Übungsstunde — Informatik — 02

Adel Gavranović

Boolean Expressions, `for`-Schleifen, Debugging, Selection Statements

Übersicht

Heutige Themen

Boolean Expressions
`for`-Schleifen
Debugging
Debugging in **code** expert
Selection Statements

`n.ethz.ch/~agavranovic`

Material
Webpage
E-Mail

1. Lernziele

Ziele für Heute

Lernziele

- ☐ Verstehen, was ein *Short Circuit* ist
- ☐ Verstehen, was Precedence und Assoziativität im Kontext von Bool'schen Operatoren sind
- ☐ (Bool'sche) Expressions von Hand Evaluieren können
- ☐ Expressions vereinfachen können
- ☐ *Program Tracing* verstehen und anwenden können
- ☐ `for`-Schleifen verstehen und anwenden können
- ☐ Simple, "manuelle" Debugging-Strategien anwenden können

2. Zusammenfassung

Fragen/Unklarheiten?

3. Boolean Expressions

Booleans

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann:

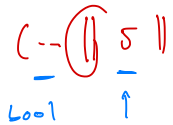
Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung "casting"



Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung

- Bool zu Zahl

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung

- Bool zu Zahl
  - wenn `true` in eine Zahl umgewandelt wird,

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung

- Bool zu Zahl
  - wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung

- Bool zu Zahl
  - wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein
  - wenn `false` in eine Zahl umgewandelt wird,

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung

- Bool zu Zahl
  - wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein
  - wenn `false` in eine Zahl umgewandelt wird, wird es die Zahl 0 sein

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung

- Bool zu Zahl
  - wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein
  - wenn `false` in eine Zahl umgewandelt wird, wird es die Zahl 0 sein
- Zahl zu Bool

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung

- Bool zu Zahl
  - wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein
  - wenn `false` in eine Zahl umgewandelt wird, wird es die Zahl 0 sein
- Zahl zu Bool
  - wenn 0 in einen `bool` umgewandelt wird,

Booleans

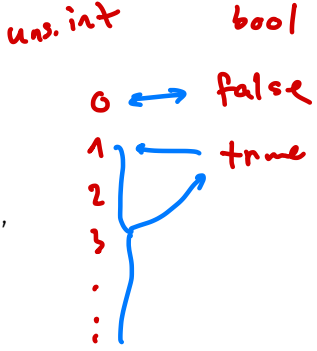
`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}

Konvertierung

- Bool zu Zahl
  - wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein
  - wenn `false` in eine Zahl umgewandelt wird, wird es die Zahl 0 sein
- Zahl zu Bool
  - wenn 0 in einen `bool` umgewandelt wird, wird es der Wert `false` sein
  - wenn eine Zahl $\neq 0$ in einen `bool` umgewandelt wird,

Booleans

`bool` ist ein Datentyp der nur zwei Werte annehmen kann: {`true`, `false`}



Konvertierung

- Bool zu Zahl
  - wenn `true` in eine Zahl umgewandelt wird, wird es die Zahl 1 sein
  - wenn `false` in eine Zahl umgewandelt wird, wird es die Zahl 0 sein
- Zahl zu Bool
  - wenn 0 in einen `bool` umgewandelt wird, wird es der Wert `false` sein
  - wenn eine Zahl $\neq 0$ in einen `bool` umgewandelt wird, wird es der Wert `true` sein

Precedence Ranking

Precedence Ranking[1]

[1] Gezeigt sind hier nur die z. Z. wichtigsten. Für den Rest, siehe cppreference

Precedence Ranking

Precedence Ranking[1]

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

(noch nicht relevant)

a = a + 5;

a += 5;

[1]Gezeigt sind hier nur die z. Z. wichtigsten. Für den Rest, siehe cppreference

(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

(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

(Nutzt) (Klammern)

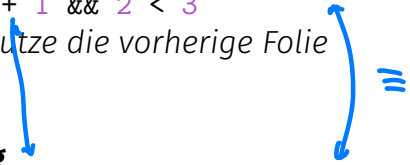
- Klammern funktionieren wie in der Mathematik
- sie werden oft gebraucht, um die korrekte Evaluation offensichtlich zu machen
- …oder eben um die Evaluationsreihenfolge abzuändern

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

Assoziativität

Mehrere Operatoren mit gleicher Präzedenz

Assoziativität

- Sagt aus, in welcher Reihenfolge Operationen evaluiert werden

Mehrere Operatoren mit gleicher Präzedenz

Assoziativität

- Sagt aus, in welcher Reihenfolge Operationen evaluiert werden
- Entweder Right-to-left ($\leftarrow$) oder Left-to-right ($\rightarrow$)

```
a = (b = (c = (d = 5)));
```

Mehrere Operatoren mit gleicher Präzedenz

Assoziativität

- Sagt aus, in welcher Reihenfolge Operationen evaluiert werden
- Entweder Right-to-left ($\leftarrow$) oder Left-to-right ($\rightarrow$)
- 💡 Am besten eine Tabelle auf die Zusammenfassung mit Präzedenzen und Assoziativitäten schreiben

Mehrere Operatoren mit gleicher Präzedenz

Kurzaufgabe

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

Tipp: Die Assoziativität von `&&` ist Left-to-right

```
(a && b)&& c
```

Mehrere Operatoren mit gleicher Präzedenz

Kurzaufgabe
Wie würdest du klammern, damit die Evaluation der unteren Expression offensichtlich wird?
Tipp: Die Assoziativität von `&&` *ist Left-to-right*

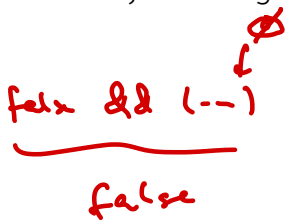
```
a && b && c
```

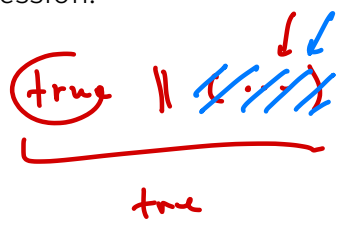
Lösung

```
(a && b) && c
```

Short Circuits (deutsch: Kurzschluss)

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





Short Circuits (deutsch: Kurzschluss)

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.

Short Circuits in Code

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

Short Circuits in Code

```
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 I

Aufgabe

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

```
2>3 && 17u - 55 <= ++x + y
```

17u

Verständniskontrolle I

Aufgabe

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

```
2>3 && 17 u - 55 <= ++x + y
```

Lösung

```
(2>3) && 17 u - 55 <= ++x + y
```

beginne links

false

Verständniskontrolle I

Aufgabe

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

```
2>3 && 17 u - 55 <= ++x + y
```

Lösung

```
(2>3) && 17 u - 55 <= ++x + y    beginne links
false && 17 u - 55 <= ++x + y    short circuit!
```

Verständniskontrolle I

Aufgabe

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

```
2>3 && 17 u - 55 <= ++x + y
```

Lösung

```
(2>3) && 17 u - 55 <= ++x + y    beginne links
false && 17 u - 55 <= ++x + y    short circuit!
false
```

Merke: (`true || ...`) evaluiert immer zu `true`

Verständniskontrolle II

Aufgabe

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

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

Verständniskontrolle II

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
```

Verständniskontrolle II

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
```

Verständniskontrolle II

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
```

Verständniskontrolle II

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
```

Verständniskontrolle II

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
```

Verständniskontrolle II

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
```

Verständniskontrolle II

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
```

Verständniskontrolle III

Aufgabe

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

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

Verständniskontrolle III

Aufgabe

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

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

Lösung

Zuerst: Klammern!

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

beginne links

Verständniskontrolle III

Aufgabe

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

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

Lösung

Zuerst: Klammern!

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

beginne links

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

Verständniskontrolle III

Aufgabe

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

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

Lösung

Zuerst: Klammern!

```
(x == 1) || ((1 / (x - 1)) < 1)    beginne links
(1 == 1) || ((1 / (x - 1)) < 1)
true || ((1 / (x - 1)) < 1)    short circuit!
```

Verständniskontrolle III

Aufgabe

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

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

Lösung

Zuerst: Klammern!

```
(x == 1) || ((1 / (x - 1)) < 1)    beginne links
(1 == 1) || ((1 / (x - 1)) < 1)
true || ((1 / (x - 1)) < 1)    short circuit!
true
```

Fragen/Unklarheiten?

Minimale Expression

Oft kann man Ausdrücke vereinfachen, um sie leichter zu verstehen oder um sie eleganter zu gestalten und weniger Aufmerksamkeit zu erregen. Beispielsweise:

```
(x > 3) == true || z - 3 >= 0
```

kann vereinfacht werden zu

Minimale Expression

Oft kann man Ausdrücke vereinfachen, um sie leichter zu verstehen oder um sie eleganter zu gestalten und weniger Aufmerksamkeit zu erregen. Beispielsweise:

```
(x > 3) == true || z - 3 >= 0
```

kann vereinfacht werden zu

```
x > 3 || z >= 3
```

Verständniskontrolle I

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)
```

Verständniskontrolle I

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)
```

Lösung

```
b && n > 0 && n < 6
```

Verständniskontrolle I

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)
```

Lösung

```
b && n > 0 && n < 6
```

oder um es noch deutlicher zu machen:

```
b && (n > 0) && (n < 6)
```

Verständniskontrolle I

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)
```

Lösung

```
b && n > 0 && n < 6
```

oder um es noch deutlicher zu machen:

```
b && (n > 0) && (n < 6)
```

Das ist sehr falsch:

```
b && (0 < n < 6)
```

Verständniskontrolle I

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)
```

Lösung

`b && n > 0 && n < 6`

oder um es noch deutlicher zu machen:

`b && (n > 0) && (n < 6)`

Das ist sehr falsch:

`b && (0 < n < 6)` Aber wieso?

Verständniskontrolle II

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
a && !(a == true) || (b == false)
```

Verständniskontrolle II

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
a && !(a == true) || (b == false)
```

Lösung

```
!b
```

Verständniskontrolle III

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
!(a != false && !(b == false) == false)
```

Verständniskontrolle III

Aufgabe

Minimiere die die folgenden booleschen Expressions. Nehme an `bool a,b; int n;`:

```
!(a != false && !(b == false) == false)
```

Lösung

```
!a || b
```

Fragen/Unklarheiten?

4. `for`-Schleifen

Kurzeinführung zu Scopes

- Praktisch immer, wenn Ihr {diese geschweiften Klammern} verwendet, erstellt ihr einen {Scope}

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

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

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; condition; expression){
    statement 1;
    statement 2;
    // ....
}
```

Wichtige Bemerkung

Der `expression`-Teil wird *nach* den `statements` 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`-Schleifen-Beispiel

`for` Loop

Example – `for` Loop

```
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

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

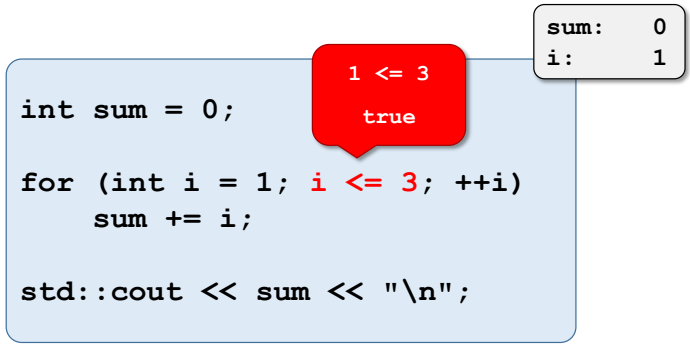
```
sum:    0
i:      1
```

```
int sum = 0;

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

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

Example – `for` Loop



Example – `for` Loop

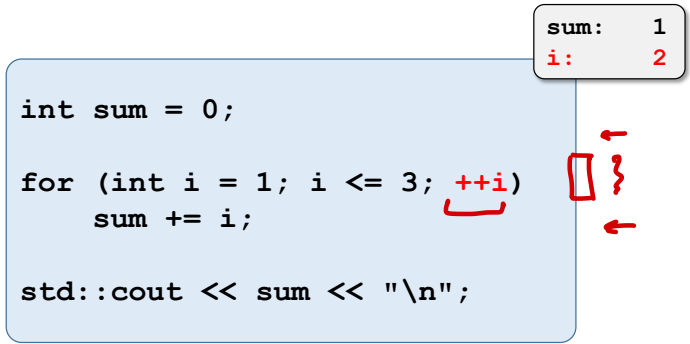
```
int sum = 0;

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

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

sum:	1
i:	1

Example – `for` Loop



Example – `for` Loop

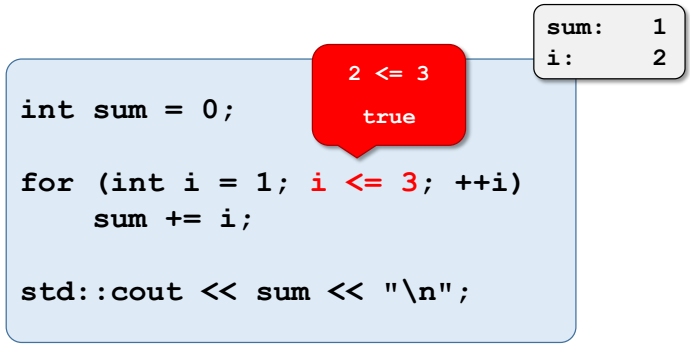
```
int sum = 0;

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

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

```
sum:    1
i:      2
```

Example – `for` Loop



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

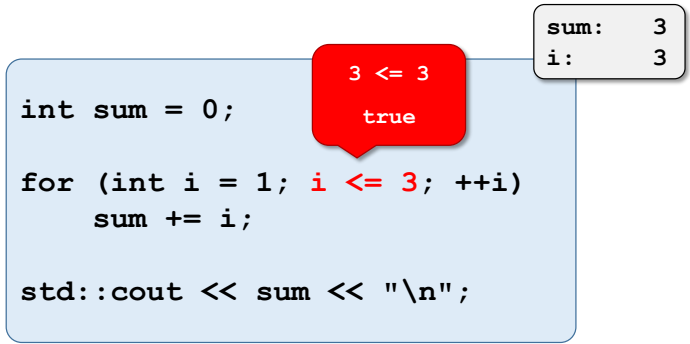
```
int sum = 0;

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

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

```
sum:    3
i:      3
```

Example – `for` Loop



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

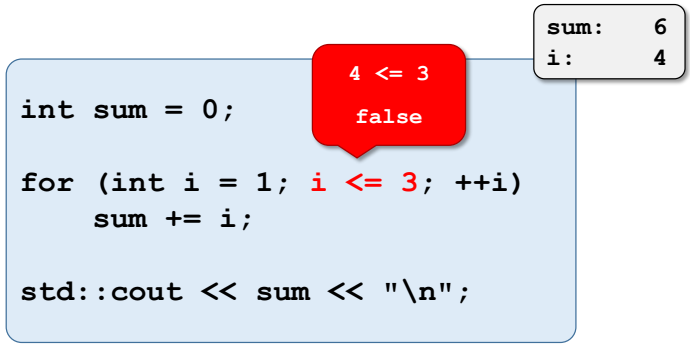
```
int sum = 0;

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

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

```
sum:    6
i:      4
```

Example – `for` Loop



Example – `for` Loop

```
int sum = 0;

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

```
sum:    6
```

Fragen/Unklarheiten?

Aufgabe *Strange Sum*

n

i:

Aufgabe

Öffnet *Strange Sum* auf **code** expert und versucht es zuerst mit Stift und Papier zu lösen. (~~10min~~ ~5min)

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.

std::cin >> n;

if (i % 2 == 0) // even

i

if (i % 2)

&&

if (i % 5)

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 *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 *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*

```cpp
#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?

5. Debugging

Debugging

Debugging
…ist der Prozess des Auffindens und der Behebung von Fehlern (Defekte oder Probleme, die den korrekten Betrieb verhindern) in Programmen, Software oder Systemen.

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 - Live Demo

Frage

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

Debugging - Live Demo

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

Antwort
Sachen ausgeben lassen an interessanten Stellen im Code

Debugging - Live Demo

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?

Debugging - Live Demo

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!

Debugging - Live Demo

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 - Live Demo

Frage
Wie können wir ermitteln, wieso trotzdem nichts ausgegeben wird?

Debugging - Live Demo

Frage
Wie können wir ermitteln, wieso trotzdem nichts ausgegeben wird?

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

Debugging - Live Demo

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?

Debugging - Live Demo

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?

Debugging in **code** expert

Debugging in code expert

```
int main() {

    long start = 2;
    // Finding the largest power of 2 representable by long
    while (true) {
      if (start * 2 > LONG_MAX)
        std::cout << "Stopping..." << std::endl;
        break;

      start *= 2;
    }

    std::cout << "Largest power of two representable by long: "
              << start << std::endl;
    // ...
```

Debugging in code expert

```
    // ...
    // Finding the smallest negative power of 2 representable by long
    start = -2;
    while (true) {
      if (start * 2 < LONG_MIN)
        std::cout << "Stopping..." << std::endl;
        break;

      start *= 2;
    }

    std::cout << "Smallest negative power of two representable by long: "
              << start << std::endl;

    return 0;
}
```

Debugging in code expert

Aufgabe
Öffnet das Beispiel "Debugging in CodeExpert" auf **code** expert

Description
Find the largest (N) and smallest negative (n) power of two that is representable by `long`. Formally that is:

$$\begin{aligned} N &= \max(\{2^i : i \in N \text{ and } 2^i \text{ is representable by } \texttt{long}\}) \\ n &= \min(\{-2^i : i \in N \text{ and } -2^i \text{ is representable by } \texttt{long}\}) \end{aligned}$$

Debugging in code expert

Frage
Wie soll das Programm funktionieren?

Debugging in code expert

Frage
Wie soll das Programm funktionieren?

Antwort
2er-Potenzen berechnen, bis man den maximalen Wert von `long` erreicht

Debugging in code expert

Frage
Wie soll das Programm funktionieren?

Antwort
2er-Potenzen berechnen, bis man den maximalen Wert von `long` erreicht

Frage
Es gibt einen Syntaxfehler im Code. Kann jemand erkennen, was falsch ist?

Debugging in code expert

Frage
Wie soll das Programm funktionieren?

Antwort
2er-Potenzen berechnen, bis man den maximalen Wert von `long` erreicht

Frage
Es gibt einen Syntaxfehler im Code. Kann jemand erkennen, was falsch ist?

Antwort
die `{}` fehlen nach dem `if`-Zweig.

Debugging in code expert

Frage
Was ist an diesem Programm noch falsch?

Debugging in code expert

Frage
Was ist an diesem Programm noch falsch?

Antwort
Der Ausdruck `*2` führt zu einem **overflow**, was ein undefiniertes Verhalten für `signed int` ist. Hier führt es zu einer Endlosschleife!

Debugging in code expert

Frage
Was ist an diesem Programm noch falsch?

Antwort
Der Ausdruck `*2` führt zu einem **overflow**, was ein undefiniertes Verhalten für `signed int` ist. Hier führt es zu einer Endlosschleife!

Frage
Wie können wir also den Code korrigieren?

Debugging in code expert - Lösung

```cpp
int main() {
    long start = 2;
    // Finding the largest power of 2 representable by long
    while (true) {
      if (start > LONG_MAX / 2) {
        std::cout << "Stopping..." << std::endl;
        break;
      }
      start *= 2;
    }

    std::cout << "Largest power of two representable by long: "
              << start << std::endl;
    // ...
```

Debugging in code expert - Lösung

```
    // ...
    // Finding the smallest negative power of 2 representable by long
    start = -2;
    while (true) {
      if (start < LONG_MIN / 2) {
        std::cout << "Stopping..." << std::endl;
        break;
      }
      start *= 2;
    }

    std::cout << "Smallest negative power of two representable by long: "
              << start << std::endl;

    return 0;
}
```

Fragen/Unklarheiten?

7. Selection Statements

Even Numbers

Even Numbers

Aufgabe
Öffnet das Beispiel "Even Numbers" auf **code** expert und versucht es zu lösen

Description
Write a program that accepts an input value n (as an `int`). If the input value is non-negative, it outputs the biggest even number m that does not exceed the input value. If the input value is negative, then the program should output 0.

Input A whole number n

Output

- If $n \geq 0$, output the biggest even number $m \leq n$
- If $n < 0$, output 0

Examples $2 \to 2, \qquad 13 \to 12, \qquad 43 \to 42$

Even Numbers - Lösung

Even Numbers - Lösung

