

# Übungsstunde

## Woche 03

Adel Gavranović  
adel.gavranovic@inf.ethz.ch

# Overview

## Heutige Themen

Intro

Boolean Expressions

for-Schleifen

assert()

Simple Manual Debugging

Outro

## Links

▶ polybox zum Material für die Übungsstunden

▶ Mail an Assistenten

# Follow-up aus vorheriger Übungsstunde

- Fehler im Handout W02 zum Thema Modulo-rechnen auf S.16

# Kommentare zu [code] expert

- Korrekturen stehen noch aus

# Fragen zu [code]expert eurerseits?

# Lernziele

- boolean expressions* richtig evaluieren
- wissen, was *short circuiting* ist und wo es als Fehlerquelle vorkommen kann
- Programme mit for-Schleifen *tracen*
- Einfache Programme, die for-Schleifen enthalten, schreiben können

# Expressions

- Repetition: was war nochmal ein `bool`?
- (Klammern) sind eure besten Freunde
- die Reihenfolge ist beim Auswerten von Bedeutung (Links nach rechts und Operator Precedences)
- Vorsicht vor *short circuits*

# Booleans

- kurz und einfach: `bools`
- immer entweder `true` oder `false`
- wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein
- wenn eine Zahl  $\neq 0$  in einen `bool` umgewandelt wird, wird es der Wert `true` sein
- wenn `false` in eine Zahl umgewandelt wird, wird es die Zahl 0 sein
- wenn 0 in einen `bool` umgewandelt wird, wird es der Wert `false` sein
- Kurze `[code]`expert Demonstration



# Precedences Ranking<sup>1</sup>

1. `a++`, `a--`
2. `++a`, `--a`, `-a`, `!a`, `*a`, `&a`
3. `*`, `/`, `%`
4. `+`, `-`
5. `<`, `<=`, `>`, `>=`
6. `==` `!=`
7. `&&`
8. `||`
9. `=`, `+=`, `-=`, `*=`, `/=`, `%=`

---

<sup>1</sup>Gezeigt sind hier nur die z. Z. wichtigsten. Für den Rest, siehe

# (Nutzt) (Klammern)

- Klammern funktionieren wie in der Mathematik
- sie werden oft gebraucht, um die korrekte Evaluation offensichtlich zu machen
- ... oder eben um die Evaluationsreihenfolge abzuändern

## Aufgabe

Mache die Evaluation mittels Klammern offensichtlich:

$3 < 4 + 1 \ \&\& \ 2 < 3$

*Tipp: nutze die vorherige Folie*

## Lösung

$(3 < (4 + 1)) \ \&\& \ (2 < 3)$

# Mehrere Operatoren mit gleicher Präzedenz

## Kurzaufgabe

Wie würdest du klammern, damit die Evaluation der unteren Expression offensichtlich wird?

```
false && false && true
```

## Lösung

Einfach von links nach rechts:

```
(false && false) && true
```

# Short Circuits

## Short Circuit

Die bool'schen Operatoren `&&` und `||` evaluieren zuerst die linke Expression und dann *nur falls nötig* die rechte Expression.

Insbesondere heisst das, dass die rechte Expression *nicht* evaluiert wird, wenn man das Ergebnis der gesamten Evaluation bereits herleiten kann

Was sind die Implikationen für uns?

# Short Circuits in Code

```
if (3 > 2 && 10 > 11){
    std::cout << "Of course not!\n";
} // not a short circuit evaluation

int a = 3;

if (false && ++a < 2){
    std::cout << "Of course not!\n";
} // a short circuit evaluation

std::cout << a << "\n"; // what will be the output?

if (++a < 2 && false){
    std::cout << "Of course not!\n";
} // another short circuit evaluation

std::cout << a << "\n"; // what will be the output?
```

# Verständniskontrolle

## Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
x == 1 || 1 / (x - 1) < 1
```

Vergesst nicht: Klammern sind eure Freunde

## Lösung

Zuerst: Klammern!

