Computergrundlagen 2025

Blatt 13: C++

- Abgabetermin für die Lösungen: **25.01.2026, 20 Uhr/ für Montagsgruppe: 23.01.2026, 12 Uhr**
- Bei Fragen wendet euch bitte an eure/n Tutor/in:
  - Mo 11:30: Stephan Haag: `st170833@stud.uni-stuttgart.de`
  - Di 09:45: Julian Hoßbach: `julian.hossbach@icp.uni-stuttgart.de`
  - Mi 14:00: Julian Peters: `julian.peters@icp.uni-stuttgart.de`
  - Do 09:45: Rebecca Stephan: `rebecca.stephan@icp.uni-stuttgart.de`
  - Fr 09:45: Jonas Höpker: `st182335@stud.uni-stuttgart.de`
- Die Übungsaufgaben sollen in der Regel in **Zweiergruppen** bearbeitet werden. Nur in **begründeten Ausnahmefällen** sind Dreiergruppen möglich.
- Die Abgabe der Übungsblätter erfolgt über Ilias.
- Mit Abgabe der Lösungen erklärt Ihr, dass Ihr die Lösung euren Mitstudierenden im Rahmen der Übungsbesprechung vorstellen könnt. Um dies zu überprüfen, muss mindestens zweimal von jedem Teilnehmenden vorgetragen werden. Wenn Ihr das nicht könnt, werden euch die Punkte für die entsprechenden Aufgaben wieder abgezogen.
- **Befehle, die nicht in der Vorlesung besprochen wurden, müssen gegebenenfalls recherchiert werden.**
- **Alle erstellen Skripte sowie ein mit markdown oder Latex erstellter Report (`.pdf`) sind Teil der Abgabe**

Berechnung der Eulerschen Zahl mit C++ (4 Punkte)

Die Eulersche Zahl e kann durch die folgende unendliche Reihe angenähert werden:

$$e = \sum_{k=0} \frac{1}{k!} \tag{1}$$

Vervollständige das folgende C++ Programm.

1. Schreibe eine Funktion `fakultaet`, die die Fakultät einer Zahl berechnet. Nutze dafür eine for-Schleife.
2. Vervollständige die Funktion `approximate_e`, die die Summe der ersten n Terme der Reihe berechnet.
3. Achte auf die korrekten Datentypen: Welche Datentypen sollten `fakultaet` und `approximate_e` zurückgeben? Warum ist diese Wahl sinnvoll?
4. Gebe das Ergebnis mit einer hohen Präzision (bis zu 8 Nachkommastellen) aus.

```
#include <iostream>
#include <iomanip>

// Funktion zur Berechnung der Fakultaet k!
long fakultaet(int k) {
    long result = 1;
    // --- HIER CODE ERGÄNZEN: Berechne die Fakultät mit einer Schleife ---
```

```
    return result;
}

// Funktion zur Annaeherung von e durch n Terme
double approximate_e(int n) {
    double summe = 0.0;
    for (int k = 0; k < n; ++k) {
        // --- HIER CODE ERGÄNZEN: Addiere 1.0 / k! zur Summe ---
        // Hinweis: Nutze die Funktion fakultaet(k)

    }
    return summe;
}

int main() {
    int iterationen = 12;

    // Berechne e
    double e_approx = approximate_e(iterationen);

    // Ausgabe des Ergebnisses
    std::cout << "Annaeherung von e nach " << iterationen << " Iterationen:" << std::endl;

    // --- HIER CODE ERGÄNZEN: Gib e_approx mit 8 Nachkommastellen aus ---
    // Tipp: Nutze std::setprecision aus <iomanip>

    return 0;
}
```

Tipp: Mit `std::setprecision(n)` kannst du festlegen, wie viele Stellen einer Zahl ausgegeben werden sollen.

Vektoren I (2 Punkte)

Implementiere die Funktion `sum_below_limit`, welche alle Elemente eines Vektors aufsummiert, die kleiner als ein gegebenes Limit sind. Vervollständige dafür die mit `...` markierten Lücken im Code unten. Verwende in deiner Implementierung eine Range-basierte Schleife. Warum ist es sinnvoll, dass der Funktion `const std::vector<double>& vector` übergeben wird?

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

... sum_below_limit(const std::vector<double>& vector, double limit) {
    ...
}

int main()
{
    std::vector<double> v = {1.0, 2.0, 3.0, 4.0, 5.0, 42.0};
```

```
    std::cout << sum_below_limit(v, 4.0) << "\n";

    return 0;
}
```

Vektoren II (2 Punkte)

Implementiere die Funktion `scalar_product`, welche das Skalarprodukt zweier Vektoren $\mathbf{v}_1$ und $\mathbf{v}_2$ identischer Länge berechnet. Vervollständige dafür die mit `...` markierten Lücken im Code unten.

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

... scalar_product(const std::vector<double>& vector1,
                   const std::vector<double>& vector2) {
    ...
}

int main()
{
    std::vector<double> v1 = {1.0, 2.0, 3.0, 4.0};
    std::vector<double> v2 = {1.0, -1.0, -1.0, 1.0};

    std::cout << scalar_product(v1, v2) << "\n";

    return 0;
}
```

Vektoren III (2 Punkte)

Implementiere eine Funktion `normalize_vector`, die einen Vektor entgegennimmt und einen neuen Vektor zurückgibt. In diesem neuen Vektor soll jedes Element des ursprünglichen Vektors durch den größten absoluten Wert des Vektors geteilt werden (Normalisierung auf das Maximum). Überlege dir, was passiert, wenn das Maximum 0 ist. Vervollständige dafür die mit `...` markierten Lücken im Code unten. Achte darauf, wie der Rückgabetyp der Funktion definiert sein muss, um einen Vektor zurückzugeben.

```
#include <iostream>
#include <vector>
#include <algorithm> // Tipp: std::max_element könnte hilfreich sein

... normalize_vector(const std::vector<double>& input_vec) {
    // 1. Finde das Maximum im Vektor
    // (Falls der Vektor leer ist, gib einen leeren Vektor zurück)
    double max_val = ...;

    // 2. Erstelle einen neuen Vektor für die Ergebnisse
    std::vector<double> result;
```

```cpp
    // 3. Berechne die normalisierten Werte und füge sie 'result' hinzu
    ...

    return result;
}

int main() {
    std::vector<double> v = {2.0, 5.0, 10.0, 4.0};

    std::vector<double> normalized = normalize_vector(v);

    // Ausgabe der neuen Werte
    for (double val : normalized) {
        std::cout << val << " ";
    }
    std::cout << std::endl;

    return 0;
}
```