

## Kurseinheit 4: Kontrollstrukturen

1. Allgemein
2. Sequenz
3. Selektion
4. Kopfgetestete Schleife
5. Endegetestete Schleife
6. Geschachtelte Kontrollstrukturen und Idiome
7. Strukturiertes Programmieren
8. Metriken
9. Rekursion

## Übersicht KE 4

Lehrveranstaltung Ingenieur-Informatik – 2 SWS/2 Credits: EI1, EI+1, MKA1, MK+1, EI3nat3  
Lehrveranstaltung Programmierung 2 (Teil C) – 2 SWS/2 Credits: AI2

Unterrichtsdauer für diese Kurseinheit: 90 Minuten

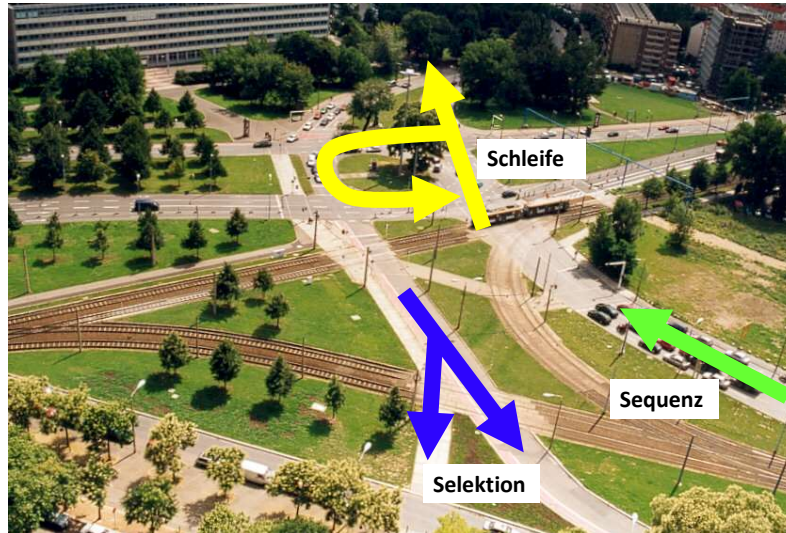
Korrespondierende Kapitel aus *C-Programmierung – Eine Einführung*: Kapitel 6

Zusatzthemen: Idiome, strukturierte Programmierung und Metriken

## 1. Allgemein

3

### Metapher Kontrollstrukturen



Kontrollstrukturen dienen zur Steuerung des Programmablaufs innerhalb von Funktionen.

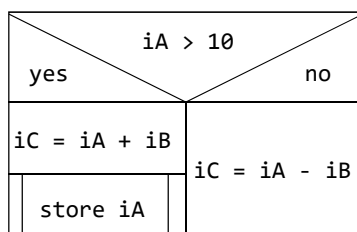
## 1. Allgemein

4

### Modellierung Kontrollstrukturen

Kontrollstrukturen und damit die gesamte Funktion lassen sich mittels Struktogrammen (SG) oder Flussdiagrammen (FD) modellieren. SG überwiegen bei der strukturierten Programmierung, während FD meist im Umfeld der Programmierung in Assembler zu finden sind. Ein FD für eine strukturierte Funktion lässt sich in ein SG überführen.

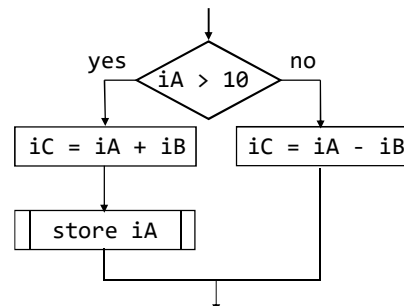
#### Struktogramm (SG)



<https://de.wikipedia.org/wiki/Nassi-Shneiderman-Diagramm>

Synonym für Struktogramm ist:  
Nassi-Shneiderman-Diagramm

#### Flussdiagramm (FD)



Synonym für Flussdiagramm ist:  
Programmablaufplan

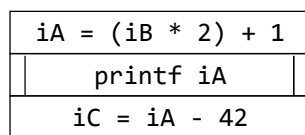
## 2. Sequenz

5

### Prinzip Sequenz

```
iA = (iB * 2) + 1;
printf("%d\n", iA);
iC = iA - 42;
```

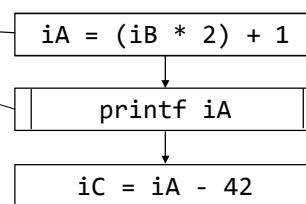
#### Struktogramm (SG)



Anweisung

(Unter-)  
Funktionsaufruf

#### Flussdiagramm (FD)



(Unter-)Funktionsaufruf: Eigene Funktionen oder Funktionen aus Bibliotheken.  
Allgemein: FD benötigen mehr Platz – SD deutlich kompakter!

## 3. Selektion

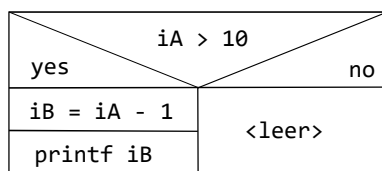
6

### Einfache Selektion

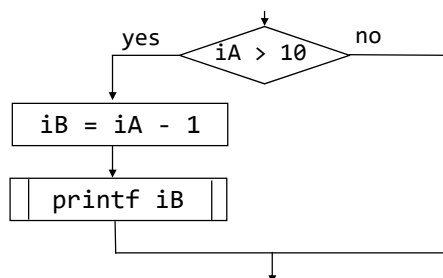
```
if (iA > 10)
{
    iB = iA - 1;
    printf("%d\n", iB);
}
```

Kein Semikolon!

#### Struktogramm (SG)



#### Flussdiagramm (FD)



### 3. Selektion

7

#### Einfache Selektion – mit und ohne Blockklammerung

Anweisungen können mit { } in Blöcke zusammengefasst werden.

```
if (iA > 10)
{
    iB = iA - 1;
    printf("%d\n", iB);
}
```

Selektion bezieht sich auf den gesamten Block.

Ohne Block bezieht sich die Selektion nur auf den nächsten Befehl. Einrückungen haben keine Wirkung!

```
if (iA > 10)
    iB = iA - 1;
    printf("%d\n", iB);
```

printf wird immer ausgeführt.

Dieses fehleranfällige Verhalten wird vermieden, indem immer hinter if, else, while, do, for usw. ein Block implementiert wird, auch wenn dieser nur eine Anweisung enthält.

C-Coding Styleguide: CL6 und K9

### 3. Selektion

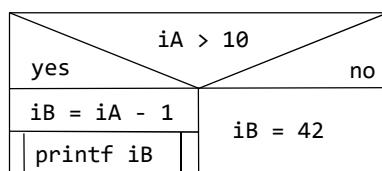
8

#### Zweifache Selektion

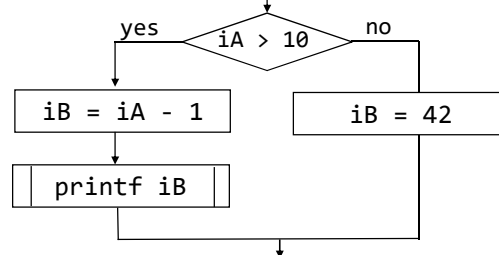
```
if (iA > 10)
{
    iB = iA - 1;
    printf("%d\n", iB);
}
else
{
    iB = 42;
}
```

Kein Semikolon!

Struktogramm (SG)



Flussdiagramm (FD)



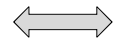
### 3. Selektion

9

#### Zweifache Selektion mit dem Bedingungsoperator

```
if (iA > 10)
{
    iB = iA - 1;
}
else
{
    iB = 42;
}
```

äquivalent



Bedingung

`(iA > 10) ? (iB = iA - 1) : (iB = 42);`

wahr

falsch

Der Bedingungsoperator `?:` ist der einzige C-Operator mit drei Operanden. Dieser ist äquivalent zu einer zweifachen Selektion.

Da dieser somit nicht unbedingt nötig ist und der Quellcode dadurch schlechter zu lesen ist, sollte dieser vermieden werden (C-Coding Styleguide, K7 -> **empfohlene** Regel).

In der Praxis findet sich der Bedingungsoperator häufig in Bibliotheken und Makros (siehe KE 9)

### 3. Selektion

10

#### Mehrfache Selektion

```
switch (iA)
{
    case 1:
        iB = 1;
        break;
    case 2:
        iB = 73;
        break;
    case 3:
        iB = 42;
        break;
    default:
        iB = -1;
        break;
}
```

Bei switch sollte ein numerischer Ganzzahlwert verwendet werden (signed char, unsigned char, short int, unsigned short int, int, unsigned int, ... sowie enums (siehe später)). Danach sind die verschiedenen Fälle (cases) aufzuführen. Mit break wird ans Ende der mehrfachen Selektion gesprungen.

