Informatik I - Exercise Session
Expressions and Loops

Recap: Expressions

→ verschiedene Operatoren ⇒ Präzedenz, welche Operatoren müssen zuerst evaluiert werden?

→ gleiche Operatoren ⇒ Assoziativität

→ Operatoren versch. Datentypen ⇒ type conversion (implizite Konversion)

```
bool -> char -> short int -> int ->

unsigned int -> long -> unsigned ->

long long -> float -> double -> long double
```

die wichtigsten für euch:

bool → int → unsigned int → float → double

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 decisions.
3. Determine the values of the expressions and explain how these values are obtained. Assume that initially `x == 1` and `y == -1`.

use then change

Exercise I: Lösungen 1)

```
(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

Exercise I: Lösungen 2)

```
y = (x++ = 3)
```

INVALID

Exercise I: Lösungen 3)

```
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

Exercise I: Lösungen 4)

```
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

Scopes

```
int a = 2;
if (x < 7) {
    int a = 8;
    std::cout << a;
}
std::cout << a;
```

→ hier ist das lokale a "wichtiger" d.h. 8 wird geprinted

// hier zählt wieder das äussere a, das innere existiert nicht mehr.

⇒ Ausgabe: 82

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

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`.

Loop Conversion

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

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

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

```
while (condition)
    BODY
```

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

```
do
    BODY
while (condition);
```

Loop Conversion - Solution

A possible way to convert a `for`-loop into an equivalent `while`-loop:

```
{   // This additional block restricts the scope of i.
  int i = 0;
  while (i < n) {
    BODY
    ++i;
  }
}
```

A possible way to convert a `while`-loop into an equivalent `for`-loop:

```
for ( ;condition; )
  BODY
```

A possible way to convert a `do`-loop into an equivalent `for`-loop:

```
BODY
for ( ;condition; )
  BODY
```

Exercise: Taylor Series

$n=0 \quad \frac{(-1)^0 \cdot x^{0+1}}{(0+1)!}$

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

$\cdot \left(\frac{-x^2}{2 \cdot 3}\right)$

Compute

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

up to a precision of 0.000001 = 1e-6.

$i=n-1: \frac{(-1)^{n-1} x^{2(n-1)+1}}{(2(n-1)+1)!} = \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!}$

$i=n: \frac{(-1)^n x^{2n+1}}{(2n+1)!}$

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

Exercise: Taylor Series

1. How can we compute the n-th term from the (n-1)-th term?
2. Which type of loop can we use?
3. How can we deal with the precision?

↳ when $\sum$ < precision then we are not adding anything new

i.e. if the new term is smaller than tol we are adding precision smaller than what we require.