Übungsstunde

Woche 04

Adel Gavranović
adel.gavranovic@inf.ethz.ch

Overview

Heutige Themen

Intro

Expressions

Loops

Reihen Berechnen

Outro

Links

- polybox zum Material für die Übungsstunden
- Mail an Assistenten

Informationen

Informationen

- Bitte meldet jegliche Probleme die ihr mit [code]expert habt direkt mir oder dem Head TA. Ideallerweise mit Screenshot, Console Output und einer kleinen Beschreibung des Problems/Fehlers

Informationen

- Bitte meldet jegliche Probleme die ihr mit `[code]`expert habt direkt mir oder dem Head TA. Ideallerweise mit Screenshot, Console Output und einer kleinen Beschreibung des Problems/Fehlers
- Die Textaufgabe E2:T2 ”Representation of Integers” wurde von Autograder übernommen

Informationen

- Bitte meldet jegliche Probleme die ihr mit [code]expert habt direkt mir oder dem Head TA. Ideallerweise mit Screenshot, Console Output und einer kleinen Beschreibung des Problems/Fehlers
- Die Textaufgabe E2:T2 ”Representation of Integers” wurde von Autograder übernommen
- Von nun an, erhält man bei Programmieraufgaben alle (7) Punkte oder keine
  - An der Prüfung wird es Teilpunkte geben
  - Bei den Selfassessments wird es Teilpunkte geben
  - Bei den Bonustasks wird es *wahrscheinlich* Teilpunkte geben

Kommentare zu [code]expert

Bitte bedenkt, dass euer Code auch noch von anderen gelesen wird (insb. von mir) und ihr anderen das Lesen und Verstehen eures Codes möglichst einfach machen sollt.

```
// schon kleine Kommentare
// machen einen grossen Unterschied!
```

Kommentare zu [code]expert

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen
- Tabs/Spaces um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen
- Tabs/Spaces um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)
- Nicht über kleinen grauen Strich hinausschreiben

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen
- Tabs/Spaces um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)
- Nicht über kleinen grauen Strich hinausschreiben

Kommentare

- Ihr könnt alle Kommentare von [code]expert löschen

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen
- Tabs/Spaces um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)
- Nicht über kleinen grauen Strich hinausschreiben

Kommentare

- Ihr könnt alle Kommentare von [code]expert löschen
- **Dokumentiert euren Code** (Insb. bei mathematischen Ausdrücken und Tricks wichtig)

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen
- Tabs/Spaces um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)
- Nicht über kleinen grauen Strich hinausschreiben

Kommentare

- Ihr könnt alle Kommentare von [code]expert löschen
- **Dokumentiert euren Code** (Insb. bei mathematischen Ausdrücken und Tricks wichtig)
- Fragen/Gedanken/Ansatz ganz oben als `/* Kommentar */`

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen
- Tabs/Spaces um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)
- Nicht über kleinen grauen Strich hinausschreiben

Kommentare

- Ihr könnt alle Kommentare von [code]expert löschen
- **Dokumentiert euren Code** (Insb. bei mathematischen Ausdrücken und Tricks wichtig)
- Fragen/Gedanken/Ansatz ganz oben als `/* Kommentar */`
- Englisch und Deutsch sind beide okay (bevorzugt Englisch)

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen
- Tabs/Spaces um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)
- Nicht über kleinen grauen Strich hinausschreiben

Kommentare

- Ihr könnt alle Kommentare von [code]expert löschen
- **Dokumentiert euren Code** (Insb. bei mathematischen Ausdrücken und Tricks wichtig)
- Fragen/Gedanken/Ansatz ganz oben als `/* Kommentar */`
- Englisch und Deutsch sind beide okay (bevorzugt Englisch)

Task Description/Autograder

- Ich werde inbs. anfangs sehr streng sein, was das missachten von der Task Description angeht

Kommentare zu [code]expert

Formatting und Struktur

- Leere Zeilen um Codeblöcke zu trennen
- Tabs/Spaces um Sachen auf die Gleiche Höhe zu bringen (insb. Kommentare und Scopes)
- Nicht über kleinen grauen Strich hinausschreiben