Die Sprungmarke default handelt alle anderen Fälle ab und sollte immer am Ende stehen.

Bei switch sollte auf float oder double-Werte verzichtet werden, da diese nie auf Gleichheit überprüft werden sollen (Rundungsungenauigkeiten etc.).

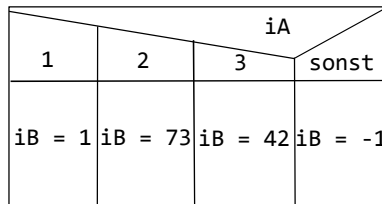
Es kann **kein Bereich als Fall** angegeben werden:  
case <4: ist nicht möglich

### 3. Selektion

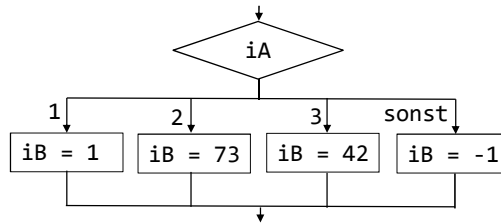
11

#### Mehrfache Selektion – Modellierung

Struktogramm (SG)



Flussdiagramm (FD)



Bei vielen Fällen (cases) wird das Diagramm zunehmend zu breit. Dann kaum noch auf einer Seite darstellbar.

Beispiel: Ein Programm erhält ein Zeichen übergeben (0-255) und muss sich immer unterschiedlich verhalten (-> 256 cases).

### 3. Selektion

12

#### Mehrfache Selektion – Fall-through

```
switch (iA)
{
    case 1:
        iB = 1;
        break;
    case 2:
        iB = 73;
        ↓ Fall-through
    case 3:
        iB = 42;
        break;
    default:
        iB = -1;
        break;
}
```

Bei einem fehlenden break wird von einem Fall-through gesprochen. Das Programm wird dann einfach im nächsten case weitergeführt.

Dies sollte nur in Ausnahmefällen gemacht werden.

Ein Ausnahmefall wäre z.B. wenn der Code in verschiedenen Fällen gleich ist.

```
switch (iA)
{
    case 1: //Fall-through is correct
    case 2:
        iC = 815; //Fall-through is correct
    case 3:
        iB = 4711;
        break;
}
```

In einem solchen Ausnahmefall sollte rechts mit Kommentar ein solcher Fall-through dokumentiert werden.

## 4. Kopfgetestete Schleife

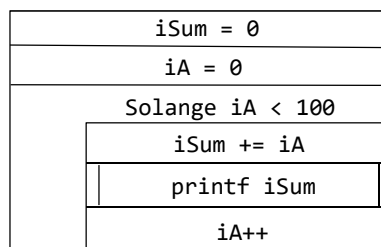
13

### Syntax: Zwei Möglichkeiten: for und while

```
iSum = 0;

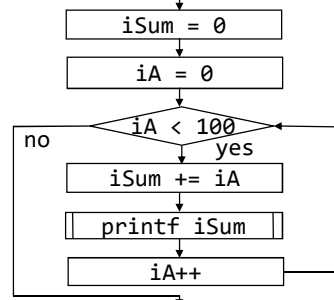
for (iA = 0; iA < 100; iA++)
{
    iSum += iA;
    printf("iSum = %d\n", iSum);
}
```

Struktogramm (SG)



```
iSum = 0;
iA = 0;
while (iA < 100)
{
    iSum += iA;
    printf("iSum = %d\n", iSum);
    iA++;
}
```

Flussdiagramm (FD)



## 4. Kopfgetestete Schleife

14

### Initialisierung, Abbruchbedingung, Inkrementierung Schleifenzähler

Initialisierung    Abbruchbedingung

```
for (iA = 0; iA < 100; iA++)
{
    //Loop Body
}
```

Wird nach dem Ende des Schleifenrumpfes ausgeführt. Hier wird meist das in- und dekrementieren des Schleifenzählers durchgeführt.

