

Einführung in die Programmierung

Klausur — WiSe 2025/26 — 11. Februar 2026

Haupttermin, Prüfungsnr. 23201019

Gruppe: Alice

Unbedingt ausfüllen

Nachname

Vorname

Matrikelnummer

Identifikator (Beliebiges Wort zur Identifikation im anonymen Notenaushang)

Unterschrift

Hiermit bestätige ich meine Prüfungsfähigkeit.

Grundregeln

- Die Bearbeitungszeit der Klausur beträgt **120 Minuten**.
- Sie schreiben diese Klausur **vorbehaltlich** der Erfüllung der **Zulassungsvoraussetzung**. Das heißt: Wir werden Ihre Zulassung vor Korrektur prüfen; die Tatsache, dass Sie die Klausur mitschreiben, bedeutet keine implizite Zulassung.
- Es sind **keine Unterlagen** und auch **keine anderen Hilfsmittel** erlaubt.
- Benutzen Sie nur dokumentenechten (blauen oder zur Not schwarzen) **Kugelschreiber**. Bleistiftlösungen werden nicht gewertet.
- Es zählt die Antwort, die sich im dafür vorgesehenen Kästchen befindet! Soll eine andere Antwort gewertet werden, so ist diese **eindeutig** zu kennzeichnen und in der vorgesehenen grauen Box darauf hinzuweisen.
- Jegliches Schummeln, und auch der Versuch desselben, führt zum Ausschluss von der Klausur und einer Bewertung mit **5,0**.

Wird vom Korrektor/Prüfer ausgefüllt

Aufgabe	1	2	3	4	5	6	7	8	Σ
Punkte (max)	10	10	8	10	10	10	10	10	78
Punkte (erreicht)									

Note:

Aufgabe 1: Kurzfragen

(10 Punkte)

Beantworten Sie die Fragen jeweils in 1–2 Sätzen.

(a) Grundbegriffe

(2 Punkte)

Was ist eine *Magic Constant*?

(b) Klassen

(2 Punkte)

Was ist ein *Destruktor*?

(c) Fehlerbehandlung

(2 Punkte)

Was ist eine *Vorbedingung*?

(d) Typsicherheit

(2 Punkte)

Was bedeutet *Überlauf*?

A large, empty rectangular box with a thin black border, intended for the student to write their answer to question (d).

(e) Funktionen

(2 Punkte)

Was bedeutet *Call-by-Reference*? Nennen Sie einen Vorteil, den dieses Konzept bietet.

A large, empty rectangular box with a thin black border, intended for the student to write their answer to question (e).

Aufgabe 2: Fehlertypen

(10 Punkte)

Betrachten Sie die folgenden C++-Programme. Gehen Sie davon aus, dass am Anfang jedes Programms noch die Zeile

```
#include "std_lib_inc.h"
```

steht. Ansonsten sind die Programme vollständig. Gehen Sie zudem von den typischen Speichergrößen der Typen aus. Jedes Programm verursacht entweder einen oder gar keinen Fehler. Kreuzen Sie entsprechend an.

Hinweis: Das Nicht-ankreuzen einer Teilaufgabe gibt 0 Punkte für diese Teilaufgabe; falsche Kreuze führen zu Punkteabzug. Die Gesamtaufgabe kann jedoch nicht weniger als 0 Punkte bringen.

	kein Fehler	Fehler zur...			Logikfehler
		Kompilierzeit	Linkzeit	Laufzeit	
<pre>int main() { int alter; string name; cin >> name >> alter; // Gibt den Namen und das Alter aus. cout << name + alter << '\n'; return 0; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<pre>class Spiel { int runde_; public: Spiel(int runde); int runde(); void starten(); }; int main() { Spiel spiel(0); // Spiel starten spiel.starten(); return 0; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<pre>// Gibt den Inhalt von v rueckwaerts aus. template<typename T> void reverse(const vector<T>& v) { for(int i = 1; i < v.size() + 1; ++i) { cout << v[v.size() - i] << '\n'; } } int main() { vector<int> v = {1, 2, 3, 4}; reverse(v); return 0; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

	kein Fehler	Fehler zur...			Logikfehler
		Kompilierzeit	Linkzeit	Laufzeit	
<pre> int addition(char* ein_zeichen_ptr) { return *(ein_zeichen_ptr + 5); } int main() { char ein_zeichen = 'C'; int wert = addition(&ein_zeichen); // Gib den Wert des Zeichens ('C' + 5) aus. cout << wert << '\n'; return 0; } </pre>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<pre> // Addiert zwei doubles und gibt das // Ergebnis zurueck. double add(double a, double b); int main() { // Gib die Summe von 3 + 4 aus. cout << add(3, 4) << '\n'; return 0; } int add(int a, int b) { return a + b; } </pre>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Aufgabe 3: Programmsemantik

(8 Punkte)

Geben Sie exakt die jeweilige Ausgabe der folgenden C++-Programme an. Sie können davon ausgehen, dass keine Compiler-Optimierung stattfindet und jedes mal die Header-Datei „std_lib_inc.h“ eingebunden ist.

