Beispiel 1: Rahmen eines ersten C-Programms

```
#include <stdio.h>
main()
  /* Programm zur Ausgabe aller
     Zweierpotenzen kleiner als 20 */
{
  Vereinbare Zähler namens "zahl"
  und setze ihn auf 1;
  while ( zahl kleiner als 20 ) {
   printf("Zweierpotenz: %d\n",zahl);
   Multipliziere zahl mit 2;
  }
}
```

Beispiel 2: Deklaration und Initialisierung

```
#include <stdio.h>
main()
  /* Programm zur Ausgabe aller
     Zweierpotenzen kleiner als 20 */
{
  int zahl = 1;
  while ( zahl kleiner als 20 ) {
   printf("Zweierpotenz: %d\n",zahl);
   Multipliziere zahl mit 2;
  }
}
```

Beispiel 3: Zuweisungen

```
int a, b, c, d, sum, prod;
a = 10; b = 20;
c = a;  d = b;
sum  = a + b;
prod = a * b;
```

Beispiel 4: Zuweisung

```
#include <stdio.h>
main()
  /* Programm zur Ausgabe aller
     Zweierpotenzen kleiner als 20 */
{
  int zahl = 1;
  while ( zahl kleiner als 20 ) {
   printf("Zweierpotenz: %d\n",zahl);
   zahl = zahl * 2;
  }
}
```

Beispiel 5: printf()

```
#include <stdio.h>
main() {
 int zahl1 = 4711, zahl2 = 1234;
 printf("1. Zahl: %d, 2. Zahl: %d\n",zahl1,zahl2);
 printf("Summe: %d",zahl1+zahl2);
}
```

Beispiel 6: scanf()

```
#include <stdio.h>
main() {
 int laenge, breite;
 printf("Bitte Laenge und Breite eingeben\n");
 scanf("%d %d",&laenge,&breite);
 ...
}
```

Beispiel 7: Ganzzahlkonstanten

```
int      i, j, k;
long     l;
unsigned u;
/* Dezimalzahlen */
i =   7;
j = +23;
k = -36;
l = -18l;   /* long */
u = 65535u; /* unsigned */
/* Oktalzahlen */
i = 0777;
j = -0123;
/* Hexadezimalzahlen */
i = 0xFFFF;
j = -0xAAAA;
```

Beispiel 8: Gleitkommakonstanten

```
float f1, f2, f3, f4, f5, f6;
f1 = -0.222;
f2 = .345;
f3 = 1.;
f4 = 1e10;
f5 = 1E+10;
f6 = 1.34e-10;
```

Beispiel 9: Typ char als Ganzzahltyp

```
main() {
 char zeichen;
 zeichen = 'A';
   /* zeichen hat jetzt den Wert 65 bzw. 'A' */
 zeichen = zeichen + 32;
   /* zeichen hat jetzt den Wert 97 bzw. 'a' */
}
```

Beispiel 10: Zeichenkonstanten

```
char c1, c2, c3;
c1 = 'A';
c2 = 'z';
c3 = '8';      /* Achtung: 8 ≠ '8'! */
c1 = '\x41';   /* Hexadezimaler ASCII-Code für 'A' */
c1 = '\101';   /* Oktaler ASCII-Code für 'A' */
/* Steuerzeichen und spezielle Zeichen */
c2 = '\0';  /* Wortende */
c2 = '\a';  /* Ton */
c2 = '\b';  /* Schritt zurück */
c2 = '\f';  /* Seitenvorschub */
c2 = '\n';  /* Zeilenvorschub */
c2 = '\r';  /* Carriage Return */
c2 = '\t';  /* Tabulator */
c2 = '\\';  /* spezielles Zeichen \ */
c2 = '\'';  /* spezielles Zeichen ' */
c2 = '\"';  /* spezielles Zeichen " */
```

Beispiel 11: Deklaration benannter Konstanten

```
int         i;               /* Variable */
const int   stink = 4711;    /* Ganzzahlkonstante */
const float pi    = 3.1415;  /* Gleitkommakonstante */
stink = 4321;   /* unzulässig!!! */
```

Beispiel 12: logische Bedingung

```
#include <stdio.h>
main()
  /* Programm zur Ausgabe aller
     Zweierpotenzen kleiner als 20 */
{
  int zahl = 1;
  while ( zahl < 20 ) {
   printf("Zweierpotenz: %d\n",zahl);
   zahl = zahl * 2;
  }
}
```

Beispiel 13: implizite Zuweisungen

```
int a, b, c, d, e, f;
...
a++;
++b;
c--;
--d;
e = 2*a--;
f = 2*--b;
```

ist äquivalent zu

```
int a, b, c, d, e, f;
...
a = a+1;
b = b+1;
c = c-1;
d = d-1;
e = 2*a;  a = a-1;
b = b-1;  f = 2*b;
```

Beispiel 14: verkürzte Schreibweise für Zuweisungen

```
int a, b, c;
...
a += b; a -= b; a *= b+c; a /= b+c; a %= c;
```

ist äquivalent zu

```
int a, b, c;
...
a = a+b; a = a-b; a = a*(b+c); a = a/(b+c); a = a%c;
```

Beispiel 15: Blöcke

```
main() {
  float giro, sparbuch, vermoegen;
  scanf("%f %f",&giro,&sparbuch);
  vermoegen = giro + sparbuch;
  {
    float sparbuch_m_zins, vermoegen;
```

```
    sparbuch_m_zins = 1.02*sparbuch;
    vermoegen = giro + sparbuch_m_zins;
    printf("Vermoegen mit Zinsen: %4.2f\n",vermoegen);
  }
  printf("Vermoegen ohne Zinsen: %4.2f\n",vermoegen);
 }
```

Beispiel 16: if-Anweisung

```
if (guthaben > 0)
  { printf("Nicht pleite\n");
    printf("Glueckwunsch!\n");
  }
 else {
    printf("Pleite\n");
    printf("Beileid!\n");
 }
printf("Bankbericht beendet\n");
```

Beispiel 17: geschachteltes if-else

```
int sender;
...
if (sender==1)
   printf("ARD");
 else
   if (sender==2)
      printf("ZDF");
    else
      if (sender==3)
         printf("Drittes");
       else
         printf("Privat");
```

Beispiel 18: einzelnes else bei mehreren ifs

```
if (a==0)
  if (b==0) <anw1>
    else <anw2>
      /* else gehört zum inneren if */
```

```
if (a==0)
  { if (b==0) <anw1>
  }
 else <anw2>
      /* else gehört zum äußeren if */
```

