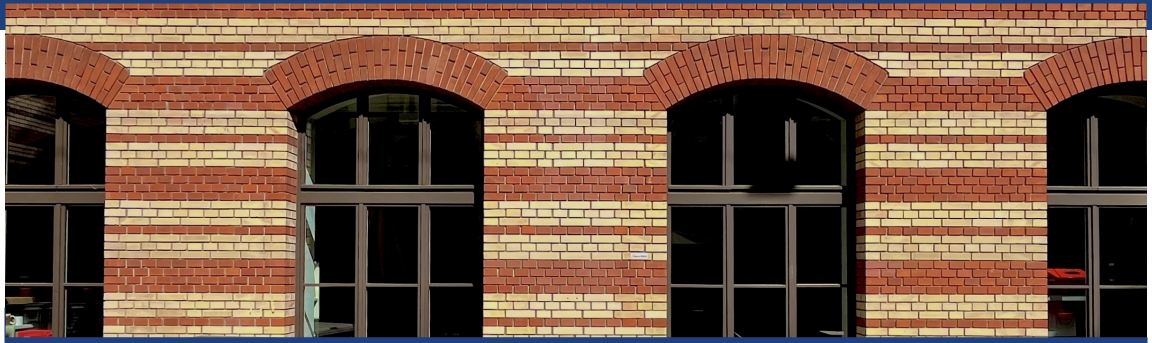

Informatik

Schleifen, Sichtbarkeitsbereiche (Scopes)

Adel Gavranović — ETH Zürich — 2025

Überblick

Lernziele
Schleifen
Berechnung von Taylor-Reihen
Sichtbarkeitsbereiche (Scopes)

n.ethz.ch/~agavranovic

Material

Webpage

Mail

1. Follow-up

Follow-up aus letzter Übungsstunde

- Bezüglich der Punktevergabe von textbasierten Aufgaben:
  - Es scheint von Aufgabe zu Aufgabe zu variieren; Mal prüfe ich manuell, mal übernimmt der Autograder
- Entschuldigt nochmal die Verwirrung bei "Strange Sum", die Musterlösungen sind beide korrekt
- Diejenige für Largest Power ebenfalls

2. Feedback zu **code** expert

Allgemeines bezüglich **code** expert

Fragen bezüglich **code** expert eurerseits?

3. Lernziele

Ziele

- ☐ Fähig sein, mathematische Reihen in C++ zu implementieren
- ☐ `for`-, `while` und `do-while`-Schleifen tracen können
- ☐ Jede Art von Schleife in jede andere Art von Schleife umformen können

4. Zusammenfassung

Getting on the same page

- Was wurde diese Woche behandelt?

5. Schleifen

Schleifenkorrektheit

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 `int` ist, deren Wert eingegeben wird.

```
int n;
std::cin >> n;
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` eine `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?

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
}
```

```
// SOLUTION

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

do-while $\rightarrow$ for

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

do{
    // BODY
}while(condition)
```

```
// SOLUTION

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

Fragen/Unklarheiten?

6. Berechnung von Taylor-Reihen

Von Summe zur Schleife

Mathematische Summen können zu Schleifen umgewandelt werden

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

wird zu

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

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

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?

```
int main(){

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


    // your code
    // goes here!


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

    return 0;
}
```

Aufwärmübungen – Lösungsbeispiel

```
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.

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

Aufgabe

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

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;

    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?

7. Sichtbarkeitsbereiche (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?

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`?

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

`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?

8. Outro

Allgemeine Fragen?

Bis nächstes Mal

Schöne Woche noch!