28 Vergleich mit anderen Programmiersprachen

GWBASIC, Turbo-Basic, Quick-Basic, Turbo-C

Gliederung

28.1 Allgemeines zum Vergleich der Programmiersprachen

Ein Vergleich einer Programmiersprache mit einer anderen ist immer problematisch. Letztlich muß die Aufgabenstellung entscheiden. Für Systemprogrammierung (Betriebssysteme, Compiler u.ä.), aber auch für anspruchsvolle Anwenderprogramme wird heute häufig die Sprache C der Assemblersprache vorgezogen, da C zum größten Teil prozessorunabhängig ist und in der Leistung dem Assembler ziemlich nahekommt. Für Gelegenheitsanwendungen ist die Sprache C weniger geeignet, wenn auch moderne Konzepte, wie sie in Turbo-C (Borland) und Quick-C (Microsoft) vorliegen, das C-Programmieren wesentlich erleichtern.

Auf der anderen Seite wurden die Basic-Dialekte weiterentwickelt; mit Turbo-Basic bzw. seinem Nachfolger Power-Basic und QuickBasic liegen auch hier moderne Konzepte vor. Sie enthalten viele Eigenschaften von Pascal, wie z.B. echte Unterprogramme (Prozeduren und Funktionen) mit lokalen Variablen und Rekursionen. Hinzu kommen komfortable Entwicklungsumgebungen, ähnlich wie in Turbo-Pascal. Mit *Visual Basic* und Visual Basic for Application stehen leistungsfähige Basic-Versionen für Windows zur Verfügung. Mit der Urform von Basic haben diese Dialekte nicht mehr viel gemeinsam.

Die folgenden Programmbeispiele zeigen die Lösung einer sehr einfachen Programmieraufgabe aus dem Bereich der Technischen Mechanik. Diese Beispiele sind auf keinen Fall repräsentativ. Es wird lediglich der Versuch unternommen, eine leicht überschaubare Aufgabe von der Programmiersprache Turbo-Pascal ausgehend in anderen Sprachen umzusetzen. Die Lösungen in den anderen Sprachen wurden so gewählt, daß sich möglichst viele Übereinstimmungen mit der Lösung in Turbo-Pascal ergeben.

Es werden folgende Versionen gezeigt:

- Turbo-Pascal
- GWBASIC/BASICA (nur aus nostalgischen Gründen)
- Turbo-Basic/Quick-Basic
- Turbo-C

Die Lösung in Turbo-Basic ist auch unter QuickBasic 4.0 lauffähig, wenn man »**incr z**« durch »**z = z + 1**« ersetzt. QuickBASIC setzt aber die reservierten Basic-Wörter in Großbuchstaben um.

GWBASIC ist ein mittlerweile veralterter Basic-Interpreter von Microsoft, zu dem es aber auch einen passenden Compiler gibt. Der Interpreter wurde mit dem Betriebssystem bis einschließlich Version 4.0 ausgeliefert, bei IBM-PCs unter der Bezeichnung BASICA. Aus diesem Grunde wurde GWBASIC mit in den Vergleich aufgenommen. GWBASIC benutzt Zeilennummern, die auch als Labels dienen werden können und setzt alle Bezeichner in Großbuchstaben um. In GWBASIC sind keine lokalen Bezeichner und auch keine Rekursionen möglich. Die Editiermöglichkeiten sind sehr beschränkt. GWBASIC-Programme werden standardmäßig nicht im ASCII-Format

gespeichert. Mit der Option »a« kann dies aber erzwungen werden. Beispiel: »SAVE "BALK-GW", a«. Der Interpreter wird mit »SYSTEM« verlassen.

Turbo-Basic und Quick-Basic sind Basic-Compiler in einer komfortablen Entwicklungsumgebung, ähnlich die der Turbo-Pascal. Zeilennummern sind nur noch optional. Es können echte Unterprogramme (Prozeduren und Funktionen mit lokalen Bezeichnern) erstellt werden. Rekursionen sind ebenfalls möglich. Das Schleifenkonzept wurde um »do/loop until ...« erweitert, was dem Pascal-»repeat/until ...« entspricht. Nach »if then/else« können im Gegensatz zu GWBASIC beliebig viele Anweisungen geschrieben werden.

Bei der Sprache C ist zu beachten, daß im Gegensatz zu den anderen Sprachen die Schreibweise der Bezeichner verbindlich ist. Die reservierten Wörter und die Standardfunktionen sind mit Kleinbuchstaben zu schreiben.