(a) (2 Punkte)

```
int main() {
    int* werte = new int[]
        {1, 2, 3, 4, 5, 6};

    int* a = &werte[3];
    int* b = a - 2;
    int* c = b + b[1];
    int* d = &c[*c - 6];

    cout << *a << *b << *c << *d;

    delete[] werte;
    return 0;
}
```

Ausgabe:

(b) (2 Punkte)

```
int f(int a, int& b) {
    b += 10;
    return b / a;
}

int main() {
    int x = 10;
    int y = 42;
    y = f(16, x);
    f(1, y);
    x = f(3, x);
    cout << x << y;
    return 0;
}
```

Ausgabe:

(c) (2 Punkte)

```
int f(int x) {
    if(x > 10)
        { throw runtime_error("X"); }
    if(x < 7)
        { throw runtime_error("Y"); }
    return 42;
}

int main() {
    try {
        cout << f(8);
        cout << f(0);
        cout << f(13);
    } catch(runtime_error& e) {
        cout << e.what();
    }
    cout << "Z";
    return 0;
}
```

Ausgabe:

(d) (2 Punkte)

```
struct O {
    virtual void f() {
        cout << "O::f()";
    }
};

struct Q : public O {
    void f() override {
        cout << "Q::f()";
    }
};

int main() {
    Q q;
    O& r = q;
    r.f();
    return 0;
}
```

Ausgabe:

Aufgabe 4: Scopes

(10 Punkte)

(a) Scopes finden

(6 Punkte)

Markieren Sie alle Scopes im folgenden Programmcode und benennen Sie die Scope-Art. Sie können das entweder durch Umkreisen/Umklamern & Beschriften tun oder als Liste mit Zeilennummern.

```
1  #include "std_lib_inc.h"
2
3  class PrivateClass {
4  private:
5      string secret_name;
6  };
7
8  int sum(int a, int b) {
9      return a + b;
10 }
11
12 int main() {
13     int n = 48, m = 18;
14     for(int i = 1; i <= 5; ++i)
15     {
16         cout << "Der berechnete Wert: " << sum(n, m) << "\n";
17         n += i;
18         m = i * i;
19     }
20     return 0;
21 }
```

(b) Scopes nachvollziehen

(4 Punkte)

Geben Sie exakt die Ausgabe des folgenden C++-Programms an. Sie können davon ausgehen, dass keine Compiler-Optimierung stattfindet.

```
# include "std_lib_inc.h"

class C {
private:
    int c;
public:
    C(int x, int y) {
        c = x;
        x = 5;
        cc = 2 * c;
        cout << c;
    };
    int cc;
    int print_c(int c) {
        return cc/2;
    }
};

int print_c(int c) {
    int cc = 86;
    cout << c;
    return cc/2;
}

int main() {
    int c = 12;
    int x = 42;
    C cc(c, x);
    cout << cc.cc;
    cc.cc = x;
    c = cc.print_c(c);
    c = print_c(c);
    cout << c << "\n";
    return 0;
}
```

Ausgabe:

Aufgabe 5: Rekursion

(10 Punkte)

Betrachten Sie die folgende Funktion $f: \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$.

