

Einführung in die Programmierung

Klausur — WiSe 2024/25 — 26. März 2025

Nebentermin, Prüfungsnr. 23201019

Gruppe: private

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	9	Σ
Punkte (max)	10	10	8	10	10	10	10	10	6	84
Punkte (erreicht)										

Note:

Aufgabe 1: Kurzfragen

(10 Punkte)

Beantworten Sie die Fragen jeweils in 1–2 Sätzen.

(a) Grundbegriffe

(2 Punkte)

Was ist die Aufgabe einer Funktions-Deklaration? Was ist der wesentliche Unterschied zu einer Funktions-Definition?

(b) Klassen-Design

(2 Punkte)

Was bedeutet Const-Correctness?

(c) Objekt-Konstruktion

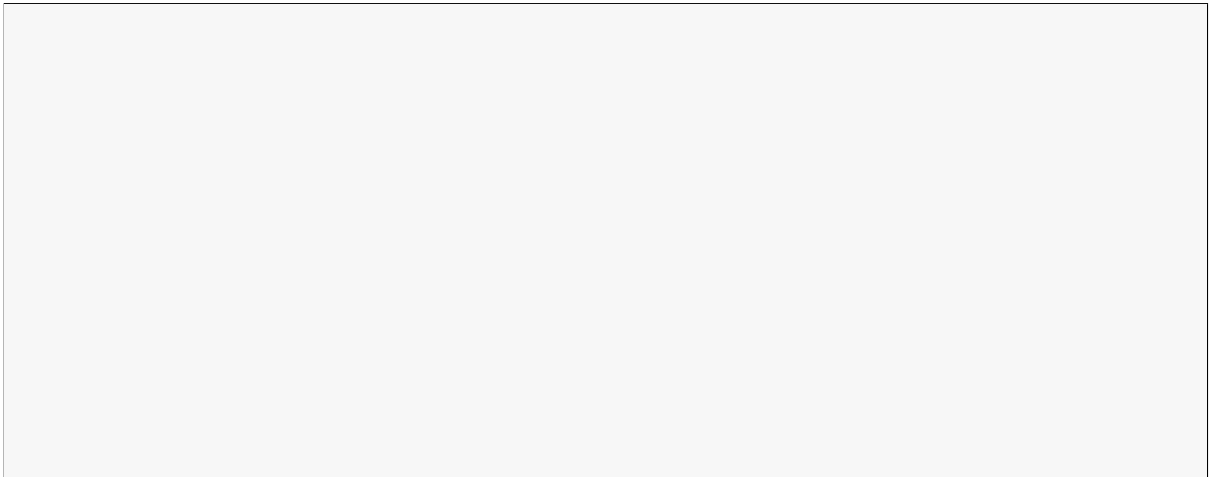
(2 Punkte)

Was ist ein konvertierender Konstruktor? Wie wird er deklariert?

(d) Programm-Sicherheit

(2 Punkte)

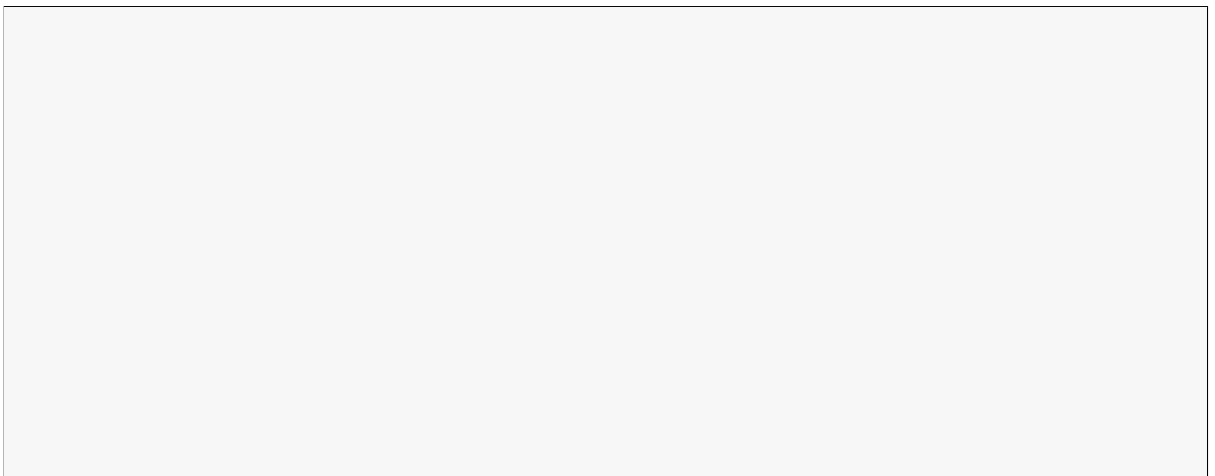
Was ist eine Invariante?

