ETH Zürich

Informatik I
Übungsstunde 03

Johannes Odenthal

ETZ J 91
cs.jodenthal.com

Hausaufgabenfragen

Welche Fragen sind offen geblieben?

Scopes

> Scopes: siehe C++-Guide auf der Kurswebseite.

For Loops

Die for-Schleife

```
for (init; condition; update) {
  // Schleifenrumpf
}
```

> Update: nach dem Rumpf. Dann die Bedingung erneut prüfen.

Eine leere Bedingung gilt als `true`.

Code Tracing

```cpp
int main() {
  int sum = 0;
  for (int i = 1; i <= 3;
       ++i) {
    sum += i;
  }
  std::cout << sum << "\n";
  return 0;
}
```

`i` ist jetzt ausserhalb seines Scopes.

Schritt	Auswertung	sum	i
Init	`sum = 0`	0	—
Init	`i = 1`	0	1
Bedingung	`1 <= 3` : **true**	0	1
Rumpf	`0 + 1 = 1`	1	1
Update	`++i`	1	2
Bedingung	`2 <= 3` : **true**	1	2
Rumpf	`1 + 2 = 3`	3	2
Update	`++i`	3	3
Bedingung	`3 <= 3` : **true**	3	3
Rumpf	`3 + 3 = 6`	6	3
Update	`++i`	6	4
Bedingung	`4 <= 3` : **false**	6	4
Ausgabe	`6` + Newline	6	—

Strange Sum

CodeExpert

Largest Power

CodeExpert

Fragen?

Debugging

Live Demo

`non_terminating.cpp`

Fragen?

Repetition

Repetition

`5 < 4 < 1`

`true > false`

Notiere die Zwischenschritte.

Repetition · Auswertung

`5 < 4 < 1`

`(5 < 4) < 1`

`false < 1`

`0 < 1`

`true`

`true > false`

`true > 0`

`1 > 0`

`true`

> Bereichstests brauchen zwei Vergleiche mit `&&`.

Fragen?

Floating Point Numbers

Floating Point: Gleichheit nach Rundung

```cpp
float a = 2.0f;
if (std::sqrt(2.0f) * std::sqrt(2.0f) == a) {
  std::cout << "A";
}
```

Was ist die Ausgabe?

Bedingung `false`. Keine Ausgabe.

Mathematisch gilt $\left(\sqrt{2}\right)^2 = 2$. $\sqrt{2}$ ist nicht exakt darstellbar; auch das Produkt wird gerundet.

> `==` nach unterschiedlichen Berechnungen oder Rundung vermeiden.

Floating Point: Gross + klein

```cpp
float a = 67108864.0f + 0.1f;
if (a > 67108864.0f) {
  std::cout << "A";
}
```

Was ist die Ausgabe?

`a` bleibt `67108864.0f`. Keine Ausgabe.

> Addition von Zahlen sehr unterschiedlicher Grösse vermeiden.

Rundung: Der kleine Beitrag fehlt

```
67108864    = 6.7108864  * 10^7
       0.1 = 0.00000001 * 10^7
```

Exakte Summe

```
67108864.1 = 6.71088641 * 10^7
```

Nach Rundung

```
67108864    = 6.7108864  * 10^7
```

Begrenzte signifikante Stellen · Dezimalskizze zur Intuition

Fragen?

Comparing Floating Point Numbers

Vergleichen mit Toleranz

Für `float` und `double`: kleine Abweichungen zulassen.

$$c > 0 \quad |\mathrm{fp1} - \mathrm{fp2}| < c$$

Mit $c = 0.001$:

$|10.0 - 12.0| = 2.0$; **nicht nahe beieinander.**

$|10.0 - 10.000013| \approx 0.000013$; **nahe beieinander.**

equals implementieren

```
1 // POST: true genau dann, wenn |x - y| < tol
2 bool equals(double x, double y, double tol) {
3   // ...
4 }
```

Funktionsaufruf

```
equals(1.0, 1.0005, 0.001)
```

Rückgabewert

```
true
```

Wie berechnest du den Betrag der Differenz?

equals: Eine Lösung

```
1 bool equals(double x, double y, double tol) {
2   double diff = x - y;
3   if (diff < 0) {
4     diff *= -1;
5   }
6   return diff < tol;
7 }
```

> Eine feste absolute Toleranz kann für grosse Werte ungeeignet sein.

Fragen?

cs.jodenthal.com

Johannes Odenthal

Optional & Nachschlagen

- Debugging in CodeExpert
- Even Numbers
- Befehlsübersicht Woche 03

Debugging in CodeExpert

CodeExpert

Even Numbers

CodeExpert

Nachschlagen: for

```cpp
int n;
std::cin >> n;
// Summe 1 + 2 + 3 + ... + n
int sum = 0;
for (int i = 1; i <= n; ++i) {
  sum += i;
}
std::cout << "1 + 2 + ... + n = " << sum << "\n";
```

Eine leere Bedingung gilt als `true`.

Nachschlagen: if–else

```cpp
int a;
int b;
std::cin >> a >> b;
if (a < b) {
  std::cout << 3 << "\n";
} else {
  std::cout << 8 << "\n";
}
```

`else` ist optional. `{ ... }` gruppiert Anweisungen.

Nachschlagen: Scope

```cpp
int main() {
  int a;
  std::cin >> a;
  if (a < 4) {
    std::cout << a << " "; // a im inneren Block gültig
    int b = 18;
    std::cout << b << " "; // b hier gültig
  } else {
    std::cout << b << " "; // Fehler: vor Deklaration
    int b = 11;
  }
  std::cout << a << " ";   // a weiterhin gültig
  std::cout << b << "\n"; // Fehler: ausserhalb des Scopes
  return 0;
}
```

Scope: ab Deklaration bis zum Ende des Blocks.