```
(x == 1) || ((1 / (x - 1)) < 1), beginne links
```

```
(1 == 1) || ((1 / (x - 1)) < 1)
```

```
true || ((1 / (x - 1)) < 1)2
```

```
true
```

---

<sup>2</sup>(true || whatever) evaluiert immer zu (true)

# Und noch eins

## Aufgabe

Evaluiere die folgende Expression von Hand und notiere jeden Zwischenschritt. Nehme an `int x = 1`:

```
!(1 < 2 && x == 1) + 1
```

## Lösung

```
!(1 < 2 && x == 1) + 1  
(!((1 < 2) && (x == 1))) + 1  
(!((true) && (true))) + 1  
!(true) + 1  
false + 1  
0 + 1  
1
```

# Fragen/Unklarheiten?



# for-Schleifen

- was ist eine Schleife
- Kurzeinführung zu *Scopes*
- Kurzeinführung ins *Program Tracing*

# Kurzeinführung zu Scopes

- praktisch immer, wenn Ihr {diese geschweiften Klammern} verwendet, erstellt ihr einen {Scope}
- stellt euch einen Scope als eine "Welt in einer Welt" vor
- Informationen/Variablen können nicht aus einem Scope rausfliessen, aber Informationen/Variablen von aussen sind im inneren Scope verfügbar
- wenn das Programm bei der rechten geschweiften Klammer des Scopes ankommt, stirbt jegliche Information/Variable innerhalb dieses Scopes

# Allgemeine Struktur einer for-Schleife

```
for(init; contition; expression){  
    statement 1;  
    statement 2;  
    ...  
}
```

## Wichtige Bemerkung

Der `expression`-Teil wird *nach* den `statemets` ausgeführt.

# Program Tracing

”*Program Tracing* ist der Prozess, des Ausführens eines Programms von Hand mit konkreten Eingaben.”

Ziemlich wichtiger Skill, insbesondere am Anfang. Irgendwann werdet ihr das quasi im Kopf können.

Einen ausführlichen Guide dazu findet ihr hier:

▶ Program Tracing

for Loop

## Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

## Example – for Loop

sum: 0

```
int sum = 0;  
  
for (int i = 1; i <= 3; ++i)  
    sum += i;  
  
std::cout << sum << "\n";
```

# Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

sum:	0
i:	1



## Example – for Loop

sum:	0
i:	1

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

## Example – for Loop

```
int sum = 0;
```

```
for (int i = 1; i <= 3; ++i)  
    sum += i;
```

```
std::cout << sum << "\n";
```

1 <= 3

true

sum:	0
i:	1

## Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

sum:	1
i:	1

## Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

sum:	1
i:	2

## Example – for Loop

sum:	1
i:	2

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

## Example – for Loop

```
int sum = 0;
```

```
for (int i = 1; i <= 3; ++i)  
    sum += i;
```

```
std::cout << sum << "\n";
```

2 <= 3

true

sum:	1
i:	2

## Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

sum:	3
i:	2

## Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

sum:	3
i:	3



## Example – for Loop

sum:	3
i:	3

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

## Example – for Loop

```
int sum = 0;
```

```
for (int i = 1; i <= 3; ++i)  
    sum += i;
```

```
std::cout << sum << "\n";
```

3 <= 3

true

sum: 3

i: 3

## Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

sum:	6
i:	3

## Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

sum:	6
i:	4

## Example – for Loop

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

sum:	6
i:	4

## Example – for Loop

```
int sum = 0;
```

```
for (int i = 1; i <= 3; ++i)  
    sum += i;
```

```
std::cout << sum << "\n";
```

4 <= 3

false

sum: 6

i: 4

## Example – for Loop

sum: 6

```
int sum = 0;

for (int i = 1; i <= 3; ++i)
    sum += i;

std::cout << sum << "\n";
```

# Fragen/Unklarheiten?



# Aufgabe *Strange Sum*

## Aufgabe

Öffnet *Strange Sum* auf [code]expert und versucht es zuerst mit Stift und Papier zu lösen. (10min)

### Description:

Write a program that reads a number  $n > 0$  from standard input and outputs the sum of all positive numbers up to  $n$  that are odd but not divisible by 5.

## Aufgabe

Und jetzt schreibt das dazugehörige Programm. (5min)

# Fragen/Unklarheiten?

# Mögliche Lösung zu *Strange Sum*

```
// input
unsigned int strangesum = 0;
unsigned int n;
std::cin >> n;

// computation
for(unsigned int i = 1; i <= n; i++){
    if((i % 2) == 1){
        if(i % 5){
            strangesum += i;
        }
    }
}

// output
std::cout << strangesum << "\n";
```

# Kompaktere Lösung zu *Strange Sum*

```
// input
unsigned int strangesum = 0;
unsigned int n;
std::cin >> n;

// computation
for(unsigned int i = 1; i <= n; i++){
    if( ((i % 2) == 1) && (i % 5) ){
        strangesum += i;
    }
}