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

(e) Vererbung

(2 Punkte)

Was ist ein Interface im Kontext von Vererbung?

A large, empty rectangular box with a thin black border, intended for the student to write their answer to the question about interfaces in inheritance.

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 a = 25; // Ist a > 10, gib diese Information aus. if(a > 10) { cout << "a ist groesser als 10\n"; } return 0; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<pre>// Template gibt Inhalt von Vector an // Index n * 5 zurueck. template<typename T> T f(int n, vector<T> v) { int i = n * 5; return v[i]; } int main() { vector<int> v(1024); cout << f(300, v) << '\n'; return 0; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<pre>int main() { vector<int> v(100); int i = 0; // Setzt i-ten Index auf i. while(i < 100) { v[i] = i; ++i; } 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>// Funktion befuellt uebergebenes int-Array. void fill(int* int_array) { int i = 0; while(int_array[i] == 0) { int_array[i] = i; ++i; } } int main() { int n = 5000; int* int_array = new int[n]; // Array int_array mit Werten befuellen. fill(int_array); delete[] int_array; return 0; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<pre>class Elefant { double groesse_; // Groesse in Metern public: Elefant(double groesse) : groesse_(0) {} double groesse() { return groesse_; } }; int main() { Elefant emil(3.21); // Gibt die Groesse des oben definierten // Elefanten zurueck. cout << emil.groesse() << "\n"; return 0; }</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 f(int a, int& b) {
    b += 5;
    return a + b;
}
int main() {
    int x = 2;
    int y = 5;
    y = f(3, x);
    f(4, y);
    x = f(5, x);
    cout << x << y;
    return 0;
}
```

Ausgabe:

(b) (2 Punkte)

```
int f(double d) {
    return d / 5;
}
int main() {
    char a = 12;
    double b = 7.5;
    int c = 42;
    cout << f(a);
    cout << f(b);
    cout << f(c);
    cout << f(0);
    return 0;
}
```

Ausgabe:

(c) (2 Punkte)

```
struct A {
    A() { cout << "V"; }
    ~A() { cout << "W"; }
    void m(int) { cout << "O"; }
    void m(string)
        { cout << "Q"; }
};
int main() {
    A a;
    a.m(5);
    A b;
    b.m(3);
    a.m("X");
    return 0;
}
```

Ausgabe:

(d) (2 Punkte)

```
int f(int x) {
    if(x % 2 == 0)
        { throw runtime_error("C"); }
    return x;
}
int main() {
    try {
        cout << f(5);
        cout << f(6);
        cout << f(7);
    } catch(runtime_error& e) {
        cout << e.what();
    }
    cout << 'Z';
}
```

Ausgabe:

(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 struct MeineKlasse {
4     int attribut_;
5
6 public:
7     MeineKlasse() : attribut_(5) {}
8 };
9
10 int main()
11 {
12     if(3 != 5)
13     {
14         for(int i = 0; i < 42; ++i)
15             cout << i * 9 << '\n';
16     }
17     return 0;
18 }
```

(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"

int x = 7;

