Programmierung 1

Schleifen

Dr. Monika Schak

Woche 3

05. November 2025

Wiederholung

- Drei Typen von Fallunterscheidungen:
  - If-else:
    `if (<Bedingung>) { ... } [else if (...) { ... }]* [else { ... }]`
    mit beliebig vielen else if-Blöcken und einem optionalen else-Block.
  - Switch-case:
    `switch(<Ausdruck>) { case <Term>: ... break; ...; default: ...;}`
  - Ternerärer Operator: `<Bedingung> ? <Anweisung1> : <Anweisung2>`
- Boolesche Operatoren: UND (`&&`), ODER (`||`), NICHT (`!`)
- Auswertungsreihenfolge: Klammern $\rightarrow$ NICHT $\rightarrow$ Vergleiche $\rightarrow$ UND $\rightarrow$ ODER

Zählschleifen

- Schlüsselwort `for`
- Danach in Klammern drei Bestandteile, je durch Semikolon getrennt
  - Initialisierung (wird zuerst ausgeführt): `int count = 1`
  - Abbruchbedingung (wann wird beendet?: tue etwas, solange `count <= 500`
  - Schritt (wird direkt nach jedem Durchlauf ausgeführt): `count++`
- Die Variable `count` wird Zähler oder Schleifenvariable genannt.
- Danach folgt ein Block, der sogenannte Schleifenrumpf
  - Bei Blöcken (auch einzeiligen) für bessere Lesbarkeit Einrücken nicht vergessen
  - Wenn obige Bedingung nicht mehr erfüllt ist, wird das Programm nach dem Rumpf fortgeführt

```
for (int count = 1; count <= 500; count = count + 1) {
  printf("Repeat this.\n");
}
```

Inkrement/Dekrement

- Inkrement: `i++` bzw. `++i`: Wert der Variable wird um 1 erhöht
- Dekrement: `i--` bzw. `--i`: Wert der Variable wird um 1 verringert
- Postfix-Schreibweise (`i++` bzw. `i--`) verringert/erhöht den Wert der Variable, gibt aber noch den alten Wert an den aktuellen Ausdruck weiter
  Analog: `i = i + 1` bzw. `i = i - 1`
- Präfix-Schreibweise (`++i` bzw. `--i`) verringert/erhöht den Wert der Variable und gibt erst danach den Wert an den aktuellen Ausdruck weiter

Inkrement/Dekrement: Beispiel

Was ist die Ausgabe der folgenden Sequenz?

```
int i = 1;
printf("i = %d\n", i);
i++;
printf("i = %d\n", i);
printf("i = %d\n", i++);
printf("i = %d\n", i);
printf("i = %d\n", ++i);
printf("i = %d\n", i);
```

Zählschleifen: Akkumulieren

- Typische Anwendung: Akkumulieren (Ansammeln) von Werten
- Verwendet eine sog. Akkumulatorvariable zum Sammeln (`smm`, `res`, o.ä.)
- Beispiele:
  - Alle geraden Zahlen von 2 bis 10 addieren
  - Alle Quadratzahlen von 1^2 bis 20^2 addieren

Zählschleifen: Suche

- Typische Anwendung: Aus einer Reihe von Zahlen eine oder mehrere mit einer bestimmten Eigenschaft herausfinden
- Häufig ist im Schleifenrumpf eine `if`-Anweisung, die auf die gesuchte Eigenschaft prüft
- Beispiele:
  - Kleinste Zahl, die durch 12, 27 und 44 teilbar ist
  - Lösung der Gleichung $x^2 - 34x + 289 = 0$ durch Ausprobieren finden

```
for (int i = __; ________; ____) {
  if (_____________________________________________) {
    printf("Gefunden: %d\n", i);
  }
}
```