Kommentare

- Ihr könnt alle Kommentare von [code]expert löschen
- **Dokumentiert euren Code** (Insb. bei mathematischen Ausdrücken und Tricks wichtig)
- Fragen/Gedanken/Ansatz ganz oben als `/* Kommentar */`
- Englisch und Deutsch sind beide okay (bevorzugt Englisch)

Task Description/Autograder

- Ich werde inbs. anfangs sehr streng sein, was das missachten von der Task Description angeht
- Achtet darauf, möglichst wenig Output zu generieren, der nicht explizit gefragt ist

Kommentare zu [code]expert

E2:T1 Expressions

- Valide Expressions müssen nicht zwingend irgendwo abgespeichert werden

Kommentare zu [code]expert

$$\frac{1}{R_{eq}} = \frac{1}{R_{12}} + \frac{1}{R_{34}}$$

$$R_{eq} = \frac{R_{12} \cdot R_{34}}{R_{12} + R_{34}} + \frac{1}{2}$$

E2:T1 Expressions

- Valide Expressions müssen nicht zwingend irgendwo abgespeichert werden

E2:T3 Equivalent Resistance

- Von denjenigen, die für einen %-Ansatz gewählt haben:
  - Wie seid Ihr darauf gekommen?
  - Könnt ihr beweisen (nicht auf DiskMath-Niveau) wieso euer Ansatz korrekt ist?

```
R_mod = ((R34 * r12)/(R12 + R34))
        + (2*((R12 * R34)%(R12 + R34)))/(R12 + R34);
R_ML  = (R12 * R34 + (R12 + R34) / 2) / (R12 + R34);
```

Fragen zu [code]expert eurerseits?

Lernziele

- ☐ komplexe Expressions (die Bools und Arithmetik enthalten) evaluieren können
- ☐ Summen in C++ darstellen und verwenden können
- ☐ alle Arten von Loops (`for`, `while`, `do-while`) tracen können
- ☐ alle Arten von Loops durch alle anderen Arten von Loops ersetzen können

Types

Bisher behandelte Types

Types

Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}

Types

Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}
- integers: `unsigned int`, `int` {`-7, 2, 0`}

Types

Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}
- integers: `unsigned int`, `int` {`-7, 2, 0`}
- floating point numbers: `float`, `double` {`1.4, -4.3, 7.0`}

Types

Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}
- integers: `unsigned int`, `int` {`-7, 2, 0`}
- floating point numbers: `float`, `double` {`1.4, -4.3, 7.0`}

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

Types

Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}
- integers: `unsigned int`, `int` {`-7, 2, 0`}
- floating point numbers: `float`, `double` {`1.4, -4.3, 7.0`}

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

Reihenfolge der (vorgestellten) Types

Types

Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}
- integers: `unsigned int`, `int` {`-7, 2, 0`}
- floating point numbers: `float`, `double` {`1.4, -4.3, 7.0`}

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

Reihenfolge der (vorgestellten) Types

`bool <`

Types

Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}
- integers: `unsigned int`, `int` {`-7, 2, 0`}
- floating point numbers: `float`, `double` {`1.4, -4.3, 7.0`}

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

Reihenfolge der (vorgestellten) Types

`bool < int < unsigned int <`

Types

Bisher behandelte Types

- logic variables: `bool` {`false`, `true`}
- integers: `unsigned int`, `int` {`-7, 2, 0`}
- floating point numbers: `float`, `double` {`1.4, -4.3, 7.0`}

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

Reihenfolge der (vorgestellten) Types

`bool < int < unsigned int < float < double`
Types konvertieren immer zum generellsten Type der Expression

Wie man sich Types vorstellen kann

Type (literal)	Approximiert

Wie man sich Types vorstellen kann

Type (literal)	Approximiert
`bool`	$\mathbb{B} = \{\texttt{false}, \texttt{true}\}$

Wie man sich Types vorstellen kann

Type (literal)	Approximiert
`bool`	$\mathbb{B} = \{\texttt{false}, \texttt{true}\}$
`unsigned int` (`u`)	$\mathbb{N}$

Wie man sich Types vorstellen kann

Type (literal)	Approximiert
`bool`	$\mathbb{B} = \{\texttt{false}, \texttt{true}\}$
`unsigned int` (u)	$\mathbb{N}$
`int`	$\mathbb{Z}$

Wie man sich Types vorstellen kann

Type (literal)	Approximiert
`bool`	$\mathbb{B} = \{\texttt{false}, \texttt{true}\}$
`unsigned int` (`u`)	$\mathbb{N}$
`int`	$\mathbb{Z}$
`float` (`f`)	$\mathbb{R}$

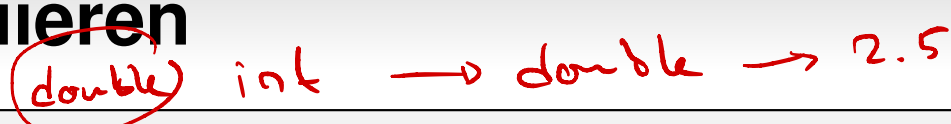
Wie man sich Types vorstellen kann

Type (literal)	Approximiert
`bool`	$\mathbb{B} = \{\texttt{false}, \texttt{true}\}$
`unsigned int` (`u`)	$\mathbb{N}$
`int`	$\mathbb{Z}$
`float` (`f`)	$\mathbb{R}$
`double`	$\mathbb{R}$, aber *double* Präzision

Types evaluieren

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

Types evaluieren




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

Lösung

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

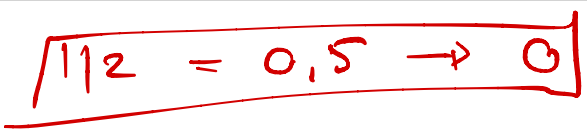
Types evaluieren

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

Lösung

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

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



Types evaluieren

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

Lösung

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

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

Lösung

`double`, `0`, weil der Compiler zuerst die linke Expression `1/2` evaluiert, welche zu `0` evaluiert (weil integer division). Der Rest ist trivial, weil `0*`anything evaluiert zu `0`. Aber diese `0` wird vom Type `double` sein.

Literale

Literale

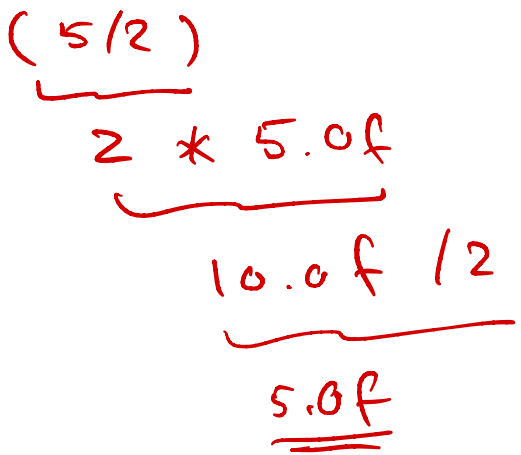
2.0f | 2.0
float | double

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;
// what type and value will this return and why?
```



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;
// what type and value will this return and why?
```

Lösung

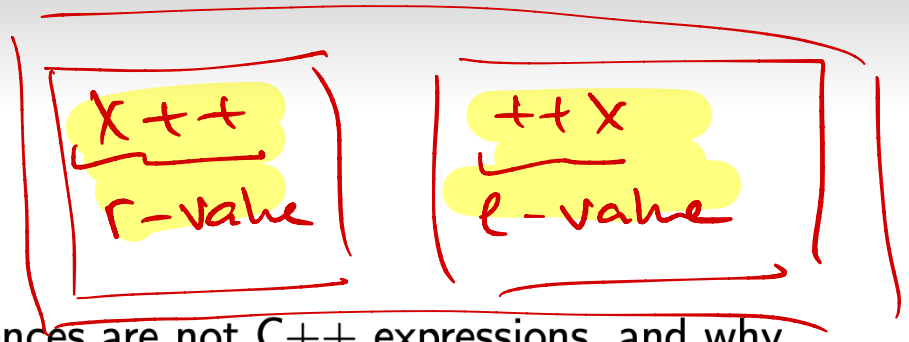
`float`, `5.0`, (kann als `5.0f` geschrieben werden).
Zuerst, evaluiert der Compiler `5/2`, was zu `2` wird (weil integer division). Dann berechnet der Compiler `2.0f*5.0f`: Die `int 2` wurde zu einer `float 2` weil `float` der generellere Type ist (von denen, die in dieser Expression sind. Das gleiche für `*2` später.

