Übungsstunde — Informatik — 03

Adel Gavranović

Expressions, Loops, Reihen Berechnen, Scopes

Übersicht

Expressions
Loops
Reihen Berechnen
Scopes
Alte Prüfungsaufgabe

n.ethz.ch/~agavranovic

Material

Webpage

Mail

Bevor es richtig losgeht…

Bevor es richtig losgeht…

- Wieso kommt ihr in *diese* Übungsstunde?

Bevor es richtig losgeht…

- Wieso kommt ihr in *diese* Übungsstunde?
- What about the english sessions?

2. Follow-up

Follow-up aus letzter Übungsstunde

Follow-up aus letzter Übungsstunde

- Habe ich etwas versäumt?

3. Feedback zu **code** expert

Allgemeines bezüglich **code** expert

Allgemeines bezüglich **code** expert

- Ich war *sehr* streng was die Korrekturen angeht. Das wird nicht immer so sein :)

Allgemeines bezüglich **code** expert

- Ich war *sehr* streng was die Korrekturen angeht. Das wird nicht immer so sein :)
- Nehmt euch (ein wenig) Zeit euren Code schön zu formatieren

Allgemeines bezüglich **code** expert

- Ich war *sehr* streng was die Korrekturen angeht. Das wird nicht immer so sein :)
- Nehmt euch (ein wenig) Zeit euren Code schön zu formatieren
- Falls beim Lösen Fragen aufkommen
  - schreibt sie bitte ganz oben im Code hin, damit ich sie direkt sehen kann (falls Sie nicht dringend sind)
  - oder schreibt mir eine E-Mail (falls Sie dingend sind) nachdem ihr bereits eine Abgabe gemacht habt und stellt sicher, dass ich die relevante Abgabe erkenne (mit einem Kommentar markieren oder so)

Allgemeines bezüglich **code** expert

- Ich war *sehr* streng was die Korrekturen angeht. Das wird nicht immer so sein :)
- Nehmt euch (ein wenig) Zeit euren Code schön zu formatieren
- Falls beim Lösen Fragen aufkommen
  - schreibt sie bitte ganz oben im Code hin, damit ich sie direkt sehen kann (falls Sie nicht dringend sind)
  - oder schreibt mir eine E-Mail (falls Sie dingend sind) nachdem ihr bereits eine Abgabe gemacht habt und stellt sicher, dass ich die relevante Abgabe erkenne (mit einem Kommentar markieren oder so)
- Woher haben so viele von euch eigentlich das "`/n`"?

Allgemeines bezüglich **code** expert

const int a = 4;

- Ich war *sehr* streng was die Korrekturen angeht. Das wird nicht immer so sein :)
  a = 7; ↯ compiler error!
- Nehmt euch (ein wenig) Zeit euren Code schön zu formatieren
- Falls beim Lösen Fragen aufkommen
  - schreibt sie bitte ganz oben im Code hin, damit ich sie direkt sehen kann (falls Sie nicht dringend sind)
  - oder schreibt mir eine E-Mail (falls Sie dingend sind) nachdem ihr bereits eine Abgabe gemacht habt und stellt sicher, dass ich die relevante Abgabe erkenne (mit einem Kommentar markieren oder so)
- Woher haben so viele von euch eigentlich das "`/n`"?
- Was bedeutet `const`?

Allgemeines bezüglich **code** expert

Allgemeines bezüglich **code** expert

- Meine Kommentare werden einheitlich auf Englisch geschrieben sein

Allgemeines bezüglich **code** expert

- Meine Kommentare werden einheitlich auf Englisch geschrieben sein
- 3/3 TA-points werden nicht häufig vergeben werden

Allgemeines bezüglich **code** expert

- Meine Kommentare werden einheitlich auf Englisch geschrieben sein
- 3/3 TA-points werden nicht häufig vergeben werden
- "Excellent" ist so ziemlich die höchste Auszeichnung

Allgemeines bezüglich **code** expert

- Meine Kommentare werden einheitlich auf Englisch geschrieben sein
- 3/3 TA-points werden nicht häufig vergeben werden
- "Excellent" ist so ziemlich die höchste Auszeichnung
- Im Editor auf **code** expert gibt es eine feine, graue Linie ganz rechts
  - Schaut, dass ihr die nie mit Code überschreitet
  - Auch nicht mit Kommentaren: spaltet sie lieber auf

```
// Das hier ist ein mehrzeiliger
// Kommentar in C++, den man
// noch immer gut lesen kann!
```

Fragen bezüglich **code** expert eurerseits?

4. Lernziele

Ziele

Ziele für heute

- ☐ Komplexe Expressions, die arithmetische und Bool'sche Operatoren beinhalten, von Hand evaluieren können
- ☐ Mathematische Serien (Summen und Produkte) in C++ kodieren/implementieren können
- ☐ `for`-, `while` und `do-while`-Schleifen tracen können
- ☐ Jede Art von Schleife in jede andere Art von Schleife umformen können

5. Zusammenfassung

6. Expressions

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 interagieren 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 interagieren verschiedene Typen miteinander?