Weitere Kontrollmöglichkeiten

- Schleifen abbrechen mit `break`
  - Die Anweisung `break` beendet eine Schleife sofort, das Programm wird nach der Schleife weiter ausgeführt
  - Typischer Einsatz: unbestimmtes Warten auf Ereignisse (z.B. ungültige Eingabenm, spezielle Werte, etc.)
- Einzelne Iterationen abbrechen mit `continue`
  - Die Anweisung `continue` erzwingt direkt den nächsten Schleifendurchlauf
  - Typischer Einsatz: bestimmte Werte in der Schleife überspringen
- Funktioniert beides bei Zählschleifen, kopf- und fußgesteuerten Schleifen
- Vorsicht: Verwendung kann Lesbarkeit von Programmen beeinträchtigen

Übung

- Lesen Sie einen Integerwert `x` ein, berechnen Sie den Teiler von `x` und geben Sie dann die Summe aller Teiler von `x` aus.
- Geben Sie folgendes Muster auf der Konsole aus (Breite 20, Höhe 30), indem Sie zwei for-Schleifen verwenden:

```
XXXXX....X
XXXXX....X
...
XXXXX....X
```

Kopfgesteuerte Schleifen

- Schlüsselwort `while`
- Dann folgt (in Klammern) die Abbruchbedingung (logischer Ausdruck)
- Danach folgt ein Block, der sog. Schleifenrumpf
  - Geschweifte Klammern `{ ... }` und richtiges Einrücken nicht vergessen!
  - Wenn Bedingung erfüllt ist, wird Rumpf ausgeführt und Abbruchbedingung erneut ausgewertet

```
int i = 0;
while (i < 20) {
  printf("%d\n", i);
  i = i + 2;
}
```

Übung

- Schreiben Sie ein Programm, das eine Integervariable `x` einliest und die Quersumme von `x` mit Hilfe einer `while`-Schleife berechnet.

for vs while-Schleife

- Kopfgesteuerte Schleifen und Zählschleifen sind gleich ausdrucksstark

```
for (int i = 0; i < 20; i = i + 2) {
  printf("%d\n.", i);
}
```

```
int i = 0;
while (i < 20) {
  printf("%d.\n", i);
  i = i + 2;
}
```

for vs while-Schleife

- `while`-Schleifen verwendet man v.a. wenn die Anzahl der Durchläufe (sog. Iterationen) nicht im Voraus bekannt ist → Wiederhole etwas so lange, bis etwas bestimmtes passiert.
- `for`-Schleifen verwendet man v.a. wenn die Anzahl der Durchläufe vorher bekannt ist → Tue etwas x-mal.
- Achtung: Bei allen Schleifenarten kann es passieren, dass der Rumpf niemals oder endlos (Endlosschleife, i.d.R. ein Fehler) ausgeführt wird
  Beispiel Endlosschleife:

```
while (true) {
  /* Endlosschleife */
}
```

Fußgesteuerte Schleifen

- Schlüsselwort `do`
- Danach folgt ein Block, der sog. Schleifenrumpf
- Geschweifte Klammern und richtiges Einrücken nicht vergessen!
- Dann folgt ein Schlüsselwort `while` und die Abbruchbedingung (in Klammern)
- Beendet wird mit einem Semikolon
- Im Unterschied zur `while`-Schleife wird die Bedingung erst am Ende geprüft, deshalb wird eine `do-while`-Schleife immer mindestens einmal durchlaufen!

```
int i = 0;
do {
  printf("%d\n", i);
  i = i + 2;
} while (i < 20);
```

Übung

Was wird hier jeweils ausgegeben? Was könnte ein Problem sein?

- ```
  int x = 0;
  while (x < 10); {
    printf("Der Wert von x beträgt %d\n", x);
    x++;
  }
  ```

- ```
  int zahl1 = 0, zahl2 = 0;
  while ((zahl1++ < 5) || (zahl2++ < 5)) {
    // do something
  }
  printf("Wert von zahl1: %d, zahl2: %d\n", zahl1, zahl2);
  ```