Aufgabe 1: Nullstellensuche (6 Punkte)

Schreiben Sie ein vollständiges C++-Programm, das per Intervallhalbierung die Nullstelle von $f(x) = e^x - x - 2$ bestimmt. Anfangsintervall soll $[0, 3]$ sein, in diesem Intervall hat f eine einfache Nullstelle.
In jedem Iterationsschritt soll f in der Mitte des aktuellen Intervalls ausgewertet werden und dann nach Vorzeichen des Funktionswerts entweder die obere oder die untere Intervallgrenze neu gesetzt werden, so dass die Nullstelle im neuen Intervall ist.
Führen Sie die Rechnung so lange fort, bis entweder die Intervallgröße kleiner $\varepsilon_x = 10^{-3}$ oder der Betrag des Funktionswerts kleiner als $\varepsilon_f = 10^{-2}$ ist. Geben Sie dann den ermittelten Wert der Nullstelle aus.
Hinweis: Betragsfunktion `abs(.)`, Exponentialfunktion: `exp(.)`.

Aufgabe 2: Charfeld umdrehen (4 Punkte)

Schreiben Sie eine C++-Funktion `reverse`, die ein Feld von `char`-Variablen umkehrt. Die Funktion soll als Argument die Länge des Feldes n und das Feld selbst erhalten. Das Feld umkehren heißt, dass der Buchstabe der vorher an erster Postion stand, an die letzte Feldposition kommen soll, der zweite Buchstabe an die zweitletzte Position, etc. Die gültigen Feldindizes bewegen sich zwischen 0 und $n - 1$.
Alle benötigten Variablen sollen lokal definiert oder Argumente der Funktion sein – verwenden Sie keine globalen Variablen.

Aufgabe 3: Was wird ausgegeben? (4 Punkte)

Setzen Sie an der markierten Stelle in folgendem Programm die letzte Ziffer Ihrer Matrikelnummer ein. Schreiben Sie dann *vollständig* auf, was von dem Programm ausgegeben wird.

```
#include <iostream>
using namespace std ;

int f(int& p) {
    p -= 1 ;    3   (2)
    if (p==0) return 1 ;
    int q = p ; 3
    return f(p)*q ;     3·2·1·1 = 6   a(wird bis 0 dekrementiert
}

int f(int* p) {
    int q = *p ;  8
    q += 2 ;  10
    *p += 3 ;   11  (1)
    return q + 5 ;  15
}

int main() {
    int a1= 2 ; // <- setzen Sie hier bitte die *letzte* Ziffer
                //    Ihrer Matrikelnummer ein.
    int a2 = 4 + (a1%10)/100 ;  = 4

    int a3 = f(&a1) ;   a3 = 15 ; a1 = 11
    cout << "Main1: " << a1 << ", " << a3 << endl ;
    a3 = f(a2) ;        a3 = 6 ; a1 = 0
    cout << "Main2: " << a2 << ", " << a3 << endl ;
}
```

Aufgabe 4: Rechteck-Klasse (8 Punkte)

Schreiben Sie eine Klasse `rechteck`, die folgendes leistet: Rechtecke werden durch Breite und Höhe charakterisiert und über Methoden werden folgende Operationen angeboten: (i) Setzen der Werte; (ii) Skalieren beider Längen; (iii) Überprüfen, ob ein Rechteck ein anderes vollständig überdeckt. Ferner soll es eine freie Funktion geben, die die Fläche eines Rechtecks ermittelt.
Die Rechtecke sind nicht an irgendwelchen Koordinaten verankert, sondern nur durch ihre zwei Kantenlängen charakterisiert. Diese Variablen sollen von außen unzugänglich sein.
Die Klasse kommt in folgendem Programm zum Einsatz:

```
#include <iostream>
using namespace std ;

// Klasse rechteck und freie Funktion zu ergaenzen

int main()
{
    rechteck r1(1.7,2) ;    // Rechteck mit Breite, Höhe vereinbaren
    rechteck r2(3,3.5) ;    // ... und noch eines
    r1 *= 2 ;               // beide Längen des Rechtecks-Arguments skalieren
                            // dabei wird r1 selbst veraendert
    if (r1.bedeckt(r2))
       cout << "Rechteck 1 kann Rechteck 2 vollst. ueberdecken" << endl ;

    cout << "Die Flaeche von Rechteck 1 ist " << flaeche(r1) << endl ;
}
```

Schreiben Sie die benötigte Klasse `rechteck` und alle benötigten Konstruktoren, Mitgliedsfunktionen und freien Funktionen, die in dem gegebenen Hauptprogramm aufgerufen werden. Die Methode `bedeckt` soll keine eventuellen Rotationen der Rechtecke beinhalten, sondern nur in der gegebenen Orientierung ermitteln, ob das erste Rechteck das zweite komplett überdecken kann. Der Rückgabewert soll ein Logikwert sein.

Aufgabe 5: Fröhliche Zahlen (8 Punkte)

Die Ziffernquadratsumme bezeichne die Summe der Quadrate der einzelnen Ziffern einer positiven ganzen Zahl. (Bsp: Die Ziffernquadratsumme von 19 ist $1^2 + 9^2 = 82$).
(a) Schreiben Sie eine Funktion `zqs` zur Berechnung der Ziffernquadratsumme.
(b) Man kann eine Folge erzeugen, indem von einem gegebenen Startwert wiederholt die Ziffernquadratsumme gebildet wird. Dabei gibt es *genau zwei* Möglichkeiten (hier ohne Beweis), was mit der Folge passieren kann: (i) die Folge kommt irgendwann zur Zahl 1 oder (ii) die Folge landet bei dem periodischen Zyklus $4 \to 16 \to 37 \to 58 \to 89 \to 145 \to 42 \to 20 \to 4$. (Mit „$\to$“ sei das Berechnen der Ziffernquadratsumme, also der Schritt zum nächsten Folgenglied, bezeichnet).
Startzahlen, bei denen die Folge irgendwann den Wert 1 erreicht, heißen *fröhliche Zahlen*.
Bsp: 19 ist fröhlich:

$19 \to 82 \to 68 \to 100 \to 1 \to \ldots$

29 ist nicht fröhlich:

$29 \to 85 \to 89 \to 145 \to 42 \to 20 \to 4 \to \ldots$

Schreiben Sie ein vollständiges C++-Programm, das für eine einzugebende Zahl ermittelt, ob diese Zahl fröhlich ist und dieses Ergebnis ausgibt.
Kommentieren Sie bitte die einzelnen Abschnitte in Ihrem Programm, so dass die einzelnen Teile sofort verständlich werden. Sie brauchen nicht zu überprüfen, ob die eingegebene Zahl positiv ist.

Name: Edelmann Vorname: Kevin Studienfach: Physik (Diplom)

Bitte geben Sie die Nummern der bearbeiteten Aufgaben deutlich lesbar an.

(1)

```
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) { return exp(x) - x - 2; }

int main()
{
    double bot = 0, top = 3, betw;    // obere und untere Schranke festlegen

    do
    {
        betw = (bot + top)/2;         // Mitte des akt. Intervalls

        if(f(betw) < 0) top = betw;   // Funktionswert bei der Mitte größer Null
                                      // => obere Schranke versetzen
        else bot = betw;              // ansonsten untere Schranke versetzen
    } while( abs(f(betw)) >= 0.01 && top - bot >= 0.001);  // Ein-Ausgabe-
                                                           // Bedingung

    cout << "Die Nullstelle liegt bei" << betw << endl;

    return 0;
}
```

(2)

```
void reverse (char (& field)[], int n)   // Eingabe erfolgt über Referenz-Parameter
                                         // der field[]
{
  char temp; int i = 0;
  while (i < (n-1)/2)                    // Feld bis zur Hälfte ablaufen (falls nicht ungerade: Mitte auslassen)
  {
    temp = field[i];                     // vorderen Wert zwischenspeichern
    field[i] = field[n-1-i];             // hinten nach vorne
    field[n-1-i] = temp                  // und vorne nach hinten
    i++;
  }
}
```

3P

(3)

Main1: 11, 15 (mit einer '8' beendet)

Main2: 0, 6

4/4

(4)

```
class rechteck
{
  public:  rechteck();
           rechteck (double w, double h);          0,5
           bool bedeckt (rechteck r2);             0,5
  void     rechteck operator * (double f);         -0,5
           friend double flaeche (rechteck r1);    1
  private: double width, height;                   0,5
};

rechteck::rechteck () { width = 0; height = 0; }
rechteck::rechteck (double w, double h) { width = w; height = h; }   0,5
bool rechteck::bedeckt (rechteck r2)                                 0,5
{
  if (r2.width >= width, && r2.height >= height) return true;        -0,5   0,5
  return false; }
```

Name: Edelmann Vorname: Kevin Studienfach: Physik (Diplom)

Bitte geben Sie die Nummern der bearbeiteten Aufgaben deutlich lesbar an.

rechteck rechteck :: operator *=(double f) −0,5

```
{
    width *= f;
    height *= f;
    return *this;
}
```

} 0,5

```
double flaeche (rechteck r1)
{
    return r1.width * r1.height;   // Breite mal Höhe, Abzug
}
```

0,5 0,5

(5)

```
#include <iostream>
#include <vector>
using namespace std;

int zqs (int n)                    // Funktion zur Berechnung der ZQS
{
    int sum = 0; int num;
    while (n > 0)
    {
        num = n % 10;              // Die letzte Ziffer von n ermitteln
        sum += num * num;          // Quadrat dieser Ziffer zur Summe hinzufügen
        n /= 10;                   // letzte Ziffer abschneiden
    }
    return sum;
}
```

4 P

```
bool istVec (vector<int> vec, int s)        // Funktion, die überprüft, ob eine Zahl
                                            // s im Vektor s enthalten ist
{
  vector<int>::iterator it;

  for (it = vec.begin(); it != vec.end(); it++)
  {
    if (*it == s) return true;              // Zahl gefunden!
  }
  return false;                             // Zahl im gs. Vektor nicht enthalten
}


int main()
{
  int n;
  vector<int> vec;

  cout << "Bitte gebt mir eine positive ganze Zahl ein: ";
  cin >> n;                                 // Zahl einlesen
  cout << "Danke" << endl;
  vec.push_back(n);                         // Zahl merken (für Zyklen-Detektion)
  while(true)                               // Endlosschleife bis zum beabsichtigten Abbruch
  {
    n = zgz(n);                             // n durch seine eigene ZGS ersetzen

    if(n == 1)                              // JUHUUU! Wir sind bei einer 1 angekommen!
    {
      cout << "Die eingegebene Zahl ist tatsächlich fröhlich!" << endl;
      break;
    }
    if(istVec(vec, n))                      // falls eine Zahl neu kommt, die wir schonmal
                                            // hatten, haben wir einen unfröhlichen Zyklus gefunden
    {
      cout << "Das ist eine traurige Zahl!" << endl;
      break;
    }                                       // (-> nächste Zeile wird gelesen!)
```

Name: Edelmann Vorname: Kevin Studiengang: Physik (Diplo

Bitte geben Sie die Nummern der bearbeiteten Aufgaben deutlich lesbar an.

```
    vec.push_back(n);      // Die aktuelle Zahl merken, um Zyklus aufzuspüren
  }  // Ende while-Schleife

  return 0;
}
```

I = 4