**Semikolon! (Kein Komma)**

```
for (iA = 0; iA < 100; iA++)
{
    //Loop Body
}
```

Initialisierung    Abbruchbedingung

```
iA = 0;
while (iA < 100)
{
    //Loop Body
    iA++;
}
```

In- oder dekrementieren des Schleifenzählers

## 4. Kopfgetestete Schleife

15

### Vergleich: for und while

Kein Semikolon!

```
iSum = 0;
for (iA = 0; iA < 100; iA++)
{
    iSum += iA;
    printf("iSum = %d\n", iSum);
}
```

```
iSum = 0;
iA = 0;
while (iA < 100)
{
    iSum += iA;
    printf("iSum = %d\n", iSum);
    iA++;
}
```

Als kopfgetestete Schleife wird die for-Schleife bevorzugt, wenn die Anzahl der Schleifendurchläufe bekannt/konstant ist. Im anderen Fall wird die while-Schleife verwendet.

Die for-Schleife ist etwas kompakter und weniger fehleranfällig. Der Schleifenrumpf der kopfgetesteten Schleife kann **kein, einmal oder mehrmals durchlaufen werden**.

Welche Werte haben iSum und iA nach dem Ende der obigen Schleife? (ohne Taschenrechner)

```
printf("iSum = %d - iA = %d\n", iSum, iA);
```

Tipp: Adam Ries (1492/3 – 1559)

## 5. Endegetestete Schleife

16

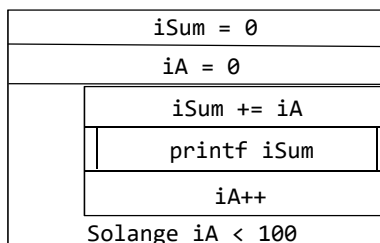
### Syntax

Der Schleifenrumpf der endegetesteten Schleife **wird mindestens einmal durchlaufen**.

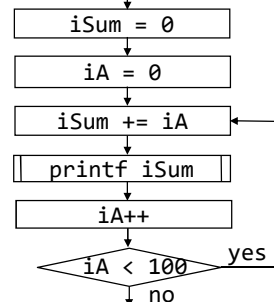
```
iSum = 0;
iA = 0;
do
{
    iSum += iA;
    printf("iSum = %d\n", iSum);
    iA++;
} while (iA < 100);
```

Semikolon!

### Struktogramm (SG)



### Flussdiagramm (FD)





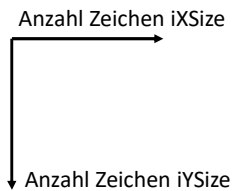
## 6. Geschachtelte Kontrollstrukturen und Idiome

17

### Beispiel: Geschachtelte Kontrollstrukturen (1)

```
int iX;  
int iY;  
int iXSize;  
int iYSize;  
  
// Enter iXSize and iYSize  
printf("Please enter iXSize (1-30):\n");  
scanf_s("%d", &iXSize);  
printf("\nPlease enter iYSize (1-30):\n");  
scanf_s("%d", &iYSize);
```

Ausgabe der folgenden Figur rechts:



```
C:\Users\Daniel Fischer\Doc...  
Please enter iXSize (1-30):  
10  
Please enter iYSize (1-30):  
10  
BBBBBBBBBB  
AABBBBBBBB  
AAABBBBBBB  
AAAABBBBBB  
AAAAABBBBB  
AAAAAABBBB  
AAAAAAABBB  
AAAAAAAABB  
AAAAAAAAB  
AAAAAAAAA
```

## 6. Geschachtelte Kontrollstrukturen und Idiome

18

### Beispiel: Geschachtelte Kontrollstrukturen (2)

```
if ((iXSize >= 1) && (iXSize <= 30) && //CRLF possible  
    (iYSize >= 1) && (iYSize <= 30)) //Better readable  
{  
    for (iY = 0; iY < iYSize; iY++)  
    {  
        for (iX = 0; iX < iXSize; iX++)  
        {  
            if (iY >= iX)  
            {  
                putchar('A');  
            }  
            else  
            {  
                putchar('B');  
            }  
        }  
        putchar('\n');  
    }  
}
```

```
C:\Users\Daniel Fischer\Doc...  
Please enter iXSize (1-30):  
10  
Please enter iYSize (1-30):  
10  
BBBBBBBBBB  
AABBBBBBBB  
AAABBBBBBB  
AAAABBBBBB  
AAAAABBBBB  
AAAAAABBBB  
AAAAAAABBB  
AAAAAAAABB  
AAAAAAAAB  
AAAAAAAAA
```

