

Schleifen / Loops: For-Loop

Einleitung

Eines der obersten Gebote im Programmieren: **Nie sich wiederholenden Code einfach kopieren!**

Soll sich eine Aktion wiederholen, dann könnte man eine Schleife, auch Loop genannt, verwenden. C kennt drei Arten von Loops:

- For-Schleifen / For-Loop
- While-Schleife / While-Loop
- Do ... While-Schleife / Do-While-Loop

Hier schauen wir uns den For-Loop genauer an.

Verständnis

For-Loops setzen wir dann ein, wenn wir eine Aktion eine bestimmte Anzahl Male durchführen möchten. Dabei muss der Code der allgemeinen Syntax entsprechen:

```
3
4 for( init; condition; increment )
5 {
6
7 }
8
```

Wir verwenden einen Loop-Iterator, damit wir die Durchläufe überwachen und im richtigen Moment abbrechen können (resp. der Loop selbst abgebrochen wird). Grundsätzlich ist es empfehlenswert ganzzahlige Variablen als Iterator zu wählen (char, short, int, long, ...) aber auch andere Datentypen sind zulässig. Dieser muss mit dem Startwert initialisiert werden. Mit der «condition» überprüfen wir, ob der Loop nochmals ausgeführt werden soll: der Loop wird ausgeführt, so lange die «condition» logisch «wahr/true» ist. Mit dem «increment» legen wir fest, in welchen Schritten der Iterator verändert wird. Dieser kann zunehmen, abnehmen, Sprünge aufweisen, etc.

Beispiel: Addiere alle Zahlen von 1 bis 10

```
1 #include <stdio.h>
2
3 int main() {
4     int sum = 0;
5     int start = 1;
6     int end = 10;
7
8     for( int i = start; i <= end; i++ )
9     {
10         sum = sum + i;
11     }
12
13     printf("The sum from %d to %d is equal to %d\n",
14           start, end, sum);
15     return 0;
16 }
```

Lege für diesen Workshop einen entsprechenden Ordner an im Ordner «21_Workshops» und lege die erarbeiteten Code-Beispiele dort ab.

Übungen

1. Schreibe das obige Beispiel ab. Wie viel ist die Summe aller Integer von 1 bis 10?
2. Summiere nur die geraden Zahlen. Hinweis: Welchen Startwert wählst Du? Welches Inkrement?
3. Verwende einen For-Loop, um folgende **Ausgabe des Iterators** an der Kommandozeile zu erzeugen («i» printen):

```
prgc@prgcvm:~/PRGModule/2021FSPRGC$ gcc forloop.c -o forloop
prgc@prgcvm:~/PRGModule/2021FSPRGC$ ./forloop
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
prgc@prgcvm:~/PRGModule/2021FSPRGC$
```

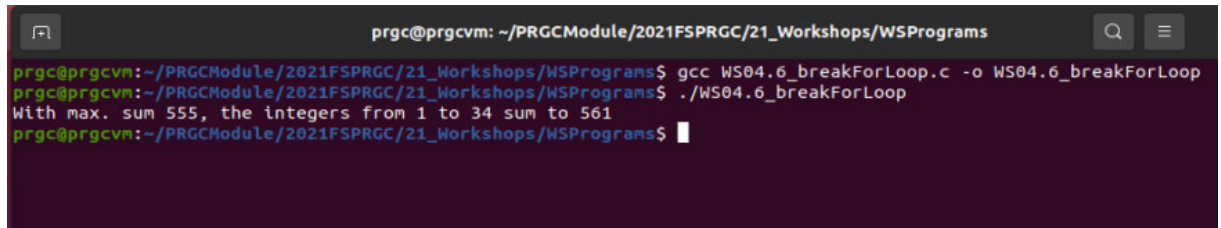
4. Verwende einen For-Loop, um folgende Ausgabe **Ausgabe des Iterators** an der Kommandozeile zu erzeugen («i» printen):

```
prgc@prgcvm:~/PRGModule/2021FSPRGC$ gcc forloop.c -o forloop
prgc@prgcvm:~/PRGModule/2021FSPRGC$ ./forloop
1
3
9
27
81
243
729
prgc@prgcvm:~/PRGModule/2021FSPRGC$
```

5. Verwende zwei ineinander **verschachtelte** For-Loops, um folgende Ausgabe zu erzeugen:

```
prgc@prgcvm:~/PRGModule/2021FSPRGC$ gcc forloop.c -o forloop
prgc@prgcvm:~/PRGModule/2021FSPRGC$ ./forloop
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
prgc@prgcvm:~/PRGModule/2021FSPRGC$
```

6. Mit dem Schlüsselwort «break» können loops (nicht nur for-loops) abgebrochen werden. Schreibe ein Programm, das so lange nach einander folgende Integer addiert (starten mit 1), bis ein gewisser Grenzwert erreicht wird. Dann wird der Loop abgebrochen, und folgende Ausgabe erzeugt:



```
prgc@prgcvm: ~/PRGModule/2021FSPRGC/21_Workshops/WSPPrograms
prgc@prgcvm:~/PRGModule/2021FSPRGC/21_Workshops/WSPPrograms$ gcc WS04.6_breakForLoop.c -o WS04.6_breakForLoop
prgc@prgcvm:~/PRGModule/2021FSPRGC/21_Workshops/WSPPrograms$ ./WS04.6_breakForLoop
With max. sum 555, the integers from 1 to 34 sum to 561
prgc@prgcvm:~/PRGModule/2021FSPRGC/21_Workshops/WSPPrograms$
```

Versuche in diesem Zusammenhang das Konzept «Scope» oder «Scope of a Variable» zu erforschen und beschreibe es im Wiki (oder verbessern/erweitern).