Generalitätsreihenfolge der Typen

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 interagieren verschiedene Typen miteinander?

Generalitätsreihenfolge der Typen

```
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 interagieren verschiedene Typen miteinander?

Generalitätsreihenfolge der Typen

```
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 interagieren verschiedene Typen miteinander?

Generalitätsreihenfolge der Typen

```
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`	{`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

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*anything` 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. Determine the values of the expressions and explain how these values are obtained. Assume that initially `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

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)

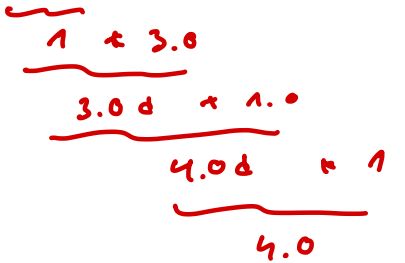
```
3.0 + 3 - 4 + 5

((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

Loop Correctness

Kann man beim Laufen dieses Programms den Unterschied zwischen den Ausgaben dieser drei Schleifen feststellen? Wenn ja, wie? Nehmen wir an, dass `n` eine Variable vom Typ `unsigned int` ist, deren Wert eingegeben wird.

Loop Correctness

Kann man beim Laufen dieses Programms den Unterschied zwischen den Ausgaben dieser drei Schleifen feststellen? Wenn ja, wie? Nehmen wir an, dass `n` eine Variable vom Typ **`unsigned int`** ist, deren Wert eingegeben wird.

```
//////////////////////////
unsigned int n;
std::cin >> n;
unsigned int i;

// loop 1 /////////////////
for (i = 1; i <= n; ++i) {
  std::cout << i << "\n";
}
```

```
// loop 2 /////////////////
i = 0;
while (i < n) {
    std::cout << ++i << "\n";
}
// loop 3 /////////////////
i = 1;
do {
  std::cout << i++ << "\n";
} while (i <= n);
```

Schleifenkorrektheit - Lösung

Lösung

Es gibt die folgenden Unterschiede:

- Im Gegensatz zu den Schleifen 1 und 2 wird in Schleife 3 für die Eingabe `n == 0` `1` ausgegeben, da die Anweisung in einer **`do`**-Schleife immer einmal ausgeführt wird, bevor die Bedingung geprüft wird
- Wenn n die grösstmögliche ganze Zahl ist, dann können die Schleifen 1 und 3 unendlich sein, weil die Bedingung `i <= n` für alle möglichen `i` wahr sein wird

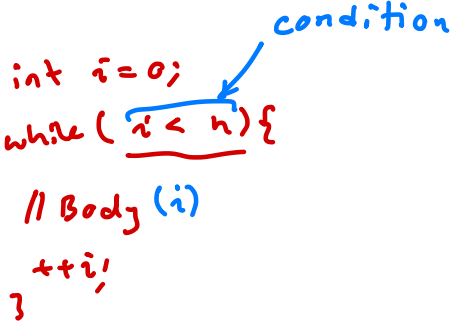
Fragen/Unklarheiten?

7. Loops

for → while

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

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



for $\rightarrow$ 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?

8. Reihen Berechnen

Von Summe zur Schleife

Mathematische Summen können zu Loops umgewandelt werden

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

Von Summe zur Schleife

Mathematische Summen können zu Loops umgewandelt werden

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

Wird zu

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

Aufwärmübungen

Betrachten wir die Formel

$$\frac{1}{n!}$$

Aufwärmübungen

Betrachten wir die Formel

$$\frac{1}{n!} = \frac{1}{1} \cdot \frac{1}{2} \cdot \ldots \cdot \frac{1}{n}$$

Aufwärmübungen

Betrachten wir die Formel

$$\frac{1}{n!} = \frac{1}{1} \cdot \frac{1}{2} \cdot \ldots \cdot \frac{1}{n}$$

Wie könnte man dies als
("multiplikative") Reihe umsetzen?

$$\frac{1}{n!}$$

Aufwärmübungen

Betrachten wir die Formel

$$\frac{1}{n!} = \frac{1}{1} \cdot \frac{1}{2} \cdot \dots \cdot \frac{1}{n}$$

Wie könnte man dies als ("multiplikative") Reihe umsetzen?

$$\frac{1}{n!} = \prod_{i=1}^{n} \frac{1}{i}$$

Wie verwandeln wir dieses Stück Mathematik in ein Stück C++-Code?

