Masterstudiengang Computational Engineering

Programmierung von CAx-Systemen – Teil 1

Name	Vorname	Matrikelnummer

Aufgabe 1	Aufgabe 2	Aufgabe 3	Summe	

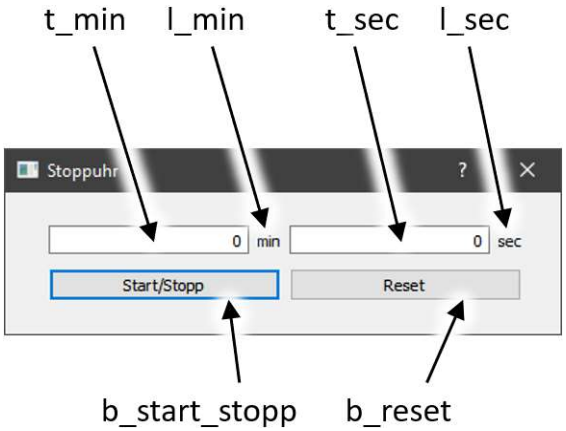
Aufgabensteller: **Dr. Reichl, Dr. Küpper**

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

Viel Erfolg!!!

Aufgabe 1: Grafische Benutzeroberfläche, endlicher Automat (ca. 15 Punkte)

Die folgende Abbildung zeigt die grafische Benutzeroberfläche einer einfachen Stoppuhr. Es handelt sich dabei um eine QtWidgets-Dialogapplikation.



1.1. Der Programmablauf wird durch einen endlichen Automaten (engl. finite state machine) gesteuert. Vervollständigen Sie die folgende grafische Darstellung des Automaten mit allen Zuständen und Zustandsübergängen (inkl. Bedingungen und Aktionen an den Zustandsübergängen).

1.2. Markieren Sie in der Datei dialog.cpp den Konstruktor und Destruktor der Dialog-Klasse.

1.3. Wenn bei einer laufenden Stoppuhr die Start/Stopp-Taste gedrückt wird, bleibt die Stoppuhr stehen.

Was passiert, wenn jetzt die Start/Stopp-Taste erneut gedrückt wird? Beginnt die Stoppuhr wieder bei 00:00 oder zählt die Stoppuhr von der aktuellen Position aus weiter? (Kurze Begründung!)

1.4. Im Attribut Dialog::current_time ist immer die aktuelle Zeit (in Sekunden) gespeichert, die von der Stoppuhr angezeigt wird. Zur Anzeige in der Benutzeroberfläche muss diese aktuelle Zeit zunächst noch in Minuten und Sekunden umgerechnet werden.

Vervollständigen Sie dazu den Quelltext am Ende der Methode Dialog::StateMachine().

Tipp: Mit ui->t_min->setText(my_qstr); kann eine QString-Zeichenkette im Textfeld t_min dargestellt werden. Dieser Aufruf von setText() bei einem Textfeld funktioniert exakt so, wie der Aufruf der Methode setText() bei einer Message-Box, der im Praktikum besprochen wurde.

Ausschnitt aus der Datei dialog.h:

```
class Dialog : public QDialog
{   Q_OBJECT

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

private slots:
    void on_b_start_stopp_clicked();
    void on_b_reset_clicked();

private:
    unsigned int state, current_time;
    Ui::Dialog *ui;
    void timerEvent(QTimerEvent *);
    void StateMachine(bool start_stopp, bool reset, bool timer);
};
```

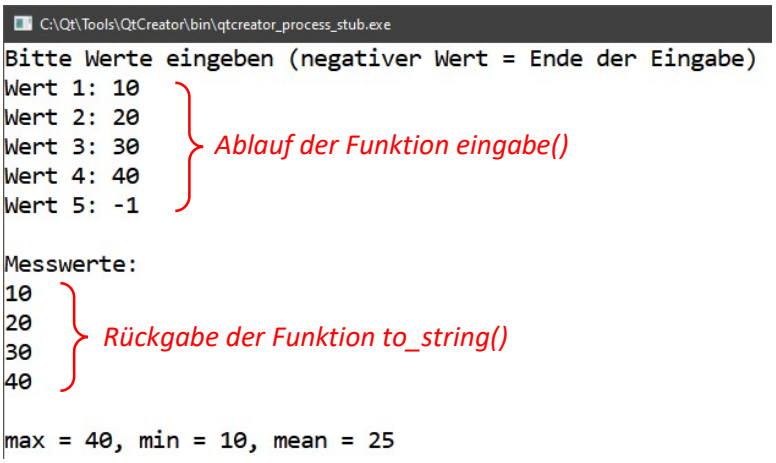
Ausschnitt aus der Datei dialog.cpp:

```
#include "dialog.h"
#include "ui_dialog.h"
#include <sstream>  // std::stringstream
using namespace std;

Dialog::Dialog(QWidget *parent)
    : QDialog(parent) , ui(new Ui::Dialog)
{
    ui->setupUi(this);
    startTimer(1000);
    current_time = state = 0;
}

Dialog::~Dialog() { delete ui; }
void Dialog::on_b_start_stopp_clicked() { StateMachine(true , false, false); }
void Dialog::on_b_reset_clicked()       { StateMachine(false, true , false); }
void Dialog::timerEvent(QTimerEvent *)  { StateMachine(false, false, true ); }
void Dialog::StateMachine(bool start_stopp, bool reset, bool timer)
{
    switch(state)
    {
    case 0:
        if(reset == true) { current_time = 0; state = 0; }
        else if(start_stopp == true) { state = 1; }
        break;

    case 1:
        if(timer == true) { ++current_time; state = 1; }
        else if(reset == true) { current_time = 0; state = 0; }
        else if(start_stopp == true) { state = 0; }
    }

    // current time in Minuten/Sekunden umrechnen und in t min/t sec ausgeben.
```

