Kontrollstrukturen

Einführung in die Programmierung

Michael Felderer

Institut für Informatik, Universität Innsbruck

Allgemein

- Bisher wurden die einzelnen Zeilen eines Programms sequentiell abgearbeitet.
- Sehr oft will man aber bestimmte Anweisungen auswählen oder wiederholen.
- Kontrollstrukturen definieren die Reihenfolge, in der Anweisungen durchgeführt werden.

Anweisungen und Blöcke

- Anweisung
  - Aus einem Ausdruck (z.B. `x * 3`) wird eine Anweisung, wenn ein Semikolon folgt.
  - Beispiele
    - `x * 3;`
    - `i++;`
    - `printf(…);`
- Block
  - Alles zwischen { und } wird zu einem Block zusammengefasst.
  - Ein Block ist syntaktisch äquivalent zu einer einzelnen Anweisung (compound statement).
    - Blöcke kann man auch ineinander schachteln.
  - Nach der rechten Klammer steht **kein** Semikolon!
  - Beispiele folgen noch.

Verzweigungen

- Mit Verzweigungen kann man den Ablauf des Programms beeinflussen, in dem man logische Bedingungen definiert und damit entscheidet, an welcher Stelle das Programms fortgesetzt werden soll.
- In C gibt es 2 solcher Verzweigungen:
  - `if`
  - `switch`
- Daneben gibt es den ternären Bedingungsoperator.

if-else

- Allgemeine Form

```
if (Ausdruck)
    Anweisung_1
else
    Anweisung_2
```

- Ablauf
  - Der Ausdruck in Klammern wird berechnet.
  - Hat der Ausdruck einen von 0 verschiedenen Wert (Ausdruck trifft zu), dann wird `Anweisung_1` ausgeführt.
  - Hat der Ausdruck den Wert 0, dann wird `Anweisung_2` ausgeführt.
- Zu beachten
  - Der `else`-Zweig ist optional, d.h. dieser Zweig kann weggelassen werden.
  - Bei mehreren Anweisungen muss ein Block verwendet werden (siehe Beispiel).
  - `if`-Anweisungen können verschachtelt werden.

Beispiel (einfach, unvollständig)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("Bitte ein Zeichen eingeben: \n");
    int ival = getchar();
    if (ival == 'a') {
        printf("a wurde eingegeben!\n");
    } else {
        printf("Ein anderes Zeichen wurde eingegeben!\n");
    }
    printf("Ausserhalb der if-Verzweigung\n");
    return EXIT_SUCCESS;
}
```

Ausgabe (zid-gpl):

```
Bitte ein Zeichen eingeben:
a
a wurde eingegeben!
Ausserhalb der if-Verzweigung
```

Beispiel (komplexer)

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void) {
    double a, b, c;
    printf("Bitte a, b und c eingeben: \n");
    scanf("%lf %lf %lf", &a, &b, &c);
    double d = b * b - 4 * a * c;
    if (d < 0)
        printf("Keine Lösung\n");
    else {
        if (d == 0)
            printf("1 Lösung: %lf\n", -b / (2 * a));
        else {
            double sqrt_d = sqrt(d);
            printf("2 Lösungen: %lf und %lf\n",
                   (-b + sqrt_d) / (2 * a),
                   (-b - sqrt_d) / (2 * a)    );
        }
    }
    return EXIT_SUCCESS;
}
```

Achtung:
Unter Linux muss beim gcc-Aufruf noch **`-lm`** am Ende angegeben werden (für `math.h`).

Ausgabe:

```
Bitte a, b und c eingeben:
1.5 4 2.25
2 Lösungen: -0.806287 und -1.860380
```

Achtung:
Hier treten zwei typische numerische Fallstricke von Fließkommarechnungen auf!

else-if

- Mehrfache Alternative
- Allgemeine Form:

```
if (Ausdruck_1)
    Anweisung_1
else if (Ausdruck_2)
    Anweisung_2
…
else if (Ausdruck_n)
    Anweisung_n
else
    Anweisung_else              /* optional */
```

- Ablauf:
  - Ein Vergleich wird nach dem anderen durchgeführt.
  - Bei der ersten Bedingung, die wahr ist, wird die zugehörige Anweisung abgearbeitet und die Mehrfach-Selektion abgebrochen.

Beispiel (else-if, mit Bereichsabfragen)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int c = getchar();
    if (c >= '0' && c <= '9')
        printf("Sie haben eine Ziffer eingegeben!\n");
    else if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        printf("Sie haben einen Buchstaben eingegeben!\n");
    else if (c == ' ' || c == '\n' || c == '\t')
        printf("Sie haben ein Whitespace-Zeichen eingegeben!\n");
    else
        printf("Sie haben ein anderes Zeichen eingegeben!\n");
    return EXIT_SUCCESS;
}
```

Beispiel (else-if, mit Bibliotheksfunktionen)

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void) {
    int c = getchar();
    if (isdigit(c))
        printf("Sie haben eine Ziffer eingegeben!\n");
    else if (isalpha(c))
        printf("Sie haben einen Buchstaben eingegeben!\n");
    else if (isspace(c))
        printf("Sie haben ein Whitespace-Zeichen eingegeben!\n");
    else
        printf("Sie haben ein anderes Zeichen eingegeben!\n");
    return EXIT_SUCCESS;
}
```

Dangling else

- Bekanntes Problem bei der Programmierung
- Ein `else`-Zweig bei zwei `if`-Anweisungen
  - Wohin gehört der `else`-Zweig?
- Beispiel (schlecht formatiert)

```
if (counter < 5)
    if (counter % 2 == 0)
        printf("HERE1");
else
    printf("HERE2");
```

- Auflösung durch Compiler
  - `else`-Zweig gehört zum textuell letzten freien `if` im selben Block!
  - `counter` mit Wert 7 führt zu keiner Ausgabe.
  - `counter` mit Wert 3 führt zur Ausgabe *"HERE2"*.
  - Erzeugt Warnings!
    - In solchen Fällen die Blockstruktur immer mit geschweiften Klammern klarstellen!

Interaktive Aufgabe

- Das nachfolgende Programm überprüft die Eingabe von zwei Ganzzahlen über `scanf`. Allerdings gibt das Programm immer zurück, dass es sich nicht um zwei Ganzzahlen handelt, auch wenn es zwei Ganzzahlen waren. Was wurde hier falsch gemacht?

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {

  int chck, val1, val2;
  printf("Zwei Ganzzahlen eingeben: ");
  chck = scanf("%d %d", &val1, &val2);
  if (chck != 2); {
    printf("Das waren nicht zwei Ganzzahlen");
  }
  return EXIT_SUCCESS;
}
```

switch

- Allgemeine Form:

```
switch (Ausdruck_1) {
    case k1:
          Anweisungen_1
          break;
    case k2:
          Anweisungen_2
          break;
    …
    case kn:
          Anweisungen_n
          break;
    default:
          Anweisungen_default
          break;      /* Nicht notwendig, aber hilfreich */
}
```

switch – case-Marken (1)

- Jeder Alternative geht eine, oder eine Reihe, von `case`-Marken mit Integer-Konstanten `k1`, …, `kn` oder konstanten Integer-Ausdrücken voraus.
- Hat der Ausdruck der `switch`-Anweisung den gleichen Wert wie einer der konstanten Ausdrücke der `case`-Marken, dann wird die Ausführung des Programms mit der Anweisung (den Anweisungen) hinter dieser `case`-Marke weitergeführt.
- Stimmt keiner der konstanten Ausdrücke mit dem `switch`-Ausdruck überein:
  - Es wird zu `default` gesprungen, falls vorhanden (`default` ist optional).
  - Wenn kein `default` vorhanden ist, dann wird die Anweisung nach der `switch`-Anweisung ausgeführt.

switch – case-Marken (2)

- Nach der Abarbeitung der Anweisungen wird die `switch`-Anweisung verlassen, falls ein `break` eingefügt wurde.
- Fehlt die `break`-Anweisung, dann werden alle nachfolgenden Anweisungen bis zum nächsten `break` oder dem Ende der `switch`-Anweisung abgearbeitet („fall through“).
  - Beispiele folgen noch!

Beispiel (switch)

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void) {
    int c = getchar();
    if (isdigit(c))
        switch(c) {
            case '0': printf("Sie haben 0 eingegeben!\n"); break;
            case '1': printf("Sie haben 1 eingegeben!\n"); break;
            case '2': printf("Sie haben 2 eingegeben!\n"); break;
            case '3':
            case '4': printf("Sie haben 3 oder 4 eingegeben!\n"); break;
            default:  printf("Größer vier!\n"); break;
        }
    else
        printf("Bitte geben Sie eine Zahl an!\n");
    return EXIT_SUCCESS;
}
```

Interaktive Aufgabe

- Formulieren Sie die nachfolgende if-Anweisung in eine switch-Anweisung um!

```
printf("-1- Option 1\n");
printf("-2- Option 2\n");
printf("Funktion auswählen: \n");
scanf("%d", &option);

if(option == 1) {
  printf("Option 1 gewählt!\n");
}
else if (option == 2) {
  printf("Option 2 gewählt!\n");
}
else {
  printf("Falsche Eingabe!\n");
}
```

switch und else-if im Vergleich

- `switch` prüft nur auf die Gleichheit von Werten im Gegensatz zur `if-else`-Anweisung, bei der logische Ausdrücke ausgewertet werden.
- Der Bewertungsausdruck der `switch`-Anweisung kann nur ganzzahlige, konstante Werte oder Zeichen verarbeiten.
  - Der generierte Code ist effizienter als äquivalente `if-else if`-Ketten.
- `switch` ist übersichtlicher und leichter zu erweitern.

Iteration – while-Schleife

- Allgemeine Form

```
while (Ausdruck)
    Anweisung
```

- Ablauf
  - Zunächst wird der Ausdruck ausgewertet.
  - Ist der Ausdruck von 0 verschieden, so wird die nachfolgende Anweisung ausgeführt.
  - Anschließend wird wieder der Ausdruck ausgewertet.
- Wichtig
  - Beliebiger Teil des Ausdrucks muss im Schleifenrumpf (`Anweisung`) verändert werden (ansonsten erhält man eine Endlosschleife).
  - Bei mehreren Anweisungen muss ein Block verwendet werden.

Beispiel (while-Schleife)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    long i = 2;
    printf("N \tN^2 \tN^3 \n");
    while (i <= 1024) {
        printf("%ld", i);
        printf("%c", '\t');
        printf("%ld", i * i);
        printf("%c", '\t');
        printf("%ld", i * i * i);
        printf("\n");
        i *= 2;
    }
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
N       N^2     N^3
2       4       8
4       16      64
8       64      512
16      256     4096
32      1024    32768
64      4096    262144
128     16384   2097152
256     65536   16777216
512     262144  134217728
1024    1048576 1073741824
```

Interaktive Aufgabe

- Das nachfolgende Codefragment soll die Zahlen von 0 bis 9 untereinander ausgeben. Korrigieren Sie die Fehler!

```
float ival=0;
while ( ival > 10 )
  printf("%d\n",ival);
  ival++;
```

for-Schleife (als Zählschleife)

- Allgemeine Form

```
for (Ausdruck_1; Ausdruck_2; Ausdruck_3)
    Anweisung
```

- Ablauf in 3 Schritten (typischer Ablauf)
  - Initialisierung einer Laufvariable, welche die Anzahl der Schleifendurchläufe zählt (`Ausdruck_1`).
    - Die Laufvariable kann auch vom Typ `float` oder `double` sein.
      - **ABER: Nicht sicher**, da Gleichheitsprüfung problematisch sein kann!
    - Hier kann auch ein beliebiger Ausdruck stehen.
  - Prüfen der Schleifenbedingung im `Ausdruck_2`.
  - Gegebenenfalls Ausführung der Anweisung und Erhöhen des Wertes der Laufvariable im `Ausdruck_3` (falls kein Abbruch erfolgte).
    - Hier kann auch ein beliebiger Ausdruck stehen.
    - Danach wird wieder die Schleifenbedingung überprüft.

Beispiel (for-Schleife)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    long i;
    printf("N \tN^2 \tN^3 \n");
    for (i = 2; i <= 1024; i *= 2) {
        printf("%ld", i);
        printf("%c", '\t');
        printf("%ld", i * i);
        printf("%c", '\t');
        printf("%ld", i * i * i);
        printf("\n");
    }
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
N       N^2     N^3
2       4       8
4       16      64
8       64      512
16      256     4096
32      1024    32768
64      4096    262144
128     16384   2097152
256     65536   16777216
512     262144  134217728
1024    1048576 1073741824
```

Interaktive Aufgabe

- Auf den ersten Blick scheint bei der nachfolgenden for-Schleife alles in Ordnung zu sein. Auch logisch liegt kein Fehler vor. Warum läuft diese Schleife trotzdem in einer Endlosschleife, und was können Sie dagegen tun?

```
float fval;
for(fval = 0.0f; fval != 1.0f; fval += 0.1f) {
  printf("%f\n",fval);
}
```

C99 (for-Schleifen)

- Im C99-Standard kann man die Schleifenvariablen auch gleich im ersten Ausdruck der for-Schleife deklarieren.
- Beispiel

```
for (int i = 10; i > 0; i--) {
    // Anweisungen
}
```

- Schleifenvariablen, die innerhalb der Schleife deklariert werden, stehen nach dem Ende der `for`-Schleife nicht mehr zur Verfügung!

Komma-Operator

- Es ist auch möglich, im Schleifenkopf einer for-Schleife mehrere Anweisungen mit dem Komma-Operator getrennt zu verwenden.
- Dies wird zum Beispiel dazu verwendet, im ersten Ausdruck mehrere Variablen mit einem Wert zu initialisieren und/oder im dritten Ausdruck mehrere Variablen zu reinitialisieren.

Beispiel (Komma-Operator)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    for (int i = 1, j = 10; i < j; i++, j--)
        printf("i=%d, j=%d\n", i, j);
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
i=1, j=10
i=2, j=9
i=3, j=8
i=4, j=7
i=5, j=6
```

Beispiel (geschachtelte for-Schleifen)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    for(int i = 1; i < 5; i++)
        for (int j = 1; j < 5; j++)
            printf("%d * %d = %d\n", i, j, i * j);
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
```

for-Schleife (Spezialfälle)

- Jeder der drei Ausdrücke kann entfallen.
  - Fehlt `Ausdruck_1`, dann entfällt die Initialisierung.
  - Fehlt `Ausdruck_2`, so ist die Bedingung immer wahr (Endlosschleife).
  - Fehlt `Ausdruck_3`, so fehlt die Erhöhung der Laufvariable.
- Abbruch einer Endlosschleife
  - Die Schleife kann dann durch eine `break`-Anweisung abgebrochen werden.
    - Sollte nur in Ausnahmefällen verwendet werden.
- Leere Anweisung
  - Soll keine Anweisung ausgeführt werden, dann muss trotzdem ein einzelner Strichpunkt oder ein leerer Block angegeben werden.
  - `for(;;)` oder `for(;1;)` ist immer wahr

while- oder for-Schleife?

- Wann ist eine `for`- und wann eine `while`-Schleife zu verwenden ?
  - `for`-Schleifen werden üblicherweise dann verwendet, wenn ein ganzer Bereich beginnend bei einem bestimmten Startwert bis zu einem vorgegebenen Endwert mit einer festen Schrittweite zu durchlaufen ist.
  - `while`-Schleifen dagegen werden verwendet, wenn lediglich ein Endkriterium gegeben ist, und es nicht vorhersagbar ist, wie oft die Schleifenanweisungen ausgeführt werden müssen, bis das Endkriterium zutrifft.
  - Generell wähle man die besser lesbare Variante.
- Jede `for`-Schleife lässt sich durch eine `while`-Schleife formulieren und umgekehrt!

`for`-Schleife	entsprechende `while`-Schleife

for (*ausdr1*; *ausdr2*; *ausdr3*)
  Schleifenanweisung
	ausdr1;
	while (*ausdr2*) {
	  Schleifenanweisung;
	  ausdr3;
	}

Interaktive Aufgabe

- Was wird durch das folgende Codefragment ausgegeben? Formulieren Sie die while-Schleife als for-Schleife!

```
int ival = 0;
int sum = 0;
while (ival <= 5) {
  if(ival % 2)
    sum += ival;
  ival++;
}
printf("%d\n",sum);
```

do-while-Schleife

- Allgemeine Form:

```
do {
    Anweisung
} while (Ausdruck);
```

- Ablauf
  - Zuerst wird die Anweisung der Schleife einmal ausgeführt (annehmende Schleife).
  - Danach wird der Ausdruck bewertet.
  - Die Anweisung und die Bewertung des Ausdrucks werden solange fortgeführt, solange der Ausdruck wahr ist (ungleich 0).
- Hinweis
  - Am Ende steht ein Strichpunkt, da hier das Ende der Anweisung erreicht wird.

Beispiel (do-while-Schleife)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    long i = 2;
    printf("N \tN^2 \tN^3 \n");
    do {
        printf("%ld", i);
        printf("%c", '\t');
        printf("%ld", i * i);
        printf("%c", '\t');
        printf("%ld", i * i * i);
        printf("\n");
        i *= 2;
    } while (i <= 1024);
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
N       N^2     N^3
2       4       8
4       16      64
8       64      512
16      256     4096
32      1024    32768
64      4096    262144
128     16384   2097152
256     65536   16777216
512     262144  134217728
1024    1048576 1073741824
```

Beispiel (Zeichen etc. zählen)

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void) {
    int c, nwhite, nletter, nother, ndigit;
    nwhite = nletter = nother = ndigit = 0;
    while ((c = getchar()) != '!') {
        if (isdigit(c))
            ndigit++;
        else if (isalpha(c))
            nletter++;
        else if (isspace(c))
            nwhite++;
        else
            nother++;
    }
    printf("Statistik für den eingegebenen Text: \n");
    printf("%d Ziffern\n", ndigit);
    printf("%d Buchstaben\n", nletter);
    printf("%d Zwischenräume\n", nwhite);
    printf("%d andere Zeichen\n", nother);
    return EXIT_SUCCESS;
}
```

Sprunganweisungen

- **`break`**
  - Wird verwendet, um eine `switch`-Anweisung **oder** eine Schleife zu beenden (zu verlassen).
  - Bei verschachtelten Schleifen beendet `break` immer nur die innerste Schleife, in der `break` benutzt wird!
- **`continue`**
  - Mit dieser Anweisung wird der Rest der Anweisungsfolge einer Schleife übersprungen und ein neuer Schleifendurchlauf gestartet.

Beispiel (break und continue)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (j == 2)
                break;
            printf("j *** %d\n", j);
        }
        if (i > 1)
            continue;
        printf("i --- %d\n", i);
    }
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
j *** 0
j *** 1
i --- 0
j *** 0
j *** 1
i --- 1
j *** 0
j *** 1
j *** 0
j *** 1
j *** 0
j *** 1
```

Interaktive Aufgabe

- Was gibt das Programm tatsächlich aus und warum hängt es in einer Endlosschleife fest. Nehmen Sie eine entsprechende Korrektur vor, damit das Programm die ungeraden Zahlen bis 20 ausgibt!

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

Endlosschleifen

- Schleifen, die nach ihrer Abarbeitung erneut ausgeführt werden und bei denen kein Abbruchkriterium definiert wurde
- Programme in Multitasking-Systemen oder auf einem Server laufen häufig beabsichtigt als Endlosschleife und warten auf eine Aktion (Polling-Verhalten)
- Beendigung von Endlosschleifen
  - Sprung mit `break` aus Endlosschleife
  - Beendigung einer Funktion in der Endlosschleife läuft mit `return`
  - Beendigung eines Programms mit `exit`
- Typische Erstellungsmöglichkeiten von Endlosschleifen
  - `while(1) {…}`
  - `for(;;) {…}` oder `for(;1;) {…}`

goto

- Diese Anweisung bewirkt einen Sprung zu der Programmzeile, die mit einer Sprungmarke markiert wurde.
- Nur in **ganz wenigen** Ausnahmefällen sinnvoll!
  - Jedes Programm kann auch ohne goto realisiert werden!
- „Theoretisches Beispiel“:

```
for (int i = 0; i < 5; i++)
    for (int j = 0; j < 5; j++) {
        for (int k = 0; k < 5; k++)
            if (i + j + k == 10)
                goto breakout;
        printf("%d %d\n", i, j);
    }
breakout:
printf("Nach goto!\n");
```

Ausgabe:

```
0 0
0 1
0 2
0 3
0 4
1 0
1 1
1 2
1 3
1 4
2 0
2 1
2 2
2 3
Nach goto!
```