

Exercise Week 02

GianAndrea Müller
`mailto:muellegi@student.ethz`

March 12, 2018

Time Schedule

- 5' Nachbesprechung
- 15' while, do while mit Übung
- 3' break und continue
- 7' Gültigkeitsbereich
- 5' float und double mit Übung
- 10' Typenumwandlungen mit Übung
- 20' Fehlersuche mit assert

Learning Objectives

- Kenntnis aller Schleifen und erweiterte Flusskontrolle
- Verständnis aller grundlegenden Variablentypen und deren Umwandlung
- Nutzung erweiterter Debugging-Methoden

Nachbesprechung

```
1 8>4>2>1
2 true>2>1
3 1>2>1
4 false>1
5 0>1
6 false
7
8 2<a<4      2<a<4
9 true<4     false<4
10 1<4       0<4
11 true      true
12
13 2<a && a<4
14
15 //Comment your code
```

Schleifen

```
1 while(condition)
2     statement
3
4
5 do{
6     statement1
7     statement2
8 }
9 while(condition);
```

```
1 //loop 1
2 for(int i = 1; i<=n; ++i)
3     cout << i << "\n";
4
5 //loop 2
6 int i = 0;
7 while(i < n)
8     cout<< ++i << "\n";
9
10 //loop 3
11 i = 1;
12 do
13     cout<<i++<< "\n";
14 while (i <= n)
```

Solution 03_1

```
1 //if n == INT_MAX -> infinite loop
2 for(int i = 1; i<=n; ++i)
3     cout << i << "\n";
4
5 //loop 2
6 int i = 0;
7 while(i < n)
8     cout<< ++i << "\n";
9
10 //n<= 0 -> still outputs 1, if n=INT_MAX
11 //    -> infinite loop
12 i = 1;
13 do
14     cout<<i++<< "\n";
15 while (i <= n)
```

Welche Schleife in welchen Fall?

Motivation

- So wenig code wie möglich.
- Einfach lesbarer code.

Möglichkeiten

- **for**: Es wird ein Zähler benötigt, Zähler wird nach der Schleife nicht mehr benötigt.
Wiederhole ein statement n mal.
- **while**: Die Bedingung hängt von einer Variable ab, die bereits vor der Schleife existiert.
Dekrementiere x bis es ein Vielfaches von 5 ist.
- **do**: Die Bedingung hängt von einer Variable ab, die erst in der Schleife erhalten wird.
Führe cin >> x aus bis $x > 3$

break und continue

```
1 while(true){
2     statement1;
3     if(condition){
4         break;
5     }
6 }
7
8 for (int i = 0; i < 10; i++){
9     statement1;
10    if(condition){
11        continue;
12    }
13    statement2;
14 }
```

Gültigkeitsbereich - Scope

```
1  if (x < 7) {  
2  int a=8;  
3  }  
4  std::cout << a; // Compiler error, a does  
   not exist.
```

Gültigkeitsbereich - Scope

```
1  int a = 2;
2  if (x < 7) {
3  a=8;
4  }
5  std::cout << a; // Outputs 2 or 8,
    depending on the if-statement.
```

Gültigkeitsbereich - Scope

```
1 for (int i = 0; i < 5; ++i) {  
2     std::cout << i << "\n"; // Outputs i  
3 }  
4 std::cout << i << "\n"; // Compiler error,  
    i does not exist
```

Gültigkeitsbereich - Scope

```
1 unsigned int x = 2;
2 int i = 5;
3 if (x > 1) {
4     int i = 3;
5     std::cout << i; // outputs 3
6 }
7 std::cout << i; // outputs 5
```

float und double

$$\pm \underbrace{3.4}_{\sim 7 \text{ Stellen}} \cdot 10^{\pm 38}$$

$$\pm \underbrace{1.7}_{\sim 15 \text{ Stellen}} \cdot 10^{\pm 308}$$

```
1 float a = 1.0/6.0;
2 double b = 2/5;
3
4 cout << a; //outputs 0.166666672
5
6 float c = (a - 0.1)*10;
7
8 cout << c; //outputs 0.666666746
```

Datentypen

Typenumwandlung

`char, bool < int < unsigned int < float < double`

Typenumwandlung

```
1 #include <iostream>
2
3 int main()
4 {
5     int a = -4;
6     unsigned int b = 2u;
7
8     std::cout << a + b; //output:
9     4294967294
10
11     std::cout << int(a+b); //output: -2
12
13     unsigned int c = a;
14
15     return 0;
16 }
```


Exercise 03_2 ~ 5'

Evaluate by hand. Assume $x = 1$ is of type *int*

① `3.0 + 3 - 4 + 5`

② `5 % 4 * 3.0 + true * x++`

③ `- 3 - 4u + 8.0`

Solution 03_2 ~

Solution (1)

- ❶ $3.0 + 3 - 4 + 5$
- ❷ $((3.0 + 3) - 4) + 5$
- ❸ $((3.0 + 3.0) - 4) + 5$
- ❹ $(6.0 - 4) + 5$
- ❺ $(6.0 - 4.0) + 5$
- ❻ $2.0 + 5$
- ❼ $2.0 + 5.0$
- ❽ 7.0

Solution 03_2 ~

Solution (2)

- ① $- 3 - 4u + 8.0$
- ② $(4294967283u - 4u) + 8.0$
- ③ $4294967289u + 8.0$
- ④ $4294967289.0 + 8.0$
- ⑤ 4294967297.0

Solution 03_2 ~

Solution (1)

- ❶ `5 % 4 * 3.0 + true * x++`
- ❷ `((5 % 4) * 3.0) + (true * (x++))`
- ❸ `(1 * 3.0) + (true * (x++))`
- ❹ `(1.0 * 3.0) + (true * (x++))`
- ❺ `3.0 + (true * (x++))`
- ❻ `3.0 + (true * 1)`
- ❼ `3.0 + (1 * 1)`
- ❽ `3.0 + 1`
- ❾ `3.0 + 1.0`
- ❿ `4.0`