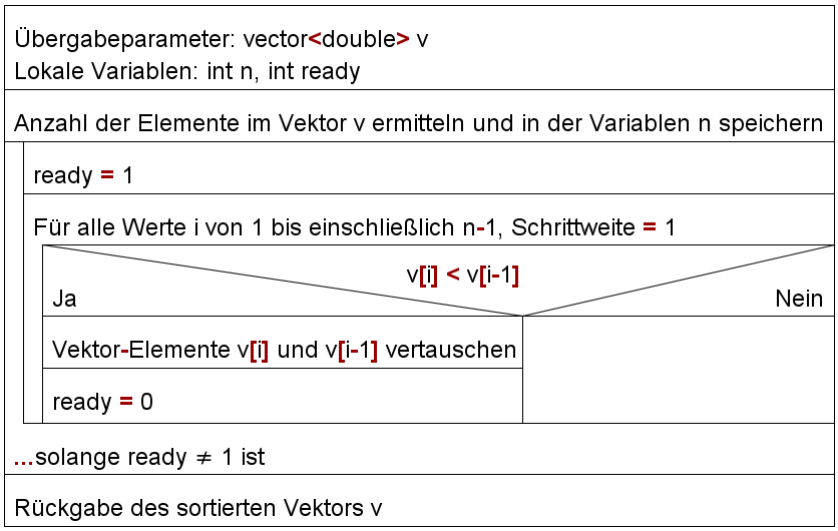
Aufgabensteller: **Dr. Reichl, Dr. Küpper**

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

Viel Erfolg!!!

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

1.1. Ergänzen Sie den auf der rechten Seite abgebildeten Quelltext um eine Funktion **sort_vec** zum Sortieren eines Vektors. Der Ablauf des Sortierverfahrens ist im Struktogramm dargestellt, es handelt sich um das sog. Bubble-Sort-Verfahren. Die von Ihnen erstellte Funktion **sort_vec** soll dem Struktogramm entsprechen:



1.2. Erstellen Sie ein Hauptprogramm (Funktion **main**), welches die folgenden Aufgaben erledigt:

- Ein Vektor wird mit 100 zufälligen double-Zahlen im Bereich von 0,25…0,75 gefüllt.
- Der Vektor wird durch Aufruf der Funktion **sort_vec** sortiert.
- Die Elemente des sortierten Vektors werden der Reihe nach auf dem Bildschirm ausgegeben.

```cpp
#include <vector>
#include <iostream>

using namespace std;
typedef vector<double> d_vec;

d_vec sort_vec(d_vec v)
{
      // ***** AUFGABE 1.1 *****









}
int main(void)
{
    // ***** AUFGABE 1.2 *****









}
```

Aufgabe 2: Objektorientierte Programmierung (ca. 17 Punkte)

Es soll eine Klasse **Vec3D** zur Arbeit mit dreidimensionalen Vektoren (in kartesischen Koordinaten) erstellt werden.

2.1. Definieren Sie einen Konstruktor, der die x-, y- und z-Komponenten des Vektors auf null setzt.

2.2. Definieren Sie die Methode **Length** zur Berechnung der Länge (euklidische Norm) des Vektors.

2.3. Definieren Sie einen Operator zur Addition von zwei **Vec3D**-Instanzen.

2.4. Definieren Sie einen Operator zur Ausgabe von **Vec3D**-Instanzen auf Streams. Die Ausgabe soll in dem folgenden Format erfolgen (inkl. der eckigen Klammern): `[1.111111; 2.222222; 3.333333]`

2.5. Schreiben Sie ein Hauptprogramm, in dem die folgenden drei Dinge passieren:

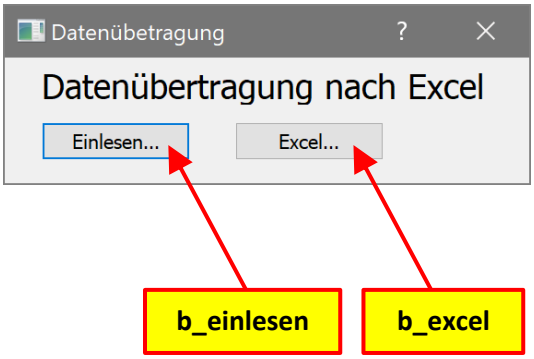
- Der Anwender gibt (nach einer Eingabeaufforderung) jeweils einen x-, y- und z-Wert über die Tastatur ein.
- Es wird eine neue Instanz der Klasse **Vec3D** angelegt. Die x-, y- und z-Komponenten dieser Instanz werden auf die vom Anwender eingegebenen Werte gesetzt.
- Der Vektor und seine Länge werden auf dem Bildschirm ausgegeben. Dabei sollen die in den Unterpunkten 2.1 bis 2.4 definierten Funktionen benutzt werden.

```
#include <iostream>
#include <math.h>
using namespace std;

class Vec3D
{
private:
    double x, y, z;

public:

    // ***** AUFGABE 2.1 *****









    double Length()
    {
        // ***** AUFGABE 2.2 *****









    }
```

```
    void Set(double x0, double y0, double z0)
    {
       x = x0;  y = y0;  z = z0;
    }

    double GetX()
    {
        return x;
    }

    double GetY()
    {
        return y;
    }

    double GetZ()
    {
        return z;
    }
};
Vec3D operator+ (Vec3D a, Vec3D b)
{
    // ***** AUFGABE 2.3 *****




}
ostream& operator<< (ostream& strm, Vec3D v)
{
    // ***** AUFGABE 2.4 *****




}
int main(void)
{
    // ***** AUFGABE 2.5 *****









}
```