Beispiel 19: switch-Anweisung

```
int sender;
...
switch(sender) {
 case 1: printf("ARD"); break;
 case 2: printf("ZDF"); break;
 case 3: printf("Drittes"); break;
 default: printf("Privat"); break;
}

switch(sender){
 case 1:
 case 2:
 case 3: printf("Oeffentlich-rechtlich"); break;
 default: printf("Privat"); break;
}
```

Beispiel 20: for-Schleife zur Ausgabe der ersten zehn Quadrat- und Kubikzahlen

```
int i;       /* Laufvariable */
for (i=1;i<=10;i++)
  printf("i, i hoch 2, i hoch 3: %d, %d, %d\n",i,i*i,i*i*i);
```

Beispiel 21: break-Anweisung zum Verlassen einer Schleife

```
for (i=1;;i++) {
  if (i==10)
    { printf("Letzte Runde\n"); break; }
  printf("%d-te Runde\n",i);
 }
```

äquivalent zu

```
for (i=1;i<=10;i++) {
  if (i==10)
   printf("Letzte Runde\n");
   else printf("%d-te Runde\n",i);
 }
```

Beispiel 22: continue-Anweisung zum Rücksprung an den Schleifenkopf

```
for (i=5;i>=-5;i--) {
  if (i==0) continue;
  printf("Quotient = %f\n",10.0/i);
 }
```

Beispiel 23: fußgesteuerte Schleife

```
int choice;
do {
 printf("Bitte Wert zwischen 0 und 9 eingeben");
 scanf("%d",&choice);
}
while (choice<0||choice>9);
```

Beispiel 24: Felder und for-Schleifen

```
float umsaetze[12];
int   i;

/* Initialisierung aller Feldkomponenten mit 0 */
for (i=0;i<=11;i++)
  umsaetze[i] = 0;

/* Einlesen aller Feldkomponenten von der Tastatur */
for (i=0;i<=11;i++)
  scanf("%f",&umsaetze[i]);

/* in jedem Monat zwei Prozent mehr Umsatz */
umsaetze[0] = 1234.56;
for (i=1;i<=11;i++)
  umsaetze[i] = 1.02 * umsaetze[i-1];
```

Beispiel 25: Zuweisung zwischen Feldvariablen

```
int a[3], b[3] = {1,2,3};
int i;
a = b;                          /* falsch! */
for (i=0;i<3;i++) a[i] = b[i];  /* richtig */
```

Beispiel 26: mehrdimensionale Felder

```
float umsaetze2[10][12];
          /* für 10 Firmen jeweils 12 Monatsumsätze */
int   firma, monat;

for (firma=0;firma<=9;firma++)
 for (monat=0;monat<=11;monat++)
  umsaetze2[firma][monat]=0.;

umsaetze2[2][1] = 12345.67;  /* Februar-Umsatz der 3. Firma */
```

Beispiel 27: Stringvariable

```
char text1[20], text2[20];
int  i;

/* Ein- und Ausgabe eines Strings */
printf("Bitte Wort eingeben (Laenge<20!): ");
scanf("%s",text1);
printf("Eingegebenes Wort: %s\n",text1);

/* Übertragung von text1 nach text2 und Ausgabe */
for (i=0;i<=19;i++)
  text2[i]=text1[i];
for (i=0;i<=19;i++)
  printf("%c\n",text2[i]); /* je Zeile ein Zeichen */
```

Beispiel 28: Stringfunktionen

```
#include <stdio.h>
#include <string.h>
main() {
 char text1[20], text2[20], text3[20];
 scanf("%s %s",text1,text2);
 printf("Laenge String 1: %d\n",strlen(text1));
 printf("Laenge String 2: %d\n",strlen(text2));
 if (strcmp(text1,text2)==0)
   printf("Strings sind gleich\n");
  else printf("Strings sind ungleich\n");
 strcpy(text3,text1);  /* Kopie von text1 nach text3 */
}
```

Beispiel 29: Zeigervariablen - Deklaration und Zugriffe

```
float *pt;
float konto1 = 1000.0, konto2 = 2000.0;
int wahl;

/* dynamische Auswahl einer Kontovariablen */
printf("Bitte waehlen: 1 = Konto1, 2 = Konto2\n");
scanf("%d",&wahl);
if (wahl==1)
  pt = &konto1;
 else pt = &konto2;

/* pt zeigt nun auf ausgewähltes Konto,
   Zugriffe darauf über *pt */
...
*pt = *pt + 100.0; /* erhöht Stand des gewählten Kontos */
...
```

Beispiel 30: Operationen auf Zeigervariablen

```
int *pt1, *pt2;
int var1 = 100, var2 = 200;

pt1  = &var1;       /* pt1 zeigt nun auf var1 */
pt2  = pt1;         /* pt2 zeigt nun auch auf var1 */
```

```
*pt1 = *pt1 + 1;    /* entspricht var1 = var1 + 1; */
pt1  = &var2;       /* pt1 zeigt nun auf var2 */
(*pt1)++;           /* entspricht var2 = var2 + 1; */
*pt2 = 150;         /* entspricht var1 = 150; */

pt1  = &var1;
pt2  = &var2;
*pt2 = *pt1;        /* entspricht var2 = var1; */
*pt2 = 300;         /* entspricht var2 = 300; */
pt2  = pt1;         /* pt2 und pt1 zeigen nun auf var1 */
*pt2 = 300;         /* entspricht var1 = 300; */
```

Beispiel 31: dynamische Speicherverwaltung

```
#include <stdlib.h>   /* für malloc und free */
main() {
 float *pt;
 pt = (float *) malloc(sizeof(float));
     /* belegt sizeof(float) = 4 Bytes im Hauptspeicher
        und weist Adresse pt zu */
 if (pt==NULL) {
   printf("Fehler: Kein Speicher frei!");
   ... Programmabbruch ...
 }
 ...
 free(pt);    /* gibt Speicherplatz wieder frei */
 pt = 0;      /* markiert Zeiger als ungültig */
 pt = NULL;   /* äquivalent */
 ...
 if (pt == NULL)
   ... Zeiger nicht gültig ...
}
```

