

Übungsstunde Woche 5

Fynn Krebser

14.10.2026



Frage 1 ■ 50 € Was beschreibt die Ausgabe am besten?

```
int sum = 17;  
int i = 1;  
  
do {  
    i += sum;  
    sum = sum/2;  
} while (i > sum && sum >= 0);  
  
std::cout << sum;
```

A: 17

B: 8

C: Terminiert nicht

D: 18

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

Frage 1 ■ 50 € Was beschreibt die Ausgabe am besten?

```
int sum = 17;  
int i = 1;  
  
do {  
    i += sum;  
    sum = sum/2;  
} while (i > sum && sum >= 0);  
  
std::cout << sum;
```

A: 17

B: 8

C: Terminiert nicht

D: 18

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

50:50

Pub

Tel

Frage 2 ■ 100 €

Welche Pre-Condition ist zwingend nötig
für eine Funktion, die $1.0 / k$ berechnet?

A: $k > 0$ B: $k \neq 0$ C: $k == 0$ D: $k < \text{INT_MAX}$

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

50:50

Pub

Tel

Frage 2 ■ 100 €

Welche Pre-Condition ist zwingend nötig
für eine Funktion, die $1.0 / k$ berechnet?

A: $k > 0$ B: $k \neq 0$ C: $k == 0$ D: $k < \text{INT_MAX}$

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

50:50

Pub

Tel

Frage 3 ■ 200 €

Was passiert, wenn eine Funktion mit Rückgabetyt `int` das Ende erreicht, ohne ein `return`-Statement auszuführen?

A: Gibt automatisch 0 zurück

B: Kompiliert nicht

C: Undefiniertes Verhalten (Undefined Behavior)

D: Programm bricht sauber mit Errorcode -6 ab

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

50:50

Pub

Tel

Frage 3 ■ 200 €

Was passiert, wenn eine Funktion mit Rückgabetyt `int` das Ende erreicht, ohne ein `return`-Statement auszuführen?

A: Gibt automatisch 0 zurück

B: Kompiliert nicht

C: Undefiniertes Verhalten (Undefined Behavior)

D: Programm bricht sauber mit Errorcode -6 ab

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

Serie letzter Woche

- **Loop mix up:** Viele haben sehr ausführlich begründet wieso man welchen Loop wählt. Sehr cool!
- **Loop Analysis:** Passt immer auf Randfälle auf

Prüfungstipp

Prüft, was bei Randfällen, insbesondere $n = 0$, $n < 0$ und $n = \text{INT_MAX}$ passiert. Diese Fälle werden in der Prüfung oft getestet.

Fragen?

Updates auf [code]expert

- Versteckte Testcases: Grundsätzlich nur eine Methode gegen Hardcoding.
- Regrading: Testcases werden geändert
- Eigene Inputs: Spart euch copy & paste, aber nicht für die Bewertung relevant.

Ziel von Heute

- Grundlagen von Funktionen
- Debugging mit `assert()`

Switch Statements: Fall-Through

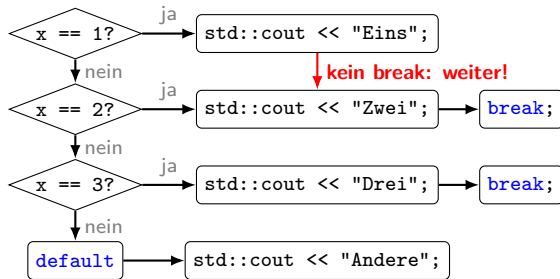
- Springt zum passenden case und führt von dort aus alles weiter unten aus
- Erst ein `break` verlässt den `switch` (*Fall-Through*)
- `case` kann in etwa mit `==` übersetzt werden. $\implies$ Aufpassen was verglichen wird!

```
switch (x) {  
  case 1:  
    std::cout << "Eins";  
  case 2:  
    std::cout << "Zwei";  
    break;  
  case 3:  
    std::cout << "Drei";  
    break;  
  default:  
    std::cout << "Andere";  
}
```

Switch Statements: Fall-Through

- Springt zum passenden case und führt von dort aus alles weiter unten aus
- Erst ein `break` verlässt den switch (*Fall-Through*)
- case kann in etwa mit `==` übersetzt werden. $\implies$ Aufpassen was verglichen wird!

```
switch (x) {  
  case 1:  
    std::cout << "Eins";  
  case 2:  
    std::cout << "Zwei";  
    break;  
  case 3:  
    std::cout << "Drei";  
    break;  
  default:  
    std::cout << "Andere";  
}
```