Aufgabe 3: Grafische Benutzeroberflächen, COM-Schnittstellen (ca. 15 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 gelöscht 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:

```
0.123 0.135
0.286 0.151
0.217 0.298
0.101 0.182
```

- 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:

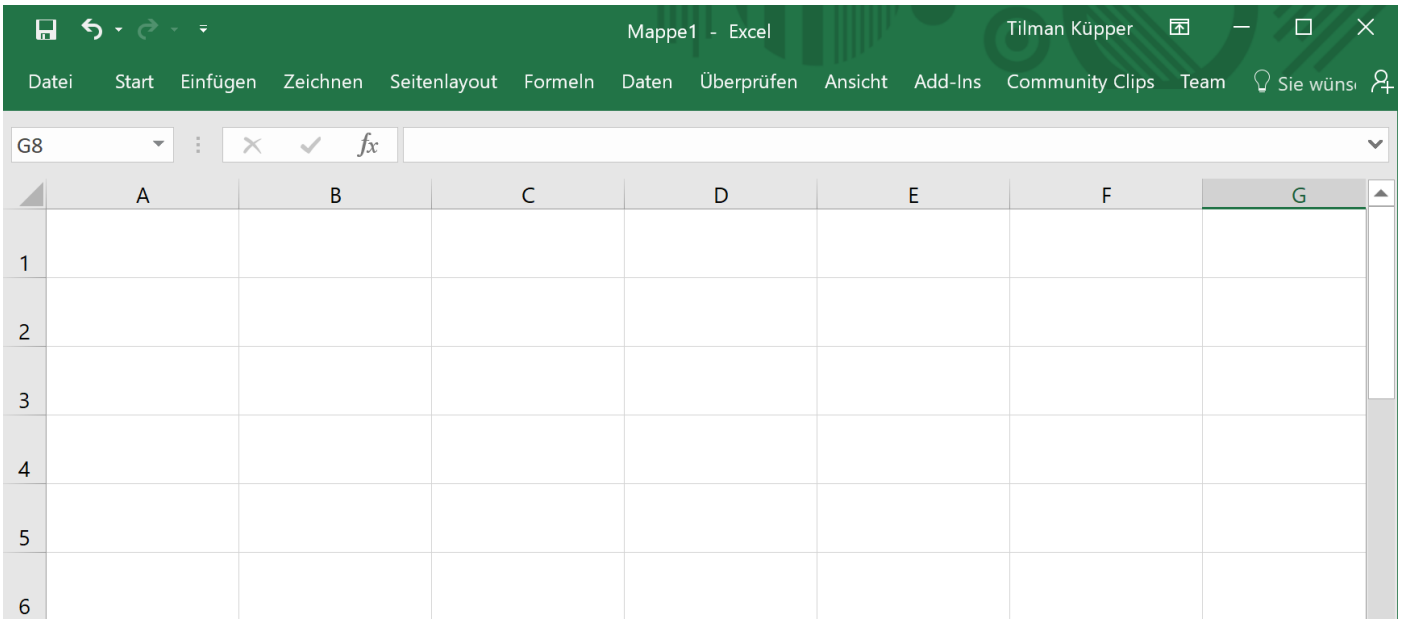
```
class Punkt
{
public:
    double x, y;
};

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" löschen; 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 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, 4);
    cells->setProperty("Value", "Anzahl:");

    cells = active->querySubObject("Cells(int,int)", 1, 5);
    cells->setProperty("Value", data.size());

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

        cells = active->querySubObject("Cells(int,int)", row, 2);
        cells->setProperty("Value", p.y);

        ++row;
    }
}
```



(Platz für Notizen und Nebenrechnungen)