void f(int& x) {
    x += 3;
}

int main()
{
    int x = 5;
    {
        int x = 10;
        f(x);
        cout << x;
    }
    for(int x = 0; x <= 17; ++x)
    {
        f(x);
    }
    cout << x;
    {
        int x = ::x;
        f(x);
        cout << x;
    }
    cout << ::x;
    return 0;
}
```

Ausgabe:

Aufgabe 5: Rekursion

(10 Punkte)

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

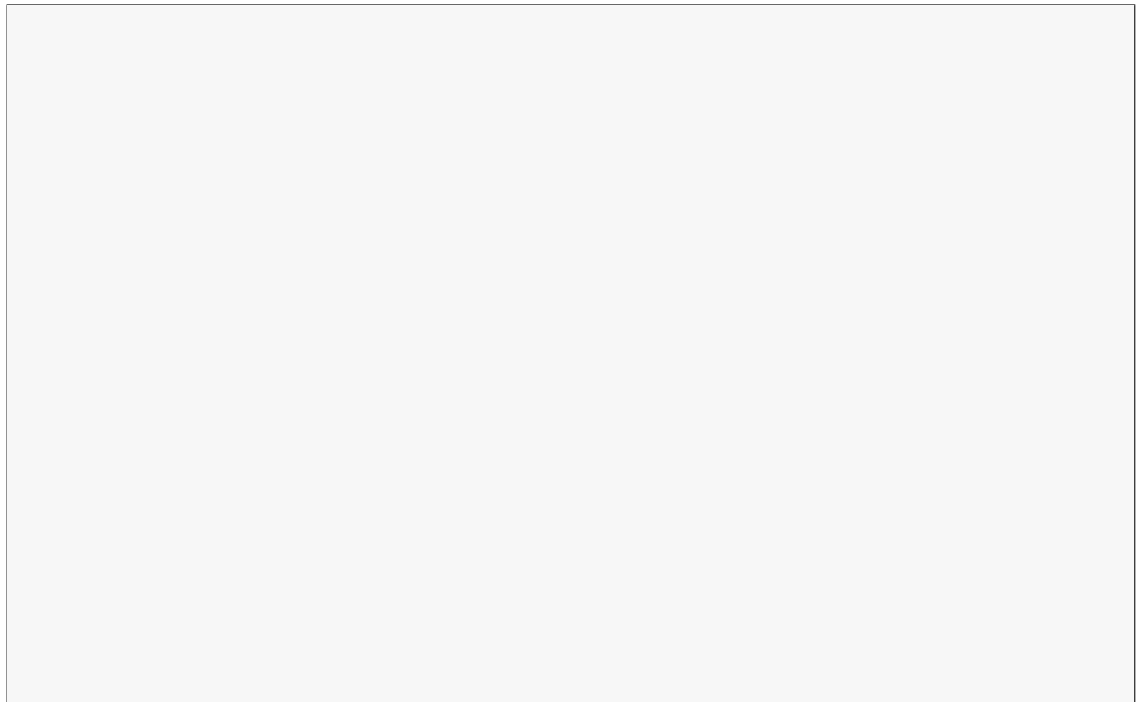
$$f(n, m) = \begin{cases} 0 & m = 0 \\ n & m = 1 \\ 2f(n, \frac{m}{2}) & m > 1 \text{ und } m \text{ gerade} \\ n + f(n, m - 1) & \text{sonst} \end{cases}$$

(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) {
```

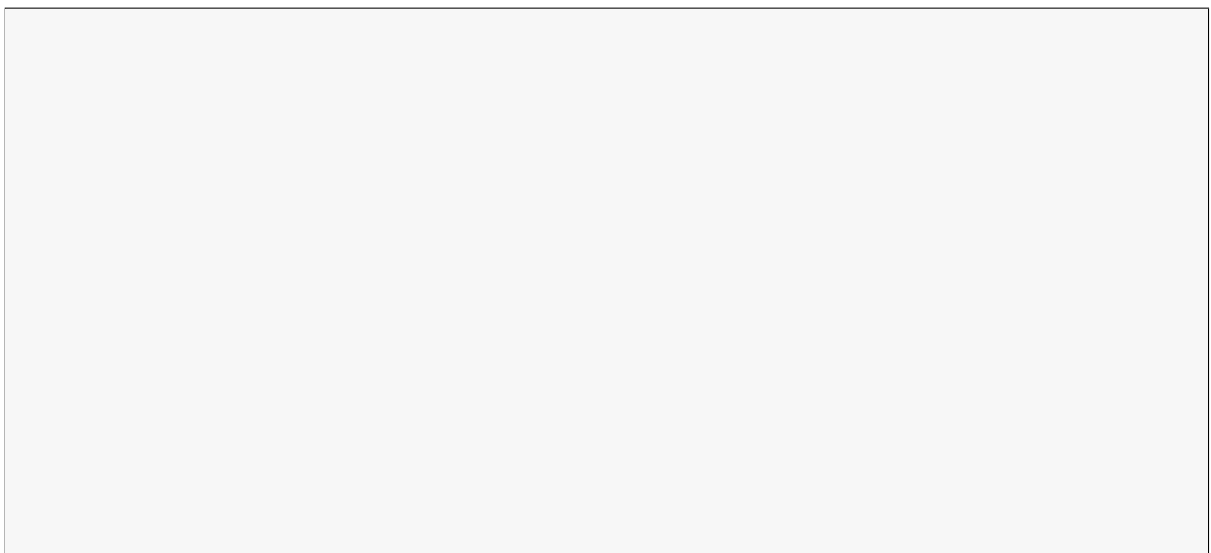


```
}
```

(b) Rekursive Berechnung

(4 Punkte)

Was ist $f(6, 7)$ mit Rechenweg?



