C / C++ kurz und knapp Rev. 207 / 16.09.2015

```
// einzeiliger Kommentar

if (var >= 45) // Kommentar hinter Code
```

```
/*
      Kommentar über
      mehrere Zeilen
*/
```

Datentypen

Alle hier gemachten Angaben beziehen sich auf den Microsoft Visual C Compiler. Bei anderen Compilern sind Typbezeichnungen, Anzahl der Bit und damit der Wertebereich u.U. abweichend.

Zahlenformate und andere Festlegungen des IEEE-754-Standards

		Wertebereich	
Typ	**Bits**	**Unsigned**	**Signed**
char	8	0 … 255	-128 … 127
long int	32	0 … 4.294.967.295	-2.147.483.648 …. 2.147.483.647
short int	16	0 … 65.535	- 32768 … 32767

Der Datentyp int ist vom verwendeten Prozessorsystem (16/32 bit) abhängig

Typ	Bits	Genauigkeit (Nachkommastellen)	kleinste Zahl (Betrag)	größte Zahl (Betrag)
float	32 1 – Vorzeichen 8 – Exponent 23 – Mantisse	7 – 8	$2^{-23} \times 2^{-126} = 2^{-149} \approx 1{,}401 \cdot 10^{-45}$	$(1-2^{-24}) \times 2^{128} \approx 3{,}403 \cdot 10^{38}$
double	64 1 – Vorzeichen 11 – Exponent 52 – Mantisse	15 - 16	$2^{-52} \times 2^{-1022} = 2^{-1074} \approx 4{,}941 \cdot 10^{-324}$	$(1-2^{-53}) \times 2^{1024} \approx 1{,}798 \cdot 10^{308}$

Variablen

	Deklaration	Initialisierung	Wertzuweisung
einfache Variable	`DATATYPE varname;`	`DATATYPE varname = value;` `z.B.:  char varname = 'z';`	`varname = value;`
Array	`DATATYPE arrname[COUNT];`	`DATATYPE arrname[4] = {1,2,3,4};`	`arrname[index] = value;` `*(arrname + index) = value; (s. Pointer)`
Aufzählung (Enum)	`enum ENUM_NAME {NAME1,NAME2,NAME3=5,NAME4};`	-	-
Struktur	`struct structname{` `  short int var1;` `  double var2;` `  float var3;` `};` ← nicht vergessen !!!	`structname varname = {5, 2.3, 5.234};`	`varname.var2 = 2.345;`
Pointer	`DATATYPE *pointername;`	`DATATYPE *pointername = &varname;`	`*pointername = value;`
Referenz	`DATATYPE &refname;`	`DATATYPE &refname = varname;`	`int a; int &b = a; int &c = b;` `int x=3;` `c = x;` → a,b,c haben den Wert 3
Typvereinbarung	`typedef TYPNAME ALIASNAME` oder `typedef struct{int var1, int var2} ALIASNAME`	**Beispiele**	
		`typedef unsigned int uint;` `uint var;`	`typedef struct {double re, double im} complex;` `complex var;` `var = {2.045,-3.56}`
Typumwandlung (Cast)	`(DATATYPE)varname`	`TYP1 a=<WERT>;` `TYP2 b;` `b = (TYP2) a;`	`float a=5.1234;` `float b;` `b = (int)a;  // b=5.0`

Werden Funktions-/Methodenparameter als Referenzen angegeben, wird keine Kopie angelegt. wird der Rückgabewert einer Funktion/Methode als Referenz angegeben, wird vom Rückgabewert auch keine Kopie angelegt.