Beispiel 32: Zeigerarithmetik

```
float *pt1, *pt2;
float var1 = 1.0, var2 = 3.0, var3 = 5.0;
/* Annahme im folgenden:
   var1, var2, var3 sind aufeinanderfolgend abgespeichert.
   Kann nicht allgemein vorausgesetzt werden! */
```

```
pt1 = &var1;
pt2 = pt1+2;        /* pt2 zeigt nun auf var3 */
pt2--;              /* pt2 zeigt nun auf var2 */
pt2++;              /* pt2 zeigt nun wieder auf var3 */

var3 = *pt1 + 1;    /* entspricht var3 = var1 + 1; */
var3 = *(pt1 + 1);  /* entspricht var3 = var2; */
```

Beispiel 33: Felder und Zeiger

```
float umsaetze[12];
float *zeiger;

umsaetze[0]   = 1.23;
*umsaetze     = 1.23;    /* äquivalent */

umsaetze[2]   = 4.56;
*(umsaetze+2) = 4.56;    /* äquivalent */

zeiger = umsaetze;       /* zulässig   */
zeiger = &umsaetze[0];   /* äquivalent */

zeiger = &umsaetze[2];
zeiger = umsaetze+2;     /* äquivalent */

umsaetze = zeiger;       /* unzulässig, da Zeigerkonstante */
```

Beispiel 34: Strukturen - Deklaration und Operationen

```
struct {
 char  name[20];
 int   pers_nr;
 float gehalt;
} angestellter;

struct {
 ...
 } ang_1 = { "Gustav Gans", 4567, 4444.44 },
   ang_2, ang_3,
   *ang_zeiger;

ang_2 = ang_1;
```

```
ang_3.pers_nr = 4567;
ang_3.gehalt  = ang_1.gehalt + ang_2.gehalt;
strcpy(ang_3.name,"Donald Duck");

ang_zeiger = &ang_1;
```

Beispiel 35: Strukturtypen

```
struct ang_info {   /* ang_info ist der Typname */
 char  name[20];
 int   pers_nr;
 float gehalt;
} ang_1, ang_2, ang_3;

struct ang_info  ang_4, ang_5;
```

äquivalent zu

```
struct ang_info {
 char  name[20];
 int   pers_nr;
 float gehalt;
} ang_1, ang_2, ang_3, ang_4, ang_5;
```

äquivalent zu

```
struct ang_info {   /* nur Typdefinition */
 char  name[20];
 int   pers_nr;
 float gehalt; };

struct ang_info  ang_1, ang_2, ang_3, ang_4, ang_5;
                                      /* Variablendeklaration */
```

Beispiel 36: geschachtelte Strukturen

```
struct datum {
 int tag, monat, jahr; };

struct ang_info {
 char  name[20];
 int   pers_nr;
 float gehalt;
```

```
 struct datum eintritt;
};

struct ang_info ang_1;

ang_1.eintritt.jahr = 1990;
```

Beispiel 37: Zeiger in Strukturen

```
struct ang_info {
 char  name[20];
 int   pers_nr;
 float gehalt;
 struct ang_info chef;    /* !!! unzulässig */
};

struct ang_info {
 char  name[20];
 int   pers_nr;
 float gehalt;
 struct ang_info *chef;   /* zulässig (da Zeiger) */
};

struct ang_info *ang_zeiger;

ang_zeiger =
           (struct ang_info *) malloc(sizeof(struct ang_info));

(*ang_zeiger).chef =
           (struct ang_info *) malloc(sizeof(struct ang_info));
(*(*ang_zeiger).chef).chef =
           (struct ang_info *) malloc(sizeof(struct ang_info));

(*ang_zeiger).pers_nr = 1234;
ang_zeiger->pers_nr   = 1234;    /* äquivalent */
ang_zeiger->chef->chef->pers_nr = 007;
ang_zeiger->chef->chef->chef    = NULL;  /* wichtig! */
```

Beispiel 38: Strukturen und Felder

```
struct ang_info {
... wie oben ...
};

struct ang_info  alle_ang[100];

alle_ang[50].pers_nr = 1234;
```

alternativ: Feld mit Zeigern auf Strukturen

```
struct ang_info  *alle_ang_zeiger[100];

alle_ang_zeiger[50] =
           (struct ang_info *) malloc(sizeof(struct ang_info));
alle_ang_zeiger[50]->pers_nr = 1234;
```

Beispiel 39: Typdefinitionen

```
typedef  int  ganzzahl;
ganzzahl  zahl;        /* entspricht  int zahl; */

typedef  float  *flt_zeiger;
flt_zeiger  pointer;   /* entspricht  float *pointer; */

typedef  char  text[30];
text  zeile1, zeile2;  /* entspricht:
                                 char zeile1[30], zeile2[30]; */

typedef  struct { int tag, monat, jahr; }  datum;
datum  heute;
      /* entspricht: struct { int tag, monat, jahr; } heute; */

typedef  struct {...wie oben...}  ang_info;
typedef  ang_info  *ang_info_feld[100];
ang_info_feld fabriken[10];
```

Beispiel 40: grundlegende Dateioperationen

```
#include <stdio.h>          /* für Dateioperationen */
main() {
 FILE *datzeig;             /* Dateizeiger */
 datzeig = fopen("c:\\goofy\\out","w");
                            /* Öffnen der Datei zum Schreiben */
 fprintf(datzeig,"Hello World!");
                            /* Schreiben in die Datei */
 fclose(datzeig);           /* Schließen der Datei */
}
```

Beispiel 41: Funktionen

Drei Maximumsberechnungen:

- Maximum der Punktezahlen in der DV-Klausur
- Maximum der Punktezahlen in der Mathematik-Klausur
- Maximum der Punktezahlen in der Physik-Klausur,

jeweils für drei Studenten

ohne C-Funktionen:

```
main() {

 int punkte_dv_1, punkte_dv_2, punkte_dv_3, punkte_dv_max;
 int punkte_ma_1, punkte_ma_2, punkte_ma_3, punkte_ma_max;
 int punkte_ph_1, punkte_ph_2, punkte_ph_3, punkte_ph_max;

/* ... hier Initialisierungen ... */

 punkte_dv_max = punkte_dv_1;
 if (punkte_dv_2>punkte_dv_max)
                 punkte_dv_max = punkte_dv_2;
 if (punkte_dv_3>punkte_dv_max)
                 punkte_dv_max = punkte_dv_3;

 punkte_ma_max = punkte_ma_1;
 if (punkte_ma_2>punkte_ma_max)
                 punkte_ma_max = punkte_ma_2;
 if (punkte_ma_3>punkte_ma_max)
                 punkte_ma_max = punkte_ma_3;
```

```
 punkte_ph_max = punkte_ph_1;
 if (punkte_ph_2>punkte_ph_max)
                 punkte_ph_max = punkte_ph_2;
 if (punkte_ph_3>punkte_ph_max)
                 punkte_ph_max = punkte_ph_3;

}
```