Aufgabe 2: Arbeiten mit der C++-Standardbibliothek (ca. 15 Punkte)

Das folgende C++-Programm liest zunächst eine Reihe von Messwerten über die Tastatur ein. Alle Messwerte werden anschließend wieder ausgegeben, ebenso das Maximum, das Minimum und der Mittelwert.

2.1. Vervollständigen Sie den Quelltext so, dass die folgende Bildschirm-Ausgabe erscheint. Beachten Sie dabei die Kommentare im Quelltext:



2.2. Ändern Sie das Hauptprogramm so, dass die Ausgaben des Programms nicht mehr auf den Bildschirm, sondern in die Datei c:/temp/test.dat geschrieben werden.

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>  // std::ostringstream
#include <fstream>  // std::ofstream
using namespace std;

typedef vector<double> d_vec;

// In dieser Klasse werden die Ergebnisse der Funktion statistik() gespeichert.
class StatResult
{
public:
    double min, max, mean;  // Minimum, Maximum und Mittelwert;
    StatResult(double min_, double max_, double mean_)
    { min = min_; max = max_; mean = mean_; }
};

// Die Funktion statistik() ermittelt das Minimum, das Maximum und den Mittelwert
// von allen Messwerten im Vektor "v". Die Ergebnisse werden als StatResult-Instanz
// zurückgegeben. Falls "v" leer ist, werden alle Ergebnisse auf -1 gesetzt.
StatResult statistik(d_vec v)
{
    // Vektor leer?
    if(v.size() == 0) return StatResult(-1, -1, -1);

    // TODO: Minimum, Maximum, Mittelwert ermitteln und zurückgeben.
```

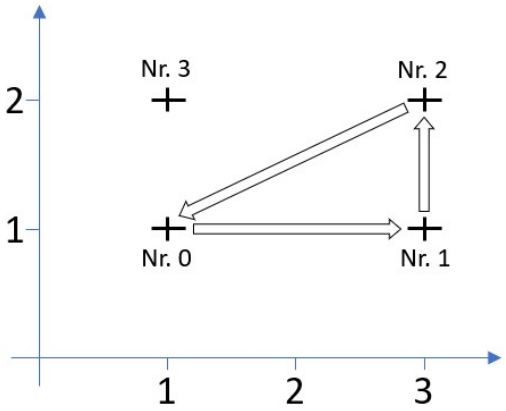
```
// Die Funktion eingabe() liest solange Messwerte von der Tastatur ein, bis ein
// negativer Wert eingegeben wird. Die eingegebenen Werte werden als double-Vektor
// zurückgegeben. Die Funktion eingabe() soll so ablaufen, wie im Bildschirmfoto
// gezeigt. (Eingabeaufforderungen "Wert 1:" usw. nicht vergessen!)
d_vec eingabe(void)
{
    cout << "Bitte Werte eingeben (negativer Wert = Ende der Eingabe)" << endl;

    // TODO: Schleife zur Eingabe der Messwerte programmieren.









}

// Die Funktion to_string() gibt alle Messwerte im Vektor "v" als std::string
// zurück. (Beachten Sie das Bildschirmfoto für ein konkretes Beispiel!)
string to_string(d_vec v)
{
    // TODO: Quelltext der Funktion to_string() schreiben.









}

int main(void)
{
    d_vec v = eingabe();

    StatResult s = statistik(v);

    // TODO: Hauptprogramm so ändern, dass alle Ausgaben in die Datei
    // c:/temp/test.dat geschrieben werden (und nicht mehr auf den Bildschirm).