Assert: Besseres Debugging

- `assert(condition);`
- `condition falsch` $\implies$ Programmabbruch mit Fehlermeldung
- Schöner als `std::cout << condition << std::endl;`

Fehlercodes

- Tritt bei einem unnormalen Programmabbruch auf, z. B. durch `assert(false);`

Fehlercode	Bedeutung
------------	-----------

-6	Abbruch durch ein Assert
----	--------------------------

-8	Abbruch durch Division durch Null
----	-----------------------------------

-9	Programm hat nicht terminiert oder zu viel Speicher gebraucht
----	---

-11	Abbruch durch Segmentation Fault
-----	----------------------------------

50:50

Pub

Tel

Frage 4 ■ 300 €

Wieso sind Asserts nützlich?

A: Um Fehler wie Division durch Null zu beheben

B: Stellt sicher, dass das Programm nicht abstürzt

C: Prüfen Code bereits zu Kompilierzeit

D: Um undefiniertes Verhalten zu vermeiden

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

50:50

Pub

Tel

Frage 4 ■ 300 €

Wieso sind Asserts nützlich?

A: Um Fehler wie Division durch Null zu beheben

B: Stellt sicher, dass das Programm nicht abstürzt

C: Prüfen Code bereits zu Kompilierzeit

D: Um undefiniertes Verhalten zu vermeiden

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

Pre- und Post-Conditions

- Werden verwendet um Funktionen zu dokumentieren
- Pre-Condition: Was muss erfüllt sein, damit die Funktion korrekt arbeitet?
- Post-Condition: Was garantiert die Funktion, wenn sie korrekt arbeitet?

Pre- und Post-Conditions: Übungen

Was sind geeignete Pre- und Post-Conditions für die folgende Funktion?

```
1 double f (double i, double j, double k){  
2     if (i > j){  
3         if (i > k) return i;  
4         else return k;  
5     }  
6     else{  
7         if (j > k) return j;  
8         else return k;  
9     }  
10 }
```

Pre- und Post-Conditions: Übungen

Was sind geeignete Pre- und Post-Conditions für die folgende Funktion?

```
1 double f (double i, double j, double k){  
2     if (i > j){  
3         if (i > k) return i;  
4         else return k;  
5     }  
6     else{  
7         if (j > k) return j;  
8         else return k;  
9     }  
10 }
```

- Pre-Condition: Keine
- Post-Condition: Gibt das Maximum der drei Zahlen zurück

Pre- und Post-Conditions: Übungen

Was sind geeignete Pre- und Post-Conditions für die folgende Funktion?

```
1 double g(int i, int j){  
2     double r = 0.0;  
3     for (int k = i; k <= j; ++k){  
4         r += 1.0 / k;  
5     }  
6     return r;  
7 }
```

Pre- und Post-Conditions: Übungen

Was sind geeignete Pre- und Post-Conditions für die folgende Funktion?

```
1 double g(int i, int j){  
2     double r = 0.0;  
3     for (int k = i; k <= j; ++k){  
4         r += 1.0 / k;  
5     }  
6     return r;  
7 }
```

- Pre-Condition: $i \leq j$, $0 \notin [i, j]$ und $j < \text{INT_MAX}$
- Post-Condition: Gibt die Summe der Kehrwerte aller ganzen Zahlen von i bis j zurück

50:50

Pub

Tel

Frage 5 ■ 500 €

Wer ist primär dafür verantwortlich, dass die Pre-Condition einer Funktion erfüllt ist?

A: Der Aufrufer der Funktion**B:** Die Funktion selbst**C:** Der C++ Compiler**D:** Das Betriebssystem

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

50:50

Pub

Tel

Frage 5 ■ 500 €

Wer ist primär dafür verantwortlich, dass die Pre-Condition einer Funktion erfüllt ist?

A: Der Aufrufer der Funktion**B:** Die Funktion selbst**C:** Der C++ Compiler**D:** Das Betriebssystem

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

Funktionen: Übung

Was ist die Ausgabe des Programms? Überläufe könnt ihr ignorieren.

```
1  #include <iostream>
2
3  int f (int i) return i * i;
4
5  int g (int i) return i * f(i) * f(f(i));
6
7  void h (int i) std::cout << g(i) << "\n"
8
9  int main () {
10     int i;
11     std::cin >> i;
12     h(i);
13     return 0;
14 }
```

Funktionen: Übung (Lösung)

```
1  int f (int i) {  
2      return i*i;  
3  }  
4  
5  int g (int i) {  
6      return i*f(i)*f(f(i));  
7  }
```