Operatoren

binärer Operator : ergebnis = x op y `(a=1; b=2; c=a+b → c=3)`
unärer Operator : ergebnis = op x `(a=1; b=-a → b=-1)`
Zuweisungsoperator : ergebnis op= x `(a=1; a+=5 → a=6)`

Sequenz

Zusammenfassung mehrerer Anweisungen zu einer Anweisung

Operator	Beispiel
, (Komma)	int i=0, j=2, k=4; i=3, j*=i, k+=i; → i=3 j=6 k=7

arithmetisch

	Operator	Ganzzahl	Fließkomma
Addition	+[1]	7 + 5 → 12	4.3 + 2.3 → 6.6
Subtraktion	-[1]	5 – 2 → 3	4.3 – 1.2 → 3.1
Multiplikation	*	5 * 2 → 10	1.5 * 2 → 3
Division	/	13 / 5 → 2	15.0 / 4 → 3.75
Modulo	%	13 % 5 → 3	

[1] binär

inkrement / dekrement

Operator post	Operator pre	Äquivalent	Beispiel
a++	++a	a = a + 1	a = 1; c = ++a → c = 2, a=2 a = 1; c = a++ → c = 1, a=2
a--	--a	a = a - 1	a = 3; c = --a → c = 2, a=2 a = 3; c = a-- → c = 3, a=2

bitweise

Operation	Operator	Wertigkeit	Beispiel	
AND	&	binär	a = 0xE0; b = 0x70; a & b = 0x60	`1110 0000 & 0111 0000 → 0110 0000`
OR	|	binär	a = 0xE0; b = 0x93; a | b = 0xF3	`1110 0000 | 1001 0011 → 1111 0011`
XOR	^	binär	a = 0x33; b = 0x0F; a ^ b = 0x3C	`0011 0011 ^ 0000 1111 → 0011 1100`
SHIFT LEFT	<<	binär	5 << 2 → 20 (0x14)	`0000 0101 << 2 → 0001 0100`
SHIFT RIGHT	>>	binär	0xF0 >> 2 → 0x3C	`1111 0000 >> 2 → 0011 1100`
1-COMPLEMENT	~	unär	~0xC1 → 0x3E	`~1100 0001 → 0011 1110`
2-COMPLEMENT	-	unär	-0x3E → 0xC2 (~0x3E + 1)	`-0011 1110 → 1100 0010`

logisch (boolesch) / Vergleich

false: 0; true: alle anderen Zahlen

Operation	Operator	Beispiel
GREATER (OR EQUAL)	> >=	5 > 3 → true; 5 >= 5 → true; 3 > 5 → false;
SMALLER (OR EQUAL)	< <=	5 < 3 → false; 5 <= 5 → true; 3 < 5 → true;
AND	&&	3 < 5 && 5 < 10 → true; 3 > 5 && 5 < 10 → false
OR	||	3 < 5 || 5 > 10 → true; 3 > 5 || 5 > 10 → false
NOT	!	!(3 > 5 || 5 > 10) → true; !(3 < 5 || 5 > 10) → false
EQUAL	==	5 == 5 → true; 5 ==4 → false; 6-2*3 && 5 < 10→ false, da 6-2*3=0 (Punkt vor Strich)
NOT EQUAL	!=	5 != 5 → false Äquivalent: !(5==5); 5 !=4 → true;

Verweis- / Adressoperator

Operator	Zeichen	Beispiel
Adress-operator	&	`int a=5,*p;` `p = &a;  // Adresse von a in p` `cout << p;  // Adresse von a` `cout << *p;  // Inhalt von a`
Verweis-operator	*	

Bedingung (ternärer Operator)

Operator	Beispiel
`Bedingung ? Wahr-Teil : False-Teil`	x=5>3 ? 12 : 22 → x=12; x=5<3 ? 12 : 22 → x=22

Enums (Aufzählungen)

```
enum colour {RED=3,YELLOW,GREEN,BLUE};

colour x;
x = PURPLE;   // Error
x = GREEN;    // Ok
cout << x;    // 5

// benutzerdefinierte Werte
enum colour {ON=2,STANDBY,OFF=12};  // -> STANDBY=3
```

Arrays

Deklaration	Initialisierung	Matrixdarstellung	Speicherdarstellung
`TYP NAME[DIM]`	`a[3]={1,3,5};`	$(1\,3\,5)$	1 | 3 | 5
`TYP NAME[DIM1][DIM2]`	`a[2][3]={{1,3},{2,4,6}};` *Element a_{13} wird nicht initialisiert* Vereinfacht: `a[2][3]={1,3,5,2,4,6};` *alle Elemente müssen angegeben werden*	$\begin{pmatrix}1&3&5\\2&4&6\end{pmatrix}$	1 | 3 | ? | 2 | 4 | 6 1 | 3 | 5 | 2 | 4 | 6

Array m. Speicherallokation

```
TYP *ARRAYNAME = new TYP[DIM];          // Allokation von Speicher
delete[] ARRAYNAME;                     // Freigabe von Speicher
```

Arrays, Adressierung

über Index

```
int a[2][3]={1,3,5,2,4,6};

for (spalte=0; spalte<3; spalte++){
  for (zeile=0; zeile<2; zeile++)
    cout << a[zeile][spalte] << endl;
}
// a zeigt immer noch auf erstes Element
```

über Pointer

```
int a[2][3]={1,3,5,2,4,6};

for (i=0; i<6; i++)
  cout << *a++ << endl;

// a zeigt nun auf letztes Element
```

Priorität, Assoziativität

Priorität	Operator	Assoziativität
15	( ) [ ] -> .	links → rechts
14	! ~ ++ -- +[1] -[1] (TYP) * & sizeof	rechts → links
13	* / % (Rechenoperationen)	links → rechts
12	+[2] -[2]	links → rechts
11	<< >>	links → rechts
10	< <= > >=	links → rechts
9	== !=	links → rechts

[1] unär [2] binär

Priorität	Operator	Assoziativität
8	&	links → rechts
7	^	links → rechts
6	|	links → rechts
5	&&	links → rechts
4	||	links → rechts
3	?:	rechts → links
2	= += -= /= *= %= >>= <<= &= |= ^=	rechts → links
1	, (Sequenz-Operator)	links → rechts

Programmstrukturen

Verzweigungen

if

```
if (Bedingung){
      // Anweisungen wenn Bedingung wahr
}
else{
      // Anweisungen, wenn Bedingung falsch
}
```

```
if (Bedingung1){
      // Anweisungen wenn Bedingung wahr
}
elseif (Bedingung2){
      // Anweisungen, wenn Bedingung1 falsch und Bedingung2 wahr
}
```

switch

```
switch (x*3){
      case 15:
            // Anweisungen für X = 5
            break;
      case 30:
      case 45:
            // Anweisungen für x=10 oder x=15
            break;
      case 33:
            // Anweisungen für x=11
      case 66:
            // Anweisungen für x=11 oder x=22
            break;
      default:
            // Anweisungen für alle anderen Fälle
}
```

kein break

Schleifen

while	do while	for
abweisende / kopfgesteuerte Schleife; wird so lange ausgeführt, bis Bedingung falsch; wird nicht ausgeführt, wenn Bedingung von Anfang an falsch	nicht abweisende / fußgesteuerte Schleife; wird so lange ausgeführt bis Bedingung falsch; wird einmal ausgeführt, wenn Bedingung von Anfang an falsch	Zählschleife Anzahl der notwendigen Schleifendurchläufe ist vorher bekannt wird so lange ausgeführt bis Ende-Bedingung falsch

```
while (Bedingung){
      Anweisungen
}

Beispiel:
int x=0
while (x < 20) {      // 10 Durchläufe
  x += 2;
}
```

```
do{
      Anweisungen
}while(Bedingung);

Beispiel:
int x = 5;
do {                  // 1 Durchlauf
  x *= 3;
} while (x < 5);
```

nicht vergessen !!!!

```
for (Initialisierung; Ende-Bedingung; Aktualisierung){
      Anweisungen
}

Beispiel:
for (i=5; i<23; i++){        // 18 Durchläufe
}
```

Funktionen

Deklaration / Prototyp (im h-File)

```
DATENTYP FUNKTIONSNAME(TYP PARAM1, TYP PARAM2 …);
```

Definition

```
DATENTYP FUNKTIONSNAME(TYP PARAM1,TYP PARAM2 ….){
      Anweisungen
}
```

nicht vergessen !!!!

Beispiel

```
int myfunction(int var1, char var2, float var3);

int myfunction(int var1,char var2,float var3){
      Anweisungen
}
```

verschachtelte Funktionsaufrufe

Der Aufruf einer Funktion mit Rückgabewert kann direkt als Parameter eines weiteren Funktionsaufrufs verwendet werden.

