Informatik

Ganzzahl-Division und Modulo, Ausdrücke

Adel Gavranović — ETH Zürich — 2025

Überblick

Lernziele
Ganzzahlige Division und Modulo
Logische Ausdrücke
    Kurzschlussauswertung (“Short Circuit Evaluation”)
Ausdrücke mit gemischten Typen
Gültige und ungültige Ausdrücke

n.ethz.ch/~agavranovic

Material
Webpage
Mail

1. Follow-up

Follow-up aus letzter Übungsstunde

Follow-up aus letzter Übungsstunde

- LaTeXTutorial (optional, *nicht* prüfungsrelevant) Youtube Tutorial

2. Feedback zu **code** expert

Allgemeines bezüglich **code** expert

Allgemeines bezüglich **code** expert

- Läuft alles wie gehofft?

Fragen bezüglich **code** expert eurerseits?

3. Lernziele

Ziele

Ziele

- ☐ Ganzzahl- und Modulo-Division verstehen und berechnen können
- ☐ (Gemischte) Ausdrücke von Hand evaluieren können und Evaluationsschritte begründen können
- ☐ ~~Ein Programm schreiben können, das eine positive Zahl in eine bestimmte Basis umrechnet~~

4. Zusammenfassung

Getting on the same page

Getting on the same page

- Was wurde diese Woche behandelt?

5. Ganzzahlige Division und Modulo

Integer Division & Modulo

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler “ignoriert” Dezimalstellen, wenn er (**`unsigned`**) **`int`** durch (**`unsigned`**) **`int`** teilt.

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler “ignoriert” Dezimalstellen, wenn er (`unsigned`) `int` durch (`unsigned`) `int` teilt.

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler "ignoriert" Dezimalstellen, wenn er (`unsigned`) `int` durch (`unsigned`) `int` teilt.

```
7/3
```

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler "ignoriert" Dezimalstellen, wenn er (**`unsigned`**) **`int`** durch (**`unsigned`**) **`int`** teilt.

```
7/3    == 2
15/4
```

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler "ignoriert" Dezimalstellen, wenn er (**`unsigned`**) **`int`** durch (**`unsigned`**) **`int`** teilt.

```
7/3    == 2
15/4   == 3
16/4
```

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler "ignoriert" Dezimalstellen, wenn er (**`unsigned`**) **`int`** durch (**`unsigned`**) **`int`** teilt.

```
7/3   == 2
15/4  == 3
16/4  == 4
```

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

```
7%3
```

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler "ignoriert" Dezimalstellen, wenn er (**`unsigned`**) **`int`** durch (**`unsigned`**) **`int`** teilt.

```
7/3    == 2
15/4   == 3
16/4   == 4
```

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

```
7%3    == 1
15%4
```

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler "ignoriert" Dezimalstellen, wenn er (**`unsigned`**) **`int`** durch (**`unsigned`**) **`int`** teilt.

```
7/3    == 2
15/4   == 3
16/4   == 4
```

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

```
7%3    == 1
15%4   == 3
16%4
```

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler "ignoriert" Dezimalstellen, wenn er (**`unsigned`**) **`int`** durch (**`unsigned`**) **`int`** teilt.

```
7/3    == 2
15/4   == 3
16/4   == 4
```

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

```
7%3    == 1
15%4   == 3
16%4   == 0
```

Integer Division & Modulo

Integer Division (`a/b`)

Der Compiler "ignoriert" Dezimalstellen, wenn er (**`unsigned`**) **`int`** durch (**`unsigned`**) **`int`** teilt.

```
7/3    == 2
15/4   == 3
16/4   == 4
```

Modulo (`a%b`)

Division mit Rest, aber *nur* den Rest.

```
7%3    == 1
15%4   == 3
16%4   == 0
```

Wichtige Identität

```
(a / b) * b + a % b == a
```

Fragen/Unklarheiten?

Code Snippet Quiz

Frage: Welche Frage beantwortet dieses Code Snippet?

```
int a;
std::cin >> a;

if (a % 2 == 0) {
    std::cout << "Yes" << std::endl;
} else {
    std::cout << "No" << std::endl;
}
```

Antwort:

Code Snippet Quiz

Frage: Welche Frage beantwortet dieses Code Snippet?

```
int a;
std::cin >> a;

if (a % 2 == 0) {
    std::cout << "Yes" << std::endl;
} else {
    std::cout << "No" << std::endl;
}
```

Antwort: Die Ausgabe beantwortet, ob die eingegebene Zahl (die in `a` gespeichert wird) eine gerade Zahl ist.

Mal schauen, was ihr gelernt habt

- Geht auf `expert.ethz.ch`
- Logt euch ein
- Geht zu “Code Examples”
- Unter “Lecture 2: Exercise Session”, öffnet “Last Three Digits”

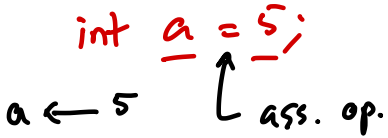
Mal schauen, was ihr gelernt habt

- Geht auf `expert.ethz.ch`
- Logt euch ein
- Geht zu “Code Examples”
- Unter “Lecture 2: Exercise Session”, öffnet “Last Three Digits”
- Versucht Ansätze für diese Aufgabe zu finden (5 Minuten)

Mal schauen, was ihr gelernt habt

- Geht auf `expert.ethz.ch`
- Logt euch ein
- Geht zu “Code Examples”
- Unter “Lecture 2: Exercise Session”, öffnet “Last Three Digits”
- Versucht Ansätze für diese Aufgabe zu finden (5 Minuten)
- Und nun diskutiert eure Ansätze mit euren Sitznachbar:innen

Aufgabe



Task “Last Three Digits”

Write a program which reads in an integer `a` larger than 1000 and outputs its last three digits with a space between them.

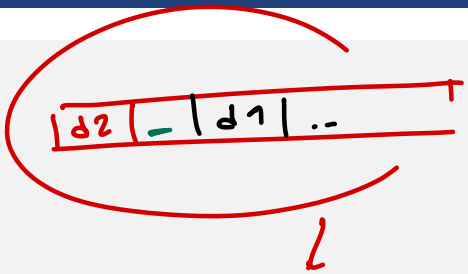
For example, if `a = 14325`, the output should be `3 2 5`.

mind the gap!

Lösung

Lösung

```
#include <iostream>
int main() {
    int a;
    // input
    std::cin >> a;
    // computation
    int digit0 = a % 10;
    int remainder0 = a / 10;
    int digit1 = remainder0 % 10;
    int remainder1 = remainder0 / 10;
    int digit2 = remainder1 % 10;
    // output
    std::cout << digit2 << " " << digit1 << " " << digit0 << std::endl;
    return 0;
}
```



Fragen/Unklarheiten?

6. Logische Ausdrücke

Was sind Expressions?

Expressions

…sind Ausdrücke die evaluiert werden können. Sie kommen oft in Form von mathematischen Ausdrücken vor, die man dann Stück für Stück evaluiert.

Beispiel

Evaluiere den Ausdruck

```
3 < 4 + 1 && 2 < 3
```

Beispiel

Evaluiere den Ausdruck

```
3 < 4 + 1 && 2 < 3
(3 < 5) && (2 < 3)
```

Beispiel

Evaluiere den Ausdruck

```
3 < 4 + 1 && 2 < 3
3 < 5 && 2 < 3
True && 2 < 3
```

Beispiel

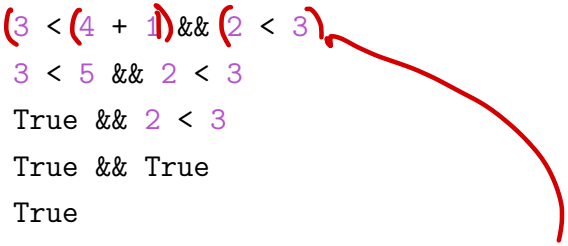
Evaluiere den Ausdruck

```
3 < 4 + 1 && 2 < 3
3 < 5 && 2 < 3
True && 2 < 3
True && True
```

Beispiel

Evaluiere den Ausdruck

```
(3 < (4 + 1)) && (2 < 3)
3 < 5 && 2 < 3
True && 2 < 3
True && True
True
```



Wie können wir mit diesem Wissen den obigen Ausdruck in Klammern setzen?

Beispiel

Evaluiere den Ausdruck

```
3 < 4 + 1 && 2 < 3
3 < 5 && 2 < 3
True && 2 < 3
True && True
True
```

Wie können wir mit diesem Wissen den obigen Ausdruck in Klammern setzen?

```
(3 < (4 + 1)) && (2 < 3)
```

6. Logische Ausdrücke

6.1. Kurzschlussauswertung (“Short Circuit Evaluation”)

Beispiel

Manchmal gibt es zwei Operatoren mit demselben Vorrang, zum Beispiel

`false && false && true`.

Beispiel

Manchmal gibt es zwei Operatoren mit demselben Vorrang, zum Beispiel

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

In diesem Fall müssen wir die Assoziativität berücksichtigen, d.h. `&&` ist ein *links*-assoziativer Operator. Das bedeutet hier, dass der linke der beiden Operatoren *zuerst* berechnet werden muss, d.h. die Klammern sind so zu setzen

Beispiel

Manchmal gibt es zwei Operatoren mit demselben Vorrang, zum Beispiel

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

In diesem Fall müssen wir die Assoziativität berücksichtigen, d.h. `&&` ist ein *links*-assoziativer Operator. Das bedeutet hier, dass der linke der beiden Operatoren *zuerst* berechnet werden muss, d.h. die Klammern sind so zu setzen

```
(falsch && falsch) && wahr
```

Kurzschlussauswertung (“Short-Circuit Evaluation”)

Für die Operatoren `&&` und `||` gibt es eine spezielle Regel namens *Kurzschlussauswertung*. Für diese Operatoren garantiert der Compiler, dass er immer den *linken Ausdruck zuerst* auswertet. *Wenn das Gesamtergebnis allein aus dem linken Ausdruck abgeleitet werden kann*, dann wird der rechte Ausdruck *nicht* ausgewertet, was unnötige Berechnungen erspart.

Kurzschlussauswertung (“Short-Circuit Evaluation”)

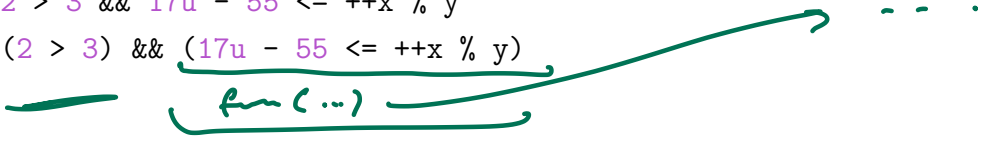
Für die Operatoren `&&` und `||` gibt es eine spezielle Regel namens *Kurzschlussauswertung*. Für diese Operatoren garantiert der Compiler, dass er immer den *linken Ausdruck zuerst* auswertet. *Wenn das Gesamtergebnis allein aus dem linken Ausdruck abgeleitet werden kann*, dann wird der rechte Ausdruck *nicht* ausgewertet, was unnötige Berechnungen erspart.

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

Kurzschlussauswertung (“Short-Circuit Evaluation”)

Für die Operatoren `&&` und `||` gibt es eine spezielle Regel namens *Kurzschlussauswertung*. Für diese Operatoren garantiert der Compiler, dass er immer den *linken Ausdruck zuerst* auswertet. *Wenn das Gesamtergebnis allein aus dem linken Ausdruck abgeleitet werden kann*, dann wird der rechte Ausdruck *nicht* ausgewertet, was unnötige Berechnungen erspart.

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



Kurzschlussauswertung (“Short-Circuit Evaluation”)

Für die Operatoren `&&` und `||` gibt es eine spezielle Regel namens *Kurzschlussauswertung*. Für diese Operatoren garantiert der Compiler, dass er immer den *linken Ausdruck zuerst* auswertet. *Wenn das Gesamtergebnis allein aus dem linken Ausdruck abgeleitet werden kann*, dann wird der rechte Ausdruck *nicht* ausgewertet, was unnötige Berechnungen erspart.

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

Kurzschlussauswertung (“Short-Circuit Evaluation”)

Für die Operatoren `&&` und `||` gibt es eine spezielle Regel namens *Kurzschlussauswertung*. Für diese Operatoren garantiert der Compiler, dass er immer den *linken Ausdruck zuerst* auswertet. *Wenn das Gesamtergebnis allein aus dem linken Ausdruck abgeleitet werden kann*, dann wird der rechte Ausdruck *nicht* ausgewertet, was unnötige Berechnungen erspart.

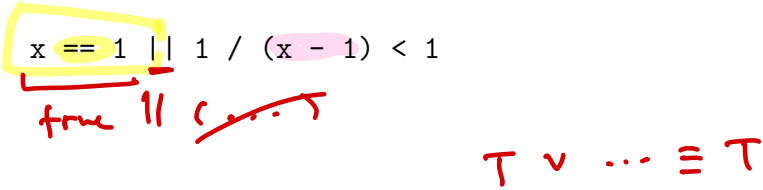
```
2 > 3 && 17u - 55 <= ++x % y
(2 > 3) && (17u - 55 <= ++x % y)
false && (17u - 55 <= ++x % y)
false
```

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung



Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

```
(x == 1) || ((1 / (x - 1)) < 1)
(1 == 1) || ((1 / (x - 1)) < 1)
true || ((1 / (x - 1)) < 1) // Short Circuit ↯
true
```

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

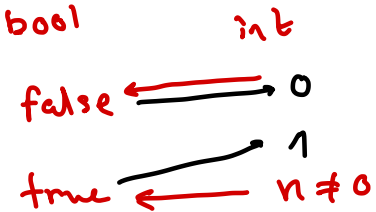
Short Circuit!

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung



Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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

"implicitly casted" to an int

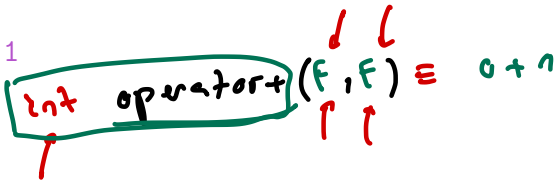
Quiz

Aufgabe Werten Sie die folgenden Ausdrücke von Hand aus und notieren Sie alle Zwischenschritte. Sie können davon ausgehen, dass `x` eine Variable vom Typ `int` mit dem Wert `1` ist.

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

Lösung

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



Quiz

Aufgabe Minimiere den folgenden booleschen Ausdruck, wobei `a` und `b` Variablen vom Typ `bool` sind und `n` ein `int` ist:

Quiz

Aufgabe Minimiere den folgenden booleschen Ausdruck, wobei `a` und `b` Variablen vom Typ `bool` sind und `n` ein `int` ist:

1. `b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)`
2. `a && !(a == true) || (b == false)`
3. `!(a != false && !(b == false) == false)`

Quiz

Aufgabe Minimiere den folgenden booleschen Ausdruck, wobei `a` und `b` Variablen vom Typ `bool` sind und `n` ein `int` ist:

1. `b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)`
2. `a && !(a == true) || (b == false)`
3. `!(a != false && !(b == false) == false)`

Lösung

Quiz

Aufgabe Minimiere den folgenden booleschen Ausdruck, wobei `a` und `b` Variablen vom Typ `bool` sind und `n` ein `int` ist:

1. `b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)`
2. `a && !(a == true) || (b == false)`
3. `!(a != false && !(b == false) == false)`

Lösung

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

Quiz

Aufgabe Minimiere den folgenden booleschen Ausdruck, wobei `a` und `b` Variablen vom Typ `bool` sind und `n` ein `int` ist:

1. `b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)`
2. `a && !(a == true) || (b == false)`
3. `!(a != false && !(b == false) == false)`

Lösung

1. `b && n > 0 && n < 6`
2. `!b`

Quiz

Aufgabe Minimiere den folgenden booleschen Ausdruck, wobei `a` und `b` Variablen vom Typ `bool` sind und `n` ein `int` ist:

1. `b == true && !(n < 0) && (n != 0) && (n != 6) && (n < 6)`
2. `a && !(a == true) || (b == false)`
3. `!(a != false && !(b == false) == false)`

Lösung

1. `b && n > 0 && n < 6`
2. `!b`
3. `!a || b`

Fragen/Unklarheiten?

bool, int, double, ...

7. Ausdrücke mit gemischten Typen

Types

Bislang haben wir in den Vorlesungen drei Gruppen von Typen gesehen:

- Logische Variablen: `bool`;
- Ganzzahlen: `int`, `unsigned int`;
- Fliesskommazahlen: `float`, `double`.

Manchmal sind mehrere Types in einer Expression. Wie interagieren verschiedene Typen miteinander?

Types

Bislang haben wir in den Vorlesungen drei Gruppen von Typen gesehen:

- Logische Variablen: `bool`;
- Ganzzahlen: `int`, `unsigned int`;
- Fliesskommazahlen: `float`, `double`.

Manchmal sind mehrere Types in einer Expression. Wie interagieren verschiedene Typen miteinander?

Generalitätsreihenfolge der Typen

Types

Bislang haben wir in den Vorlesungen drei Gruppen von Typen gesehen:

- Logische Variablen: `bool`;
- Ganzzahlen: `int`, `unsigned int`;
- Fliesskommazahlen: `float`, `double`.

Manchmal sind mehrere Types in einer Expression. Wie interagieren verschiedene Typen miteinander?

Generalitätsreihenfolge der Typen

```
bool <
```

Types

Bislang haben wir in den Vorlesungen drei Gruppen von Typen gesehen:

- Logische Variablen: `bool`;
- Ganzzahlen: `int`, `unsigned int`;
- Fliesskommazahlen: `float`, `double`.

Manchmal sind mehrere Types in einer Expression. Wie interagieren verschiedene Typen miteinander?

Generalitätsreihenfolge der Typen

```
bool < int < unsigned int <
```

Types

Bislang haben wir in den Vorlesungen drei Gruppen von Typen gesehen:

- Logische Variablen: `bool`;
- Ganzzahlen: `int`, `unsigned int`;
- Fliesskommazahlen: `float`, `double`.

Manchmal sind mehrere Types in einer Expression. Wie interagieren verschiedene Typen miteinander?

Generalitätsreihenfolge der Typen

least gen. ↙ ↘ most gen.

```
bool < int < unsigned int < float < double
```

Types konvertieren immer zum generellsten Typen der Expression

Wie man sich Types vorstellen kann

Type	Literal	Approximiert

Wie man sich Types vorstellen kann

Type	Literal	Approximiert
`bool`		{`false`, `true`}

Wie man sich Types vorstellen kann

Type	**Literal**	**Approximiert**
`bool`		$\{\texttt{false}, \texttt{true}\}$
`unsigned int`	u	$\mathbb{N}$

Wie man sich Types vorstellen kann

Type	Literal	Approximiert
`bool`		$\{\texttt{false}, \texttt{true}\}$
`unsigned int`	u	$\mathbb{N}$
`int`		$\mathbb{Z}$

Wie man sich Types vorstellen kann

Type	Literal	Approximiert
`bool`		$\{\texttt{false}, \texttt{true}\}$
`unsigned int`	`u`	$\mathbb{N}$
`int`		$\mathbb{Z}$
`float`	`f`	$\mathbb{R}$

Wie man sich Types vorstellen kann

Type	Literal	Approximiert
`bool`		{`false`, `true`}
`unsigned int`	`u`	$\mathbb{N}$
`int`		$\mathbb{Z}$
`float`	`f`	$\mathbb{R}$
`double`		$\mathbb{R}$, aber *double* Präzision

```
int x = 2;
int y = ...
int z = x * y;
```

Types evaluieren I

```
std::cout << 5.0/2 << std::endl;
// what type and value will this return and why?
```

Types evaluieren I

```
std::cout << 5.0/2 << std::endl;
// what type and value will this return and why?
```

Lösung

`double`, `2.5`, weil die `int` `2` zuerst in eine `double` `2.0` konvertiert wird, um diese Expression zu berechnen.

Types evaluieren II

```
std::cout << (1/2)*5.0/2 << std::endl;
// what type and value will this return and why?
```

Types evaluieren II

```
std::cout << (1/2)*5.0/2 << std::endl;
// what type and value will this return and why?
```

Lösung

`double`, `0`, weil zuerst die linke Expression `1/2` evaluiert wird, welche zu `0` evaluiert (Integer-Division). Der Rest ist trivial, weil `0*`irgendetwas evaluiert zu `0`. Aber diese `0` wird vom Type `double` sein.

Literale

Literale

Es gibt bestimmte Buchstaben, die der Compiler mit bestimmten Types verbindet. Wenn ihr dem Compiler sagen möchtest *"Hey, don't treat this `2.0` as a `double`, but instead as a `float`"* müsst ihr ein `f` am Ende des Werts hinzufügen. Etwa so:

Literale

Es gibt bestimmte Buchstaben, die der Compiler mit bestimmten Types verbindet. Wenn ihr dem Compiler sagen möchtest *"Hey, don't treat this `2.0` as a `double`, but instead as a `float`"* müsst ihr ein `f` am Ende des Werts hinzufügen. Etwa so:

```
std::cout << (5/2)*5.0f/2 << std::endl;
```

Types evaluieren III

```
std::cout << (5/2)*5.0f/2 << std::endl;
// what type and value will this return and why?
```

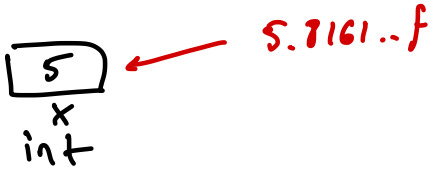
Types evaluieren III

```
std::cout << (5/2)*5.0f/2 << std::endl;
// what type and value will this return and why?
```

Lösung

`float`, `5.0`, (kann als `5.0f` geschrieben werden).

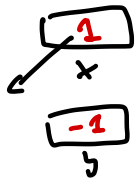
Zuerst, wird `5/2` evaluiert, was zu `2` wird (integer division). Dann wird `2.0f*5.0f` berechnet: Die `int 2` wurde zu einer `float 2`, weil `float` der generellere Type (in dieser Expression) ist. Dito für `/2` später.



Exercise I

1. Welche der folgenden Zeichenfolgen sind keine C++ Expression, und warum nicht? Hierbei seien `x` und `y` Variablen vom Typ `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. Für alle gültigen Expression, die oben identifiziert wurden, entscheide, ob es sich um L-Werte oder R-Werte handelt und begründe deine Entscheidung.
3. Bestimmen Sie die Werte der Ausdrücke und erklären Sie, wie diese Werte erhalten werden. Angenommen, dass anfangs `x == 1` and `y == -1`.

Expression Evaluation - Lösungen a)

```
(y++ < 0 && y < 0) + 2.0
```

Expression Evaluation - Lösungen a)

```
(y++ < 0 && y < 0) + 2.0
(-1 < 0 && y < 0) + 2.0  // after this step: y==0
```

Expression Evaluation - Lösungen a)

```
(y++ < 0 && y < 0) + 2.0
(-1 < 0 && y < 0) + 2.0  // after this step: y==0
(true && y < 0) + 2.0
```

Expression Evaluation - Lösungen a)

```
(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
```

Expression Evaluation - Lösungen a)

```
(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
```

Expression Evaluation - Lösungen a)

```
(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
```

Expression Evaluation - Lösungen a)

```
(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
```

Expression Evaluation - Lösungen a)

```
(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-Wert

Expression Evaluation - Lösungen b)

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

Expression Evaluation - Lösungen b)

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

Invalid

hier wird r-Value (nämlich "alter" Wert von x) zurückgegeben

a == 5 "infix"

|||

equals(a, 5)

= assignment op.

== { return a boolean

== boolean operator

Expression Evaluation - Lösungen c)

```
3.0 + 3 - 4 + 5
```

Expression Evaluation - Lösungen c)

```
3.0 + 3 - 4 + 5
((3.0 + 3) - 4) + 5
```

Expression Evaluation - Lösungen c)

```
3.0 + 3 - 4 + 5
((3.0 + 3) - 4) + 5
((3.0 + 3.0) - 4) + 5
```

Expression Evaluation - Lösungen c)

```
3.0 + 3 - 4 + 5
((3.0 + 3) - 4) + 5
((3.0 + 3.0) - 4) + 5
(6.0 - 4) + 5
```

Expression Evaluation - Lösungen c)

```
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
```

Expression Evaluation - Lösungen c)

```
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
```

Expression Evaluation - Lösungen c)

```
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
```

Expression Evaluation - Lösungen c)

```
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
```

Expression Evaluation - Lösungen c)

```
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-Wert

Expression Evaluation - Lösungen d)

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

Expression Evaluation - Lösungen d)

```
5 % 4 * 3.0 + true * x++
((5 % 4) * 3.0) + (true * (x++))
```

Expression Evaluation - Lösungen d)

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

Expression Evaluation - Lösungen d)

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

Expression Evaluation - Lösungen d)

```
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++))
```

Expression Evaluation - Lösungen d)

```
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)
```

Expression Evaluation - Lösungen d)

```
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)
```

Expression Evaluation - Lösungen d)

```
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
```

Expression Evaluation - Lösungen d)

```
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
```

Expression Evaluation - Lösungen d)

```
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
```