Hinweise:

- Borland hat Turbo-Basic 1990 eingestellt und die Rechte verkauft. Die Weiterentwicklung von Turbo-Basic wird unter dem Namen **PowerBASIC** von Spectra Publishing, USA, vermarktet.
- Mit dem Betriebssystem ab Version 5.0 (1991) liefert Microsoft an Stelle des GWBASIC-Interpreters eine Interpreterversion von Quick-Basic aus, die mit **QBasic** aufgerufen wird. Microsoft vertreibt separat die Compilerversion von Quick-Basic. Ab Windows 95 wird QBasic nicht mehr ausgeliefert.
- Bei Microsoft-Anwenderprogrammen wie *Word für Windows*, *Excel* u.a. dient ein spezieller Basic-Dialekt als Makro-Programmiersprache. Seit "Office 97" sind alle Microsoft-Office-Programme (Word, Excel, Access usw.) mit der Makrosprache Visual Basic for Application ausgestattet.

28.2 Allgemeines zu den Demo-Programmen

Es werden die Auflagerkräfte bei einem auf zwei Stützen gelenkig gelagerten Balken berechnet. Siehe spätere Skizze im Programmteil. Die Anzahl n der Einzelkräfte ist praktisch beliebig. Die Kräfte können innerhalb und außerhalb der Auflager A und B angreifen.

Gegeben sind die äußeren Einzelkräfte F_i (i = 1, 2, 3, ..., n), die Angriffslängen der Einzelkräfte l_i (i = 1, 2, 3, ..., n), gemessen vom linken Auflager aus und der Lagerabstand der Auflager LA. Gesucht sind die Auflagerkräfte F_a und F_b.

Anmerkung: Mit »*LA*« ist nach späterer Skizze der Lagerabstand gemeint. In der Mechanik würde man dafür wahrscheinlich den Großbuchstaben »*L*« verwenden. Pascal unterscheidet bei den Bezeichnern nicht zwischen Groß- und Kleinbuchstaben. Der Bezeichner »*l*« wird aber im Programm für eine indizierte Variable benutzt (Kraftangriffslängen). In Pascal darf der gleiche Bezeichner nicht für indizierte und nicht-

indizierte Variablen benutzt werden, im Gegensatz zu GWBASIC und Turbo-Basic. In der Sprache C wird im Gegensatz zu GWBASIC, Turbo-Basic und Pascal zwischen Groß- und Kleinbuchstaben unterschieden. Mit Ausnahme von Pascal könnte also in den anderen Sprachen statt »*LA*« das »*L*« benutzt werden. Es wird aber einheitlich »*LA*« benutzt.

Einzelkräfte, die von unten kommen, sind mit negativem Vorzeichen zu versehen. Bei Kräften, die links vom Auflager A angreifen, ist die zugehörige Kraftangriffslänge »*l*« mit negativem Vorzeichen zu versehen.

Aus den beiden Forderungen nach Kräftegleichgewicht und Momentengleichgewicht (z.B. um Punkt A) ergibt sich:

```
(1)   Σ F = 0      ——>      (1a)    Σ Fi  - (Fa + Fb)      = 0
(2)   Σ M = 0      ——>      (2a)    Σ (Fi * li) - Fb * LA = 0
```

Führt man zur Abkürzung und im Hinblick auf die Programmierung ein:

```
(3) | SummeF  =  Σ Fi           |   Summe der äußeren Kräfte
(4) | SummeM  =  Σ (Fi * li)    |   Summe der Momente der äußeren Kräfte

    ( i = 1, 2, 3, ..., n )
```

so ergeben sich aus (1a) und (2a) folgende Gleichungen für die gesuchten Auflagerkräfte:

```
(5) | Fa = SummeF – SummeM/LA |   Auflagerkraft bei A
(6) | Fb = SummeF – Fa        |   Auflagerkraft bei B
```

28.3 Zu den Sprachunterschieden in den Demo-Programmen

	Turbo–Pascal	GWBASIC (BASICA)	Turbo–Basic Quick–Basic	Turbo–C
Kommentarbeginn: Kommentarende:	{ oder (* } oder *)	' oder REM (Zeilenende)	' oder REM (Zeilenende)	/* */
Schreibweise g/k verbindlich:	nein	nein	nein	ja
Anweisung ab–schließen mit:	;	: oder RETURN	: oder RETURN	;
Blockbeginn: Blockende:	begin end	(fehlt) (fehlt)	(Unter– streichung)	{ }
Variablen deklarieren:	ja	nein	nein (ja)	ja
Indexklammern:	[ ]	( )	( )	[ ]

	od. (. .)			
Stringkonstanten	'.....'	"....."	"....."	"....."
Zuweisungsoperator:	:=	=	=	=
Gleichheit:	=	=	=	==
Inkrementieren, z.B. z := z + 1	Inc(z)	Z = Z + 1	incr z *)	z++ od. ++z
Dekrementieren:	Dec(z)	Z = Z – 1	decr z *)	z-- od. --z
Bildschirm löschen	ClrScr	CLS	cls	clrscr()
Cursorzeile lösch.:	DelLine	(fehlt)	(fehlt)	delline()
Zeile ab Cursor löschen:	ClrEoL	(fehlt)	(fehlt)	clreol()
Cursor position. Spalte s, Zeile z:	GotoXY(s, z)	LOCATE Z, S	locate z, s	gotoxy(s, z)
Schreiben:	Write(....) od. WriteLn(	PRINT	print	cprintf(...) printf(...)
Daten einziehen:	Read(.....)	INPUT	input	scanf(....)
Ein Zeichen von Tastatur einziehen:	ReadKey	INKEY$	inkey$	getchar()
Bildschirm– helligkeit:	NormVideo HighVideo LowVideo	COLOR v, h	color v, h	normvideo() highvideo() lowvideo()

In Quick-Basic statt »inc z«: z = z + 1
In Quick-Basic statt »dec z«: z = z - 1

28.4 Programmversion 1: Turbo-Pascal

```
program Balk_TP;  { Balken-Auflagerkräfte, Version Turbo-Pascal     }
uses
  CRT;
const
  s    =  15;    { Bildschirm-Spaltenposition }
  nMax = 100;    { Maximal 100 äußere Kräfte. Kann aber im Rahmen
                   der Speicherkapazität beliebig erhöht werden    }
var
  SummeF, LA,
  SummeM, Fa, Fb:  Real;
  F, l:            array[1..nMax] of Real;
  i, n:            Integer;
  Antwort:         Char;
```

```
begin
  repeat              { Beginn der Schleife: repeat ... until }
    ClrScr;           { Bildschirm löschen }
    HighVideo;
    GotoXY(s,  1); Write('Auflagerkräfte: Balken auf 2 Stützen ');
    LowVideo;
    GotoXY(s,  2); Write('--- Version: Turbo-Pascal -----------');
    GotoXY(s,  4); Write('  ‖              ‖   ‖+F          ‖  ');
    GotoXY(s,  5); Write('  v              v   v            v  ');
    GotoXY(s,  6); Write('═══════════════════════════════════');
    GotoXY(s,  7); Write('      ‖ Fa              |    ^    ‖ Fb   ');
    GotoXY(s,  8); Write('   A ═╩═               |    ‖    ═╩═ B   ');
    GotoXY(s,  9); Write('     |<──── +l ─────>|         |      ');
    GotoXY(s, 10); Write('     |<──────── LA ──────────>|      ');

    GotoXY(s, 12); Write('Eingabe Abstand LA der Auflager:     ');
    ReadLn(LA);
    GotoXY(s, 14); Write('Eingabe Anzahl n der äußeren Kräfte: ');
    ReadLn(n);
    GotoXY(s, 16); Write('- Kräfte von unten negativ eingeben  ');
    GotoXY(s, 17); Write('- Links vom Auflager A liegende Kraft');
    GotoXY(s, 18); Write('  angriffslängen l negativ eingeben  ');

    SummeF := 0.0; { Summe der äußeren Kräfte.             Init. }
    SummeM := 0.0; { Summe der Momente der äußeren Kräfte. Init. }

    for  i := 1 to n do
      begin
        GotoXY(1,  20); DelLine;  { delete line: Zeile löschen    }
        GotoXY(s,  20); Write(i, '. Kraft: ');          ReadLn(F[i]);
        GotoXY(34, 20); Write('Kraftangriffslänge: ');  ReadLn(l[i]);
        SummeF := SummeF + F[i];
        SummeM := SummeM + F[i] * l[i];
      end;

    Fa := SummeF - SummeM/LA;
    Fb := SummeF - Fa;
    GotoXY(s, WhereY + 1);  Write('Die Auflagerkraft Fa = ', Fa);
    GotoXY(s, WhereY + 1);  Write('Die Auflagerkraft Fb = ', Fb);

    GotoXY(s, 24); Write('Wiederholung (j/n): ');
    Antwort := ReadKey;

  until (Antwort = 'n');  { Ende der Schleife:  repeat .... until }
end.
```

28.5 Programmversion 2: GWBASIC/BASICA (Nostalgie)

```
100 PROGRAMM$ = "Balk-GW.BAS" '**** Version GWBASIC, BASICA **********
110 '                         'Alle Bezeichner in Großbuchstaben
120 NMAX = 100  'Maximal 100 äußere Kräfte. Kann aber im Rahmen
130             'der Speicherkapazität beliebig erhöht werden.
140 DIM F(NMAX), L(NMAX)    'Dimensionierung der beiden Arrays
150 '
```

```
160 FALSE = 0: TRUE = NOT FALSE
170 WIEDERHOLUNG = TRUE
180     'In GWBASIC bzw. BASICA nur FOR/NEXT- und WHILE/WEND-Schleifen
190 WHILE WIEDERHOLUNG   'Hier Ersatz für Pascal "repeat/until"
200   S = 15             'Bildschirm-Zeile, -Spalte
210   CLS                'Bildschirm löschen
220   COLOR 15, 9        'Statt »HighVideo«.
230   LOCATE  1, S: PRINT "Auflagerkräfte: Balken auf 2 Stützen "
240   COLOR 7, 9         'Statt »LowVideo«.
250   LOCATE  2, S: PRINT "--- Version: GWBASIC, BASICA --------"
260   LOCATE  4, S: PRINT "  ‖              ‖   ‖+F          ‖  "
270   LOCATE  5, S: PRINT "  v              v   v            v  "
280   LOCATE  6, S: PRINT "═══════════════════════════════════════"
290   LOCATE  7, S: PRINT "      ‖ Fa           |   ^    ‖ Fb   "
300   LOCATE  8, S: PRINT "   A ═╩═             |   ‖   ═╩═ B   "
310   LOCATE  9, S: PRINT "     |─── +l ──->|        |        "
320   LOCATE 10, S: PRINT "     |<──── LA ──────────>|        "
330   '
340   LOCATE 12, S: PRINT "Eingabe Abstand LA der Auflager:     ";
350   INPUT "", LA
360   LOCATE 14, S: PRINT "Eingabe Anzahl n der äußeren Kräfte: ";
370   INPUT "", N
380   LOCATE 16, S: PRINT "- Kräfte von unten negativ eingeben  "
390   LOCATE 17, S: PRINT "- Links vom Auflager A liegende Kraft"
400   LOCATE 18, S: PRINT "  angriffslängen l negativ eingeben  "
410   '
420   SUMMEF = 0  'Summe der äußeren Kräfte.             Initialis.
430   SUMMEM = 0  'Summe der Momente der äußeren Kräfte. Initialis.
440   '
450   FOR I = 1 TO N
460        LOCATE 20, 1:      PRINT STRING$(80, " ")
470            'Zeile mit Blanks löschen
480        LOCATE 20, S:      PRINT I; ". Kraft: ";
490        INPUT "", F(I)
500        LOCATE 20, 34:     PRINT "Kraftangriffslänge: ";
510        INPUT "", L(I)
520        SUMMEF = SUMMEF + F(I)
530        SUMMEM = SUMMEM + F(I) * L(I)
540   NEXT I
550   '
560   FA = SUMMEF - SUMMEM / LA
570   FB = SUMMEF - FA
580   '
590   PRINT;
600   LOCATE , S: PRINT "Die Auflagerkraft Fa = "; FA
610   LOCATE , S: PRINT "Die Auflagerkraft Fb = "; FB
620   '
630   LOCATE 24, S, 1: PRINT "Wiederholung (j/n): ";
640   ANTWORT$ = ""
650   WHILE NOT (ANTWORT$ = "j" OR ANTWORT$ = "n")
660      ANTWORT$ = INKEY$
670   WEND: IF ANTWORT$ = "n" THEN WIEDERHOLUNG = FALSE
680 WEND
690 END    'in Basic ist "END" nur optional
```

28.6 Programmversion 3: Turbo-Basic/Quick-Basic

```
Programm$ = "Balk-TB.BAS" '*** Version Turbo-Basic, Power-Basic
                          'oder Quick-Basic. Quick-Basic stellt
                          'Basic-Begriffe in Großschreibung dar.
nMax = 100              'Maximal 100 äußere Kräfte. Kann aber im Rahmen
                        'der Speicherkapazität beliebig erhöht werden.

dim F(nMax), l(nMax)  'Dimensionierung der beiden Arrays
do                      'Beginn der Schleife: do ... loop until
    s = 15              'Bildschirm-Spalte
    cls                 'Bildschirm löschen
    color 15, 9         'Statt »HighVideo«.
    locate  1, s:  print "Auflagerkräfte: Balken auf 2 Stützen "
    color 7 , 9:        'Statt »LowVideo«.
    locate  2, s:  print "--- Version: Turbo-Basic/Quick-Basic "
    locate  4, s:  print "  ‖              ‖   ‖+F          ‖  "
    locate  5, s:  print "  v              v   v            v  "
    locate  6, s:  print "═══════════════════════════════════════"
    locate  7, s:  print "      ‖ Fa            |   ^    ‖ Fb   "
    locate  8, s:  print "   A ═╝               |   ‖    ╚═ B   "
    locate  9, s:  print "     |<──── +l ─────>|        |       "
    locate 10, s:  print "     |<──────── LA ──────────>|       "
    locate 12, s:  print "Eingabe Abstand LA der Auflager:     ";
    input "", LA
    locate 14, s:  print "Eingabe Anzahl n der äußeren Kräfte: ";
    input "", n
    locate 16, s:  print "- Kräfte von unten negativ eingeben  "
    locate 17, s:  print "- Links vom Auflager A liegende Kraft"
    locate 18, s:  print "  angriffslängen l negativ eingeben  "

    SummeF = 0    'Summe der äußeren Kräfte.             Initialis.
    SummeM = 0    'Summe der Momente der äußeren Kräfte. Initialis.

    for i = 1 to n
         locate 20, 1:      print string$(80, " ")
                              'Zeile mit Blanks löschen
         locate 20, s:      print i; ". Kraft: ";
         input "", F(i)
         locate 20, 34:     print "Kraftangriffslänge: ";
         input "", l(i)
         SummeF = SummeF + F(i)
         SummeM = SummeM + F(i) * l(i)
    next i

    Fa = SummeF - SummeM / LA
    Fb = SummeF - Fa

    print
    locate , s: print "Die Auflagerkraft Fa = "; Fa
    locate , s: print "Die Auflagerkraft Fb = "; Fb

    locate 24, s, 1: print "Wiederholung (j/n): ";
    do
       Antwort$ = inkey$
    loop until Antwort$ = "j" or Antwort$ = "n"

loop until Antwort$ = "n"
end   'in Basic ist "end" nur optional
```

28.7 Programmversion 4: Turbo-C

```
#include <conio.h>    /* **** Beginn »Balk-TC.C« (Turbo-C 2.0) **** */
   /* Zu »conio.h«: Header-Datei "Console Input/Output"   */
main()
{                     /* Hier beginnt der main-Block.               */
  float  SummeF, LA,  /* float = Real    */
         SummeM, Fa, Fb,
         F[100],      /* Maximal 100 Kräfte. Kann im Rahmen der     */
         l[100];      /* Speicherkapazität beliebig erhöht werden.  */
  int    i, n;        /* int   = Integer */
  char   Antwort;     /* char  = Char    */
  #define S 15        /* Konstanten üblicherweise in Großschreibung */
  do
    { /* --- Hier beginnt der do-Block --- */
      clrscr();      /* Bildschirm löschen        */
      highvideo();   /* Nur für »cprintf("...")«  */
      gotoxy(S, 1);  cprintf("Auflagerkräfte: Balken auf 2 Stützen ");
      lowvideo();
      gotoxy(S,  2); cprintf("─── Version: Turbo─C 2.0 ──────────────");
      gotoxy(S,  4); cprintf("  ‖              ‖   ‖+F          ‖  ");
      gotoxy(S,  5); cprintf("  v              v   v            v  ");
      gotoxy(S,  6); cprintf("═══════════════════════════════════");
      gotoxy(S,  7); cprintf("      ‖ Fa              │   ^    ‖ Fb   ");
      gotoxy(S,  8); cprintf("   A ═╩═               │   ‖    ═╩═ B   ");
      gotoxy(S,  9); cprintf("     ├──── +l ─────>│        │      ");
      gotoxy(S, 10); cprintf("     │<──────── LA ──────────>│      ");
      gotoxy(S, 12); cprintf("Eingabe Abstand LA der Auflager:     ");
      scanf("%f", &LA);
      gotoxy(S, 14); cprintf("Eingabe Anzahl n der äußeren Kräfte: ");
      scanf("%d", &n);
      gotoxy(S, 16); cprintf("- Kräfte von unten negativ eingeben  ");
      gotoxy(S, 17); cprintf("- Links vom Auflager A liegende Kraft");
      gotoxy(S, 18); cprintf("  angriffslängen l negativ eingeben  ");

      SummeF = 0;  /* Summe der äußeren Kräfte.             Init. */
      SummeM = 0;  /* Summe der Momente der äußeren Kräfte. Init. */

      for  (i = 1; i <= n; i++)
        {
          gotoxy(1, 20);      delline();  /* Zeile löschen  */
          gotoxy(S, 20);      printf("%1d. Kraft: ", i);
          scanf("%f", &F[i]);
          gotoxy(34, 20);     printf("Kraftangriffslänge: ");
          scanf("%f", &l[i]);
          SummeF = SummeF + F[i];
          SummeM = SummeM + F[i] * l[i];
        }

      Fa = SummeF - SummeM/LA;
      Fb = SummeF - Fa;
      gotoxy(S, wherey() + 1);
      cprintf("Die Auflagerkraft Fa = %f", Fa);
      gotoxy(S, wherey() + 1);
      cprintf("Die Auflagerkraft Fb = %f", Fb);
```

```
      gotoxy(S, 24); cprintf("Wiederholung (j/n): ");
      Antwort = getch();
            /* »getch« = get character: Ein Zeichen einlesen   */
    } /* ---- Hier endet der do-Block ---- */
  while (Antwort == 'j');
} /*************************************************************/
```

28.8 Weitere Tropfen auf den heißen C-Stein

Die Unterschiede in der Variablendeklaration zeigt die folgende Gegenüberstellung:

Pascal:	**C:**
`var`	
`i, j, k: Integer;`	`int i, j, k;`
`x, y, z: Real;`	`float x, y, z;`

In Pascal wird streng zwischen Funktionen (liefern Wert zurück) und Prozeduren (liefern keinen Wert zurück) unterschieden. Die Sprache C kennt diese Unterscheidung nicht; dort gibt es nur Funktionen. C-Funktionen die keinen Wert zurückliefern (in Pascal Prozeduren) werden in der Funktionsdeklaration mit einem vorgesetzten **`void`** gekennzeichnet. Funktionen mit Rückgabewert werden in der Deklaration mit dem Ergebnisdatentyp *vor* dem Funktionsbezeichner gekennzeichnet.

Alle Funktionen müssen in C Parameterklammern besitzen, auch wenn keine Parameter übergeben werden. Beispiel: `clrscr()`

In C können Funktionen nicht geschachtelt werden. Die Deklaration von lokalen Funktionen ist im Gegensatz zu Pascal leider nicht möglich.

Weitere Gegenüberstellungen zwischen Pascal und C:

	Pascal:	**C**:	Bemerkungen zu C
Zuweisungsoperator			
	`:=`	`=`	
Blockbildung			
	`begin`	{	
	`end`	}	
Rechenoperatoren			
Multiplikation	`*`	`*`	
Division	`/`	`/`	
Integer-Division	**`div`**	(fehlt)	
Integer-Modulo	**`mod`**	`%`	

Addition	+	+	
Subtraktion	-	-	
Logische Operatoren			
	and	&&	
	or	||	
	not	!	
Bit-Operatoren			
	shl	<<	
	shr	>>	
	and	&	
	or	|	
	xor	^	(Hochpfeil)
	not	~	(Tilde)
Vergleichsoperatoren			
größer als	>	>	
größer oder gleich	>=	>=	
gleich	=	==	
ungleich	<>	!=	
kleiner oder gleich	<=	<=	
kleiner als	<	<	
Verzweigung			
	if *b* **then** *a1* **else** *a2;*	**if** (*b*) *a1;* **else** *a2;*	Bei mehr als einer Anweisung Blockbildung mit { } notwendig

Man beachte, daß im Gegensatz zu Pascal in C das "then" nicht angeschrieben wird und daß auch nach der then-Anweisung, also vor dem "else" ein Semikolon stehen muß, ausgenommen bei Blockbildung mit { und }. Die Bedingungen sind in C immer zu klammern.

Für das **Inkrementieren** und **Dekrementieren** von Variablen stehen in C die Operatoren `++` und `--` zur Verfügung. Sie können *vor* oder *nach* dem Variablenbezeichner stehen, auch im Rahmen einer anderen Anweisung. In diesem Fall wird bei *vorgesetztem* Operator das Inkrementieren bzw. Dekrementieren *vor* dem Ausführen der Anweisung durchgeführt, bei *nachgesetzem* Operator *nach* dem Ausführen der Anweisung.

Beispiele:

```
n = 0;                          n = 0;
printf("%2d", n++);             printf("%2d", ++n);
printf("%2d", n);               printf("%2d", n);
```

```
/*  Ausgabe: 0 1 */        /*  Ausgabe: 1 1 */
```

Zulässig sind auch Kombinationen von Operatoren und dem Zuweisungsoperator. Damit erreicht man eine verkürzte Darstellung. Einige Beispiele:

Normale Darstellung:	**Verkürzte Darstellung:**
`summe = summe + a;`	`summe += a;`
`summe = summe - a;`	`summe -= a;`
`a = a * b;`	`a *= b;`
`a = a / b;`	`a /= b;`
`a = a % b;`	`a %= b;`
`a = a & b;`	`a &= b;`

Zur formatierten Ausgabe:

Für allgemeine Ausgaben (Bildschirm, Drucker und Dateien) dient die Funktion `printf`, für Bildschirmausgaben speziell die Funktion `cprintf` (console print) die einen kompakteren Code erzeugt als `printf`. Darüber hinaus stehen mit `putchar` (Ausgabe Character) und `puts` (Ausgabe String) noch zwei weitere spezielle Ausgabefunktionen zur Verfügung. Bei `puts` wird nach der Ausgabe standardmäßig ein Zeilenvorschub erzeugt, bei allen anderen Funktionen dagegen erst beim Ausdruck eines speziellen Steuerzeichens.

Das Format für `printf` bzw. `cprintf`:

```
printf(formatstring, ausdruck1, ausdruck2, ...);
```

Der Formatstring (Stringkonstante oder Stringvariable) enthält die Formatierangaben. Es können folgende Formatierelemente verwendet werden:

`%u`	unsigned, 16-Bit-Integer ohne Vorzeichen
`%x`	hex, 16-Bit-Integer in Hexadezimal-Schreibweise
`%X`	wie %x
`%d`	decimal, 16-Bit-Integer mit Vorzeichen
`%ld`	long decimal, 32-Bit-Integer mit Vorzeichen
`%f`	float, Fließkommazahl
`%e`	exponential, Fließkommazahl in normierter Exponential-Schreibweise
`%c`	character, ein einzelnes Zeichen
`%s`	string, Zeichenkette
`%p`	pointer, Zeigerwert (Speicheradresse)

Mit Ausnahmen von `%x` und `%X` ist, wie in C allgemein gültig, die Schreibweise verbindlich. Wenn das Prozentzeichen als Textzeichen ausgegeben werden soll, dann ist es zweimal anzuschreiben.

Mit zusätzlichen Integerangaben **zwischen** dem einleitenden Prozentzeichen und dem Formatierelement läßt sich optional die (rechtsbündige) Ausgabebreite festlegen.

Beispiele:

```
printf("%d", 47);
printf("%5d", 11);        /* Schreibbreite 5 Zeichen          */
printf("%5.2f", 47.11);   /* 5 Schreibstellen, 2 Nachkommast. */
```

Der Formatstring muß soviele Formatierangaben enthalten, wie Ausgabeparameter aufgeführt sind. Der Formatstring kann zusätzlich Ausgabetexte enthalten.

Beispiel:

```
i = 3;
printf("Die Wurzel aus %d ist: %5.2f", i, sqrt(i));
/*      Die Wurzel aus 3 ist:  1.73    */
```

Der Formatstring kann zudem an beliebigen Stellen **Steuerzeichen** enthalten, die mit einem Rückwärtsstrich (Backslash, in C das Symbol für Escape) eingeleitet werden. Wenn der Backslash als Textzeichen ausgegeben werden soll, dann ist er zweimal anzuschreiben.

Die wichtigsten Steuerzeichen:

`\n`	LF, line feed, new line, Zeilenvorschub
`\t`	HT, tab, Tabulator
`\f`	FF, form feed, Bildschirm löschen, bei Drucker Seitenvorschub
`\b`	BS, backspace, Cursor (Schreibkopf) eine Zeichen zurück
`\r`	CR, carriage return, Wagenrücklauf
`\ta`	BEL, Piepton
`\v`	VT, vertical tab, vertikaler Tabulator
`\x`*hh*	hex, Darstellung des Zeichens, dessen Code in hex (1 bis 2 Hex-Zeichen *hh*) angegeben ist
`\`*ooo*	oktal, Darstellung des Zeichens, dessen Code in oktal (1 bis 3 Oktal-Zeichen *ooo*) angegeben ist

Beispiel:

```
printf("\n%10s\n%10s\n%10.2f\n", "Anton", "Huber", 47.11);
```

erzeugt die Ausgabe:

```
     Anton
     Huber
     47.11
```

Die momentane Druckzeile wird abgeschlossen oder eine Leerzeile erzeugt, dann wird in der neuen Zeile `Anton` rechtsbündig mit 10 Schreibstellen gedruckt. Anschließend erfolgt wieder ein Zeilenvorschub und das Wort `Huber` wird ebenfalls mit 10 Schreibstellen rechtsbündig in die neue Zeile gedruckt. Abschließend erfolgt nochmals ein Zeilenvorschub und die Zahl `47.11` wird rechtsbündig in ein Feld mir 10 Schreibstellen gedruckt, davon 2 Nachkommastellen.

Zur Eingabe:

Zur Eingabe dient die Funktion scanf, die ähnlich universell ausgelegt ist wie die Funktion printf. Das Format:

```
scanf(formatstring, adresse_var1, adresse_var2, ...);
```

Der Formatstring ist genauso aufzubauen wie bei printf. Die weiteren Parameter von scanf sind die Variablen, auf die einzulesen ist. Allerdings nicht die Variablen selbst, sondern deren Adressen, was in C durch den vorgestellten Adreß-Operator **"&"** bewerkstelligt wird, wenn der Bezeichner nicht bereits ein Zeiger ist.

Beispiel:

```
int a, b;
...
scanf("Eingabe zwei Integer %d %d", &a, &b);
```

28.9 Druckerausgabe in C, Dateien bearbeiten in C

Die folgende Programm "Drucker.C" demonstriert die Druckerausgabe und den Umgang mit Dateien in C:

```
#include <stdio.h>     /* stdio.h  auch Dateioperationen */
main()                 /* Programm »Drucker.C«, Demo Drucker        */
{
  FILE *Dr;                     /*   Siehe Nr 1   */

  Dr = fopen("PRN", "w");       /*   Siehe Nr 2   */

  fprintf(Dr, "Drucker mit »fprintf« = 'file printf' ansprechen\n\n");
  fprintf(Dr, "1. »FILE *Dr;«   Datei-Variable deklarieren     \n  ");
  fprintf(Dr, "   »Dr«          frei gewählter Datei-Bezeichner\n\n");
  fprintf(Dr, "2. »Dr = fopen(''PRN'', ''w'');«                  \n");
  fprintf(Dr, "   »fopen«   File open, Datei öffnen              \n");
  fprintf(Dr, "   »PRN«     Geräte-Dateibezeichner für Printer   \n");
  fprintf(Dr, "   »w«       Write-Modus = Schreiben            \n\n");
  fprintf(Dr, "   Weitere Geräte: »CON« = Console = Bildschirm   \n");
  fprintf(Dr, "   Für Disk-Datei: Dateibezeichner nach MS-DOS    \n");
  fprintf(Dr, "       Beispiel:  C:\\TC\\Haller\\Umsatz.DAT    \n\n");
  fprintf(Dr, "   Weitere Modi:                                  \n");
  fprintf(Dr, "   »r«  = Read. Lesen nicht für Drucker/Bildschirm\n");
  fprintf(Dr, "   »a«  = Append. Anhängen an evtl. bereits vor-  \n");
  fprintf(Dr, "          handene Datei zum Schreiben.           \n");
  fprintf(Dr, "          Ggf. neue Datei.                       \n");
  fprintf(Dr, "   »r+« = Schreiben/Lesen. Datei muß existieren.  \n");
  fprintf(Dr, "   »w+« = Neue Datei für Schreiben/Lesen anlegen. \n");
  fprintf(Dr, "          Evtl. existierende Datei wird gelöscht! \n");
  fprintf(Dr, "   »a+« = Datei zum Lesen öffnen und Anhängen von \n");
  fprintf(Dr, "          Daten durch Schreiben. Ggf. neue Datei\n\n");
  fprintf(Dr, "3. »fclose(Dr);«  : File close, Datei schließen.\n\n");
  fprintf(Dr , "------------------------------------------------ \f");
  fclose(Dr);                /*   Siehe Nr 3   */
}
```