Aufgaben

(y++) : 1. gib -1 zurück
2. inkrementiere y um 1

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` → 2.0 double
   b) `y = (x++ = 3)`
   c) `(3.0 + 3) - 4 + 5` → 7.0 double
   d) `(5 % 4 * 3.0) + (true * x++)` = (3.0) + (1 * 1) = 3.0 + 1.0 = 4.0 double
   1.0 * 3.0
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`.

Aufgaben



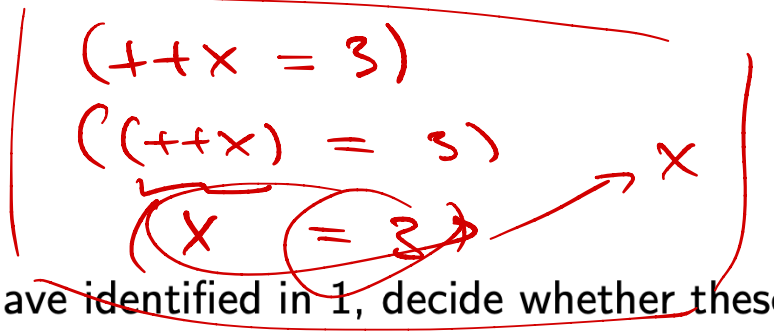
1. Which of the following character sequences are not C++ expressions, and why not? Here, `x` and `y` are variables of type `int`.

   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`.

Lösungen hierzu sind (nach der Lektion) in der polybox unter solutions.pdf.

Fragen/Unklarheiten?

for → while

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

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

for → while

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

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

```
// SOLUTION:

{
int i = 0; ⟶

while(i < n){
 BODY
 ++i;
}
}
```

while → for

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

while(condition){
 BODY
}
```

while → for

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

while(condition){
 BODY
}
```

```
// SOLUTION:

for(;condition;){
 BODY
}
```

do-while → for

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

do{
 BODY
}while(condition)
```

do-while → for

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

do{
 BODY
}while(condition)
```

```
// SOLUTION:

BODY

for(;condition;){
 BODY
}
```

Fragen/Unklarheiten?

Δ sum: <= oder <

→ werden wir nächstes Mal anschauen

Von Summe zur Loop

Mathematische Summen können zu Loops umgewandelt werden

Mathematik:

$$\sum_{i=0}^{n} f(i)$$

```
for (int i=0 ; i<=n; i++){
    f += i
}
```

Von Summe zur Loop

Mathematische Summen können zu Loops umgewandelt werden

Mathematik:

$$\sum_{i=0}^{n} f(i)$$

C++:

```
int n = 0;
int sum = 0;

for(int i = 0; i <= n; i++){
  sum += f(i);
}
```

user input

sum = sum + f(i);

Von Reihe zu Loop

Taylor Series auf `[code]expert`

Schreibe ein Programm, dass $\sin(x)$ bis auf sechs Stellen berechnet.
Tipp: Welchen Loop sollte man hierfür verwenden?
Tipp: MacLaurin-Reihe verwenden.

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

Von Reihe zu Loop

Taylor Series auf [code]expert

Schreibe ein Programm, dass $\sin(x)$ bis auf sechs Stellen berechnet.
Tipp: Welchen Loop sollte man hierfür verwenden?
Tipp: MacLaurin-Reihe verwenden.

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

Aufgabe

- Mit Stift und Papier versuchen (10min)

Von Reihe zu Loop

Taylor Series auf [code]expert

Schreibe ein Programm, dass $\sin(x)$ bis auf sechs Stellen berechnet.
Tipp: Welchen Loop sollte man hierfür verwenden?
Tipp: MacLaurin-Reihe verwenden.

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

wait til functions are introduced and then try (again)

Aufgabe

- Mit Stift und Papier versuchen (10min)
- Mit Person neben euch versuchen in [code]expert zu implementieren (10min)

Fragen/Unklarheiten?

Tipps für [code]expert

== nie mit floating pt numbers

- Nicht ==, sondern Ungleichheiten verwenden

Bis zum nächsten Mal

- ☐ macht eure Hausaufgaben
- ☐ bleibt gesund

Allgemeine Fragen?