Mit C-Funktion:

```
int max(int wert_1,int wert_2,int wert_3) {

 int wert_max;

 wert_max = wert_1;
 if (wert_2>wert_max)
            wert_max = wert_2;
 if (wert_3>wert_max)
            wert_max = wert_3;
 return wert_max;

}

main() {

 int punkte_dv_1, punkte_dv_2, punkte_dv_3, punkte_dv_max;
 int punkte_ma_1, punkte_ma_2, punkte_ma_3, punkte_ma_max;
 int punkte_ph_1, punkte_ph_2, punkte_ph_3, punkte_ph_max;

/* ... hier Initialisierungen ... */

 punkte_dv_max = max(punkte_dv_1,punkte_dv_2,punkte_dv_3);
 punkte_ma_max = max(punkte_ma_1,punkte_ma_2,punkte_ma_3);
 punkte_ph_max = max(punkte_ph_1,punkte_ph_2,punkte_ph_3);

}
```

Beispiel 42: Funktionsprototypen

```
/* Prototypen = Funktionsdeklarationen */

int max(int x,int y);
int min(int x,int y);

/* Funktionsdefinitionen */

int max(int x,int y) {
 if (x>y) return x;
  else return y;
}

int min(int x,int y) {
 if (x<y) return x;
  else return y;
}
```

Beispiel 43: geschachtelte Funktionsaufrufe

```
int fct_2(int a) {
 return a*a;
}

int fct_1(int a, int b) {
 int c;
 c = 2*a;
 a = fct_2(3*c);
 return a+b;
 }

main() {
 int c,d,e;
 c = 2; d = 3;
 e = fct_1(c,d);     /* e = 147 */
}
```

Beispiel 44: Zeiger als Parameter

Funktion zum Austausch zweier Variableninhalte - falsch:

```
void austausch(int a, int b) {
  /* Funktion beeinflußt nur lokale Variable a und b */
 int hilf;
 hilf = b;
 b = a;
 a = hilf;
}

main() {
 int x = 1,y = 2;
 austausch(x,y);
}
```

Funktion zum Austausch zweier Variableninhalte - richtig:

```
void austausch(int *a, int *b) {
  /* Funktion beeinflußt aktuelle Parameter */
 int hilf;
 hilf = *b;
 *b = *a;
 *a = hilf;
 }

main() {
 int x = 1,y = 2;
 austausch(&x,&y);
}
```

Beispiel 45: Felder als Parameter

```
void vektorsumme(int a[], int b[], int c[], int laenge) {
 /* Funktion beeinflußt aktuelle Parameter, da Feld = Zeiger */
 int i;
 for (i=0;i<laenge;i++)
  c[i] = a[i] + b[i];
}
```

```
main() {
 int x[5], y[5], summe[5];
 ...
 vektorsumme(x,y,summe,5);
 }
```

Beispiel 46: Zeiger als Rückgabetyp

```
int *max(int *a,int *b) {
 if (*a>*b)
   return a;
  else return b; }

main() {
 int x = 1, y = 2, *m;
 m = max(&x,&y);   /* Zeiger auf Variable mit maximalem Wert */
}
```

Beispiel 47: statische Variable

```
int zaehler() {
 static int count;
 count++;
 return count;
}

main() {
 printf("%d\n",zaehler());  /* Ausgabe: 1 */
 printf("%d\n",zaehler());  /* Ausgabe: 2 */
 printf("%d\n",zaehler());  /* Ausgabe: 3 */
}
```

Beispiel 48: globale Variable

```
int zaehler;   /* globale Variable, überall benutzbar */

void fct_1() {
 zaehler++;
}
```

```
void fct_2() {
 zaehler++;
}

main() {
 zaehler++;
 fct_1();
 fct_2();
 printf("Anzahl durchlaufener Funktionen: %d\n",zaehler);
                                          /* Ausgabewert = 3 */
}
```

Beispiel 49: globale und lokale Variable

```
int zaehler;          /* globale Variable */

void fct() {
 int zaehler = 0;     /* lokale Variable */
 zaehler++;           /* Operation auf lokaler Variable */
}

main() {
 fct();
 zaehler++;           /* Operation auf globaler Variable */
}
```

Beispiel 50: globale Variable in mehreren Programm-Modulen

modul1.c:

```
int ueberall = 100;    /* Deklaration und Initialisierung
                          einer globalen Variablen */
```

modul2.c:

```
#include <stdio.h>

extern int ueberall;   /* Bezug auf globale Variable
                          in modul1.c */
main() {
 printf("%d\n",ueberall);     /* Ausgabe: 100 */
}
```

Beispiel 51: Präprozessoranweisung #include

structdf.h:

```
struct datum {
 int tag, monat, jahr; };
```

programm.c:

```
#include "structdf.h"
main() {
 struct datum date;
 date.tag = 1;
 ...
}
```

Beispiel 52: Abkürzungen mit #define

```
#define PI 3.14159265389793
main() {
 double a_grad, a_bogen;
 ...
 a_bogen = PI/180.*a_grad;
}
```

wird durch Präprozessor umgewandelt in

```
main() {
 double a_grad, a_bogen;
 ...
 a_bogen = 3.14159265389793/180.*a_grad;
}
```

Beispiel 53: Konstantendefinition mit #define

```
#define grenze 20
...
float vektor[grenze];
...
for (i=0;i<grenze;i++)
  vektor[i] = ...;
...
```

wird durch Präprozessor umgewandelt in

```
...
float vektor[20];
...
for (i=0;i<20;i++)
  vektor[i] = ...;
...
```

Beispiel 54: Makrodefinition mit #define

```
#define average(a,b,c) (a+b+c)/3
main() {
float f_1,f_2,f_3,f_avg;
int i_1,i_2,i_3,i_avg;     ...
f_avg = average(f_1,f_2,f_3);
i_avg = average(i_1,i_2,i_3);
```

wird durch Präprozessor umgewandelt in

```
main() {
float f_1,f_2,f_3,f_avg;
int i_1,i_2,i_3,i_avg;     ...
f_avg = (f_1+f_2+f_3)/3;
i_avg = (i_1+i_2+i_3)/3;
```

Beispiel 55: Bedingte Compilierung mit #ifdef

```
#define TESTVERSION
...
vektor[i] = j;
#ifdef TESTVERSION
printf("i = %d, vektor[i] = %d",i,vektor[i]);
#endif
...
```

wird durch Präprozessor umgewandelt in

```
...
vektor[i] = j;
printf("i = %d, vektor[i] = %d",i,vektor[i]);
...
```

```
#undef TESTVERSION
...
vektor[i] = j;
#ifdef TESTVERSION
printf("i = %d, vektor[i] = %d",i,vektor[i]);
#endif
...
```

wird durch Präprozessor umgewandelt in

```
...
vektor[i] = j;
...
```

Beispiel 56: Struct als Basis zur Darstellung eines Graphen

```
struct ice_knoten {
 char       stadt[20];         /* Knotenattribut */
 struct ice_knoten *nachbar1;  /* Kante zum ersten Nachbarn */
 unsigned   entf1;             /* Attribut der ersten Kante */
 struct ice_knoten *nachbar2;
 unsigned   entf2;
 struct ice_knoten *nachbar3;
 unsigned   entf3;
}
```

Beispiel 57: Typdefinitionen für lineare Listen

```
/* Einfach verkettete Liste */

typedef struct listelem {
  int              wert;  /* Eintrag des Knotens */
  struct listelem  *next; /* Zeiger auf nächstes Listenel't */
} listelem;

listelem *head;      /* Listenkopf = Zeiger auf erstes
                        Listenelement */
```

```
/* Doppelt verkettete Liste */

typedef struct doplistelem {
  int wert;                    /* Eintrag des Knotens */
  struct doplistelem  *next;   /* Zeiger auf Nachfolger */
  struct doplistelem  *prev;   /* Zeiger auf Vorgänger */
} doplistelem;

doplistelem *dophead;  /* Listenkopf = Zeiger auf erstes
                          Listenelement */
```

Beispiel 58: Einfügen in einfach verkettete Liste

Einfügen nach einem bestimmten Element:

```
int einfueg(listelem *nachdiesem, listelem *einzufueg) {
    /* nachdiesem: Element, nach dem eingefügt werden soll
       einzufueg:  einzufügendes Element */
 if ((nachdiesem==NULL)||(einzufueg==NULL)) return -1;
                          /* Rückgabe eines Fehlercodes */
 /* Einketten des neuen Elements */
 einzufueg->next = nachdiesem->next;
 nachdiesem->next = einzufueg;
 return 0;
}
```

Einfügen eines neuen ersten Elements:

```
int neuerst(listelem **liste, listelem *einzufueg) {
    /* liste:      Liste, in die eingefügt werden soll
       einzufueg:  einzufügendes Element */
 if ((einzufueg==NULL)||(liste==NULL)) return -1;
                            /* Rückgabe eines Fehlercodes */
 /* Einketten des neuen Elements mit Änderung von liste */
 einzufueg->next = *liste;
 *liste = einzufueg;
 return 0;
}
```

Einfügen eines neuen letzten Elements:

```
int neuletzt(listelem **liste, listelem *einzufueg) {
    /* liste:      Liste, in die eingefügt werden soll
       einzufueg:  einzufügendes Element */
 listelem *hilfzeig;
 if ((einzufueg==NULL)||(liste==NULL)) return -1;
                          /* Rückgabe eines Fehlercodes */
 if (*liste==NULL) {
 /* Falls Liste leer, Änderung von liste */
  einzufueg->next = NULL;
  *liste = einzufueg; }
 else {
 /* Falls Liste nicht leer: Durchlaufen und Einketten hinten */
  hilfzeig = *liste;
  while (hilfzeig->next!=NULL)
    hilfzeig = hilfzeig->next;
  einfueg(hilfzeig,einzufueg);
 }
 return 0;
}
```

Beispiel 59: Ausfügen aus einfach verketteter Liste

```
int ausfueg(listelem **liste, listelem *auszufueg) {
    /* liste:      Liste, aus der ausgefügt werden soll
       auszufueg:  auszufügendes Element */
 listelem *hilfzeig;
 if ((auszufueg==NULL)||(liste==NULL)) return -1;
 if (*liste==NULL) return -1;    /* -1 = Fehler */
 if (auszufueg==*liste) {
  /* neues erstes Element */
  *liste=(*liste)->next;
  return 0;   /* 0 = ok */
 }
 hilfzeig = *liste;
 /* Suche des Vorgängers des auszufügenden Elements */
 while (hilfzeig->next!=auszufueg) {
  if (hilfzeig->next == NULL)
   return -1;
```

```
  hilfzeig = hilfzeig->next;
 }
 /* Ausketten des Elements */
 hilfzeig->next = hilfzeig->next->next;
 return 0;
}
```

Beispiel 60: Durchlaufen einer einfach verketteten Liste

```
void durchlauf(listelem *liste) {
 /* liste: zu durchlaufende Liste */
 listelem *hilfzeig;
 hilfzeig = liste;
 while (hilfzeig!=NULL) {
  /* Schleife durch alle Listenelemente */
  printf("%d ",hilfzeig->wert);
  hilfzeig = hilfzeig->next;
 }
 printf("\n");
}
```

Beispiel 61: Einfügen in doppelt verkettete Liste

Einfügen nach einem bestimmten Element:

```
int dopeinfueg(doplistelem *nachdiesem, doplistelem *einzufueg)
    /* nachdiesem: Element, nach dem eingefügt werden soll
       einzufueg:  einzufügendes Element */
{
 if ((nachdiesem==NULL)||(einzufueg==NULL)) return -1;
                          /* Rückgabe eines Fehlercodes */
 /* Einketten des neuen Elements */
 einzufueg->next = nachdiesem->next;
 einzufueg->prev = nachdiesem;
 nachdiesem->next->prev = einzufueg;
 nachdiesem->next = einzufueg;
 return 0;
}
```

Einfügen eines neuen ersten Elements:

```
int dopneuerst(doplistelem **liste,doplistelem *einzufueg) {
    /* liste:      Liste, in die eingefügt werden soll
       einzufueg:  einzufügendes Element */
 if ((einzufueg==NULL)||(liste==NULL)) return -1;
                            /* Rückgabe eines Fehlercodes */
 /* Einketten des neuen Elements mit Änderung von liste */
 if (*liste==NULL) {
  *liste = einzufueg;
  (*liste)->next = (*liste)->prev = *liste;
 }
 else {
  dopeinfueg((*liste)->prev,einzufueg);
  *liste = einzufueg;
 }
 return 0;
}
```

Einfügen eines neuen letzten Elements:

```
int dopneuletzt(doplistelem **liste,doplistelem *einzufueg) {
    /* liste:      Liste, in die eingefügt werden soll
       einzufueg:  einzufügendes Element */
 if ((einzufueg==NULL)||(liste==NULL)) return -1;
                            /* Rückgabe eines Fehlercodes */
 if (*liste==NULL) {
 /* Falls Liste leer, Änderung von liste */
  *liste = einzufueg;
  (*liste)->next = (*liste)->prev = *liste;
 }
 else
  /* Einketten des neuen Elements */
  dopeinfueg((*liste)->prev,einzufueg);
 return 0;
}
```

Beispiel 62: Ausfügen aus doppelt verketteter Liste

```
int dopausfueg(doplistelem **liste,doplistelem *auszufueg) {
    /* liste:      Liste, aus der ausgefügt werden soll
       auszufueg:  auszufügendes Element */
 if ((liste==NULL)||(auszufueg==NULL)) return -1;
 if (*liste==NULL) return -1;
                            /* Rückgabe eines Fehlercodes */
 if (auszufueg->next==auszufueg) {
  /* Falls einziges Element: Liste ist jetzt leer */
  *liste=NULL;
  return 0;
 }
 if (*liste==auszufueg)
  /* Änderung von liste */
  *liste=(*liste)->next;
 /* Ausketten des Elements */
 auszufueg->next->prev = auszufueg->prev;
 auszufueg->prev->next = auszufueg->next;
 return 0;
}
```

Beispiel 63: Durchlaufen einer doppelt verketteten Liste

```
void dopdurchlauf(doplistelem *liste) {
 /* liste: zu durchlaufende Liste */
 doplistelem *hilfzeig;
 if (liste==NULL) return;
 hilfzeig = liste;
 do {
  /* Schleife durch alle Listenelemente */
  printf("%d ",hilfzeig->wert);
  hilfzeig = hilfzeig->next;
 }
 while (hilfzeig!=liste);
 printf("\n");
}
```

Beispiel 64: Typdefinition für binären Baum

```
typedef struct treeelem {
  int wert;                 /* Eintrag des Knotens */
  struct treeelem  *left;   /* Zeiger auf linken Sohn */
  struct treeelem  *right;  /* Zeiger auf rechten Sohn */
} treeelem;

treeelem *head;      /* Wurzel des Baums */
```

Beispiel 65: rekursive Funktion - Fakultätsberechnung

```
long fak(unsigned n) {
 long hilf;
 if (n<=1) hilf = 1;              /* 0! = 1! = 1 */
 if (n>1)  hilf = n * fak(n-1);   /* n! = n*(n-1)! */
 return hilf;
}

main() {
 unsigned eingabe;
 printf("Bitte n>=0 eingeben: ");
 scanf("%u",&eingabe);
 printf("fak(%u) = %ld",eingabe,fak(eingabe));
}
```

optimierte Fassung:

```
long fak(unsigned n) {
 if (n<=1) return 1;
   else return n*fak(n-1);
}
```

Beispiel 66: rekursive Funktion - Umkehren eines Strings

```
void umkehren(char strg[], unsigned laenge) {
 char hilf;
 if (laenge>1) {
  hilf=strg[0]; strg[0]=strg[laenge-1]; strg[laenge-1]=hilf;
  umkehren(&strg[1],laenge-2); }
}
```

Beispiel 67: Durchlaufen eines Binärbaums in Pre-, In- oder Postorder

```
#define preorder  0
#define inorder   1
#define postorder 2

void durchlauf(treeelem *baum,int order) {

 if (baum==NULL)
  return;

 switch(order) {

  case preorder:
   printf("%d ",baum->wert);
   durchlauf(baum->left,order);
   durchlauf(baum->right,order);
   break;

  case inorder:
   durchlauf(baum->left,order);
   printf("%d ",baum->wert);
   durchlauf(baum->right,order);
   break;

  case postorder:
   durchlauf(baum->left,order);
   durchlauf(baum->right,order);
   printf("%d ",baum->wert);
   break;

  }
}
```

Beispiel 68: Ausdrucken einer Baumstruktur (mit Einrückungen)

```
void drucke_rek(treeelem *baum,int space);

void drucke(treeelem *baum) {
    /* treeelem: auszudruckender Baum */
 drucke_rek(baum,0);
}

void drucke_rek(treeelem *baum,int space) {
    /* treeelem: auszudruckender Baum
       space:    Anzahl der Einrückungen */

const int tab=4;  /* Leerzeichen bei Einrückung */
int i;

 if (baum==NULL)
  return;

 drucke_rek(baum->right,space+1);

 for (i=1;i<=space*tab;i++)
  printf(" ");
 printf("%d\n",baum->wert);

 drucke_rek(baum->left,space+1);

}
```

Beispielausgabe:

```
        10
    8
        6
4
    2
```

Beispiel 69: Ausdrucken einer Baumstruktur (mit Kanten) - nur Borland-C unter DOS

```
void drucke_rek(treeelem *baum,
                        int left,int top,int right,int bottom);

void drucke(treeelem *baum) {
 clrscr();
 /* Drucke gesamten Baum auf dem gesamten Bildschirm aus */
 drucke_rek(baum,1,1,80,25);
 /* Definiere wieder Gesamtbildschirm als Ausgabefenster */
 window(1,1,80,25);
}

void drucke_rek(treeelem *baum,
                        int left,int top,int right,int bottom) {
 /* baum:            auszudruckender Baum */
 /* left,...,bottom: Bildschirmausschnitt für das Ausdrucken
                     (siehe window()-Funktion) */

int i;

 if (baum==NULL)
  return;

 /* Setzen des Bildschirmausschnitts */
 window(left,top,right,bottom);

 /* Ausdrucken der Kante, die von oben zur Wurzel führt */
 gotoxy(1+(right-left)/2,1);
 printf(":");
 gotoxy(1+(right-left)/2,2);
 printf(":");

/* Ausdrucken der Kante, die zum linken Unterbaum führt */
 if (baum->left!=NULL) {
  gotoxy((right-left)/4+1,3);
  for (i=(right-left)/4+1;i<(right-left)/2;i++)
   printf("-");
 }
```

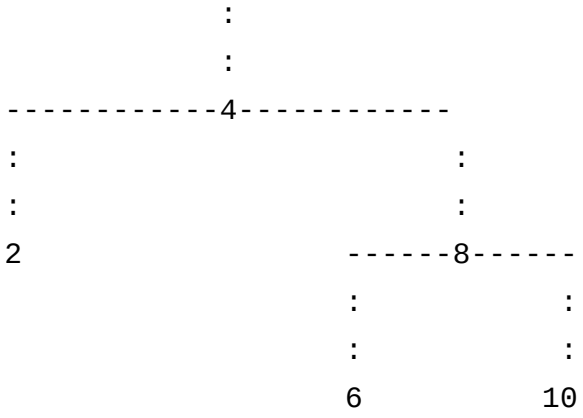
```
  /* Ausdrucken der Kante, die zum rechten Unterbaum führt */
   if (baum->right!=NULL) {
    gotoxy((right-left)/2,3);
    for (i=(right-left)/2;i<=1+(3*(right-left))/4;i++)
     printf("-");
   }

   /* Ausdrucken des Wurzelwerts */
   gotoxy((right-left)/2,3);
   printf("%3d",baum->wert);

   /* rekursives Ausdrucken des linken und rechten Unterbaums
      in entsprechenden Teilbereichen des Bildschirms */
   drucke_rek(baum->left,left,top+3,left+(right-left)/2,bottom);
   drucke_rek(baum->right,left+(right-left)/2+1,top+3,right,
                                                       bottom);

  }
```

Beispielausgabe (schematisch):



Beispiel 70: Löschen aller Einträge eines Baums

```
void loeschen(treeelem **baum) {

 if ((*baum)==NULL)
  return;

 /* rekursives Löschen des linken und rechten Unterbaums */
 loeschen(&((*baum)->left));
 loeschen(&((*baum)->right));

 /* Löschen der Wurzel */
 printf("Loesche %d\n",(*baum)->wert);
 free(*baum);
 *baum=NULL;

}
```

Beispiel 71: Suchen in einem Suchbaum

```
treeelem *suche(treeelem *baum, int suchwert) {
  /* baum:     zu durchsuchender Baum
     suchwert: zu suchender Wert */

 if (baum==NULL)
  /* Suchwert nicht gefunden */
  return NULL;

 if (baum->wert==suchwert)
  /* Suchwert gefunden */
  return baum;

 else if (baum->wert>suchwert)
  /* Suchwert im linken Unterbaum */
   return suche(baum->left,suchwert);

  else  /* Suchwert im rechten Unterbaum */
   return suche(baum->right,suchwert);

}
```

Beispiel 72: Einfügen in einen Suchbaum

```
int insuchbaum(treeelem **baum,int einwert) {
  /* baum:    Baum, in den eingefügt werden soll
     einwert: einzufügender Wert */

 if ((*baum)==NULL) {
  /* Blatt erreicht → Erzeugen und Einfügen
     eines neuen Elements */
  *baum=(treeelem *) malloc(sizeof(treeelem));
  (*baum)->wert=einwert;
  (*baum)->left=(*baum)->right=NULL;
  return 0; }

 if ((*baum)->wert==einwert)
  /* Wert ist schon im Baum vorhanden */
  return -1;

 else if ((*baum)->wert>einwert)
   /* Einfügen in den linken Unterbaum */
   return insuchbaum(&((*baum)->left),einwert);

  else
   /* Einfügen in den rechten Unterbaum */
   return insuchbaum(&((*baum)->right),einwert);

}
```

Beispiel 73: Ausfügen aus einem Suchbaum

```
int del_min(treeelem **baum);
 /* Fügt Element mit minimalem Eintrag aus einem Baum aus
    und liefert diesen Eintrag zurück - siehe unten */

int aussuchbaum(treeelem **baum,int auswert) {
 /* baum:    Baum, aus dem ausgefügt werden soll
    auswert: auszufügender Wert */

 treeelem *hilf;

 if ((*baum)==NULL)
  /* Wert nicht gefunden */
  return -1;
```

```
 if ((*baum)->wert==auswert) {
  /* Wert gefunden */

  if ((*baum)->left==NULL) {
   /* Hochholen des rechten Unterbaums */
   hilf=*baum;
   *baum=(*baum)->right;
   free(hilf);
  }

  else if ((*baum)->right==NULL) {
   /* Hochholen des linken Unterbaums */
   hilf=*baum;
   *baum=(*baum)->left;
   free(hilf);
  }

  else
   /* Hochholen des minimalen Elements aus rechtem Teilbaum */
   (*baum)->wert=del_min(&((*baum)->right));

  return 0;

 }

 else if ((*baum)->wert>auswert)
  /* Suche im linken Unterbaum */
   return aussuchbaum(&((*baum)->left),auswert);
  else /* Suche im rechten Unterbaum */
   return aussuchbaum(&((*baum)->right),auswert);
}

int del_min(treeelem **baum) {
 /* Fügt Element mit minimalem Eintrag aus einem Baum aus
    und liefert diesen Eintrag zurück */

 treeelem *hilf;
 int retwert;

 if ((*baum)->left==NULL) {
  /* minimaler Wert gefunden */
  hilf=*baum;
```

```
  retwert = (*baum)->wert;
  /* Ausfügen */
  *baum = (*baum)->right;
  free(hilf);
  return retwert; }

  else /* Fortsetzung der Suche */
   return del_min(&((*baum)->left));

}
```

Beispiel 74: Interpreter für arithmetische Ausdrücke

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct token { /* Einträge der Tokenliste */
 char zeichen;         /* zulässige Zeichen: (,),+,-,*,/,' ' */
 int  wert;            /* wert nur gültig, wenn zeichen==' ' */
 struct token *next;
} token;

typedef struct baumeintrag {  /* Einträge des Strukturbaums */
 char operator;      /* zulässige Zeichen: +,-,*,/,' ' */
 int  wert;          /* wert nur gültig, wenn operator==' ' */
 struct baumeintrag *links, *rechts;
} baumeintrag;

token *lexikal(char *ausdr) {
 /* führt für ausdr eine lexikalische Analyse durch und liefert
    als Rückgabewert eine entsprechende Tokenliste (zulässige
    Token: Zahlenwerte, Klammern, Operatoren). Die Tokenliste
    ist rückwärts verkettet! */
 token *toklist=NULL, *neu;  /*zum Aufbau der Tokenliste */
 unsigned lg=strlen(ausdr);
 int i=0,j;
 char zahlstring[15];  /* zur Aufnahme einer Zahl */
```

```
 while (i<lg) {
  /* Durchlauf des Strings */
  neu=(token *)malloc(sizeof(token));
  if (neu==NULL)
   { printf("Fehler bei malloc\n"); exit(-1); }
  if (ausdr[i]<'0'||ausdr[i]>'9')
   /* Zeichen gefunden */
    neu->zeichen=ausdr[i++];
   else {
     /* Zahl gefunden.
        Übertrage ihre Ziffern nach zahlstring */
     j=0;
     do {
      zahlstring[j++]=ausdr[i++];
     } while(ausdr[i]>='0'&&ausdr[i]<='9');
     zahlstring[j]='\0';
     neu->wert=atoi(zahlstring);  /* Umwandlung nach int */
     neu->zeichen=' ';
    }
  /* Einketten des neuen Token */
  if (toklist==NULL) {
     toklist=neu;
     toklist->next=NULL;
    }
   else {
     neu->next=toklist;
     toklist=neu;
    }
 }
 return toklist;
}

void error() {
 /* Zur Anzeige eines syntaktischen Fehlers */
 printf("syntaktischer Fehler!\n");
 exit(-1);
}
```

```
/* Die folgenden Funktionen führen für eine Tokenliste eine
   Syntaxanalyse durch und liefern als Rückgabewert den zu-
   gehörigen Strukturbaum des arithmetischen Ausdrucks. Die
   Tokenliste wird als rückwärts verkettet vorausgesetzt, die
   Analyse des Ausdrucks geht also von hinten nach vorne vor.
   Die einzelnen Funktionen analysieren ganze Ausdrücke, Terme
   und Faktoren. */

baumeintrag *syntausdr(token **toklist);
                                /* Prototyp wg. Rekursion */

baumeintrag *syntfaktor(token **toklist) {
 baumeintrag *pt;
 if ((*toklist)->zeichen==')') {
   /* Ende eines Teilausdrucks gefunden */
   (*toklist)=(*toklist)->next;
   pt=syntausdr(toklist);
   /* Test, ob zugehörige Klammer vorhanden */
   if ((*toklist==NULL)||((*toklist)->zeichen!='('))
    error();
   (*toklist)=(*toklist)->next;
   }
  else {
   /* Faktor ist einzelne Zahl */
   pt=(baumeintrag *)(malloc(sizeof(baumeintrag)));
   if (pt==NULL)
    { printf("Fehler bei malloc\n"); exit(-1); }
   if ((*toklist)->zeichen!=' ')
    error();
   pt->operator=' ';
   pt->wert=(*toklist)->wert;
   pt->links=pt->rechts=NULL;
   (*toklist)=(*toklist)->next;
  }
 return pt;
}
```

```
baumeintrag *syntterm(token **toklist) {
 baumeintrag *pt, *pthilf;
 pt=syntfaktor(toklist);
 if ((*toklist!=NULL)&&((*toklist)->zeichen=='*'
                        ||(*toklist)->zeichen=='/')) {
  /* Term besteht aus mindestens zwei Faktoren
     -> Aufbau eines entsprechenden Teilbaums */
  pthilf=(baumeintrag *)(malloc(sizeof(baumeintrag)));
  if (pthilf==NULL)
   { printf("Fehler bei malloc\n"); exit(-1); }
  pthilf->operator=(*toklist)->zeichen;
  pthilf->rechts=pt;
  (*toklist)=(*toklist)->next;
  pthilf->links=syntterm(toklist);
  pt=pthilf;
 }
 return pt;
}

baumeintrag *syntausdr(token **toklist) {
 baumeintrag *pt, *pthilf;
 pt=syntterm(toklist);
 if (((*toklist)!=NULL)&&((*toklist)->zeichen=='-')
          &&((*toklist)->next==NULL
                  ||(*toklist)->next->zeichen=='(')) {
  /* Ausdruck ist einzelne Zahl mit negativem Vorzeichen */
  pt->wert=-pt->wert;
  (*toklist)=(*toklist)->next;
  return pt;
 }
 if (*toklist==NULL||
         (((*toklist)->zeichen!='+')
                  &&((*toklist)->zeichen!='-')))
  /* Ausdruck besteht nur aus einem Term */
  return pt;
 /* Ausdruck besteht aus mindestens zwei Termen
    -> Aufbau eines entsprechenden Teilbaums */
 pthilf=(baumeintrag *)(malloc(sizeof(baumeintrag)));
```

```
if (pthilf==NULL)
  { printf("Fehler bei malloc\n"); exit(-1); }
 pthilf->rechts=pt;
 pthilf->operator=(*toklist)->zeichen;
 (*toklist)=(*toklist)->next;
 pthilf->links=syntausdr(toklist);
 return pthilf;
}

int auswert(baumeintrag *struktbaum) {
 /* wertet den durch struktbaum beschriebenen arithmetischen
    Ausdruck aus und liefert seinen Wert zurück */
 switch (struktbaum->operator) {
  case ' ': return struktbaum->wert;
  case '+': return auswert(struktbaum->links)
                                  +auswert(struktbaum->rechts);
  case '-': return auswert(struktbaum->links)
                                  -auswert(struktbaum->rechts);
  case '*': return auswert(struktbaum->links)
                                  *auswert(struktbaum->rechts);
  case '/': return auswert(struktbaum->links)
                                  /auswert(struktbaum->rechts);
 }
}

main() {

 char ausdruck[80]; /* auszuwertender arithm. Ausdruck */
 token *tokenliste;
 baumeintrag *strukturbaum;

 /* Ausdruck einlesen */
 printf("Bitte arithmetischen Ausdruck eingeben: ");
 gets(ausdruck);

 /* Lexikalische Analyse = Aufbau der Tokenliste */
 tokenliste=lexikal(ausdruck);
```

```
/* Syntaxanalyse = Aufbau des Strukturbaums */
 strukturbaum=syntausdr(&tokenliste);
 if (tokenliste!=0) error();

 /* Auswertung des Ausdrucks */
 printf("\nWert des Ausdrucks: %d\n",auswert(strukturbaum));

}
```