```cpp
#include <iostream>
int main () {

    int n;
    std::cin >> n;
    if ( n < 0) {
        n = 0;
    }
    if (n % 2 == 1) {
        n = n - 1;
    }

    std::cout << n << std::endl;

    return 0;
}
```

Fragen/Unklarheiten?

Alternative Basen

Zahlen in anderen Basen funktionieren konzeptionell genau gleich

Alternative Basen

Zahlen in anderen Basen funktionieren konzeptionell genau gleich

binär	hexadezimal	dezimal
1111	F	15
1’1111	1F	31
11’0111’1100’0101	37C5	14’277
1010’1100’1101’1100	ACDC	44’252
1’0000’0000’0000’0000	1’0000	65’536
1010’1111’1111’1110’0000’1000’0001’0101	AFFE’0815	2’952’661’013

Fragen/Unklarheiten?

Two’s complement

Konversion von Dezimahl zu Binär:

1. Nehme den Betrag der Zahl und schreibe die Binärrepräsentation davon
2. Bits flippen: 1 wird zu 0 und 0 wird zu 1
3. 1 addieren und den Overflow ignorieren

Two’s complement

Konversion von Dezimahl zu Binär:

1. Nehme den Betrag der Zahl und schreibe die Binärrepräsentation davon
2. Bits flippen: 1 wird zu 0 und 0 wird zu 1
3. 1 addieren und den Overflow ignorieren

Beispiel: -6

1. $+6$ lässt sich als 0110 schreiben
2. Nach Bit Flipping erhalten wir 1001
3. $1001 + 1 = 1010$

Two's complement

Tipps zu Two's Complement:

- Das erste Bit einer Zahl n heisst **most significant bit** und gibt das Vorzeichen an

$$\text{msb} \begin{cases} = 1 \rightarrow n < 0 \\ = 0 \rightarrow n >= 0 \end{cases}$$

- Die Grösse (= Anzahl Bits) einer Zahl bestimmt, welche Zahlen dargestellt werden können
  - Mit n Bits können Zahlen von $-(2^{n-1})$ bis $2^{n-1} - 1$ dargestellt werden

Fragen/Unklarheiten?

9. Alte Prüfungsaufgabe

Prüfung 2018 "Typen und Werte"

1 Typen und Werte / Types and values *(9 Punkte)*

Geben Sie für jeden der sechs Ausdrücke unten jeweils C++-Typ (0.5 P) und Wert (1 P) an! Nehmen Sie für Fliesskommazahlen den Standard IEEE 754 an! *For each of the six expressions below, provide the C++ type (0.5 P) and value (1 P)! For floating point numbers, assume the IEEE 754 standard!*

Prüfung 2018 "Typen und Werte"

(a) `8 + 5 / 3` — 1.5 P

Typ/ *Type* | Wert/ *Value*

(b) `1.0 * 0.1 == 0.1 * 1.0` — 1.5 P

Typ/ *Type* | Wert/ *Value*

(c) `1e1 * 2e2` — 1.5 P

Typ/ *Type* | Wert/ *Value*

Prüfung 2018 "Typen und Werte" – Lösung

(a) `8 + 5 / 3` — 1.5 P

Typ/ *Type*	Wert/ *Value*
`int`	`9`

(b) `1.0 * 0.1 == 0.1 * 1.0` — 1.5 P

Typ/ *Type*	Wert/ *Value*
`bool`	`true`

(c) `1e1 * 2e2` — 1.5 P

Typ/ *Type*	Wert/ *Value*
`double`	`2000`

Prüfung 2018 "Typen und Werte"

(d) `5 / 2.0f + 5 / 2.0` — 1.5 P

Typ/ *Type*	Wert/ *Value*

(e) `!false || true && false` — 1.5 P

Typ/ *Type*	Wert/ *Value*

(f) `17 % 5u` — 1.5 P

Typ/ *Type*	Wert/ *Value*

Prüfung 2018 "Typen und Werte" – Lösung

(d) `5 / 2.0f + 5 / 2.0` — 1.5 P

Typ/ *Type*	Wert/ *Value*
`double`	`5`

(e) `!false || true && false` — 1.5 P

Typ/ *Type*	Wert/ *Value*
`bool`	`true`

(f) `17 % 5u` — 1.5 P

Typ/ *Type*	Wert/ *Value*
`unsigned int`	`2`

10. Tipps zu **code** expert

Tipps für **code** expert

Task "two-complement integer representation (Optional)"

- nichts $\rightarrow$ Basis 10
- `0b` $\rightarrow$ Basis 2
- `0` $\rightarrow$ Basis 8 (also `10` $\neq$ `010`)
- `0x` $\rightarrow$ Basis 16

Task "From decimal to binary representation"

- Denkt an letzte Übungsstunde…

Task "Fibonacci overflow check"

- ***Most importantly***: *exit the print loop as soon as you detect that an overflow would occur.*, also darf die Addition *nicht* passieren!

11. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche noch!