



# Übungsstunde W04

Informatik (RW) – HS 23

# Übersicht

## Heutiges Programm

Follow-up

Feedback zu **code** expert

Expressions

Loops

Reihen Berechnen

Tipps zu **code** expert

Outro



[rwko.ch/lily](https://rwko.ch/lily)

# 1. Follow-up

---

# Follow-up aus vorherigen Übungsstunden

- Die Übungsstunde bleibt auf Deutsch. Die Übung am Dienstag (HIT K 51, 13:45-15:30) wird neu auf Englisch gehalten.

## 2. Feedback zu **code** expert

---

# Allgemeines zu **code expert**

- Die Aufgaben der ersten 2 Wochen sind korrigiert<sup>1</sup>
  - Fragen zu Stoff/Aufgabe → Mail an TA
  - Fragen zu Korrekturen → Mail an TA
  - Bugs in **code expert** → Mail an Head TA

---

<sup>1</sup>Falls ihr bei mir in eingeschrieben seid auf **code expert**

# Ziele

- Komplexe Expressions (die Booleans und Arithmetik enthalten) evaluieren können
- Summen in C++ darstellen und verwenden können
- Alle Arten von Loops (`for`, `while`, `do-while`) tracen können
- Alle Arten von Loops durch alle anderen Arten von Loops ersetzen können

# Kommentare zu **code expert**

Bitte bedenkt, dass euer Code auch noch von anderen gelesen wird (insb. von mir) und ihr anderen das Lesen und Verstehen eures Codes möglichst einfach machen solltet.

```
// even small comments  
// can make a big difference
```



# Kommentare zu **code expert**

## **Formatting und Struktur**

- Leere Zeilen, um Codeblöcke zu trennen
- Tabs/Spaces, um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)
- Nicht über kleinen grauen Strich am rechten Rand hinausschreiben

## **Kommentare**

- Dokumentiert euren Code (Insb. bei mathematischen Ausdrücken und Tricks wichtig)
- Fragen/Gedanken/Ansatz ganz oben als Kommentar vermerken
- Englisch und Deutsch sind beide okay, Feedback schreibe ich in der jeweiligen Sprache

## **Task Description/Autograder**

- Inbs. anfangs sehr streng bewertet, was das Missachten von der Task Description angeht

# Kommentare zu **code expert**

## **E2:T1 Expressions**

- Valide Expressions müssen nicht zwingend irgendwo abgespeichert werden

Fragen zu **code** expert ?

# 3. Expressions

---

# Types

## Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}
- integers: `unsigned int`, `int` {-7, 2, 0}
- floating point numbers: `float`, `double` {1.4, -4.3, 7.0}

Manchmal sind mehrere Types in einer Expression.  
Wie interagieren verschiedene Typen miteinander?

## Generalitätsreihenfolge der Typen

`bool < int < unsigned int < float < double`

Types konvertieren immer zum generellsten Type der Expression

# Wie man sich Types vorstellen kann

Type (literal)	Approximiert
bool	$\mathbb{B} = \{\text{false}, \text{true}\}$
unsigned int (u)	$\mathbb{N}$
int	$\mathbb{Z}$
float (f)	$\mathbb{R}$
double	$\mathbb{R}$ , aber <i>double</i> Präzision

# Types evaluieren I

```
std::cout << 5.0/2 << std::endl;  
// what type and value will this return and why?
```

## Lösung

double, 2.5, weil die `int` 2 zuerst in eine `double` 2.0 konvertiert wird, um diese Expression zu berechnen.

# Types evaluieren II

```
std::cout << (1/2)*5.0/2 << std::endl;  
// what type and value will this return and why?
```

## Lösung

`double`, 0, weil zuerst die linke Expression `1/2` evaluiert wird, welche zu 0 evaluiert (Integer-Division). Der Rest ist trivial, weil `0*anything` evaluiert zu 0. Aber diese 0 wird vom Type `double` sein.



# Literale

Es gibt bestimmte Buchstaben, die der Compiler mit bestimmten Types verbindet. Wenn ihr dem Compiler sagen möchtet “*Hey, don’t treat this 2.0 as a double, but instead as a float*” müsst ihr ein `f` am Ende des Werts hinzufügen. Etwa so:

```
std::cout << (5/2)*5.0f/2 << std::endl;
```

# Types evaluieren III

```
std::cout << (5/2)*5.0f/2 << std::endl;  
// what type and value will this return and why?
```

## Lösung

float, 5.0, (kann als 5.0f geschrieben werden).

Zuerst, wird  $5/2$  evaluiert, was zu 2 wird (integer division). Dann wird  $2.0f*5.0f$  berechnet: Die `int` 2 wurde zu einer `float` 2, weil `float` der generellere Type (in dieser Expression) ist. Dito für  $/2$  später.

# Exercise I

1. Which of the following character sequences are not C++ expressions, and why not? Here, `x` and `y` are variables of type `int`.
  - a) `(y++ < 0 && y < 0) + 2.0`
  - b) `y = (x++ = 3)`
  - c) `3.0 + 3 - 4 + 5`
  - d) `5 % 4 * 3.0 + true * x++`
2. For all of the valid expressions that you have identified in 1, decide whether these are lvalues or rvalues and explain your decision.
3. Determine the values of the expressions and explain how these values are obtained. Assume that initially `x == 1` and `y == -1`.

# Expression Evaluation - Lösungen a)

```
(y++ < 0 && y < 0) + 2.0
```

```
(-1 < 0 && y < 0) + 2.0 // after this step: y==0
```

```
(true && y < 0) + 2.0
```

```
(true && false) + 2.0
```

```
(false) + 2.0
```

```
0.0 + 2.0
```

```
2.0
```

R-VALUE

# Expression Evaluation - Lösungen b)

`y = (x++ = 3)`

INVALID

# Expression Evaluation - Lösungen c)

$$3.0 + 3 - 4 + 5$$

$$((3.0 + 3) - 4) + 5$$

$$((3.0 + 3.0) - 4) + 5$$

$$(6.0 - 4) + 5$$

$$(6.0 - 4.0) + 5$$

$$2.0 + 5$$

$$2.0 + 5.0$$

$$7.0$$

R-VALUE

# Expression Evaluation - Lösungen d)

5 % 4 \* 3.0 + true \* x++

((5 % 4) \* 3.0) + (true \* (x++))

(1 \* 3.0) + (true \* (x++))

(1.0 \* 3.0) + (true \* (x++))

3.0 + (true \* (x++))

3.0 + (true \* 1)

3.0 + (1 \* 1)

3.0 + 1

3.0 + 1.0

4.0

R-VALUE

# Loop Correctness

Can a user of the program observe the difference between the output produced by these three loops? If yes, how? Assume that `n` is a variable of type `unsigned int` whose value is given by the user.

```
unsigned int n; std::cin >> n;  
unsigned int i;
```

```
// loop 1 //////////////////////////////////////  
for (i = 1; i <= n; ++i) {  
    std::cout << i << "\n";  
}
```

```
// loop 2 //////////////////////////////////////  
i = 0;  
while (i < n) {  
    std::cout << ++i << "\n";  
}
```

```
// loop 3 //////////////////////////////////////  
i = 1;  
do {  
    std::cout << i++ << "\n";  
} while (i <= n);
```



# Loop Correctness - Solution

## Solution

There are the following differences:

- Unlike loops 1 and 2, loop 3 does output `|1|` for input `|n == 0|` because the statement in a `|do|-loop` is always executed once before the condition is checked.
- If  $n$  is the largest possible integer, then the loops 1 and 3 may be infinite because the condition `|i <= n|` is going to be true for all possible `|i|`.

# Questions?

# 4. Loops

---

# for → while

```
// TASK: Convert the following for-loop  
// into an equivalent while-loop:
```

```
for (int i = 0; i < n; ++i) {  
    BODY  
}
```

```
// SOLUTION
```

```
int i = 0;  
  
while(i < n){  
    BODY  
    ++i;  
}
```

# while → for

```
// TASK: Convert the following while-loop  
// into an equivalent for-loop:
```

```
while(condition){  
    BODY  
}
```

```
// SOLUTION  
for(;condition;){  
    BODY  
}
```

# do-while → for

```
// TASK: Convert the following do-while-loop  
// into an equivalent for-loop:
```

```
do{  
    BODY  
}while(condition)
```

```
// SOLUTION
```

```
BODY
```

```
for(;condition;){  
    BODY  
}
```

# Questions?

# 5. Reihen Berechnen

---



# Von Summe zur Loop

Mathematische Summen können zu Loops umgewandelt werden

$$\sum_{i=0}^n f(i)$$

Wird zu

```
int n = 0;
int sum = 0;

for(int i = 0; i <= n; i++){
    sum += f(i);
}
```

# Von Reihe zu Loop

## Taylor Series auf **code expert**

Schreibe ein Programm, das  $\sin(x)$  bis auf sechs Stellen berechnet.

Tipp: Welchen Loop sollte man hierfür verwenden?

Tipp: MacLaurin-Reihe verwenden.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

## Aufgabe

- Mit Stift und Papier versuchen (10min)
- Mit Person neben euch versuchen, in **code expert** zu implementieren (10min)

# Questions?

We first split up the sum:

$$\sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} \left. \begin{array}{l} \text{introduce 2 variables} \\ \text{double } n \text{ (numerator)} \\ \text{double } d \text{ (denominator)} \end{array} \right\}$$

Assume we are at iteration  $(n-1)$ . Our variables will be:

$$n = (-1)^{n-1} x^{2(n-1)+1} = (-1)^{n-1} \cdot x^{2n-1}$$

$$d = (2(n-1)+1)! = (2n-1)!$$

At iteration  $n$ , our variables should be:

$$n = (-1)^n \cdot x^{2n+1}$$

$$d = (2n+1)!$$

How do we get these?

$$(-1)^{n-1} \cdot x^{2n-1} \cdot (-1)(x^2) = (-1)^n \cdot x^{2n+1}$$

$$(2n-1)! \cdot (2n)(2n+1) = (2n+1)!$$

- These will be the updates in the body of the loop!

## 6. Tipps zu **code** expert

---

# Tipps für **code expert**

## **Tasks 1 and 2: “Loop mix-up”**

- Falls ihr einen Loop nicht direkt erkennt, versucht zuerst, ein paar Zahlenwerte einzufügen

## **Task 3: “Loop Analysis”**

- Q2: Welche Werte können Variablen vom Typ `unsigned int` annehmen?

# 7. Outro

---

# Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche!