Hochschule **Rosenheim**
University of Applied Sciences

Prozedurale Programmierung

Iterationen

Hochschule Rosenheim - University of Applied Sciences

WS 2018/19

Prof. Dr. F.J. Schmitt

Iterationen (1)

- Problem: Wiederholtes Ausführen von Anweisungen
- Beispiel: Ausgabe des Alphabets

```
printf(" A\n ");
printf(" B\n ");
printf(" C\n ");
.
.
.
printf(" Z\n ");
```

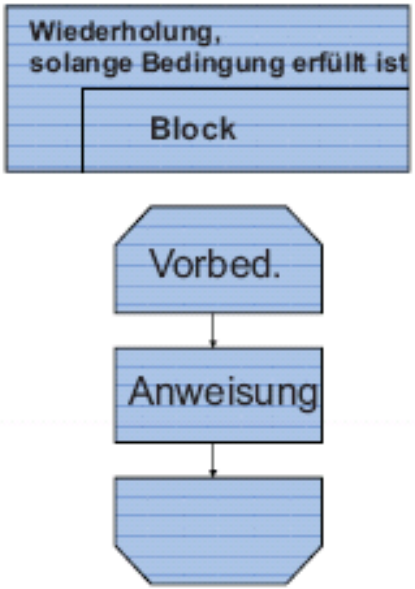
```
char c = 0;
for ( c = 'A'; c <= 'Z'; c = c + 1)
        printf(" %c\n ", c);
```

- Ist dies eine gute Lösung?
  - Lange, unübersichtliche Programme
  - Codeduplizierung

Iterationen (2)

- Schleifen werden durchlaufen, solange die Schleifenbedingung erfüllt ist
- Zwei Arten von Schleifen:

Vorprüfende Schleife

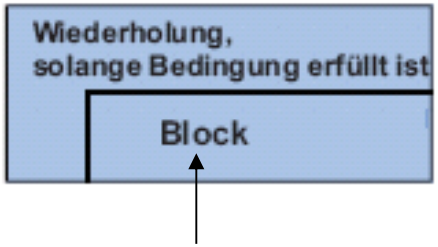


Nachprüfende Schleife

`while`-Anweisung (1)

- Vorprüfende Schleife
- Syntax:

```
while (Bedingung)
   Block
```



Schleifenrumpf

Anweisungsblock / Schleifenrumpf wird solange wiederholt bis Bedingung „falsch“ ist

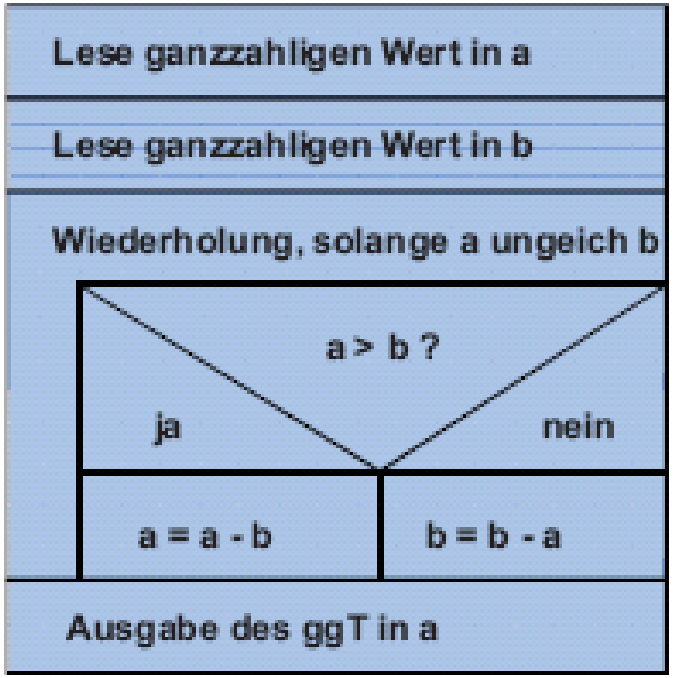
`while`-Anweisung (2)

- Abarbeitungsreihenfolge:
  1. Auswertung der Bedingung
     - ✓ Bedingung falsch (0)
       gesamte Schleife wird übersprungen und es wird mit der nächsten Anweisung nach der Schleife fortgesetzt
     - ✓ Bedingung wahr (ungleich 0)
       Durchlaufen des Schleifenrumpfs/Anweiungsblocks
  2. Erneutes Überprüfen der Bedingung

- Zu Schleifenbeginn wird immer erst die Bedingung ausgewertet

while-Anweisung (3)

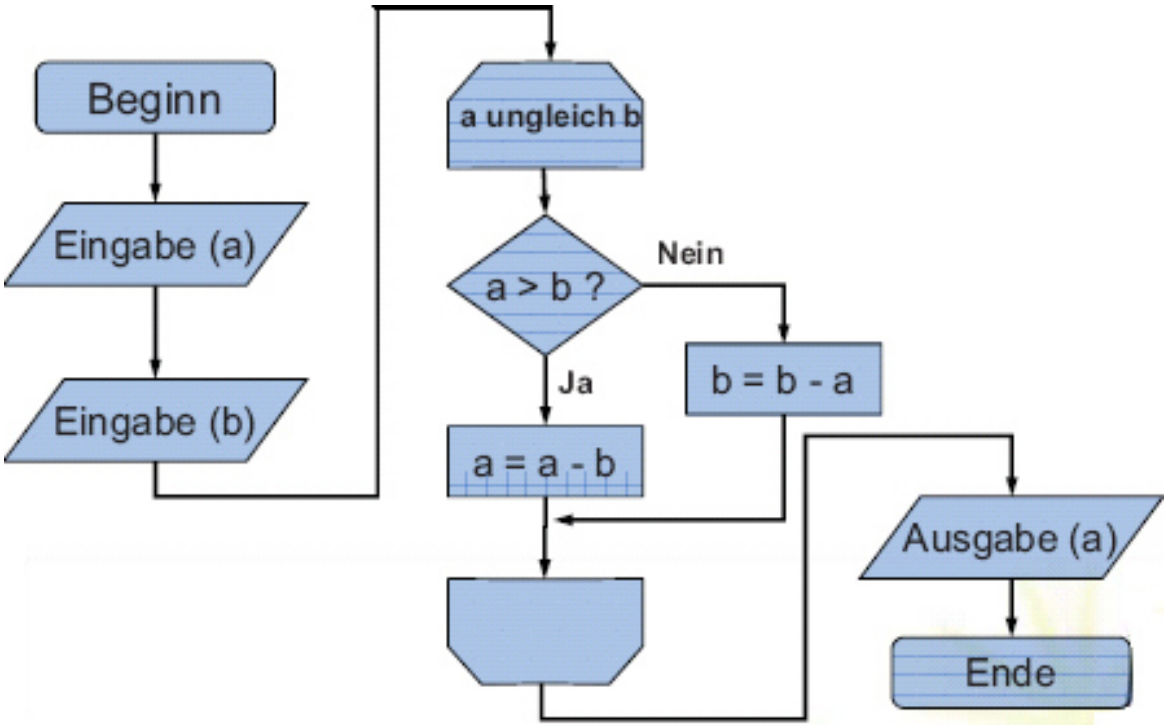
- Beispiel: Euklidischer Algorithmus zur Bestimmung des ggT zweier ganzer Zahlen




```
int main(void)
{
  int a,b;
  scanf("%d", &a);
  scanf("%d", &b);
  while (a != b)
   {
      if (a > b)
        a = a - b;
      else
        b = b - a;
   }
 printf("ggT = %d\n", a);
 return 0;
}
```

`while`-Anweisung (4)

- Programmablauf Euklidischer Algorithmus



Häufiger Fehler

- Falsches Semikolon

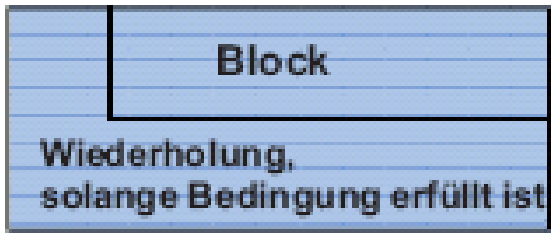
```
while (Bedingung);
   Block  oder Anweisung
```

- Auswirkung:
  - Semikolon stellt den Schleifenrumpf dar (Leeranweisung)
  - Ist Schleifenbedingung „wahr“ ⇒ es wird *nichts* gemacht
  - Erneute Prüfung der Schleifenbedingung (Endlosschleife)

do-while-Anweisung (1)

- Nachprüfende Schleife
- Syntax:

```
do
    Block
while (Bedingung);
```



- Vorsicht: Semikolon hinter der Bedingung ist notwendig!
- Unterschied zur `while`-Schleife:

Der Block wird mindestens einmal durchlaufen!

do-while-Anweisung (2)

- Beispiel: wiederholtes Einlesen eines Zeichens

```
#include <stdio.h>
int main(void)
{
  char c;
  do
  {
    //...
    printf(" nochmal?\n ");
    c = getchar();
    getchar();     //entfernt das \n aus dem Eingabepuffer
   }
  while (c == 'J'|| c == 'j');
}
```

for-Anweisung (1)

- Vorprüfende Schleife: Bedingung wird vor Ausführung des Anweisungsblocks ausgewertet
- Syntax:

```
for (Initialisierung; Bedingung; Anweisung)
    Block
```

- Beispiel: Zähler

```
for (i = 0; i < 10; i = i + 1)
  printf("%ld\n", i);
```

- Welchen Wert hat i nach Durchlauf der Schleife?

`for`-Anweisung (2)

- Ablauf Beispiel:
  - Vor dem ersten Durchlauf wird die Variable i auf 0 gesetzt
  - Bedingung wird überprüft (⇒ „wahr“)
  - Schleifenrumpf wird einmal komplett durchlaufen mit i = 0 (⇒ Ausgabe: 0)
  - Ausführung Inkrement-Ausdruck: i wird um eins erhöht (i = 1)
  - Bedingung wird überprüft (⇒ „wahr“)
  - Erneutes Durchlaufen des Schleifenrumpfs
  - . . .

for-Anweisung (3)

```
for (i = 0; i < 10; i = i + 1)
  printf("%ld\n", i);
```

- Ablauf Beispiel:
  - ...
  - Variable i = 9: Bedingung erfüllt => Ausgabe 9
  - Erhöhung i auf 10
  - Auswertung der Bedingung (=> „falsch“)
  - Schleifenabbruch und Programmfortsetzung nach der Schleife
  - i hat den Wert 10
- Fazit: Wird eine for-Schleife nicht frühzeitig beendet, so hat nach der Schleife der Zähler den Wert, der die Bedingung nicht mehr erfüllt

Besonderheiten der `for`-Anweisung (1)

- Die Semikolons innerhalb der `for`-Anweisung dienen als Markierungen.
  Sie dürfen nicht weggelassen werden!
- Bsp.: i ist bereits initialisiert

```
for ( ; i < 10; i = i + 1)
  printf("%ld\n", i);
```

das ist übrigens schlechter Programmierstil…

- Wie oft wird diese Schleife durchlaufen?

Lässt sich nicht bestimmen – abhängig davon welchen Wert die Variable i vor der Schleife hat!

Besonderheiten der `for`-Anweisung (2)

- Eine `for`-Schleife lässt sich auch als `while`-Schleife darstellen

```
i = 1;
for (; i<=10;)
 {
   printf("%ld\n", i);
   i = i + 1;
 }
```

```
i = 1;
while (i<=10)
 {
   printf("%ld\n", i);
   i = i + 1;
 }
```

- Fazit: Ist kein Inkrement als solches notwendig, sollte eine `while`-Schleife verwendet werden.
- Achtung: Das `for`-Beispiel ist schlechter Stil und dient nur der Verdeutlichung im Vergleich zu `while`!

Besonderheiten der `for`-Anweisung (3)

- Endlosschleifen
  - Bedingung ist weggelassen

```
for (i = 0; ; i = i + 1)
  printf("%ld\n", i);
```

  - Alle Ausdrücke sind weggelassen

```
for (;;)    //unabhängig von Variablen wird Vorgang
  //…       //wiederholt
```

  - Werden verwendet, wenn eine Schleife nicht vor oder nach sondern innerhalb des Schleifenrumpfs beendet werden muss
  - werden mit `break`-Anweisung abgebrochen
- schlechter Stil, sollte vermieden werden!

break-Anweisung

- Einsatz: Abbrechen einer Schleife innerhalb des Anweisungsblocks

```
long zahl1, zahl2;
while(1)
{
  printf("1. Zahl:");
  scanf("%ld", &zahl1);
  printf("2. Zahl:");
  scanf("%ld", &zahl2);
  if(zahl1 == 0 || zahl2 == 0)
    {
     printf("Schleifenende\n");
     break;
    }
  printf("Der Quotient lautet: %ld\n", zahl1/zahl2);
}
```

Vermeidung `break`-Anweisung

```
long zahl1, zahl2, ende;
do
{
  printf("1. Zahl:");
  scanf("%ld", &zahl1);
  printf("2. Zahl:");
  scanf("%ld", &zahl2);

  ende = (zahl1 == 0) || (zahl2 == 0);

  if (!ende)
    printf("Der Quotient lautet: %ld\n", zahl1/zahl2);
}while (!ende);
```

continue-Anweisung

- Einsatz: Vorzeitiges Auslösen der nächsten Schleifeniteration

```
#include <stdio.h>
int main(void)
{
  char zeichen;
  while((zeichen = getchar()) != EOF)
  {
    if( (zeichen < 'A') || (zeichen > 'Z'))
        continue;
    printf("%c", zeichen);
  }
}
```

Vermeidung `continue`-Anweisung

- Umgehen durch `if`-Anweisung normalerweise sinnvoller

```
#include <stdio.h>
int main(void)
{
  char zeichen;
  while((zeichen = getchar()) != EOF)
  {
    if( (zeichen >= 'A') && (zeichen <= 'Z'))
      printf("%c", zeichen);
  }
}
```

Aufgabe

- Schreiben Sie ein C-Programm, dass alle Zahlen zwischen 1 und 10 ausgibt, die nicht durch 3 teilbar sind.

Zusammenfassung

- for-Schleifen
  - sollten nur zum Zählen eingesetzt werden
- while-Schleifen
  - abweisende Schleife
- do-while Schleifen
  - nicht-abweisende Schleife
- break/continue
  - continue braucht niemand – vergessen Sie das wieder
  - break ist in Ausnahmefällen ok, kann aber immer vermieden werden (bei switch ist es natürlich nötig)
- durch while alleine sind alle anderen Konstrukte darstellbar

Fazit:

Nicht alles, was erlaubt ist, sollte man auch tun. Vieles ist einfach nur schlechter Programmierstil!