Aufgabe 6: Kopier-Kontrolle

(10 Punkte)

Betrachten Sie die folgende Klasse:

```
#include "std_lib_inc.h"

class DatenKlasse {
    double* datum_;

public:
    DatenKlasse() : datum_(new double{0.0}) {}
    DatenKlasse(double datum) : datum_(new double{datum}) {}
};
```

Schreiben Sie einen Destruktor, Copy-Konstruktor und Copy-Assignment-Operator, die gemeinsam sicherstellen, dass beim Kopieren von Objekten der Klasse `DatenKlasse` eine tiefe Kopie stattfindet. Schreiben Sie nur die Implementierung außerhalb der Klasse auf. Sie können annehmen, dass die Deklaration in der Klasse ihrer Implementierung entspricht.

Aufgabe 7: Listen

(10 Punkte)

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

```
#include "std_lib_inc.h"

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

    Eintrag(double w, Eintrag* v, Eintrag* n)
        : wert(w), vorgaenger(v), nachfolger(n) {}
};

Eintrag* loesche(Eintrag* element);
void alles_ausgeben(Eintrag* liste, ostream& stream);
```

Implementieren Sie die Funktionen `loesche` und `alles_ausgeben`: Die Funktion `loesche` erhält einen Zeiger `element` auf das zu löschende Element und soll dieses Element aus der Liste löschen und den Speicher für das Element freigeben. Die Liste soll nach der Operation wieder eine korrekte Liste sein. War die Operation erfolgreich, soll ein Zeiger auf den ursprünglichen Nachfolger von `element` zurückgegeben werden. Wird `nullptr` übergeben, soll `nullptr` zurückgegeben werden. Die Funktion `alles_ausgeben` soll die Werte der `wert`-Attribute aller Listenelemente kommasepariert (und ohne abschließendes Komma) an den übergebenen Ausgabe-Stream `stream` ausgeben. Sie dürfen dabei annehmen, dass der Zeiger `liste` auf das erste Element der Liste zeigt, d.h. der `vorgaenger` von `liste` ist `nullptr`.

Aufgabe 8: Templates

(10 Punkte)

Erstellen Sie ein Funktionstemplate `existiert_wert`, welches einen `begin`- und einen `end`-Iterator, sowie einen `int`-Wert als Parameter übergeben bekommt und einen `bool`-Wert zurückgibt. Das Template soll genau dann `true` zurückgeben, wenn der übergebene `int`-Wert zwischen den beiden Iteratoren vorkommt. Verwenden Sie dabei das Iterator-Entwurfsmuster und nur Operationen aus dem Interface eines Iterators, insbesondere ohne `std::find` zu nutzen. Schreiben Sie den vollständigen Quelltext des Funktionstemplates auf.

Hinweis: Sie dürfen davon ausgehen, dass `begin` nicht hinter `end` liegt und dass die Werte im Container `int`-Werte sind.

Beispiel

Übergeben: Begin- und End-Iterator eines `std::vector<int>` mit Inhalt (4, 5, 2, 2, 4, 3), sowie der Wert 2.

Rückgabe: `true`

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

Aufgabe 9: Vererbung

(6 Punkte)

Betrachten Sie das folgende Teilprogramm:

```
#include "std_lib_inc.h"

class Konto
{
    double saldo_;    // Kontostand

public:
    explicit Konto(double anfangs_wert) : saldo_(anfangs_wert) {}
    double saldo() const { return saldo_; }
    void einzahlen(double wert);
    void abheben(double wert);
};
```

Erweitern Sie die Funktionalität der Klasse Konto durch Vererbung: Es soll dadurch eine neue Klasse Sparkonto entstehen, die zusätzlich zu der Funktionalität von Konto ein privates double-Attribut zinssatz_ und eine neue Methode void zinsen_berechnen() enthält. Schreiben Sie die Definition der Klasse Sparkonto auf. Fügen Sie auch einen Konstruktor hinzu, der sowohl einen anfänglichen Wert des Kontos erhält, als auch den anfänglichen Zinssatz des Kontos. Definieren Sie diesen Konstruktor vollständig. Die Methode void zinsen_berechnen() brauchen Sie nicht zu definieren, die Deklaration genügt.

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

Raum für Notizen: