260 069 Problemlösungen in C

Einführung

Was ist Programmierung ?

Generationen von Programmiersprachen

Überblick über gängige Sprachen

Einführung in C

Beispiele

Algorithmen

Einführung in C++

Beispiele

1. Was ist Programmierung?

Umsetzung einer Aufgabenstellung in eine formalisierte Form.
Dazu bedient man sich meist einer Programmiersprache, seltener noch der Hilfe von Hardwareelementen, zunehmend aber der Hilfe graphischer Methoden, deren Ergebnisse mittels eines Programmgenerator zu einen ausführbaren Programm umgewandelt werden.

Der klassische Weg zu einem ausführbaren Programm:

- Editor (einfaches Texteingabe/bearbeitungsprogramm)
- Compiler (Übersetzungsprogramm, das den Quelltext in eine für den Rechner ausführbare Form bringt = Objektcode)
- Linker (Bindeprogramm, das den Objektcode mit anderen Objektmodulen, Bibliotheken, ... zu einem ausführbaren Programm verbindet)

Die Entwicklung eines Programmes:

- Definitionsphase: Problemanalyse, Definition der Anforderungen, ...
- Entwurfsphase: Darstellung des Problems in formalisierter Form
- Implementationsphase: Editieren, Compilieren, Linken, Testen (!)
- Dokumentationsphase: Benutzerhandbuch, ..., Archivierung (Sicherungskopien)

Hilfsmittel für die Entwurfsphase:
Flußdiagramm
Datenflußplan
Struktogramm
...

2. Generationen von Programmiersprachen

Generation	Name	Beispiele
0	Maschinensprache	8088, PDP-11, VAX, ...
1	Assembler	MASM86, MACRO, 370, ...
2	Problemorientierte Sprachen	Fortran, Basic, ...
3	Strukturierte Sprachen	Pascal, C, Fortran 77, Basicdialekte, ...
4	Datenbanksprachen	SQL, ...
5	Logische Sprachen	Prolog, Lisp
„6“	Objektorientierte Sprachen	Smalltalk, C++, Fortran 90, ...
„7“	Graphische Sprachen	Objektvision, ...
„8“	Expertensysteme	„Expert System Shells“
„9“	Linguistische Programmierung	„Fuzzyshells“
„10“	Neuronale Programmierung	ECANSE

```
Programm Example
    for INDEX ← 1 bis 2
        A[INDEX] ← LESE("Bitte eine Zahl")
    while (A[1] ≠ 0) OR (A[2] ≠ 0)
        SUMME ← A[1] + A[2]
        DIFFERENZ ← A[1] – A[2]
        PRODUKT ← A[1] * A[2]
        if A[2] ≠ 0
        then                                  else
        QUOTIENT ← A[1] / A[2]                FEHLER ← TRUE
        FEHLER ← FALSE
        SCHREIBE (A[1], A[2], SUMME, "+")
        SCHREIBE (A[1], A[2], DIFFERENZ, "-")
        SCHREIBE (A[1], A[2], PRODUKT, "*")
        if FEHLER
        then                                  else
        A: "Division durch 0 nicht möglich"   SCHREIBE (A[1],A[2],QUOTIENT, "/")
        for INDEX ← 1 bis 2
            A[INDEX] ← LESE("Bitte eine Zahl")

Prozedur LESE(TEXT)
    A: TEXT
    E: HILFSVAR
    LESE ← HILFSVAR

Prozedur SCHREIBE(A, B, ERG, RZ)
    A: A, "  ", RZ, "  ", B, "  =  ", ERG
```

3. Die 7 Elemente einer Programmiersprache

Element	Basic	C	Fortran	Pascal
Kommentar	REM text	/* text */	C text	{text} (* text *)
Deklaration	Letztes Zeichen im Namen der Variablen	typ name;	typ name	var name:typ;

Ein-/Ausgabe	INPUT liste
	
	PRINT liste
		scanf(for,liste);
		
		printf(for,liste);
			READ(*.*) liste
			
			WRITE (*,*) liste
				read[ln](liste);
				
				write[ln](liste);

Zuweisung	[LET] var=ausdruck	var=ausdruck	var=ausdruck	var:=ausdruck
Verzweigung	IF bed THEN befehl	if(bed) befehl else befehl; switch(auswahl) {case aus1:befehl …}	IF (bed) THEN befehl [ELSE befehl] ENDIF	if bed then befehl else befehl; case var of wert1:befehl; … end;

Schleife	FOR var=anf TO end [STEP sw]
	
	…
	
	NEXT
	
	IF … GOTO
		for (var=anf;bed; änderung var) …;
		
		while(bed) …;
		
		do … while(bed);
			DO l var=a,e,sw
			
			….
			
			l CONTINUE
				repeat ... until bed;
				
				while bed do …;
				
				for var:=anf [down]to end do…;

Unter-programm	GOSUB zeile RETURN	typ name …	typ FUNCTION … SUBROUTINE …	function name … procedure …

Beispielprogramme

BASIC

```
10 INPUT "Hallo, bitte geben Sie eine Zahl ein: ", ZAHL
20 ERG=2*ZAHL
30 PRINT "Das war: ", ZAHL, ". Das Doppelte wäre: ", ERG
```

C

```
main()
{
        float zahl;
        printf ("Hallo, bitte geben Sie eine Zahl ein: ");
        scanf("%f", &zahl);
        erg=2*zahl;
        printf("Das war: %8.3f. Das Doppelte wäre: %8.3f\n", zahl, erg);
}
```

FORTRAN

```
        PROGRAM LITTLE
        REAL ZAHL
        WRITE(6,*) ´Hallo, bitte geben Sie eine Zahl ein: ´
        READ(5,*) ZAHL
        ERG=2*ZAHL
        WRITE(6,100) ZAHL, ERG
100     FORMAT(1X,´Das war: ´,G8.3,´. Das Doppelte wäre: ´,G8.3)
        END
```

PASCAL

```
programm little(input, output);
var zahl, erg:real;
begin
        write('Hallo, bitte geben Sie eine Zahle ein:');
        readln(zahl);
        erg:=2*zahl;
        writeln('Das war: ', zahl:8:3, '. Das Doppelte wäre: ', erg:8:3);
end.
```

BASIC

```
1000  REM IDENT: EXAMPLE.BAS V1.0 written by K. Coufal on 1989 04 14
1010  DIM A(2)
1020  FOR INDEX%=1 TO 2
1030  FRAGE$="Bitte eine Zahl: "
1040  GOSUB 1500
1050  A(INDEX%)=LESE
1060  NEXT INDEX%
1070  WHILE ((A(1)<>0) OR (A(2)<>0)
1080  SUMME=A(1)+A(2)
1090  DIFFERENZ=A(1)-A(2)
1100  PRODUKT=A(1)*A(2)
1110  IF A(2)<>0 THEN QUOTIENT=A(1)/A(2): FEHLER%=0 ELSE FEHLER%=1
1120  ERG=SUMME: RZ$="+"
1130  GOSUB 1600
1140  ERG=DIFFERENZ : RZ$="-"
1150  GOSUB 1600
1160  ERG=PRODUKT : RZ="*"
1170  GOSOB 1600
1180  IF FEHLER% THEN PRINT "Division durch 0 nicht möglich": GOTO 1210
1190  ERG=QUOTIENT : RZ="/"
1200  GOSUB 1600
1210  FOR INDEX%=1 TO 2
1220  FRAGE$="Bitte eine Zahl: "
1230  GOSUB 1500
1240  A(INDEX%)=LESE
1250  NEXT INDEX%
1260  WEND
1270  END

1500  REM Subroutine lese
1510  PRINT FRAGE$;
1520  INPUT LESE
1530  RETURN

1600  REM Subroutine Schreibe
1610  PRINT A(1),RZ$,A(2)," = ",ERG
1620  RETURN
```

C

```
/* IDENT: EXAMPLE.C V1.1 written by K. Coufal on 1996 10 28 */
/* IDENT: EXAMPLE.C V1.0 written by K. Coufal on 1989 04 14 */
#include <stdio.h>
#define TRUE -1
#define FALSE 0
void main()
{
      float a[2],summe,differenz,produkt,quotient,lese();
      int index,fehler;
      void schreibe();
      for (index=0; index<=1; index++) a[index]=lese("Bitte eine Zahl: ");
      while((a[0]!=0) || (a[1]!=0))
      {
            summe=a[0]+a[1];
            differenz=a[0]-a[1];
            produkt=a[0]*a[1];
            if (a[1]!=0)
            {
                  quotient=a[0]/a[1];
                  fehler=FALSE;
            }
            else
                  fehler=TRUE;
            schreibe(a[0],a[1],'+',summe);
            schreibe(a[0],a[1],'-',differenz);
            schreibe(a[0],a[1],'*',produkt);
            if (fehler)
                  printf("Division durch 0 nicht möglich\n");
               else
                  schreibe(a[0],a[1],'/',quotient);
            for (index=0; index<=1; index++)
                  a[index]=lese("Bitte eine Zahl: ");
      }
}

float lese (text)
char *text;
{
      float aux;
      printf("%s",text);
      scanf("%f",&aux);
      return(aux);
}

void schreibe (a,b,rz,erg)
float   a,b,erg;
char    rz;
{
      printf("%8.3f %c %8.3f = %8.3f\n",a,rz,b,erg);
      return;
}
```

FORTRAN

```
C IDENT: EXAMPLE.FOR V1.0 written by K. Coufal on 1989 04 14
      PROGRAM EXAMPLE
      REAL    A(2), SUMME, DIFF, PROD, QUOT, LESE
      LOGICAL FEHLER
      INTEGER INDEX
1000  CONTINUE
      DO      1         INDEX=1,2
      A(INDEX)=LESE(´Bitte eine Zahl: ´)
1     CONTINUE
      IF((A(1) .EQ. 0) .AND. (A(2) .EQ. 0))    GOTO  2000
      SUMME=A(1)+A(2)
      DIFF=A(1)-A(2)
      PROD=A(1)*A(2)
      IF(A(2) .NE. 0)   THEN
                  QUOT=A(1)/A(2)
                  FEHLER=.FALSE.
            ELSE
                  FEHLER=.TRUE.
      ENDIF
      CALL  SCHREIBE(A(1),A(2),´+´,SUMME)
      CALL  SCHREIBE(A(1),A(2),´-´,DIFF)
      CALL  SCHREIBE(A(1),A(2),´*´,PROD)
      IF (FEHLER) THEN
                  WRITE(6,*)  ´Division durch 0 unmöglich´
            ELSE
                  CALL  SCHREIBE(A(1),A(2),´/´,QUOT)
      ENDIF
      GOTO  1000
2000  CONTINUE
      END

REAL  FUNCTION          LESE(TEXT)
CHARACTER*(*)     TEXT
WRITE(6,*)        TEXT
READ(5,*)         LESE
END

SUBROUTINE        SCHREIBE(A, B, RZ, ERG)
REAL              A,B,ERG
CHARACTER*1       RZ
WRITE(6,100)      A,RZ,B,ERG
100   FORMAT(1X,F8.3,1X,A1,1X,F8.3,´ = ´,F8.3)
END
```

PASCAL

```
{ident: example.pas V1.0 written by K. Coufal on 1989 04 14}
program example(input, output);
type zeile=string[20];
var  a:array [1..2] of real;
     summe, differenz, produkt, quotient:real;
     fehler:boolean;
     index:integer;
function lese(text:zeile):real;
var  hilfsvar:real;
begin
      write(text);
      readln(hilfsvar);
      lese:=hilfsvar;
end;
procedure schreibe(a,b,erg:real;rz:char);
begin
      writeln(a:8:3,´ ´,rz,´ ´,b:8:3,´ = ´,erg:8:3);
end;
begin
      for index:=1 to 2 do a[index]:=lese(´Bitte eine Zahl: ´);
      while (a[1]<>0) or (a[2]<>0) do begin
            summe:=a[1]+a[2];
            differenz:=a[1]-a[2];
            produkt:=a[1]*a[2];
            if a[2]<>0 then begin
                              quotient:=a[1]/a[2];
                              fehler:=false;
                        end
                  else
                              fehler:=true;
            schreibe(a[1],a[2],summe,´+´);
            schreibe(a[1],a[2],differenz,´-´);
            schreibe(a[1],a[2],produkt,´*´);
            if fehler then
                        writeln(´Division durch 0 nicht möglich´)
                  else
                        schreibe(a[1],a[2],quotient,´/´);
            for index:=1 to 2 do a[index]:=lese(´Bitte eine Zahl: ´);
      end;
end.
```

4. Detailliertere Einführung in C

4.1. Datentypen und Umwandlung

4.1.1. elementare Datentypen

alphanumerische Datentypen	numerische Datentypen		
char	ganze Zahlen	Gleitkommazahlen	
	int	einfach genau	doppelt genau
		float	double

a.) char

1 Byte Speicherplatz, 1 Zeichen aus ASCII (ANSI, ...)

b.) int

rechnerabhängiger Speicherplatz, meist 2 oder 4 Byte

short int	meist 16 Bit (manchmal auch 32 Bit)	-32768..+32767
long [int]	4 Byte	-2 147 483 648..+2 147 483 687
unsigned [int]	2 Byte ohne Vorzeichen	0..65535

c.) float

4 Byte $\pm 10^{-37}..\pm 10^{+38}$ mit ca. 6 Nachkommastellen ($8.43*10^{-37} \leq |x| \leq 3.37*10^{+38}$)

long float = double

d.) double

8 Byte $\pm 10^{-307}..\pm 10^{+308}$ mit ca. 14 Nachkommastellen ($4.19*10^{-307} \leq |x| \leq 1.67*10^{+308}$)

4.1.2. Konstanten

char	'A'	'+'	'0'		
(short) int	255	-327			
long int	-1234567	8L	-9L		
float	3.1415926	0.27	12.5e-3	-125E-6	.5
double	9.758443L	11.0E-2L	3.1415926535L		
Mit 0 beginnend Oktalzahl			0101	01234	
Mit 0x oder 0X beginnend Hexadezimalzahl			0x27	0xABCD	

4.1.3. Umwandlung von Datentypen

a.) implizite Typumwandlung

1. Regel	Es wird mit nichts kleinerem als int gerechnet
2. Regel	Bei einer Zuweisung wird automatisch in den Datentyp des Ergebnisses konvertiert
3. Regel	Bei einer Gleitkommaoperation wird immer double gerechnet
4. Regel	Sind die Operanden eines Ausdruckes von unterschiedlichen Typ, wird mit dem speicheraufwendigsten Typ gerechnet
5. Regel	Ist einer der verwendeten Operanden vom Typ unsigned, wird der andere Operand ebenfalls in unsigned umgewandelt

b.) explizite Typumwandlung

wird über sogenannte „casts“ erreicht

(Typ) Ausdruck

z.B: (long)variable

4.1.4. Vereinbarung von Variablen

Variablenname: Buchstaben (keine Umlaute und ß), Ziffern und Underscore („_“)

1. Zeichen: Keine Ziffern

Variablennamen können im Prinzip beliebig lang sein, jedoch sind nur die ersten acht Zeichen (compilerabhängig) signifikant

Klein- und Großbuchstaben werden vom Compiler unterschieden, oftmals aber nicht vom Linker!

In C üblich: Variablennamen in Kleinbuchstaben

Symbolische Konstanten in Großbuchstaben

C-Befehle dürfen nicht als Variablennamen verwendet werden.

Schlüsselworte: auto, break, case, char, continue, default, do, double, else, entry, enum, extern, float, for, goto, if, int, long, register, return, short, sizeof, static, struct, switch, typedef, union, unsigned, viod, while

Bei machen Compilern noch zusätzliche möglich!

Vereinbarung von Variablennamen (Deklaration):

datentyp variablenname_1, ..., variablenname_n;

z.B.: int zaehler, index, auxiliary;

char antwort, zeichen;

float zahl1, zahl2;

double pi;

long int lindex;

4.2. Ausdrücke und Operatoren (siehe auch Anhang: Priorität der Operatoren)

4.2.1. einfacher Zuweisungsoperator

variable=ausdruck;

zaehler=0;

4.2.2. arithmetische Operatoren

+ Addition * Multiplikation % Modulo (Rest einer Ganzzahldivision)

- Subtraktion / Division

z.B.: flaeche=a*b;

umfang=2*(a+b);

dabei gilt: Punktrechnung vor Strichrechnung; bei Gleichheit wird von links nach rechts abgearbeitet, Klammern sind aber möglich.

4.2.3. Vergleichsoperatoren

> größer als	< kleiner als	== gleich
>= größer gleich	<= kleiner gleich	!= ungleich

Ergebnis TRUE oder FALSE

4.2.4. Logische Operatoren

! Negation	~ Einerkomplement
&& Und-Verknüpfung	& bitweises UND
|| Oder-Verknüpfung	| bitweises ODER
<< Bit nach links schieben	>> Bit nach rechts schieben
^ bitweises XOR	

4.2.5. Zusammengesetzte Zuweisungsoperatoren

„var operator= ausdruck“ steht kurz für „var = var operator ausdruck“

operator: + - * / % << >> & ^ | (z.B.: index+=2 bedeutet index=index+2)

4.2.6. Inkrement und Dekrement Operator

++

--

Zwei Arten Präfix-Schreibweise: höchste Priorität

```
var1 = ++var2 ⇔     var2 = var2 + 1;
                    var1 = var2;
```

Postfix oder Suffix-Schreibweise: niedrigste Priorität

```
var1 = var2++ ⇔     var1 = var2;
                    var2 = var2 + 1;
```

4.3. I/O (Ein-/Ausgabe)

4.3.1. include

#include <stdio.h>	Systemincludefile
#include "stdio.h"	Benutzerincludefile

4.3.2. Ein Zeichen einlesen

getchar() liest von stdin (CON, Tastatur)

4.3.3. Ein Zeichen ausgeben

putchar(...) schreibt auf stdout (CON, Bildschirm)

4.3.4. Ausgabefunktion printf

printf("kontrollzeichenkette", argument1, ..., argumentn);

Kontrollzeichenkette=Text+Sonderzeichen+Formatangaben

Sonderzeichen:

\0	Ascii 0
\´	einfaches Anführungszeichen
\\	\
\b	Backspace
\f	Formfeed
\n	Neue Zeile
\r	Carriage Return
\t	Tabulator
\xxx	Oktalcode xxx

Formatangaben: %[F][n][.m][l]U

F Formatangabe

- linksbündig

rechtsbündig (keine Angabe)

n Mindestanzahl von Stellen des Ausgabefeldes

Auffüllen erfolgt mit Leerzeichen außer die Stellenanzahl beginnt mit 0 dann wird mit „0“ aufgefüllt.

m nur bei U=s,e,f,g

s Maximale Stellen für die Ausgabe

e,f,g Anzahl der Nachkommastellen

l Long

U Interpretation des zugehörigen Arguments

c char

d dezimal

e dezimale Gleitkommazahl mit Exponent

f dezimale Gleitkommazahl ohne Exponent

g kürzere Darstellung aus e und f

o Oktalzahl

s Zeichenkette

u dezimale ganze Zahl ohne Vorzeichen

x Hexadezimale Zahl

%% Ausgabe eines %-Zeichens

4.3.5. Eingabefunktion scanf

scanf("Kontrollzeichenkette",argument1, ..., argumentn)

Kontrollzeichenkette=Text+Formatangaben

Text muß bei der Eingabe angegeben werden; Leerzeichen und Tabulatoren werden aber ignoriert

Formatangabe: %[l]U mit U=c,d,e,f,n,o,s,x

n short int

Rest siehe printf

argumente sind Zeiger auf Variable

4.4. Programmaufbau

```
/* Kommentar */
Preprocessoranweisungen
[void|int ]main([argc,argv])
[Deklaration für argc und argv]
{
        [Deklarationen]
        Programmanweisungen
}
z.B:
        /* ID: EXAMPLE01.C V1.0 written by K. Coufal on 1996 10 31 */
        /* Erstes Beispielprogramm für C;
                Programm produziert nur eine einfache Ausgabe */
        #include <stdio.h>
        void main()
        {
                printf(„Hallo C-Programmierer\n“);
        }
```

4.5. Preprocessor

4.5.1. Allgemeines

Arbeitet vor der eigentlichen Programmübersetzung, d.h. es kann noch auf den Übersetzungsvorgang Einfluß genommen werden. In der ersten Spalte einer Anweisung für den Preprocessor muß ein # sein! (Fortsetzungen sind mit einem \ in der letzten Spalte der fortzusetzenden Zeile möglich.) Die Anweisungen an den Preprocessor gehören daher nicht zum C-Code und unterliegen daher auch nicht der C-Syntax.

4.5.2. #define

a.) Konstante festlegen

```
#define name Zeichenkette
z.B.:
        #define PI 3.1415926535
        #define SIZE 100
```

b.) Makros festlegen

```
#define makro zeichenkette
z.B.:
        #define klein(c) (c|0x20)
        #define max(A,B) ((A)>(B)?(A):(B))
        #define groß(c) (c&0x5F)
        #define ctrl(c) (c&0x1F)
```

4.5.3.#include

a.) #include „Dateiname“

Dateiname bezeichnet „lokale“ Datei im eigenen Arbeitsverzeichnis

b.) #inlcude <Dateiname>

Dateiname bezeichnet Datei in einem Systemverzeichnis

4.5.4 #if

#if konstanter_ausdruck

folgende Zeilen werden nur übersetzt, wenn konstanter_ausdruck=0 ist

z.B.:

```
#define MSDOS 1
...
#if MSDOS
...
#endif
```

4.5.5 #ifdef

#ifdef name

folgende Zeilen werden nur übersetzt, wenn name zuvor definiert wurde

z.B.:

```
#define SIZE 100
...
#ifdef SIZE
....
#endif
```

4.5.6. #ifndef

#ifndef name

folgende Zeilen werden nur übersetzt, wenn name zuvor nicht definiert wurde

z.B.:

```
#ifndef SIZE
#define SIZE 10
#endif
```

4.5.7. #else

#else

ermöglicht bei #if, #ifdef und #ifndef einen „Sonst“-Zweig

z.B.:

```
#if MSDOS
        ...
#else
        ...
#endif
```

4.5.8. #endif

#endif

schließt #if, #ifdef, #ifndef und #else ab.

4.5.9. #line

#line konstante dateiname

Ermöglicht Programmgeneratorem und „Precompilern“ die korrekten Bezugspunkte für das „Debuggen“ zu setzen.

4.6. Anweisungen und Blöcke

Ein C-Programm soll nur aus einer Anweisung bestehen, um eine möglichst einfache Syntax zu haben. Daher kann man in C mehrere Anweisungen mittels { und } zu einem Block zusammenfassen, der wie eine einzige Anweisung behandelt wird.

4.7. Verzweigungen

4.7.1. if

if (ausdruck) anweisung1; [else anweisung2;]

(ausdruck ist i.a. eine Bedingung, deren Ergebnis true (!=0) oder false (=0) sein kann)

z.B.:

```
if (tag==heute) printf("Eingegebener Tag hat heutiges Datum");

if (zahl!=0)
        {
                printf(“Zahl war ungleich 0\n“);
                printf(“Ergebnis ist %d10.5 \n“, nenner/zahl);
        }
else
        {
                printf(“Zahl war 0. Bitte eine andere Zahl eingeben: “);
                scanf(“%d“, &zahl);
        }
```

else gehört immer zum nächstfrüheren if, das noch keinen else-Zweig hat!

4.7.2 Bedingte Bewertung

ausdruck1?ausdruck2:ausdruck3

Bewertung von ausdruck1 (i.a. Bedingung); wenn true dann wird ausdruck2 verwendet, sonst ausdruck3.

z.B.:

```
if (a>b) max=a; else max=b;     ⇔     max=a>b?a:b;
```

4.7.3. switch

```
switch (ausdruck)
{
        case ausdruck1:anweisungen1;
        case ausdruck2:anweisungen2;
        ...
        case ausdruckn:anweisungenn;
        default:anweisungen;
}
```

wenn ausdruck == ausdruckx ist, wird an dieser Stelle fortgefahren! D.h. auch alle Anweisungen dahinter werden ausgeführt; Ende der switch-Anweisung mit „break“.

z.B.:

```
switch (hex_ziffer)
{
        case ´0´:printf(“0000“);
                break;
        case ´1´:printf(“0001“);
                break;
        ...
        case ´a´:
        case ´A´:printf(“1010“);
                break;
        ...
        case ´f´:
        case ´F´:printf(“1111“);
                break;
        default:printf(“Keine Hexziffer“);
                break;
}
```

4.8. Zeiger

Variable belegen Platz im Speicher, wobei der Name eine symbolische Adresse dieses Speicherplatzes ist, der im ausführbaren Programm durch die wirkliche Adresse ersetzt wird. In C gibt es die Möglichkeit spezielle Variablen, die nicht Werte sondern Adressen beinhalten, zu definieren. Diese Variablen zeigen daher auf andere Variablen und werden deswegen Zeiger genannt. Zeiger werden bei der Deklaration durch Voranstellen eines ***** vor dem Zeigernamen gekennzeichnet.

```
int  *zeig_1;
double  *zeig_2;
char  *zeig_3;
```

Im Programm sind die Zeigernamen ohne * anzugeben, wenn sie als Zeiger funktionieren sollen. Mittels des Adreßoperators **&** kann die Adresse einer Variablen verwendet werden:

```
zeiger=&var_1;
```

Die Variable zeiger enthält nun die Adresse von var_1.

Der Verweisoperator * ermöglicht eine Veränderung der Variablen, deren Adresse die Zeigervariable beinhaltet:

```
*zeiger=7;        =        var_1=7;
```

Beispielprogramm:

```
void main()
{
        char zeich_1, zeich_2, *zeig_1, *zeig_2;
        zeich_2=´b´;
        zeich_1=´a´;
        zeig_1=&zeich_1;
        zeig_2=&zeich_2;
        printf("%c",*zeig_1);
        printf("%c",*zeig_2);
        *zeig_2=´c´;
        printf("%c\n",zeich_2);
        zeig_1=zeig_2;
        printf("%c",*zeig_1);
        *zeig_1=´b´;
        printf("%c",*zeig_2);
        zeig_2=&zeich_1;
        zeig_1=zeig_2;
        printf("%c\n",*zeig_1);
}
```

4.9. Vektoren

Ein Vektor ist eine Zusammenfassung gleicher Datenelemente unter einem Namen

z.B.: Zeichenketten

```
char zeile[80];
...
zeile[0]='a';
zeile[10]='e';
zeile[79]='0';
```

Mit Hilfe von Zeigern:

```
int vektor[3], *zeiger;
...
zeiger=&vektor[0];      (auch zeiger=vektor, da Name gleich der Anfangsadresse)
*(zeiger+1)=wert;       =       vektor[1]=wert;
```

Erlaubt: Addition und Subtraktion von int Konstanten oder int Variablen

Subtraktion oder Vergleich zweier Zeiger (Rest ist nicht erlaubt: z.B.: Addition von Zeigern)

```
*(zeiger+2)=*zeiger+*(zeiger+1)       =       vektor[2]=vektor[0]+vektor[1];
```

= 2*Länge eines Elements des Vektors

Zeichenketten:

```
char *zeile;
zeile="Hallo, das ist C"; (solche Zeichenketten werden mit \0 abgeschlossen)
```

Mehrdimensionale Vektoren sind ebenfalls möglich:

```
float matrix[3][3];
```

Initialisierungen:

```
int     i=0;
double pi=3.1415926535;
char *fehler="Fehler bei der Eingabe";
float matrix[3][3]={
                        {1,0,0}
                        {0,1,0}
                        {0,0,1}
                    };
```

4.10. Strukturen

Eine Struktur ist eine Zusammenfassung von Daten verschiedenen Typs zu einem Element im Gegensatz zu Vektoren, die aus Daten gleichen Typs bestehen.

z.B.:
```
struct  studentenblatt {
        char nach_name[30];
        char vor_name[20];
        char geschlecht;
        char studienkennzahl[4];
        long matrikelnummer;
        int note;
        };
struct studentenblatt student;
```

Verwendung mittels Strukturoperators

```
student.nach_name="Habicht";
student.vor_name="Hugo";
....
```

Mit Zeigern

```
struct  studentenblatt *zeiger;
zeiger=&student;
(*zeiger).nach_name="Habicht";      =   zeiger->nach_name="Habicht";
(*zeiger).vor_name="Hugo";          =   zeiger->vor_name="Hugo";
```

4.11. Schleifen

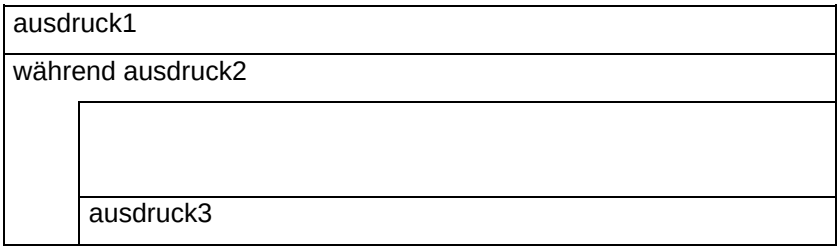
4.11.1. for

for (ausdruck1; ausdruck2; ausdruck3) anweisung;

ausdruck1: Initialisiert Schleifenvariable

ausdruck2: Abbruch, wenn dieser Ausdruck falsch ist

ausdruck3: verändert Schleifenvariable

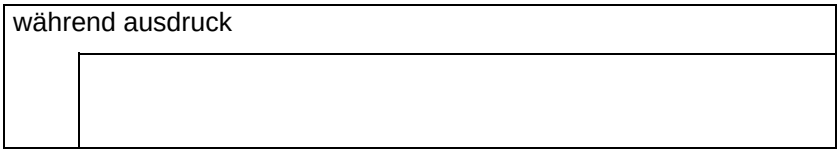


z.B.: for (i=1; i<5; ++i) printf("%d",i); ⇒ 1234

4.11.2. while

while (ausdruck) anweisung;

ausdruck: Abbruch, wenn ausdruck falsch ist

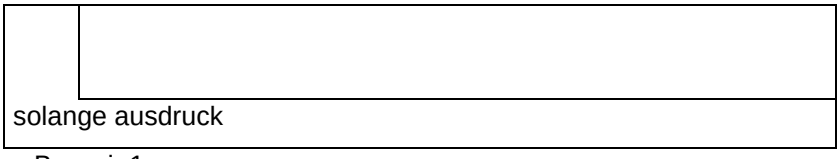



```
z.B.:   i=1;
        while(i<5)
        {
                printf("%d",i);
                ++i;
        }
```

4.11.3. do..while

do anweisung; while (ausdruck);

ausdruck Abbruch, wenn dieser ausdruck falsch ist




```
z.B.:   i=1;
        do
                printf("%d",i);
                ++i;
        while(i<5);
```

4.11.4. break

Die break-Anweisung dient dem vorzeitigen Verlassen einer for, while, do oder switch-Anweisung.

Syntax: break;

Dabei wird bei geschachtelten Schleifen nur die momentane verlassen.

4.11.4. continue

Die continue-Anweisung bricht nur den momentanen Durchlauf einer Schleife ab und beginnt den nächsten Durchlauf derselben Schleife (for, while und do).

Syntax: continue;

z.B.:
```
i=1;
while(i<5)
{
        printf("%d",i);
        ++i;
}
```

liefert 1234, und

```
i=1;
while(i<5)
{
        if(i==3)                    VORSICHT:
        {                           if(i==3) continue;
                ++i;                ⇒
                continue;           Endlosschleife, da sich i nicht
        }                           mehr ändert
        printf("%d",i);
        ++i;
}
```

liefert 124.

4.11.5. goto

Eine andere Art Schleifen aufzubauen bzw. die Programmlogik zu steuern, ist mit der Anweisung goto möglich. Dabei treten leider häufig Verletzungen der strukturierten und modularen Programmierung auf, daher ist das goto nach Möglichkeit zu vermeiden.

Syntax:

```
goto label;
...
label: (anweisung;)
```

z.B.:
```
i=1;
loop:
        printf("%d",i);
        ++i;
if (i<5) goto loop;
```

4.12. Funktionen

4.12.1. Allgemeines

Für immer wiederkehrende Aufgaben werden in allen Programmiersprachen Unterprogramme verwendet. In C nennt man solche Konstrukte „functions“ (z.B.: printf, scanf).

Bei der Verwendung von Funktionen müssen zwei Arten von Parametern unterschieden werden:

a.) formale Parameter werden für die Definition von Funktionen benötigt

b.) aktuelle Parameter werden beim Aufruf einer Funktion verwendet.

z.B.:

```
definition gelange(ziel)
        suche kürzesten Weg nach ziel ——— formaler Parameter
        gehe nach ziel
end definition
Verwendung:
        gelange(Institut) ——————————— aktueller Parameter
```

4.12.2. Definitionen von Funktionen

```
[ergebnistyp] funktionsname ([Liste der formalen Parameter])        Funktions=
[Deklaration der formalen Parameter]                                kopf
{
        [Deklarationen]                                             Funktions=
        Anweisungen                                                 bereich
        [return(Ergebnis)]
}
```

z.B.:

```
void main()
{
        float zahl1, zahl2, mittel();
        printf(“Bitte zwei Zahlen eingeben: “);
        scanf(“%f %f“, &zahl1, &zahl2);
        prinf(“Der Mittelwert ist: %8.3f“, mittel(zahl1, zahl2));
}
float mittel(a,b)
float a,b;
{
        float c;
        c=(a+b)/2;
        return(c);
}
```

Parameter werden i.a. als Wert übergeben („call-by-value“ im Gegensatz zu „call-by-reference“) z.B.:

```
void main()
{
        float zahl, doppel();
        scanf(“%f“, &zahl);
```

```
        doppel(zahl);
        printf("Das Doppelte ist %f\n", zahl);
}
float doppel(zahl)
float zahl;
{
        zahl *= 2;
        return(zahl);
}
```

führt zu falscher Ausgabe

Variablen von Funktionen haben i.a. nur für die Dauer der Funktion Gültigkeit, d.h. wenn diese Funktion nochmals aufgerufen wird, haben alle Variablen wieder undefinierten Wert. Beispiel:

```
void main()
{
        int zahl, index:
        scanf("%d", &zahl);
        for (index=1; index<=10; index++)
                zaehler;
}
int zaehler
{
        int value=0;                    Lösung: static int value=0;
        printf("%d ", ++value);
}
```

4.12.3. Rekursion

Funktionen in C können sich selbst wieder direkt oder über andere Funktionen aufrufen. Z.B.:

```
void main()
{
        int zahl;
        long fak();
        printf("Faktorielle berechnen für : ");
        scanf("%d", &zahl);
        printf("%d! = %ld\n",zahl,fak(zahl));
}
long fak(zahl)
int zahl;
{
        long fak(),ergeb;
        if (zahl>0)
                ergeb=zahl*fak(zahl-1);
        else
                ergeb=1;
        return(ergeb);
}
```

4.13. Dateien

4.13.1. FILE

Da die Bearbeitung von Dateien vom Betriebssystem abhängt, gibt es in C dafür keine Befehle, allerdings existieren auch dafür Bibliotheksfunktionen. Die dafür notwendigen Datentypen werden ebenfalls in stdio.h definiert.

In C existiert ein eigener Datentyp FILE, mit dem ein Zeiger festgelegt wird, mit dessen Hilfe dann die Verwaltung und Verarbeitung von Dateien durchgeführt wird.

```
FILE *dateivariable
```

4.13.2. Dateien öffnen

Um eine Datei verwenden zu können, muß dem „Dateizeiger“ ein Dateiname (OS-Konventionen) zugeordnet und die Datei „geöffnet“ werden. In C erfolgt dies mit fopen:

```
dateivariable=fopen(dateiname, modus)
```

mit modus	“r“	...	Lesezugriff (read)
	“w“	...	Schreibzugriff (write, vorhandene Datei wird gelöscht)
	“a“	...	Schreibzugriff am Dateiende (append)
	“r+“	...	Dateiupdate (read and write)
	“w+“	...	Dateiupdate (write, vorhandene Datei wird gelöscht)
	“a+“	...	Dateiupdate am Dateiende (append)

wenn die Datei nicht existiert, dann ist dateivariable=NULL (stdio.h).

Drei Dateiobjekte existieren automatisch:

stdin	0	CON:	STanDard Input
stdout	1	CON:	STanDard OUTput
stderr	2	CON:	STanDard ERRor

4.13.3. Dateien schließen

Da nicht beliebig viele Dateien offen sein können und um Fehler zu vermeiden, sollten nur Dateien geöffnet sein, die unmittelbar benötigt werden. Dateien werden mittels fclose geschlossen:

```
fclose(dateivariable)
```

fclose liefert 0, wenn alles in Ordnung ist und -1 bei einem Fehler.

4.13.4. Zeichenweise lesen/schreiben

zeichen=getc(dateivariable)	(zeichen==EOF bei Dateiende)
zeichen=fgetc(dateivariable)	(zeichen==EOF bei Dateiende)
putc(zeichen, dateivariable)	(Ergebnis=zeichen oder EOF bei Fehler)
fputc(zeichen, dateivariable)	(Ergebnis=zeichen oder EOF bei Fehler)
ungetc(zeichen,dateivariable)	stellt Zeichen in den Eingabepuffer zurück

```
/*
#define getchar() getc(stdin)
#define putchar(zeichen) putc (zeichen, stdout)
*/

/* Worte bearbeiten:
        integer=getw(dateivariable)
        putw(integer,dateivariable)
*/

/* Fehlerbehandlung:
        feof(dateivariable)         wahr bei EOF
        ferror(dateivariable)       wahr bei EOF und Fehlern
        clrerror(dateivariable)     Fehlerbedingung löschen
*/
```

4.13.5. Zeichenketten lesen/schreiben

fgets(vektor, maxzeichen, dateivariable)

Liest bis maxzeichen-1 Zeichen oder bis \n in vektor ein (Zeichenkette wird automatisch mit \0 abgeschlossen; Ergebnis NULL(=Zeiger) bedeutet EOF)

fputs(vektor, dateivariable)

gets(vektor) (\n wird im Gegensatz zu fgets mit \0 überschrieben)

puts(vektor) (gets und puts arbeiten mit stdin bzw. stdout)

z.B.:

```
#include <stdio.h>
#define MAX 100
void main()
{
        char    vekt[MAX];
        char    dateiname[15];
        FILE    *dateizeiger;
        puts("\nGeben Sie einen Dateinamen ein !\n");
        gets(dateiname);
        if ((dateizeiger=fopen(dateiname, "r")) == NULL)
            printf("\nDatei %s konnte nicht geöffnet werden\n",dateiname);
        else
            while (fgets(vekt,MAX,dateizeiger) != NULL) puts(vekt);
}
```

4.13.6. Blöcke lesen/schreiben

fread(vektor, byte_zahl, block_zahl, dateivariable)

fwrite(vektor, byte_zahl, block_zahl, dateivariable)

mit

vektor	...	Startadresse im Speicher
byte_zahl	...	Anzahl der Bytes / Block
block_zahl	...	Anzahl der zu lesenden/schreibenden Blöcke

4.13.7. Formatierte I/O

printf(kontrollzeichenkette, parameterliste)

fprintf(dateivariable, kontrollzeichenkette, parameterliste)

sprintf(zeichenvektor, kontrollzeichenkette, parameterliste)

scanf(kontrollzeichenkette, parameterliste)

fscanf(dateivariable, kontrollzeichenkette, parameterliste)

sscanf(zeichenvektor, kontrollzeichenkette, parameterliste)

Syntax für kontrollzeichenkette und parameterliste siehe 4.3.

fprintf schreibt auf die Datei, die mit dateivariable verbunden ist

fscanf liest von der Datei, die mit dateivariable verbunden ist

sprintf schreibt in die Zeichenkette zeichenvektor

sscanf liest von der Zeichenkette zeichenvektor

4.13.8. „Random Access“-Dateien

Bisher wurden nur sequentielle Dateien besprochen, d.h. Dateien mit denen die Information sequentiell verarbeitet wird. Allerdings spielen bei größeren Dateien und Anwendungen „Random Access“-Dateien d.h. Dateien, wo die Information auch nicht sequentiell verarbeitet werden kann, ein größere Rolle. In C wird der wahlfreie Zugriff durch Verschieben des Zeigers auf ein anderes Element „simuliert“.

fseek(dateivariable, offset, modus)

mit

dateivariable	...	Zeiger für die Datei	
offset	...	Anzahl der Bytes (longint)	
modus	...	0	Vom Anfang offset Bytes
		1	von aktueller Position offset Bytes
		2	vom Ende offset Bytes

4.13.9. Elementare Lese- und Schreiboperationen

Neben den erwähnten Routinen existieren je nach Compiler und Betriebssystem noch eine Reihe weiterer Funktionen, insbesondere elementare Dateifunktionen, die ein Programmieren auf unterster Stufe ermöglichen:

VT: höchste Flexiblität, Inhalt egal, Steuerzeichen sind selbst verwaltbar
NT: die Dateiverwaltung muß selbst programmiert werden

z.B.: read, write, open, creat, close, unlink, lseek, ...

4.14. Parameter vom Betriebssystem

z.B: PROGRAM parameter1 parameter2

Wie kann man im Programm PROGRAM die Parameter parameter1 und parameter2 verarbeiten?

Da main genau wie andere Funktionen behandelt wird, kann hier ebenfalls eine Parameterübergabe eingesetzt werden:

```
main (argc, argv)
int argc;                    Anzahl der Parameter
char *argv[];                Parameter als Feld von Zeichenketten
```

z.B.:

```
void main(argc,argv)
int argc;
char *argv[];
{
        int zaehl;
        for (zaehl=0; zaehl<argc; zaehl++)
                printf("\nWort%d : %s", zaehl, argv[zaehl]);
}
```

4.15. Zeiger auf Funktionen

Wenn an eine Funktion ein Funktion als Parameter übergeben werden soll, ist dies in C wie folgt möglich:

```
...
float rechnung(funktion,zahl)          Aufruf mit:
float zahl;                            rechnung(doppel,7);
float (*funktion)();
{                                      => return(10+doppel(7));
        return(10+(*funktion)(zahl));  => 24
}
float doppel(p)
float p;
{
        return (p*2);
}
```

4.16. TYPEDEF

Mittels typedef können neue Datentypen festgelegt werden (d.h. eigentlich keine neuen Typen, sondern nur neue Namen für bestehende Typen).

z.B.: `typedef int GANZZAHL` (⇔ #define GANZZAHL int)

`typedef int NOTEN[12]`

(z.B.: NOTEN A,B ⇔ int A[12],B[12])

vor allem bei Strukturen ist dies von Vorteil

z.B.:

```
struct person                    typedef struct person
{                                {
    char name[40];                   char name[40];
    int personalnr;                  int personalnr;
}                                } STRUKTUR
...
struct person mitarbeiter;       STRUKTUR mitarbeiter;
```

4.17. Gültigkeit von Variablen

4.17.1. Gültigkeitsbereich

a.) lokal innerhalb eines Blocks

b.) modulglobal innerhalb einer Datei

c.) programmglobal innerhalb eines Programms (u.U. mehrere Dateien)

4.17.2. Lebensdauer

a.) static

b.) automatic

4.17.3. Speicherort

a.) Hauptspeicher

b.) Register

4.17.4. Schlüsselworte

extern (4.17.1.c, 4.17.2.a, 4.17.3.a)

auto (4.17.1.a, 4.17.2.b, 4.17.3.a)

static (4.17.1.a, 4.17.2.a, 4.17.3.a) innerhalb der Funktion

(4.17.1.b, 4.17.2.a, 4.17.3.a) außerhalb der Funktion

register (4.17.1.a, 4.17.2.b, 4.17.3.b)

4.18. Bibliotheksfunktionen

Funktion	Ergebnistyp	Beschreibung
abs(i)	-	Absolutbetrag
atof(s)	double	Ascii to Double (float)
atoi(s)	int	Ascii to Integer
atol(s)	long	Ascii to long
calloc(anz,gros)	char *	Allokiert anz*gros Speicher
close(filedes)	int	Dateien schließen (open, creat)
creat(name,stat)	int	Dateien anlegen
exit(nummer)	void	Ende mit Übergabe des Fehlercodes
fclose(dateizeig)	int	Datei schließen (fopen)
fflush(dateizeig)	int	Zwischenspeicher schreiben
fgetc(dateizeig)	int	Zeichen von Datei lesen
fgets(s,n,dateiz)	char *	Zeichenkette von Datei lesen
fopen(name,m)	FILE *	Datei öffnen
fprintf(datei,c,pl)	int	Text auf Datei schreiben
fputc(z,datei)	int	Zeichen auf Datei schreiben
fputs(s,datei)	int	Zeichenkette auf Datei schreiben
fread(p,n,#,datei)	int	# Objekte in Größe n von Datei in p lesen
freopen(neudat, m,datei)	FILE*	Schließt Datei und öffnet Datei neudat mit dem selben Zeiger
fscanf(datei,c,pl)	int	Liest Text aus Datei und interpretiert in laut c in Parameterliste pl
fseek(datei,o,m)	int	Verschieben des Dateizeigers um o in Art m
ftell(datei)	long	Position des Dateizeigers
fwrite(p,g,#,datei)	int	Schreibt ab p # Objekte des Größe g in Datei
gets(s)	char *	Liest eine Zeichenkette von stdin
getw(datei)	int	Liest Wort von Datei
index(s,c)	char *	Sucht c in s
isalnum(char)	int	Ist Zeichen char alphanumerisch
isalpha(char)	int	Ist Zeichen ein Buchstabe
isascii(char)	int	Ist Zeichen ein 7-Bit-ASCII-Zeichen
iscntrl(char)	int	Ist Zeichen ein Steuerzeichen
isdigit(char)	int	Ist Zeichen eine Ziffer
islower(char)	int	Ist Zeichen ein Kleinbuchstabe
isprint(char)	int	Ist Zeichen druckbar (kein Steuerzeichen)
ispunct(char)	int	Ist Zeichen weder alphanumerisch noch ein Steuerzeichen
isspace(char)	int	Ist Zeichen ein Zwischenraum (Blank, Tabulator, Zeilentrenner)

isupper(char)	int	Ist Zeichen ein Großbuchstabe
lseek(file,o,strt)	long	Versetzt Zeiger um o auf Art strt
malloc(anzahl)	char *	Allokiert Speicher der Größe anzahl
open(name,mod)	int	Öffnet Datei
printf(c,pl)	int	Ausgabe auf stdout
puts(s)	int	Zeichenkette auf stdout ausgeben
putw(w,datei)	int	Wort auf Datei ausgeben
qsort(v,#,l,verg)	void	Sortiert ab v die # Elemente der Länge l mit verg als Vergleichsfunktion
read(file,p,#)	int	Liest # von Datei nach p
rewind(datei)	int	Setzt Dateizeiger wieder an den Anfang
rindex(s,char)	char *	Sucht char in s vom Ende weg
scanf(c,pl)	int	Liest und Interpretiert Text von stdin
sprintf(s,c,pl)	int	Gibt Text auf Zeichenkette s aus
sscanf(s,c,pl)	int	Liest und Interpretiert Text von der Zeichenkette s
strcat(s1,s2)	char *	Verkettet 2 Zeichenketten
strcmp(s1,s2)	int	Vergleicht 2 Zeichenketten
strcpy(s1,s2)	char *	Kopiert s2 auf s1
strlen(s)	int	Länge der Zeichenkette
strncat(s1,s2,n)	char *	strcat mit maximal n Zeichen von s2
strncmp(s1,s2,n)	int	strcmp mit maximal n Zeichen
strncpy(s1,s2,n)	char *	strcpy mit maximal n Zeichen von s2
toascii(i)	int	int to Ascii
tolower(c)	int	Großbuchstabe -> Kleinbuchstabe
touppwe(c)	int	Kleinbuchstabe -> Großbuchstabe
ungetc(c,datei)	int	getc rückgängig machen
unlink(datei)	int	löscht Datei
write(file,p,#)	int	Schreibt # ab p auf Datei

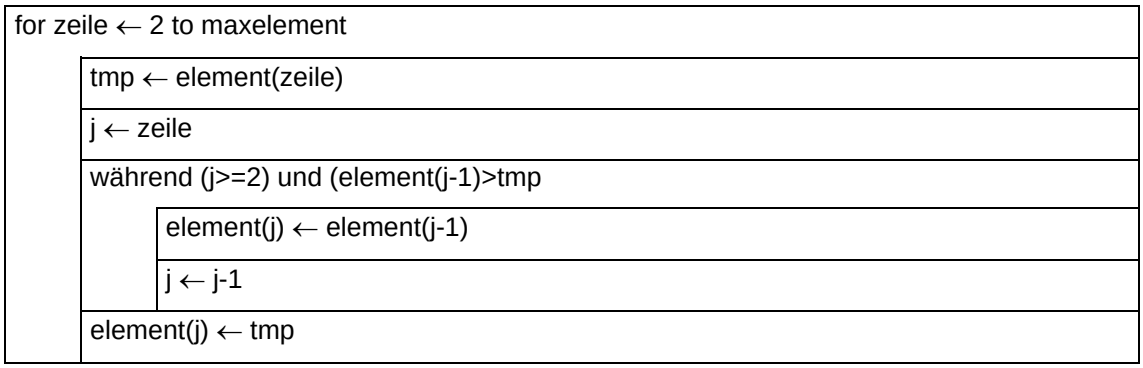
5. Algorithmen

5.1. Sortieren

Für alle Struktrogramm wird ein Feld mit dem Namen „element“ und der Anzahl „maxelement“ von Elementen angenommen. Der Datentyp ist für das Struktogramm unwesentlich, bei einer konkreten Umsetzung muß aber darauf Rücksicht genommen werden (element, tmp,... = zu sortierender Datentyp (numerisch, bei nicht numerischen Typen müssen die Vergleichsoperatoren entsprechend angepaßt werden); maxelement, zeile, j ,... = ganzahliger Typ)

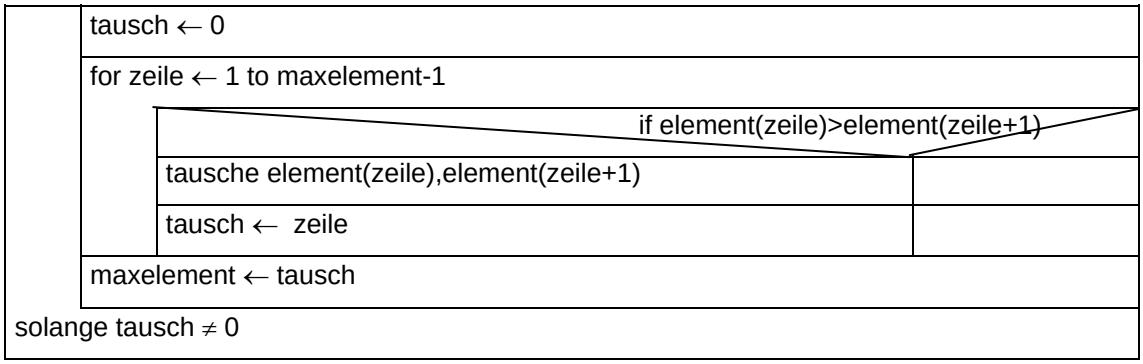
5.1.1. Einfügesort

Direktes Einfügen an die richtige Position bei allen bisherigen (schon sortierten) Elementen



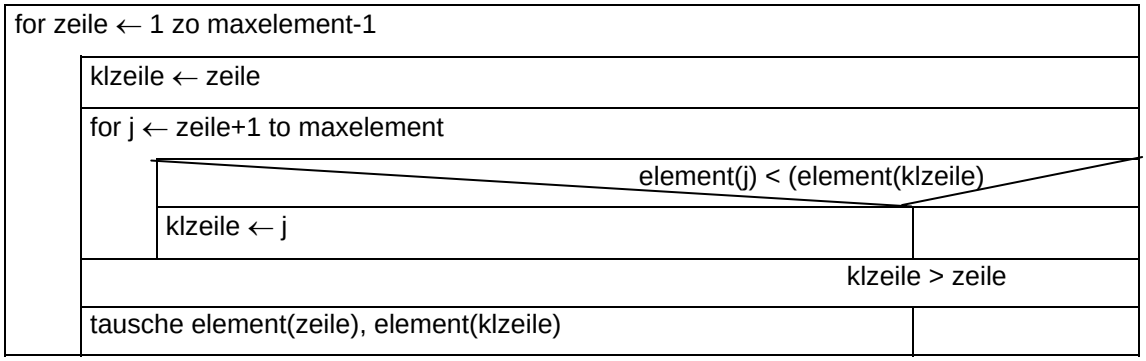
5.1.2. Bubblesort

paarweises Vertauschen bis alle Elemente in der richtigen Reihenfolge



5.1.3. Austauschsort

Austausch des aktuellen mit dem nachfolgendem kleinsten Element (wiederholte Minimumsuche)



5.1.4. Shellsort

ähnlich Bubblesort, aber nicht benachbarte Elemente werden verglichen (vertauscht), sondern zuerst entferntere und dann näher liegende; der Abstand (offset) beim Vergleichen beginnt bei maxelement/2 und endet bei 1

```
offset ← maxelement/2
während offset > 0
    grenze ← maxelement-offset
        tausch ← 0
        for zeile ← 1 to grenze
            element(zeile) > element(zeile+offset)
                tausche element(zeile), element(zeile+offset)
                tausch ← zeile
        grenze ← tausch – offset
    solange tausch ≠ 0
    offset ← offset/2
```

5.1.5. Heapsort

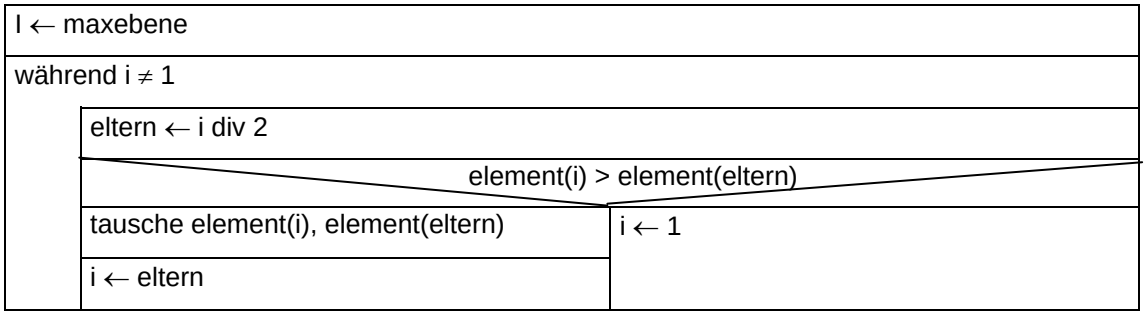
hier werden die Elemente entlang eines binären Baumes (ev. unregelmäßiger binärer Baum) angeordnet, wobei jeder Knoten größer als sein Kinder ist ⇒ erster Knoten ist die größte Zahl (= element(1)).

```
for i ← 2 to maxelement
    AUFWÄRTS(i)
for i ← maxelement to 2
    tausche element(1), element(i)
    ABWÄRTS(i-1)
```

ABWÄRTS(maxebene)

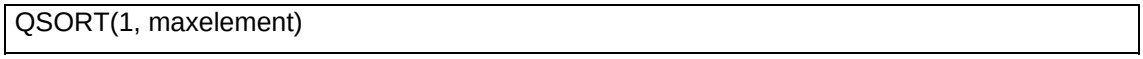
```
l ← 1
fertig ← false
während nicht fertig
    kind ← 2*i
    kind > maxebene
    fertig ← true
        kind+1<maxebene
            element(kind+1)>element(kind)
            kind ← kind + 1
                element(i) < element(kind)
                tausche element(i), element(kind)
                fertig ← true
                i ← kind
```

AUFWÄRTS(maxebene)

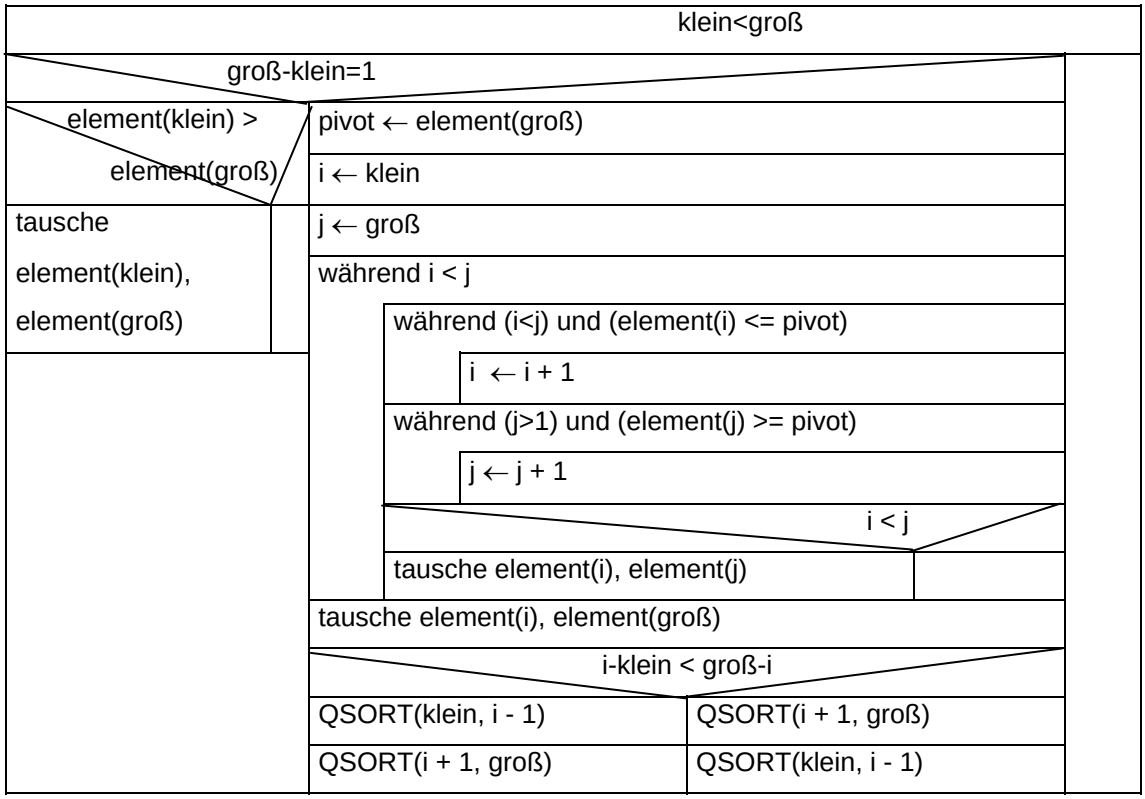


5.1.6. Quicksort

dieser Algorithmus nimmt ein zufälliges Element und teilt den Rest in zwei Untergruppen (kleinere und größere Elemente); für jede Untergruppe beginnt der selbe Vorgang wieder, ...



QSORT(klein, groß)



5.1. Datumsberechnungen

Für alle Datumsberechnungen werden einige besondere Berechnungen benötigt, von denen wir zwei herausgreifen wollen: die Bestimmung des Wochentages zu einem bestimmten Datum und die Errechnung des Osterdatums für ein bestimmtes Jahr, welches für die Berechnung der beweglichen Feiertage benötigt wird. Mit diesen beiden Methoden kann bereits eruiert werden, ob ein Tag mit gegebenen Datum ein Arbeitstag ist oder nicht. Vorausgeschickt werden muß, dass beide Arten nur für den gregorianischen Kalender ab dem 15.10.1582 Gültigkeit haben.

5.1.1. Die Bestimmung des Wochentages

Hier wird mit Hilfe von Kennzahlen für die Jahrhundert und die Monate eine einfache Bestimmung vorgenommen (Das ist nur eines von vielen möglichen Verfahren).

Die Kennzahlen für die Jahrhunderte:

15xx, 19xx, 23xx, …	0
18xx, 22xx, …	2
17xx, 21xx, …	4
16xx, 20xx, …	6

Die Kennzahlen für die Monate:

Jänner	1	Februar	4	März	3	April	6
Mai	1	Juni	4	Juli	6	August	2
September	5	Oktober	0	November	3	Dezember	5

Die Kennzahlen für die Wochentage:

Sonntag	0	Montag	1	Dienstag	2	Mittwoch	3
Donnerstag	4	Freitag	5	Samstag	6		

Jänner und Februar zählen – auch hinsichtlich der Jahrhundertkennzahlen – zum vorangegangen Jahr.

d.h.:

- Verminderung der Jahreszahl um eins, wenn das Datum im Jänner oder Februar liegt
- Addieren der 2-stelligen Jahreszahl, der Tageszahl, der Jahrhundertkennzahl, der Monatskennzahl und der ganzzahlig durch 4 geteilten 2-stelligen Jahreszahl
- das obige Ergebnis modulo 7 ergibt die Kennzahl für den Wochentag

Dazu Beispiele:

	16.11.2005	**1.1.1999**	**29.2.1996**	**1.1.2000**
2-stellige Jahreszahl	5	98	95	99
Tageszahl	16	1	29	1
Jahrhundertkennzahl	6	0	0	0
Monatskennzahl	3	1	4	1
2-stellige Jahreszahl/4	1	24	23	24
Summe	31	124	151	125
Summe modulo 7	3	5	4	6
Wochentag:	**Mittwoch**	**Freitag**	**Donnerstag**	**Samstag**

5.1.2. Die Bestimmung des Osterdatums

Dazu bedient man sich z.B.: der Formeln von Carl Friedrich Gauß, die das Datum des Ostersonntages liefern, davon abgeleitet, können dann alle bewegliche Festtage unseres Kalenders bestimmt werden (Aschermittwoch=Ostersonntag-49, Christi Himmelfahrt=Ostersonntag+39, Pfingstsonntag=Ostersonntag+49, Fronleichnam=Ostersonntag+60).

Beispielprogramm:

```
/* EASTER.C V0.3 written by K. Coufal on 1996 11 25 */
#include <stdio.h>
void main()
{
        int year,month,day;
        int d,m,gn,c,gc,cc,ed,e;
        printf("Easter V0.3c - Berechnung beweglicher Feste\n");
        printf("(c) Copyright 1996 by Mag.Dr. Klaus Coufal\n");
        printf("\nBitte Jahr eingeben  : ");
        scanf("%d",&year);
        printf("\n");
        gn=year % 19 + 1;
        if (year<=1582)
        {
                ed=(5*year) /4;
                e=(11*gn-4)%30 + 1;
        }
        else
        {
                c=year/100 + 1;
                gc=(3*c)/4-12;
                cc=(c-16-(c-18)/25)/3;
                ed=(5*year)/4-gc-10;
                e=(11*gn+19+cc-gc)%30+1;
                if(((e==25) && (gn>11)) || (e==24)) ++e;
        }
        d=44-e;
        if (d<21) d+=30;
        d=d+7-(ed+d)%7;
        if (d<=31) m=3;
        else
        {
                m=4;
                d-=31;
        }
        day=d;
        month=m;
        printf("Ostersonntag       : %2d.%02d.%d\n",day,month,year);
        d+=39;
        if (m==3)
        {
                d-=31;
                ++m;
        }
```

```
        if ((m==4) && (d>30))
        {
                d-=30;
                ++m;
        }
        if ((m==5) && (d>31))
        {
                d-=31;
                ++m;
        }
        printf("Christi Himmelfahrt: %2d.%02d.%d\n",d,m,year);
        d+=10;
        if (d>31)
        {
                d-=31;
                ++m;
        }
        printf("Pfingstsonntag     : %2d.%02d.%d\n",d,m,year);
        d+=11;
        if (d>31)
        {
                d-=31;
                ++m;
        }
        printf("Fronleichnam       : %2d.%02d.%d\n",d,m,year);
}
```

5.1.3. Sonstiges

Daneben gibt es selbstverständlich noch eine Reihe von anderen Berechnungen (z.B.: die Bestimmung der Kalenderwoche, Anzahl der Tage zwischen zwei Daten, …), die für das Arbeiten mit Datumsangaben wichtig sind, auf die hier aber nicht näher eingegangen werden soll.

6. Anhang

Zur Kontrolle von Prüfziffern am Beispiel ISBN (Internationale Standard Buch Nummer)

Problem: Eine ISBN soll zeichenweise eingelesen und auf Korrektheit geprüft werden.

Bei korrekter Nummer sollen noch Zusatzinformationen (Sprachgruppe) ausgegeben werden.

3-426-04175-8

Sprache-Verlag-Titel-Prüfziffer

Sprachgruppen:

0	englisch
2	französisch
3	deutsch
84	spanisch
86	jugoslawisch
87	dänisch
90	niederländisch
91	schwedisch
92	UNESCO
951	finnisch
963	ungarisch
977	ägyptisch
978	nigerianisch
979	indonesisch

Berechnung der Prüfziffer:

3	1	4	1	1	1	9	6	7	8
*10	*9	*8	*7	*6	*5	*4	*3	*2	*1
30	+9	+32	+7	+6	+5	+36	+18	+14	8 = 165

Prüfziffer wird so gesetzt, daß die Summe durch 11 teilbar ist (für 10 wird die Ziffer X verwendet)

Erkennt:

- falsche Ziffern
- Vertauschung von Ziffern

Andere Prüfziffernverfahren am Beispiel **EAN** (Europäische ArtikelNummer), **GTIN** (Global Trade Item Number) bzw. SSCC (Serial Shipping Container Code)

Zusammenhang zwischen GTIN und EAN

Bezeichnung	**frühere Bezeichnungen**
GTIN-14	–
GTIN-13	EAN-UCC-13, EAN-13
GTIN-12	EAN-UCC-12, UCC-12, UPC
GTIN-8	EAN UCC-8, EAN-8

Die 14-stellige GTIN wird durch Voranstellen führender Nullen aus den bisherigen 13-, 12- und 8-stelligen Artikelnummern gebildet:

Zur Berechnung der Prüfziffer:

Nummer	**Ziffernposition**																
GTIN-8											N_1	N_2	N_3	N_4	N_5	N_6	N_7
GTIN-12							N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}	N_{11}
GTIN-13						N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}	N_{11}	N_{12}
GTIN-14					N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}	N_{11}	N_{12}	N_{13}
SSCC	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}	N_{11}	N_{12}	N_{13}	N_{14}	N_{15}	N_{16}	N_{17}
Multiplikator	*3	*1	*3	*1	*3	*1	*3	*1	*3	*1	*3	*1	*3	*1	*3	*1	*3

Jede Stelle wird mit dem dazugehörigen Multiplikator multipliziert, die Ergebnisse addiert und die Prüfziffer ist dann die Differenz zum nächstgrößeren Vielfachen von Zehn. Bei einer Überprüfung wird nur die vorher genannte Summe + der Prüfziffer gebildet, diese Summe mod 10 ergibt 0, wenn die Prüfziffer korrekt war.

Beispiel:
2l-Flasche Coca Cola: EAN 5000112506396 (darin ist *6* die Prüfziffer)

Bildung der Prüfziffer:

5*1 + 0*3 + 0*1 + 0*3 + 1*1 + 1*3 + 2*1 + 5*3 + 0*1 + 6*3 + 3*1 + 9*3 =
5 + 0 + 0 + 0 + 1 + 3 + 2 + 15 + 0 + 18 + 3 + 27 = 74

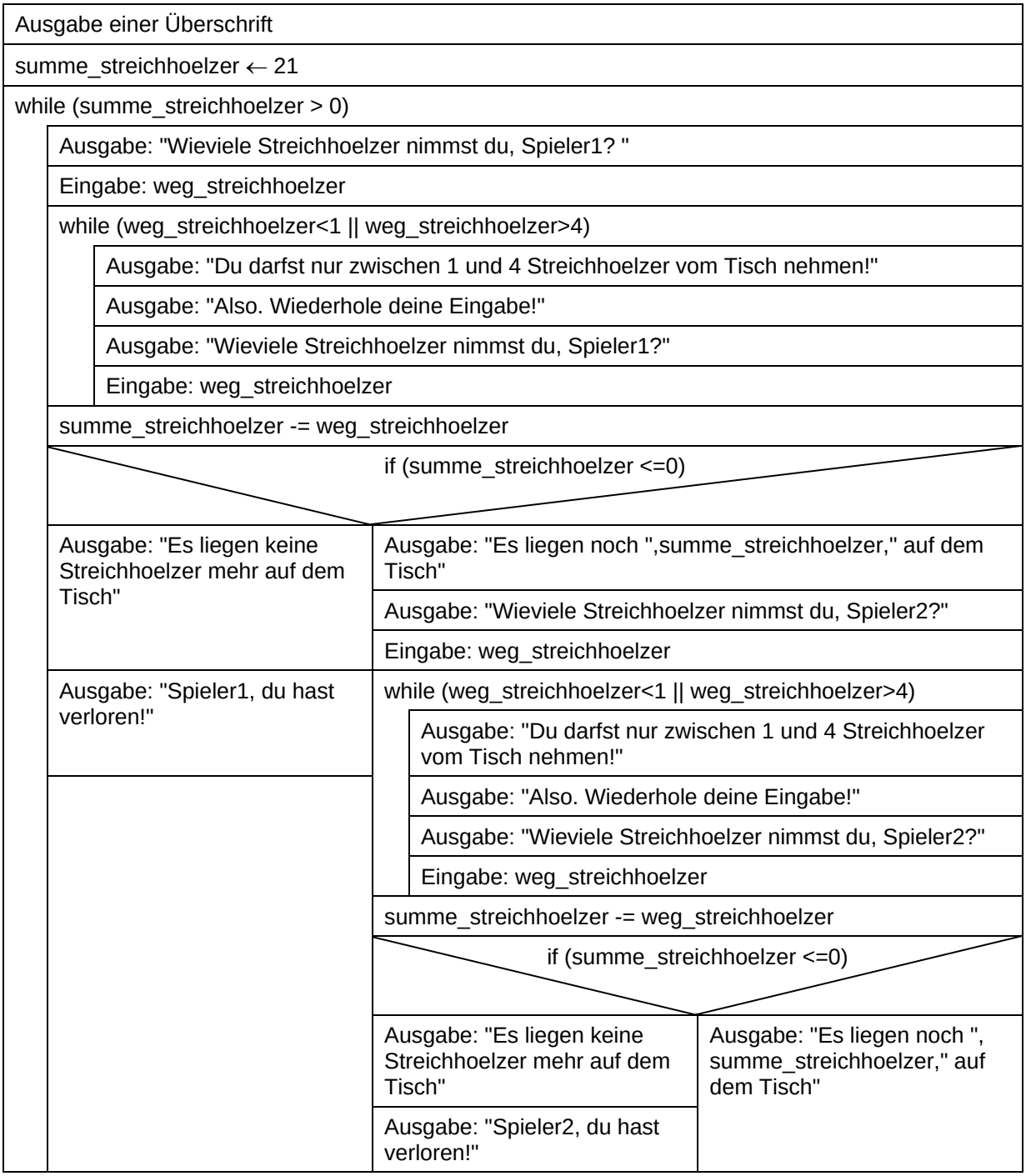
Das nächstgrößere Vielfache von 10 ist 80, daher ist die Prüfziffer 80-74=6

Überprüfung

5*1 + 0*3 + 0*1 + 0*3 + 1*1 + 1*3 + 2*1 + 5*3 + 0*1 + 6*3 + 3*1 + 9*3 + 6*1 =
5 + 0 + 0 + 0 + 1 + 3 + 2 + 15 + 0 + 18 + 3 + 27 + 6 = 80

80 mod 10 ist 0, daher ist die Prüfziffer ist korrekt

Streichholzspiel



Programmbeispiel "umdrehen" (Rekursion)

```
#include <stdio.h>
Void main()
{
      printf("\n\nGeben Sie eine Zeichenkette ein
(Ende=Leer<RETURN>)!\n");
      umdrehen();
}

void umdrehen()
{
      char zeichen;
      scanf("%c",&zeichen);
      if (zeichen!=´ ´)
            umdrehen();
      printf("%c",zeichen);
}
```

Programmbeispiel "FILES" (Dateioperationen)

```
/* IDENT: FILES.C V1.0 written by K. Coufal on 1989 05 23
Program to demonstrate some of the C file functions */
#include <stdio.h>
void main()
{
      float zahl;
      FILE *datei;
      char datnam[12];
      printf("\nName der Datei: ");
      scanf("%s", datnam);
      datei=fopen(datnam,"w");
      printf("Zahl: ");
      scanf("%f", &zahl);
      while(zahl!=0)
      {
            fprintf(datei,"%f\n",zahl);
            printf("Zahl: ");
            scanf("%f", &zahl);
      }
      fclose(datei);
      datei=fopen(datnam,"r");
      fscanf(datei,"%f",&zahl);
      while (!feof(datei))
      {
            printf("%8.3f\n", zahl);
            fscanf(datei,"%f",&zahl);
      }
      fclose(datei);
}
```

Priorität der Operatoren

Operator	bei gleicher Priorität
( ) [ ] -> .	Von links her
! ~ ++ -- - /* Minuszeichen */ (cast) * /* Verweisoperator */ & /* Adressoperator*/ sizeof	Von rechts her
* / %	Von links her
+ - /* Subtraktion */	Von links her
<< >>	Von links her
<. <= > >=	Von links her
== !=	Von links her
& /* bitweises UND */	Von links her
^	Von links her
|	Von links her
&&	Von links her
||	Von links her
?:	Von rechts her
= += -= *= /= %= >>= <<= &= |= ^=	Von rechts her
, /* Komma-Operator */	Von links her

Der in der rechten Spalte angegebene Hinweis „von links her“ oder „von rechts her“ besagt, in welcher Reihenfolge Operatoren gleicher Priorität abgearbeitet werden.

Aufgaben

Sämtliche Angaben sind nur Kurzfassungen zur Erinnerung, die Beispiele werden während der Übungen besprochen.

1. Schreiben Sie ein Programm, das einen einen einfachen Text ausgibt (z.B.: Hallo).
2. Schreiben Sie ein Programm, das Umfang und Fläche oder Volumen und Oberfläche einer einfachen geometrischen Figur ausgibt (Quadrat, Rechteck, Kreis, Zylinder, Kugel, ...)
3. Schreiben Sie ein Programm, das von einem Zahlensystem in ein anderes umrechnet.
4. Schreiben Sie ein Programm, das eine Münzliste (Liste mit der Anzahl der Münzen und Geldscheine, die für die Auszahlung bestimmter Beträge notwendig sind) für einen (fünf) Betrag (Beträge) erstellt.
5. Schreiben Sie ein Programm, das nach Eingabe zweier Zahlen und eines Rechenzeichens, die Rechnung durchführt und das Ergebnis ausgibt.
6. Schreiben Sie ein Programm, das bei gegebener Zaunlänge das größte Rechteck findet, das mit diesem Zaun an einer Felswand abgegrenzt werden kann (Hier ist nicht die mathematisch einfache Lösung gefragt, sondern an Hand dieses Beispieles soll die iterative Suche nach einer Lösung probiert werden).
7. Prüfziffernkontrolle
8. Feiertagsberechnung
9. Vektoraddition, -subtraktion, -multiplikation
10. Schreiben Sie ein Programm mit einer Funktion zur Berechnung von x^y (y ganzzahlig).
11. Schreiben Sie Programm für das Spiel „Streichholzwegnehmen“ (21 Streichhölzer, 2 Spieler, die zwischen 1 und 4 Streichhölzer wegnehmen. Wer das letzte Streichholz wegnimmt (wegnehmen muß), hat verloren).
12. Schreiben Sie ein Programm zur Berechnung der Quadratwurzel nach der Formel von Archimedes (wurzelneu=½ (wurzelalt+zahl/wurzelalt)). Vereinfachungen:wurzelalt=zahl;zahl >=1.
13. Beschäftigen Sie sich mit dem Programm zum Umdrehen von Text.
14. Schreiben Sie ein Programm zur Ausgabe von Dreiecken folgender Art:

    1
    121
    12321

nach Eingabe der Zeilenzahl.

15. Schreiben Sie ein Programm, daß eine Folge von ganzen Zahlen einliest (0=Ende) und das Minimum, das Maximum und das arithmetische Mittel ausgibt.
16. Schreiben Sie ein Programm, daß die Bestimmung von Dreiecken nach Eingabe der drei Seiten durchführt (kein, allgemeines, Strecke, rechtwinkelig, gleichseitig, ...).
17. Schreiben Sie ein Programm zur Ermittlung von ggT und kgV.
18. Schreiben Sie ein Programm zur Lösung von Gleichungen (quadratische, mehrere Variable, ...).
19. Sortieren von Elementen