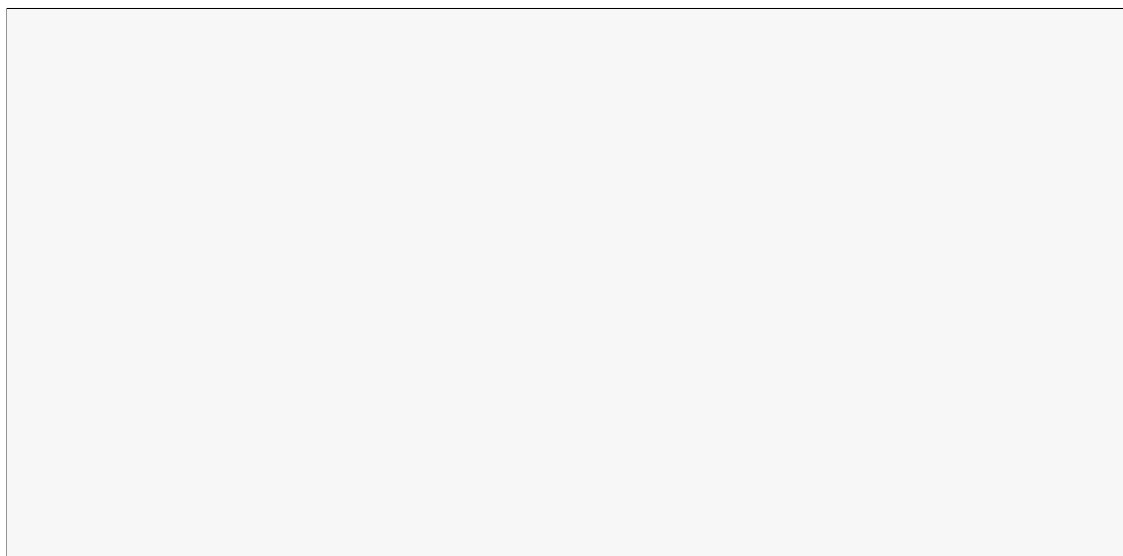
$$f(n, m) = \begin{cases} 1, & \text{falls } n \cdot m = 0 \text{ und } n + m > 0, \\ n, & \text{falls } n = m, \\ f(n - 1, m - 1) + f(n - 2, m), & \text{falls } 0 < m < n, \\ f(n - 1, m) + f(n, m - 1), & \text{sonst.} \end{cases}$$

(a) Rekursive Funktion

(6 Punkte)

Schreiben Sie eine rekursive C++-Funktion, die die obige Funktion genau nach dem Schema berechnet. Sie brauchen sich um Überläufe nicht zu kümmern und können davon ausgehen, dass nur $n \geq 0$ und $m \geq 0$ übergeben werden. Sie dürfen hierfür keine Header-Dateien laden.

```
int f(int n, int m) {
```

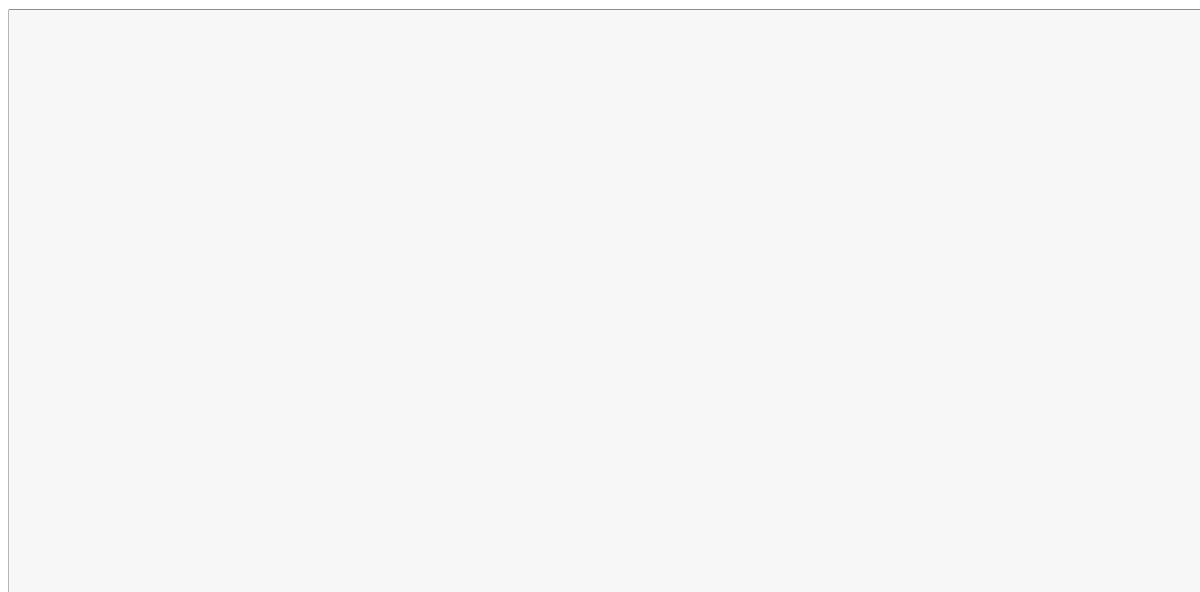


```
}
```

(b) Rekursive Berechnung

(4 Punkte)

Was ist $f(4, 3)$ mit Rechenweg?