## 6. Geschachtelte Kontrollstrukturen und Idiome

19

### Idiome Kontrollstrukturen

**Endlosschleifen:** Nur in wenigen Ausnahmefällen erwünscht!

```
while (1)
{
    // Do something
}
```

```
for (;1;)
{
    // Do something
}
```

Schlechte Variante

```
do
{
    // Do something
} while (1);
```

Schlechte Variante

**Unvollständige For-Schleifen:**

```
iA = 4;
for (;iA < 10;)
{
    // Do something
    iA++;
}
```

Vermeiden, besser hier  
while-Schleife nehmen

**Dangling else:**

```
if (iA == 1)
    if (iB == 1)
        iC = 42;
    else
        iC = 73;
```

Dangling else wird durch  
C-Coding Styleguide  
vermieden (Klammern { })

**else if:**

```
if (iA == 1)
{
}
else if (iB == 1)
{
}
```

else if ist zu vermeiden.  
Besser: Normaler Block  
nach else mit if darin

## 6. Geschachtelte Kontrollstrukturen und Idiome

20

### Idiome for-Schleifen

**Inkrementieren des Schleifenzähler**

```
for (iA = 0; iA < 10; iA++)
{
    printf("%d\n", iA);
}
```

```
for (iA = 73; iA < 83; iA++)
{
    printf("%d\n", iA);
}
```

```
for (iA = 0; iA < 10; iA = iA + 2)
{
    printf("%d\n", iA);
}
```

**Dekrementieren des Schleifenzähler**

```
for (iA = 9; iA >= 0; iA--)
{
    printf("%d\n", iA);
}
```

```
for (iA = 82; iA >= 73; iA--)
{
    printf("%d\n", iA);
}
```

```
for (iA = 8; iA >= 0; iA = iA - 2)
{
    printf("%d\n", iA);
}
```

## 6. Geschachtelte Kontrollstrukturen und Idiome

21

### Idiome Konsole

```
while (_kbhit())
{
    _getch();
}
```

#### Keyboardbuffer leeren

\_kbhit() (keyboard hit) und \_getch() (get char) sind Funktionen aus der Bibliothek conio (#include <conio.h>). Unterstrichen am Anfang signalisiert „betriebssystemnahe“ Funktionen.

```
while (!_kbhit());
while (!(_kbhit()));
while (!(_kbhit()))
{
}
```

#### Warte auf Tastatureingabe (5 Varianten)

\_kbhit() gibt einen Wert ungleich 0 (null) zurück, wenn eine Taste gedrückt wurde. Andernfalls wird 0 zurückgegeben.

```
while (_kbhit() == 0)
{
}
while (0 == _kbhit())
{
}
```

Gut lesbar!

Noch besser: Nicht jeder Compiler würde im vorherigen Beispiel einen Fehler bei nur einem = erkennen. Durch diese Schreibweise würde jeder Compiler einen Fehler melden: 0 = \_kbhit() wäre Fehler, da einer Konstanten (hier 0) nie ein Wert zugewiesen werden kann.

## 7. Strukturierte Programmierung

22

### Fünf Regeln

K10

Die Regeln der „Strukturierten Programmierung“ sind einzuhalten.

§1 Eine Funktion hat einen Eingang und einen Ausgang.

§2 Nicht in Abfragen springen.

§3 Nicht aus Abfragen herausspringen.

§4 Nicht in Schleifen springen.

§5 Nicht aus Schleifen herausspringen.

*Mit Struktogrammen (Nassi-Shneiderman Diagrammen) können nur strukturierte Funktionen realisiert werden.*

Durch die Regeln der strukturierten Programmierung ergeben sich Restriktionen für die folgenden Keywords:

- **return** darf nur einmal pro Funktion vorkommen und zwar nur am Ende (nur ein Ausgang)
- **break** darf nur in einer mehrfachen Selektion verwendet werden.
- **goto** ist grundsätzlich nicht erlaubt.

Ebenso kann auf **continue** verzichtet werden, da dies durch eine einfache Selektion realisiert werden kann.