Auswertung von g(i):

$$i * f(i) * f(f(i))$$

Funktionen: Übung (Lösung)

```
1  int f (int i) {  
2      return i*i;  
3  }  
4  
5  int g (int i) {  
6      return i*f(i)*f(f(i));  
7  }
```

Auswertung von $g(i)$:

$$\begin{aligned} & i * f(i) * f(f(i)) \\ f(i) &= i \cdot i \\ \Rightarrow & i * (i*i) * f(f(i)) \end{aligned}$$

Funktionen: Übung (Lösung)

```
1 int f (int i) {  
2     return i*i;  
3 }  
4  
5 int g (int i) {  
6     return i*f(i)*f(f(i));  
7 }
```

Auswertung von $g(i)$:

$$i * f(i) * f(f(i))$$

$$f(i) = i \cdot i$$

$$\Rightarrow i * (i*i) * f(f(i))$$

$$f(f(i)) = f(i*i) = (i \cdot i) \cdot (i \cdot i)$$

$$\Rightarrow i * (i*i) * ((i*i)*(i*i))$$

Funktionen: Übung (Lösung)

```
1 int f (int i) {  
2     return i*i;  
3 }  
4  
5 int g (int i) {  
6     return i*f(i)*f(f(i));  
7 }
```

Auswertung von $g(i)$:

$$\begin{aligned} & i * f(i) * f(f(i)) \\ f(i) &= i \cdot i \\ \Rightarrow & i * (i*i) * f(f(i)) \\ f(f(i)) &= f(i*i) = (i \cdot i) \cdot (i \cdot i) \\ \Rightarrow & i * (i*i) * ((i*i)*(i*i)) \end{aligned}$$

Ausgabe

$$i \cdot i^2 \cdot (i^2)^4 = i^7.$$

Funktionen: Fehler finden

Findet **3 Fehler** in diesem Programm.

```
1  # include <iostream>
2
3  double f (double x) {
4      return g(2.0 * x);
5  }
6
7  bool g (double x) {
8      return x % 2.0 == 0;
9  }
10
11 void h () {
12     std::cout << result;
13 }
14
15 int main () {
16     double result = f(3.0);
17     h();
18     return 0;
19 }
```

Funktionen: Fehler finden (Lösung)

```
1  # include <iostream>
2
3  double f (double x) {
4      return g(2.0 * x);
5  }
6
7  bool g (double x) {
8      return x % 2.0 == 0;
9  }
10
11 void h () {
12     std::cout << result;
13 }
14
15 int main () {
16     double result = f(3.0);
17     h();
18     return 0;
19 }
```

Funktionen: Fehler finden (Lösung)

```
1  # include <iostream>
2
3  double f (double x) {
4      return g(2.0 * x);
5  }
6
7  bool g (double x) {
8      return x % 2.0 == 0;
9  }
10
11 void h () {
12     std::cout << result;
13 }
14
15 int main () {
16     double result = f(3.0);
17     h();
18     return 0;
19 }
```

Problem 1: g noch nicht bekannt

Der Scope von g beginnt erst nach der Definition. Lösung: g vor f definieren (oder deklarieren).

Funktionen: Fehler finden (Lösung)

```
1 # include <iostream>
2
3 double f (double x) {
4     return g(2.0 * x);
5 }
6
7 bool g (double x) {
8     return x % 2.0 == 0;
9 }
10
11 void h () {
12     std::cout << result;
13 }
14
15 int main () {
16     double result = f(3.0);
17     h();
18     return 0;
19 }
```

Problem 1: g noch nicht bekannt

Der Scope von g beginnt erst nach der Definition. Lösung: g vor f definieren (oder deklarieren).

Problem 2: Modulo

Es gibt kein % für double. Lösung: int verwenden.

Funktionen: Fehler finden (Lösung)

```
1 # include <iostream>
2
3 double f (double x) {
4     return g(2.0 * x);
5 }
6
7 bool g (double x) {
8     return x % 2.0 == 0;
9 }
10
11 void h () {
12     std::cout << result;
13 }
14
15 int main () {
16     double result = f(3.0);
17     h();
18     return 0;
19 }
```

Problem 1: g noch nicht bekannt

Der Scope von g beginnt erst nach der Definition. Lösung: g vor f definieren (oder deklarieren).

Problem 2: Modulo

Es gibt kein % für double. Lösung: int verwenden.

Problem 3: h sieht result nicht

result ist nur in main gültig. Lösung: result als Parameter an h übergeben.

Funktionen: Teileranzahl (Basierend auf Sommer 2012 Prüfung)

Schreibt eine Funktion `number_of_divisors`, die ein `int` `n` entgegennimmt und die Anzahl der Teiler von `n` zurückgibt (inklusive 1 und `n`).