Expression Evaluation - Lösungen d)

```
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-Wert

Fragen/Unklarheiten?

8. Gültige und ungültige Ausdrücke

Beispiel

Evaluiere die Ausdruck `5u + 5 * 3u`

Beispiel

Evaluiere die Ausdruck `5u + 5 * 3u`

`5u + 5 * 3u`

Beispiel

Evaluiere die Ausdruck `5u + 5 * 3u`

`5u + 5 * 3u` Punkt vor Strich

Beispiel

Evaluiere die Ausdruck `5u + 5 * 3u`

`5u + 5 * 3u` Punkt vor Strich

`5u + (5 * 3u)`

Beispiel

Evaluiere die Ausdruck `5u + 5 * 3u`

`5u + 5 * 3u` Punkt vor Strich

`5u + (5 * 3u)` das Resultat wird zu **`unsigned int`**

Beispiel

Evaluiere die Ausdruck `5u + 5 * 3u`

`5u + 5 * 3u` Punkt vor Strich

`5u + (5 * 3u)` das Resultat wird zu **`unsigned int`**

`5u + 15u`

Beispiel

Evaluiere die Ausdruck `5u + 5 * 3u`

`5u + 5 * 3u` Punkt vor Strich

`5u + (5 * 3u)` das Resultat wird zu **`unsigned int`**

`5u + 15u` simple Addition

Beispiel

Evaluiere die Ausdruck `5u + 5 * 3u`

`5u + 5 * 3u` Punkt vor Strich

`5u + (5 * 3u)` das Resultat wird zu **`unsigned int`**

`5u + 15u` simple Addition

`20u`

Valide Expression erkennen

Welche sind eine valide C++-Expression und welche nicht?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Valide Expression erkennen

Welche sind eine valide C++-Expression und welche nicht?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

Valide Expression erkennen

Welche sind eine valide C++-Expression und welche nicht?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. ist nicht valide, da die Klammer `(` nicht geschlossen wird

Valide Expression erkennen

Welche sind eine valide C++-Expression und welche nicht?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. ist nicht valide, da die Klammer `(` nicht geschlossen wird
- 4. ist nicht valide, da `(a*3)` zu einem r-value wird, der Assignment-Operator `(=)` aber ein l-value zu seiner Linken erwartet

Valide Expression erkennen

Welche sind eine valide C++-Expression und welche nicht?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. ist nicht valide, da die Klammer `(` nicht geschlossen wird
- 4. ist nicht valide, da `(a*3)` zu einem r-value wird, der Assignment-Operator `(=)` aber ein l-value zu seiner Linken erwartet
- 1. und 2. sind valide C++-Expressions

Valide Expression erkennen

Welche sind L-Werte und welche sind R-Werte?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Valide Expression erkennen

Welche sind L-Werte und welche sind R-Werte?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

Valide Expression erkennen

Welche sind L-Werte und welche sind R-Werte?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. und 4. sind weder noch, da sie keine validen Expressions sind

Valide Expression erkennen

Welche sind L-Werte und welche sind R-Werte?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. und 4. sind weder noch, da sie keine validen Expressions sind
- 1. ist ein r-value, da der Multiplikation-Operator (`*`) ein r-value zurückgibt

Valide Expression erkennen

Welche sind L-Werte und welche sind R-Werte?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. und 4. sind weder noch, da sie keine validen Expressions sind
- 1. ist ein r-value, da der Multiplikation-Operator (`*`) ein r-value zurückgibt
- 2. ist ein l-Value, da der Assignment-Operator (`=`) ein l-Value zurückgibt

Valide Expression erkennen

Zu was werden die Expressions evaluiert?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Valide Expression erkennen

Zu was werden die Expressions evaluiert?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

Valide Expression erkennen

Zu was werden die Expressions evaluiert?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. und 4. können nicht evaluiert werden, da sie keine validen Expressions sind

Valide Expression erkennen

Zu was werden die Expressions evaluiert?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. und 4. können nicht evaluiert werden, da sie keine validen Expressions sind
- 1. evaluiert zu 6, also das Ergebnis der simplen Multiplikationen

Valide Expression erkennen

Zu was werden die Expressions evaluiert?

1. `1*(2*3)`
2. `(a=1)`
3. `(1`
4. `(a*3) = (b*5)`

Lösungen

- 3. und 4. können nicht evaluiert werden, da sie keine validen Expressions sind
- 1. evaluiert zu 6, also das Ergebnis der simplen Multiplikationen
- 2. evaluiert zu 1, da `a` zurückgegeben wird, dem gerade davor der Wert 1 zugewiesen wurde

Warum sind R-Werte und L-Werte wichtig?

Warum sind R-Werte und L-Werte wichtig?

Später im Kurs werden wir numerische Operatoren (wie `operator+` oder `operator=`) für neue numerische Typen (wie eigene Version von `std::complex`) implementieren müssen. Diese Typen und ihre Operatoren sollten sich wie die eingebauten Typen verhalten. So sollte `operator=` eine Rückgabe `*this` machen, aber `operator+` nicht.

9. Outro

Allgemeine Fragen?

Bis nächstes Mal

Schöne Woche noch!