Aufwärmübungen

Betrachten wir die Formel

$$\frac{1}{n!} = \frac{1}{1} \cdot \frac{1}{2} \cdot \ldots \cdot \frac{1}{n}$$

Wie könnte man dies als ("multiplikative") Reihe umsetzen?

$$\frac{1}{n!} = \prod_{i=1}^{n} \frac{1}{i}$$

Wie verwandeln wir dieses Stück Mathematik in ein Stück C++-Code?

```
int main(){

    int n;          // user input
    double result;  // main output

    [ your code goes here! ]

    std::cout << result
              << std::endl;

    return 0;
}
```

Aufwärmübungen - Lösungsbeispiel

```cpp
int main(){
    int n;
    double result = 1;
    int i = 1;

    std::cin >> n;

    while(i <= n){
        result = result/i;
        i++;
    }

    std::cout << result << std::endl;

    return 0;
}
```

Von Reihe zur Schleife

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 zur Schleife

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

Von Reihe zur Schleife

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
- Mit Person neben euch versuchen in **code** expert zu implementieren

Von Reihe zur Schleife - Lösung

Von Reihe zur Schleife - Lösung

```cpp
#include <iostream>

int main () {

  double x;
  std::cin >> x;

  double numtor = x;
  double denomtor = 1;

  double sum = x;
  double term;
  double term_abs;
  int n = 1;
```

```cpp
do {
    numtor *= -(x * x);
    denomtor *= (2 * n) * (2 * n + 1);
    term = numtor / denomtor;
    sum += term;
    if (term < 0) {
      term_abs = -term;
    } else {
      term_abs = term;
    }
    ++n;
  } while (term_abs > 0.000001);

  std::cout << sum << std::endl;
  return 0;
}
```

Fragen/Unklarheiten?

9. Scopes

Scopes

Frage

In der Vorlesung von dieser Woche wurde ein neues Konzept eingeführt: "Variable Scopes". Was sind "Variable Scopes" und warum brauchen wir sie?

Scopes

Frage

In der Vorlesung von dieser Woche wurde ein neues Konzept eingeführt: "Variable Scopes". Was sind "Variable Scopes" und warum brauchen wir sie?

Antwort

Scopes definieren die Codesegmente unseres Programms, in denen eine Variable (l-Wert) existiert. Der Scope einer Variablen beginnt an der Stelle, an der sie definiert wurde, und endet am Ende des Blocks, in dem sie definiert wurde. Zum Beispiel:

```
if (x < 7){
    int a = 8;          // <-- a's variable scope BEGINS here!
    std::cout << a;     // Fine, prints 8.
}                       // <-- a's variable scope ENDS here!
std::cout << a;         // Compiler error, a does not exist.
```

Bug?

Ein vermeindlicher Weg, den Fehler zu beheben, wäre:

```
int a = 2;

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

std::cout << a;
```

Frage
Was wird dieses Programm wiedergeben wenn `x==2`?

Bug?

Ein vermeindlicher Weg, den Fehler zu beheben, wäre:

```
int a = 2;

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

std::cout << a;
```

Frage
Was wird dieses Programm wiedergeben wenn `x==2`?

Antwort
Es wird 82 wiedergegeben.

Bug?

Ein vermeindlicher Weg, den Fehler zu beheben, wäre:

```
int a = 2;

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

std::cout << a;
```

Frage
Was wird dieses Programm wiedergeben wenn `x==2`?

Antwort
Es wird 82 wiedergegeben.
Warum? Siehe Program Tracing Guide

Bug?

Frage

Was ist der Scope von `sum`, `i` und `a` im folgenden Beispiel?

```
int sum = 0;

for (int i = 0; i < 5; ++i) {
    int a;
    std::cin >> a;
    sum += a;
}
```

Antwort

Bug?

Frage

Was ist der Scope von `sum`, `i` und `a` im folgenden Beispiel?

```
int sum = 0;

for (int i = 0; i < 5; ++i) {
    int a;
    std::cin >> a;
    sum += a;
}
```

Antwort

`sum` (Mindestens) das gesamte Snippet

Bug?

Frage

Was ist der Scope von `sum`, `i` und `a` im folgenden Beispiel?

```
int sum = 0;

for (int i = 0; i < 5; ++i) {
    int a;
    std::cin >> a;
    sum += a;
}
```

Antwort

`sum` (Mindestens) das gesamte Snippet

`i` Die gesamte `for`-Schleife

Bug?

Frage

Was ist der Scope von `sum`, `i` und `a` im folgenden Beispiel?

```
int sum = 0;

for (int i = 0; i < 5; ++i) {
    int a;
    std::cin >> a;
    sum += a;
}
```

Antwort

`sum` (Mindestens) das gesamte Snippet

`i` Die gesamte `for`-Schleife

`a` Eine Schleifeniteration. Anders gesagt: Am Anfang des Schleifenkörpers hat `a` *nicht* garantiert den Wert, den es am Ende des Schleifenkörpers in der vorherigen Schleifeniteration hatte.

Fragen/Unklarheiten?

10. Alte Prüfungsaufgabe

By the way…

By the way…

Falls ihr selbst durch alte Prüfungen gehen wollt, sind sie Hier unter "Informatik I, Computer Science Introduction, C++" auffindbar.

Eure Prüfung wird denen mit "CSE" wahrscheinlich am ähnlichsten sein.

Das Passwort sollte überall `Informatik` sein.

11. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche noch!