Aufgabe 6: Stream-Modell

(10 Punkte)

Schreiben Sie einen Eingabe-Operator nach dem C++-Stream-Modell für das folgende Struct Niederschlag:

```
#include "std_lib_inc.h"
enum class Niederschlagstyp { Schnee, Regen, Hagel };
struct Niederschlag {
    string ort; int menge; Niederschlagstyp typ;
};
// Wandelt str in den passenden Niederschlagstyp um.
// Beispielsweise wird "Schnee" zu Niederschlagstyp::Schnee.
// Vorbedingung: str muss in {"Schnee", "Regen", "Hagel"} sein.
Niederschlagstyp stringToNiederschlagstyp(string str);
```

Die Eingabe besteht aus dem nicht-leeren string ort (ohne Whitespaces), gefolgt von einem der drei Worte „Schnee“, „Regen“ oder „Hagel“ und einer natürlichen Zahl $menge \geq 0$.

Die obige Funktion stringToNiederschlagstyp dürfen Sie verwenden, um einen string in einen Niederschlagstyp umzuwandeln. Sie braucht nicht implementiert zu werden.

Beispiel **Eingabe:** „Osnabrueck Schnee 30“ (ohne Anführungszeichen), gefolgt von einem Zeilenumbruch.
Niederschlag-Objekt: ort = „Osnabrueck“, menge = 30 und typ = Schnee.

```
#include "std_lib_inc.h"
```

Aufgabe 7: Listen

(10 Punkte)

Eine doppelt-verkettete Liste ist definiert durch das folgende Struct:

```
#include "std_lib_inc.h"

struct Eintrag {
    string wert;
    Eintrag* vorgaenger;
    Eintrag* nachfolger;

    Eintrag(const string& w, Eintrag* v, Eintrag* n)
        : wert(w), vorgaenger(v), nachfolger(n) {}
};

Eintrag* finde(const Eintrag* liste, const string& wort);
Eintrag* einfuegen(Eintrag* eintrag, Eintrag* neuer_eintrag);
```

Implementieren Sie die Funktionen `finde` und `einfuegen`: Die Funktion `finde` erhält einen Zeiger `liste` auf das erstes Element einer Liste und ein Vergleichswort `wort`. Die Funktion soll einen Zeiger auf das erste Vorkommen eines Eintrages mit `wert == wort` zurückgeben, wenn so ein Eintrag existiert und ansonsten `nullptr`.

Die Funktion `einfuegen` soll einen Eintrag `neuer_eintrag` hinter dem Eintrag `eintrag` einer Liste einfügen. Nach der Anwendung soll die Liste weiterhin eine korrekte doppelt-verkettete Liste sein. Ist `neuer_eintrag` nicht `nullptr`, so wird `neuer_eintrag` zurück gegeben. Ist `neuer_eintrag == nullptr`, so wird `eintrag` zurückgegeben. Sind beide Parameter `nullptr`, so wird `nullptr` zurückgegeben.

Aufgabe 8: Vererbung

(10 Punkte)

Vervollständigen Sie den Quelltext auf dieser und der nächsten Seite, sodass der Preis des Rennrades und des Mountainbikes korrekt ausgegeben wird. Der Preis von Rennrädern und Mountainbikes berechnet sich unterschiedlich: Der Preis eines Rennrads entspricht dem Attribut `preis_`; der Preis eines Mountainbikes ist `netto_preis_ * (1 + mwst_ / 100)`. Implementieren Sie dazu ein Interface `IFahrrad` mit den entsprechenden Implementierungen in der Klasse `Rennrad` und `Mountainbike`, sowie die separate Funktion `double kosten(const IFahrrad& fr)` (nächste Seite).

Hinweis: Sie müssen sich um das Runden von Ausgabezahlen nicht kümmern.

```
class IFahrrad {
```

```
};
```

```
class Rennrad : public IFahrrad {  
    double preis_  
public:  
    Rennrad(int preis) : preis_(preis) {}
```

```
};
```

```
class Mountainbike : public IFahrrad {  
    double netto_preis_  
    double mwst_  
public:  
    Mountainbike(double netto_preis, double mwst)  
    : netto_preis_(netto_preis), mwst_(mwst) {}
```

```
};
```

```
double kosten(const IFahrrad& fr) {
```



```
}
```

```
int main() {
```

```
    Rennrad rr(300);
```

```
    Mountainbike mb(200, 19);
```

```
    cout << fixed << setprecision(2);
```

```
    cout << kosten(rr) << ", " << kosten(mb) << "\n";
```

```
    return 0;
```

```
}
```

Raum für Notizen:

Viel Erfolg!