// output
std::cout << strangesum << "\n";
```

# Noch kompaktere Lösung zu *Strange Sum*

```
// input
unsigned int strangesum = 0;
unsigned int n;
std::cin >> n;

// computation
for(unsigned int i = 1; i <= n; i+=2){
    if(i % 5){
        strangesum += i;
    }
}

// output
std::cout << strangesum << "\n";
```

# Aufgabe *Largest Power*

## Aufgabe

Öffnet *Largest Power* auf [code]expert und versucht es zuerst mit Stift und Papier zu lösen. (10min)

### Description:

Write a program that inputs a positive natural number  $n$  and outputs the largest number  $p$  that is a power of 2 and smaller or equal to  $n$ .

## Aufgabe

Und jetzt schreibt das dazugehörige Programm. (5min)

## Aufgabe

Besprecht eure Ansätze mit den Personen neben euch. Hattet ihr den gleichen Ansatz? Was könnt ihr voneinander lernen? (7min)

# Mögliche Lösung zu *Largest Power*

```
#include <iostream>
#include <cassert>

int main () {
    unsigned int n;
    std::cin >> n;
    assert(n >= 1);

    unsigned int power = 1;
    for (; power <= n / 2; power *= 2);

    std::cout << power << std::endl;

    return 0;
}
```

# Fragen/Unklarheiten?



# assert()

```
#include <cassert>

// ... mehr Code hier

assert(expression);
```

# Errors in Code...

---

- **Problem:**

We want to avoid certain values for a variable.

- **Question:**

How?

# General Hints

---

- `assert (expr) ;`
  - `expr` is `true`: nothing happens
  - `expr` is `false`: stop program

# Example

Problem: Some inputs are dangerous.

```
#include <iostream>

int main () {
    int a;
    std::cin >> a;
    int b;
    std::cin >> b;

    // Output: a/b
    std::cout << a/b << "\n";

    return 0;
}
```

Problem for:

$b == 0$

# Example

Problem: Some inputs are dangerous.

`assert ensures`

`b != 0`

```
return 0;
```

```
}
```

# Example

Problem: Some inputs are dangerous.

```
#include <iostream>
#include <cassert>

int main () {
    int a;
    std::cin >> a;
    int b;
    std::cin >> b;
    assert(b != 0);

    // Output: a/b
    std::cout << a/b << "\n";

    return 0;
}
```

# Example

Problem: Some inputs are dangerous.

```
#include <iostream>
```

```
3
```

```
0
```

```
a.out: ./Root/assert_expl.cpp:9: int main(): Assertion `b != 0' failed.
```

```
Aborted
```

```
0
```

Send

```
// Output: a/b
```

```
std::cout << a/b << "\n";
```

```
return 0;
```

```
}
```

## assert – Why?

---

- Still an easy example...
- So **why** and **where** is `assert` useful?



# assert – Why?

---

- Still an easy example...
- So **why** and **where** is `assert` useful?
  - **Long programs:** for overview
  - **User-Inputs required:** for safety
  - **Multiple programmers:** for safety

# Fragen/Unklarheiten?

# Debugging

”*Debugging* is the process of finding and resolving bugs (defects or problems that prevent correct operation) within programs, software, or systems.”

# Debugging

```
int main () {
    const int n = 6;

    // Compute n^12
    int prod = 1;
    for (int i = 1; 1 <= i < 13; ++i) {
        prod *= n;
    }

    // Output stars
    for (int i = 1; i < prod; ++i) {
        std::cout << "*";
    }
    std::cout << "\n";
    return 0;
}
```

# Debugging

## Frage

Wie könnten wir herausfinden, bei welcher Zeile das Programm feststeckt?

## Antwort

Sachen ausgeben lassen an interessanten Stellen im Code

## Frage

Wieso steckt das Programm in der ersten for-Schleife fest?

## Antwort

Die condition ist falsch geschrieben.  
Sollte sein: `1 <= i && i < 13`.

# Debugging

## Frage

Wie können wir ermitteln, wieso trotzdem nichts ausgegeben wird?

## Antwort

Einfach den Wert von `prod` nach der ersten Schleife ausgeben lassen

## Frage

Wie finden wir raus, wieso `prod` negativ wurde?

## Antwort

Einfach den Wert von `prod` in *jeder* Iteration in der Schleife ausgeben lassen

# Fragen/Unklarheiten?

# Tipps für [code]expert



# Bis zum nächsten Mal

- macht eure Hausaufgaben
- bleibt gesund

# Allgemeine Fragen?