Exercise Week 03

GianAndrea Müller
mailto:muellegi@student.ethz

March 14, 2018

Time Schedule

- 5’ Nachbesprechung
- 15’while, do while mit Übung
- 5’ break und continue
- 10’ Gültigkeitsbereich
- 5’ float und double
- 15’ Pause
- 15’ Typenumwandlungen mit Übung
- 30’ Übungsbearbeitung

Learning Objectives

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

Nachbesprechung

```
8>4>2>1
true>2>1
1>2>1
false>1
0>1
false

2<a<4     2<a<4
true<4    false<4
1<4       0<4
true      true

2<a && a<4

//Comment your code
```

Schleifen

```
while(condition)
  statement


do{
  statement1
  statement2
}
while(condition);
```

Exercise 03_1 $\sim$ 5' All positive numbers up to *n*

```cpp
//loop 1
for(int i = 1; i<=n; ++i)
  cout << i << "\n";

//loop 2
int i = 0;
while(i < n)
  cout<< ++i << "\n";

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

Solution 03_1

```
//if n == INT_MAX -> infinite loop
for(int i = 1; i<=n; ++i)
  cout << i << "\n";

//loop 2
int i = 0;
while(i < n)
  cout<< ++i << "\n";

//n<= 0 -> still outputs 1, if n=INT_MAX -> infinite loop
i = 1;
do
  cout<<i++<<"\n";
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*

```
while(true){
  statement1;
  if(condition){
    break;
  }
}

for (int i = 0; i < 10; i++){
  statement1;
  if(condition){
    continue;
  }
  statement2;
}
```

Gültigkeitsbereich - Scope

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

Gültigkeitsbereich - Scope

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

Gültigkeitsbereich - Scope

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

Gültigkeitsbereich - Scope

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

float und *double*

$$\pm \underbrace{3.4}_{\sim 7 \text{ Stellen}} \cdot 10^{\pm 38} \qquad\qquad \pm \underbrace{1.7}_{\sim 15 \text{ Stellen}} \cdot 10^{\pm 308}$$

```
float a = 1.0/6.0;
double b = 2/5;

cout << a; //outputs 0.166666672

float c = (a - 0.1)*10;

cout << c; //outputs 0.666666746
```

Datentypen

Typenumwandlung

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

Typenumwandlung

```cpp
#include <iostream>

int main()
{
    int a = -4;
    unsigned int b = 2u;

    std::cout << a + b; //output: 4294967294
    std::cout << int(a+b); //output: -2

    unsigned int c = a;

    return 0;
}
```

Typenumwandlung

```cpp
#include <iostream>

int main()
{
    int a = -4;
    unsigned int b = 2u;

    std::cout << a + b; //output: 4294967294
    std::cout << int(a+b); //output: -2

    unsigned int c = a; //output: 4294967292

    return 0;
}
```

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)

1. `3.0 + 3 - 4 + 5`

Solution 03_2

Solution (1)

1. `3.0 + 3 - 4 + 5`
2. `((3.0 + 3) - 4) + 5`

Solution 03_2

Solution (1)

1. `3.0 + 3 - 4 + 5`
2. `((3.0 + 3) - 4) + 5`
3. `((3.0 + 3.0) - 4) + 5`

Solution 03_2

Solution (1)

1. `3.0 + 3 - 4 + 5`
2. `((3.0 + 3) - 4) + 5`
3. `((3.0 + 3.0) - 4) + 5`
4. `(6.0 - 4) + 5`

Solution 03_2

Solution (1)

1. `3.0 + 3 - 4 + 5`
2. `((3.0 + 3) - 4) + 5`
3. `((3.0 + 3.0) - 4) + 5`
4. `(6.0 - 4) + 5`
5. `(6.0 - 4.0) + 5`

Solution 03_2

Solution (1)

1. `3.0 + 3 - 4 + 5`
2. `((3.0 + 3) - 4) + 5`
3. `((3.0 + 3.0) - 4) + 5`
4. `(6.0 - 4) + 5`
5. `(6.0 - 4.0) + 5`
6. `2.0 + 5`

Solution 03_2

Solution (1)

1. `3.0 + 3 - 4 + 5`
2. `((3.0 + 3) - 4) + 5`
3. `((3.0 + 3.0) - 4) + 5`
4. `(6.0 - 4) + 5`
5. `(6.0 - 4.0) + 5`
6. `2.0 + 5`
7. `2.0 + 5.0`

Solution 03_2

Solution (1)

1. `3.0 + 3 - 4 + 5`
2. `((3.0 + 3) - 4) + 5`
3. `((3.0 + 3.0) - 4) + 5`
4. `(6.0 - 4) + 5`
5. `(6.0 - 4.0) + 5`
6. `2.0 + 5`
7. `2.0 + 5.0`
8. `7.0`

Solution 03_2

Solution (2)

1. `- 3 - 4u + 8.0`

Solution 03_2

Solution (2)

1. `- 3 - 4u + 8.0`
2. `(4294967283u - 4u) + 8.0`

Solution 03_2

Solution (2)

1. `- 3 - 4u + 8.0`
2. `(4294967283u - 4u) + 8.0`
3. `4294967289u + 8.0`

Solution 03_2

Solution (2)

1. `- 3 - 4u + 8.0`
2. `(4294967283u - 4u) + 8.0`
3. `4294967289u + 8.0`
4. `4294967289.0 + 8.0`

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 (3)

1. `5 % 4 * 3.0 + true * x++`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`
3. `(1 * 3.0) + (true * (x++))`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`
3. `(1 * 3.0) + (true * (x++))`
4. `(1.0 * 3.0) + (true * (x++))`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`
3. `(1 * 3.0) + (true * (x++))`
4. `(1.0 * 3.0) + (true * (x++))`
5. `3.0 + (true * (x++))`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`
3. `(1 * 3.0) + (true * (x++))`
4. `(1.0 * 3.0) + (true * (x++))`
5. `3.0 + (true * (x++))`
6. `3.0 + (true * 1)`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`
3. `(1 * 3.0) + (true * (x++))`
4. `(1.0 * 3.0) + (true * (x++))`
5. `3.0 + (true * (x++))`
6. `3.0 + (true * 1)`
7. `3.0 + (1 * 1)`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`
3. `(1 * 3.0) + (true * (x++))`
4. `(1.0 * 3.0) + (true * (x++))`
5. `3.0 + (true * (x++))`
6. `3.0 + (true * 1)`
7. `3.0 + (1 * 1)`
8. `3.0 + 1`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`
3. `(1 * 3.0) + (true * (x++))`
4. `(1.0 * 3.0) + (true * (x++))`
5. `3.0 + (true * (x++))`
6. `3.0 + (true * 1)`
7. `3.0 + (1 * 1)`
8. `3.0 + 1`
9. `3.0 + 1.0`

Solution 03_2

Solution (3)

1. `5 % 4 * 3.0 + true * x++`
2. `((5 % 4) * 3.0) + (true * (x++))`
3. `(1 * 3.0) + (true * (x++))`
4. `(1.0 * 3.0) + (true * (x++))`
5. `3.0 + (true * (x++))`
6. `3.0 + (true * 1)`
7. `3.0 + (1 * 1)`
8. `3.0 + 1`
9. `3.0 + 1.0`
10. `4.0`