    cout << endl << "Messwerte:" << endl << to_string(v) << endl;

    cout << "max = " << s.max << ", min = " << s.min << ", mean = " << s.mean << endl;

    return 0;
}
```

Aufgabe 3: Definition eigener Klassen (ca. 20 Punkte)

Die **Klasse „Punkt“** speichert Punkt-Koordinaten in der xy-Ebene. Die Klasse „Punkt“ besitzt zwei Attribute x und y sowie einige Methoden, um diese beiden Attribute zu schreiben bzw. zu lesen.

Die **Klasse „Streckenzug“** dient dazu, die Länge von Streckenzügen über mehrere Punkte hinweg zu berechnen:

- Zunächst werden mit der Methode PunktNeu() einige Punkte definiert und (der Reihe nach) im privaten Vektor „punkte“ abgelegt. Der zuerst definierte Punkt wird am Index 0 des Vektors „punkte“ abgelegt. Die folgenden Punkte an den Indizes 1, 2, 3 usw.
- Anschließend kann die Methode Laenge() aufgerufen werden. Als Parameter wird eine Reihe von Punkt-Indizes übergeben. Die Methode Laenge() berechnet dann die Länge des Streckenzugs.



Beispiel:

- Zunächst werden die Punkte 0, 1, 2, 3 definiert.
- Anschließend wird der Streckenzug von Punkt 0 über Punkt 1 und Punkt 2 zurück zu Punkt 0 berechnet.
- Ergebnis: Länge = 5,23607
- (Der Punkt Nr. 3 wird bei diesem Streckenzug also gar nicht benötigt.)

Im Hauptprogramm main() werden zunächst die Punkte 0, 1, 2, 3 definiert. Anschließend wird die Länge des Streckenzugs von Punkt 0 über Punkt 1 und Punkt 2 zurück nach Punkt 0 berechnet.

Vervollständigen Sie den vorbereiteten Quelltext so, dass die folgende Bildschirm-Ausgabe erscheint:

```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Liste der Punkte:
(1,1)
(3,1)
(3,2)
(1,2)

Laenge des Streckenzugs: 5.23607

Leerer Streckenzug:
Keine Punkte definiert!
```

siehe Hauptprogramm, Position „xxx“

Position „yyy“

Position „zzz“

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>  // std::ostringstream
#include "math.h"
using namespace std;

// Ein einzelner Punkte mit x- und y-Koordinaten
class Punkt
{
private:
    double x, y;

public:
    Punkt(double x_, double y_) { x = x_; y = y_; }
    double GetX() { return x; }
    double GetY() { return y; }
};
```

Hinweis: Sie dürfen davon ausgehen, dass gültige x- und y-Koordinaten stets positiv sind.

```
// TODO: Abstand zweier Punkte berechnen und zurückgeben
//       (euklidische Länge, Ergebnis ist immer >= 0).
double operator- (Punkt a, Punkt b)
{









}

// Typdefinitionen für Vektoren
typedef vector<int> int_vec;
typedef vector<Punkt> p_vec;

// Klasse zur Streckenberechnung: Zunächst werden einige Punkte definiert;
// anschließend kann die Länge von Streckenzügen berechnet werden.
class Streckenzug
{
private:
    p_vec punkte;  // Liste aller Punkte

public:
    // TODO: Anzahl der Punkte im Vektor "punkte" zurückgeben.
    int AnzahlPunkte() {



    // TODO: Alle Punkte im Vektor "punkte" löschen.
    void LoeschePunkte() {



    // TODO: Einen (weiteren) Punkt zum Vektor "punkte" hinzufügen.
    void PunktNeu(Punkt p) {



    // TODO: Punkt am Index "idx" zurückgeben. Ist der Index ungültig (zu klein
    // oder zu groß) wird ein Punkt mit den Koordinaten (-1,-1) zurückgegeben.
    Punkt GetPunkt(int idx)
    {









    }
```

```
    // TODO: Länge eines Streckenzugs berechnen. Parameter: Vektor mit Punkt-Indizes.
    // Rückgabe: Berechnete Länge. Im Fehlerfall (es wurden weniger als zwei Punkte
    // definiert oder einzelne Indizes sind ungültig) wird eine -1 zurückgegeben.
    double Laenge(int_vec indizes)
    {









    }
};

// TODO: Alle Punkte eines Streckenzugs auf Stream ausgeben. Falls keine Punkte
// vorhanden sind, wird die Meldung "Keine Punkte definiert!" ausgegeben.
ostream& operator<< (ostream& strm, Streckenzug s)
{









}

int main(void)
{
    // Vier Punkte zum Streckenzug "s" hinzufügen.
    Streckenzug s;
    Punkt p0(1,1), p1(3,1), p2(3,2), p3(1,2);
    s.PunktNeu(p0); s.PunktNeu(p1); s.PunktNeu(p2); s.PunktNeu(p3);

    cout << "Liste der Punkte:" << endl << s << endl;                    // XXX

    // Länge des Streckenzugs entlang der Punkte Nr. 0, 1, 2, 0 berechnen.
    int_vec indizes { 0, 1, 2, 0 };  // Vektor mit Punkt-Indizes
    cout << "Laenge des Streckenzugs: " << s.Laenge(indizes) << endl;    // YYY

    s.LoeschePunkte();  // Zum Test: Ausgabe einer leeren Punktliste ...
    cout << endl << "Leerer Streckenzug:" << endl << s;                  // ZZZ
    return 0;
}
```