Beispiel

```
// Verschachtelung von uppercase() und getName()
cout << uppercase(getName(PersonalId));
```

Aufruf von Funktionen

Call by Value

```
int cube(int p){
  p =  p * p * p;
  return p;
}

void main(){
  int x=4;

  cube(x);
  cout << x;                  // Ausgabe: 4
  cout << cube(x);            // Ausgabe: 64
}
```

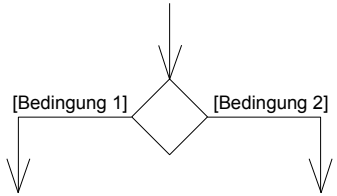
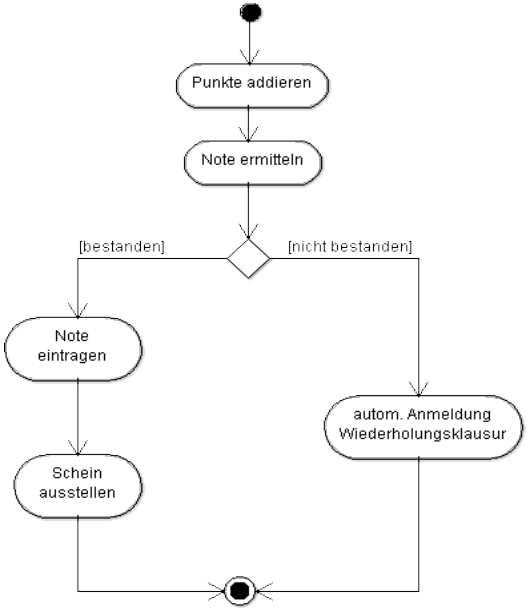
Call by Reference

```
void cube(int *p){
  *p = (*p) * (*p) * (*p);
}

void main(){
  int x=4;

  cube(&x);
  cout << x;                  // Ausgabe: 64
}
```

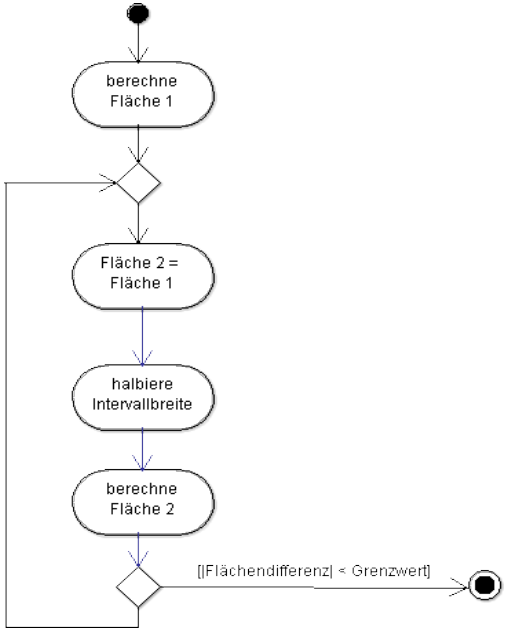
```
void cube(int& p){
  p = p * p * p;
}

void main(){
  int x=4;

  cube(x);
  cout << x;                  // Ausgabe: 64
}
```

UML-Diagramme

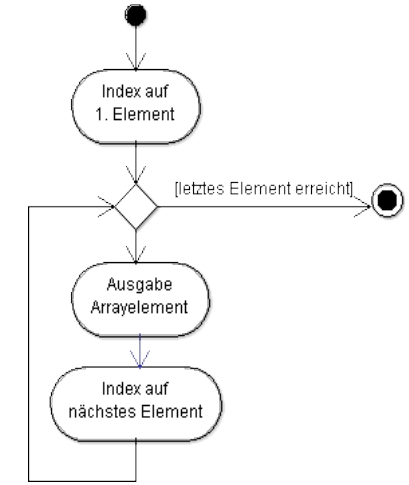
Aktivitätsdiagramm

Startpunkt	
Endpunkt	
Aktion	
Übergang	
Verzweigung	
Beispiel	

fussgesteuerte Schleife (do while)



Zählschleife (for)



kopfgesteuerte Schleife (while)