## 7. Strukturierte Programmierung

23

### §1 Eine Funktion hat einen Eingang und einen Ausgang

```
int DoItNonStructured(int iA1)
{
    if (iA1 < 5)
    {
        return 42;
    }
    else
    {
        return 73;
    }
}
```

Hier zwei Ausgänge! Nicht erlaubt!

```
int DoItStructured(int iA1)
{
    int iRet;

    if (iA1 < 5)
    {
        iRet = 42;
    }
    else
    {
        iRet = 73;
    }

    return iRet;
}
```

Eine Funktion enthält höchstens ein `return`! Nur Funktionen mit `void` als Rückgabewert müssen kein `return` haben.

## 7. Strukturierte Programmierung

24

### §2 Nicht in Abfragen springen

### §4 Nicht in Schleifen springen

```
goto Label1;

if (iVal1 < 50)
{
    printf("Hello\n");
Label1:
    printf("World\n");
}
```

„Hello“ wird nicht ausgegeben!

```
iVal2 = 0; // Otherwise uninitialized!
goto Label2;

for (iVal2 = 0; iVal2 < 3; iVal2++)
{
    printf("All is\n");
Label2:
    printf("okay\n");
}
```

```
World
okay
All is
okay
All is
okay
```

Ein „All is“ wird anfänglich nicht ausgegeben! Beim Sprung in die Schleife wäre `iVal2` ohne vorherige Initialisierung noch uninitialized.

Das Schlüsselwort `goto` darf nicht verwendet werden.

## 7. Strukturierte Programmierung

25

### §2 Nicht aus Abfragen herauspringen

### §4 Nicht aus Schleifen herauspringen

```
if (iVal1 < 50)
{
    if (iVal1 == 42)
    {
        goto Label3;
    }

    printf("World\n");
}

Label3:
```

```
for (iVal1 = 0; iVal1 < 10; iVal1++)
{
    if (iVal1 == 9)
    {
        break;
    }
    if (iVal1 == 9) // Alternative
    {               // Alternative
        goto Label4; // Alternative
    }               // Alternative

    printf("%d\n", iVal1);
}

Label4:
```

Das Schlüsselwort `goto` darf nicht verwendet werden. `break` darf nur in mehrfacher Selektion (switch) verwendet werden.

## 7. Strukturierte Programmierung

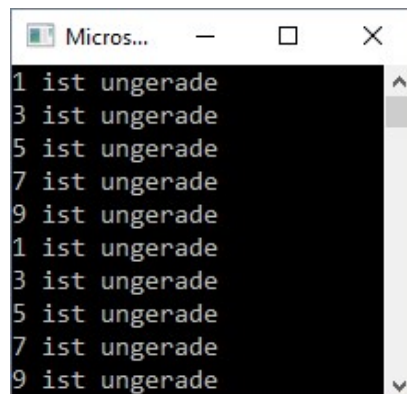
26

### continue kann vermieden werden

```
for (iVal1 = 0; iVal1 < 10; iVal1++)
{
    if ((iVal1 % 2) == 0)
    {
        continue;
    }

    printf("%d ist ungerade\n", iVal1);
}

for (iVal1 = 0; iVal1 < 10; iVal1++)
{
    if ((iVal1 % 2) != 0)
    {
        printf("%d ist ungerade\n", iVal1);
    }
}
```



```
1 ist ungerade
3 ist ungerade
5 ist ungerade
7 ist ungerade
9 ist ungerade
1 ist ungerade
3 ist ungerade
5 ist ungerade
7 ist ungerade
9 ist ungerade
```

Das Schlüsselwort `continue` kann vermieden werden.

## 8. Metriken

27

### Wie „gut“ und „komplex“ ist die Software?

```
#include <stdio.h>

// Testing of modulo operator
int main(void)
{
    int iVal;

    for (iVal = 0; iVal < 100; iVal++)
    {
        // Can iVal divided by 3 and 5
        // without remainder?
        if (((iVal % 3) == 0) && ((iVal % 5) == 0))
        {
            printf("iVal: %d", iVal);
        }
    }

    return 0;
}
```

#### Zeilenmetriken

(LOC Lines Of Code) hier bezogen auf eine Funktion  
LOCpro: Programmzeilen  
LOCcom: Kommentarzeilen  
LOCphy: Gesamtanzahl