Beispiel: 6 hat 4 Teiler (1, 2, 3, 6), also `std::cout << number_of_divisors(6);` → 4

```
1 // PRE: 1 <= n < INT_MAX
2 // POST: returns number of divisors of n (incl. 1 and n)
3 int number_of_divisors (int n) {
4     // your code
5 }
```

Funktionen: Teileranzahl (Lösung)

```
1  // PRE: 1 <= n < INT_MAX
2  // POST: returns number of divisors of n (incl. 1 and n)
3  int number_of_divisors (int n) {
4      assert(n > 0);
5      int counter = 0;
6      for (int i = 1; i <= n; ++i)
7          if (n % i == 0)
8              ++counter;
9      return counter;
10 }
```

[code]expert: Math Functions

[code]expert  Schreibt je eine Funktion, die folgendes berechnet:

- Die Summe dreier Ganzzahlen (`ints`)
- Der Durchschnitt dreier Ganzzahlen als `double`
- Das Maximum dreier Ganzzahlen
- Das Minimum dreier Ganzzahlen

Hinweis: Ihr könnt Funktionen wiederverwenden

Headers und Namespaces

Um Programme lesbarer zu machen, könnt ihr Funktionen in Header-Dateien auslagern.

- Beispiel: `#include <iostream>`.¹
- Für eigene Header-Dateien: `#include "my_header.h"` (mit Anführungszeichen)
- Bei Kompilierung wird `#include` durch den Inhalt der Datei ersetzt.

Namespaces können verwendet werden, um Namenskonflikte zu vermeiden.

¹Fun Fact: `iostream` besteht aus nur 74 Zeilen, die meisten davon sind Kommentare.²

²`iostream` importiert aber `istream` und `ostream`, die jeweils 1000+ Zeilen haben.

Headers und Namespaces

Um Programme lesbarer zu machen, könnt ihr Funktionen in Header-Dateien auslagern.

- Beispiel: `#include <iostream>`.¹
- Für eigene Header-Dateien: `#include "my_header.h"` (mit Anführungszeichen)
- Bei Kompilierung wird `#include` durch den Inhalt der Datei ersetzt.

Namespaces können verwendet werden, um Namenskonflikte zu vermeiden.

- Beispiel: `std::cout` ist in Namespace `std`.
- Werdet ihr selten brauchen, aber es ist gut zu wissen, dass es sie gibt.
- `using namespace std`; importiert alle Namen aus der Standardbibliothek, ist aber nicht empfohlen, da es Namenskonflikte verursachen kann.

¹Fun Fact: `iostream` besteht aus nur 74 Zeilen, die meisten davon sind Kommentare.²

²`iostream` importiert aber `istream` und `ostream`, die jeweils 1000+ Zeilen haben.

50:50

Pub

Tel

Frage 6 ■ 1 000 €

Warum sollte `using namespace std;` in Header-Dateien (.h) vermieden werden?

A: Es verlangsamt die Ausführung

B: Verschmutzt Namespace aller Dateien, die .h einbinden

C: Header-Dateien unterstützen keine Namespaces

D: Es führt immer zu einem Kompilierfehler

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

50:50

Pub

Tel

Frage 6 ■ 1 000 €

Warum sollte `using namespace std;` in Header-Dateien (.h) vermieden werden?

A: Es verlangsamt die Ausführung

B: Verschmutzt Namespace aller Dateien, die .h einbinden

C: Header-Dateien unterstützen keine Namespaces

D: Es führt immer zu einem Kompilierfehler

15 1 000 000 €

14 500 000 €

13 125 000 €

12 64 000 €

11 32 000 €

10 16 000 €

9 8 000 €

8 4 000 €

7 2 000 €

6 1 000 €

5 500 €

4 300 €

3 200 €

2 100 €

1 50 €

[code]expert: Stepwise Refinement

[code]expert 

- Bestimme, wie viele PERFEKTE³ Zahlen in $[a, b]$ liegen.
- $28 = 1 + 2 + 4 + 7 + 14$ und $6 = 1 + 2 + 3$ sind perfekte Zahlen.
- $12 \neq 1 + 2 + 3 + 4 + 6$ ist keine perfekte Zahl.

³Eine Zahl ist perfekt, wenn sie die Summe ihrer echten Teiler ist.

Tipps zu den Übungen

Perpetual Calendar

Tipps zu den Übungen

Perpetual Calendar

- Für die einzelnen Funktionen gibt es eigene Testcases. Prüft, ob die Funktionen korrekt arbeiten, bevor ihr die nächste Funktion implementiert.