#### c%: Kommentardichte

bezogen auf eine Funktion:

$$c\% = \frac{\text{LOCcom}}{\text{LOCphy}} \cdot 100\%$$

#### Cyclomatic Complexity (CC)

nach McCabe (in CMT++ als v(G) ausgewiesen)

$$CC = 1 + \text{Anzahl der Bedingungen in einer Funktion}$$

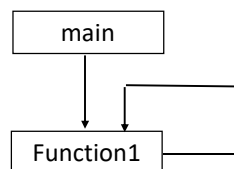
## 9. Rekursion

28

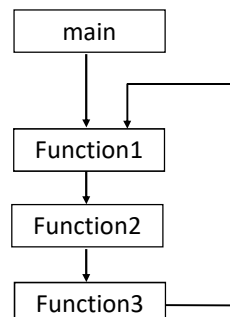
### Definition und Arten von Rekursion

Unter Rekursion (recurrere lat. = zurücklaufen) wird im Programmieren der Vorgang verstanden, wenn eine Funktion sich selbst direkt oder indirekt aufruft. Dieser Vorgang darf nicht unendlich fortgeführt werden, sondern muss irgendwann (Abbruchkriterium) abgebrochen werden.

#### Direkte Rekursion



#### Indirekte Rekursion



Mittels Rekursion lassen sich effizient (wenig Quellcode!) Algorithmen realisieren!

## 9. Rekursion

29

### Beispiel: Rekursion

```
unsigned long long int CalcFaculty(unsigned int uiN)
{
    unsigned long long int ulliRet;

    if (uiN == 0U)
    {
        ulliRet = 1ULL;
    }
    else
    {
        ulliRet = (unsigned long long int)uiN * (CalcFaculty(uiN - 1U));
    }

    return ulliRet;
}
```

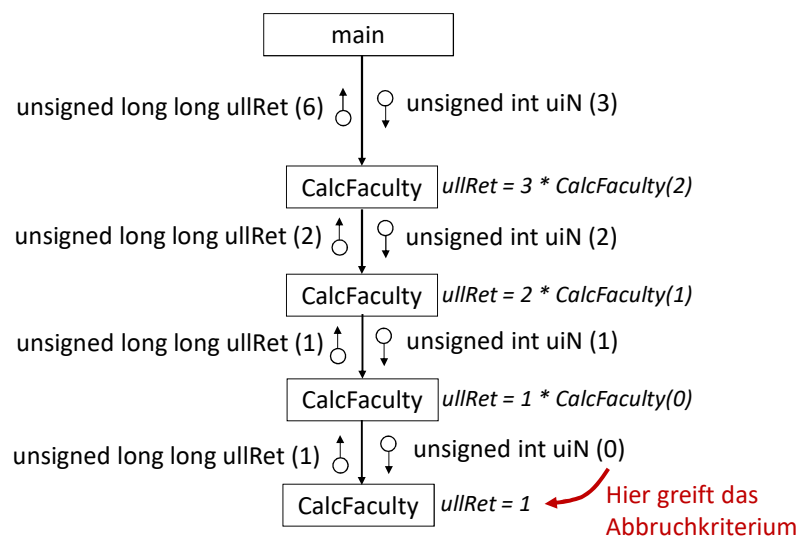
$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$

Fakultät ist nur für positive Zahlen definiert. Beim Rückgabewert kann ein Variablenüberlauf stattfinden. Durch 64-Bit Rückgabewert ist dies etwas entschärft (bis 65!), aber nicht behoben. Details (Erklärungen, Grafik) siehe [LUH17] – Implementierung ist dort aber zu hinterfragen (Datentypen, Variablenüberlauf, etc.)

## 9. Rekursion

30

### Beispiel: Rekursion für uiN = 3



## Behandelte Schlüsselwörter in KE 4

Schlüsselwörter C89:

<del>auto</del> ✓	do ✓	<del>goto</del> ✓	signed ✓	unsigned ✓
break ✓	double ✓	if ✓	sizeof ✓	void ✓
case ✓	else ✓	int ✓	static ✓	volatile ✓
char ✓	enum	long ✓	struct	while ✓
const ✓	extern ✓	register ✓	switch ✓	
<del>continue</del> ✓	float ✓	return ✓	typedef	
default ✓	for ✓	short ✓	union	

Schlüsselwörter ab C99:

_Bool ✓	_Complex ✓	_Imaginary ✓	inline	restrict
---------	------------	--------------	--------	----------

Schlüsselwörter ab C11:

_Alignas	_Alignof	_Atomic	_Generic	_Noreturn
_Static_assert	_Thread_local			