```
/*********************************************************************/
/*                                                                   */
/* Packer.C - DOS Programm zur Demonstration des Huffmancodes        */
/* (C) by Daniel Becker 1998. Entwickelt unter Borland Turbo C V2.0  */
/*                                                                   */
/*********************************************************************/
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <mem.h>
#include <io.h>
#include <fcntl.h>

unsigned long bytes[256];
unsigned int codelen,length[256];
struct liste
{
   unsigned char inhalt[256]; /* Die Inhalte */
   double p;                  /* die gemeinsamen Wahrscheinlichkeiten */
   int count;                 /* Wieviele */
   struct liste *next,*prev;
} *root;

struct baum
{
   struct baum *nullast, *einsast;
   unsigned char zeichen;
} *baumroot;

void count_bytes(int h)
{
   unsigned char c;
   int i;
   long maximum;
   struct liste *n;

   lseek(h,0,SEEK_SET);
   while (!eof(h))
   {
      _read(h,&c,1);
      bytes[c]++;
   }

   for (maximum=i=0; i<256; i++) maximum+=bytes[i];
   root=NULL; codelen=0;
   for (i=0; i<256; i++)
   {
      if (!bytes[i]) continue;
      n=(struct liste *)malloc(sizeof(struct liste));
      n->inhalt[0]=i;
      n->p=(double)(bytes[i])/(double)(maximum);
      n->count=1;
      if (root) root->prev=n;
      n->next=root;
      n->prev=NULL;
      root=n;
      codelen++;
      bytes[i]=length[i]=0;
   }
}

void generate_code()
{
   struct liste *n,*kl,*kkl;
   unsigned int z,i,oldcodelen;
   unsigned long byte;

   oldcodelen=codelen;
   while (codelen>1)
   {
      n=kl=kkl=root; /* kl und kkl nur zur initialisierung */
   /* Suche die kleineste Wahrscheinlichkeit! */
      while (n)
      {
         if (n->p<kkl->p) kkl=n;
         else
         if (n->p<kl->p) kl=n;
         else
```

```
         if (kl==kkl) kl=n;
         n=n->next;
      }
   /* Fasse zusammen */
      if (kl->p<=kkl->p) n=kl; else n=kkl; /* n ist nun der kleinste */
   /* 1. Schritt: bei kleinstem p nullen einsetzen! */
      for (i=0; i<n->count; i++)
      {
         bytes[n->inhalt[i]]=(bytes[n->inhalt[i]]<<1)|0;
         length[n->inhalt[i]]++;
      }

      if (n==kl) n=kkl; else n=kl; /* tauschen */
   /* 2. Schritt: bei anderem einsen einsetzen! */
      for (i=0; i<n->count; i++)
      {
         bytes[n->inhalt[i]]=(bytes[n->inhalt[i]]<<1)|1;
         length[n->inhalt[i]]++;
      }

   /* 3. Schritt kl und kkl zusammenfassen */
      for (i=0; i<kkl->count; i++)
         kl->inhalt[kl->count+i]=kkl->inhalt[i];
      kl->count+=kkl->count;
      kl->p+=kkl->p;
   /* kkl löschen! */
      if (kkl->prev) (kkl->prev)->next=kkl->next;
      if (kkl->next) (kkl->next)->prev=kkl->prev;
      if (kkl==root) root=kkl->next;
      free((struct liste *)kkl);
      codelen--;
   }
   codelen=oldcodelen;
   for (i=0; i<256; i++)
   {
      if (!length[i]) continue;
      byte=bytes[i]; bytes[i]=0;
      for (z=0; z<length[i]; z++)
      {
         bytes[i]<<=1;
         bytes[i]|=byte&0x1;
         byte>>=1;
      }
   }
}

/* Diese Funktion löscht alle Daten nach der Codierung */
/* und gibt den Speicher frei. */
/* Es sollte aber nur noch root übrig sein. */
void freeall()
{
   struct liste *n,*n1;

   n=root;
   while (n)
   {
      n1=n->next;
      free(n);
      n=n1;
   }
}

/* rekursive Funktion zum löschen des Codebaumes */
/* es wird ebenfalls der gesamte Speicher wieder */
/* freigegeben. */
void freebaum(struct baum *b)
{
   if (!b) return;
   if (b->nullast)
   {
      freebaum(b->nullast);
      b->nullast=NULL;
   }
   if (b->einsast)
   {
      freebaum(b->einsast);
      b->einsast=NULL;
   }
```

```
   free(b);
}

/* Diese Funktion wird im augenblick nicht mehr aufgerufen */
/* ist aber immer noch enthalten */
void ausgabe()
{
   int i,j,z;

   clrscr();
   for (i=0,j=0; i<256; i++)
   {
      if (!length[i]) continue;
      gotoxy(40*(j%2)+1,wherey());
      printf("%X(%c):",i,(unsigned char)i);
      for (z=length[i]-1; z>=0; z--)
      {
         printf("%d",(bytes[i]>>z)&0x1);
      }
      if (j%2) printf("\n");
      j++;
   }
}

/* Diese Funktion liest ein Byte, codiert es und schreibt so viele Bit, */
/* wie benötigt werden. Da nur Byteweise geschrieben werden kann, muß */
/* man hier ein bißchen tricksen. */
void write_packed_data(int rh, int wh)
{
   int i,codepos,bytepos;
   unsigned long code;
   unsigned char wrbyte,c;

   lseek(rh,0,SEEK_SET);
   lseek(wh,0,SEEK_SET);

/* Dateikopf schreiben */
   code=filelength(rh);
   _write(wh,&code,4);
   _write(wh,&codelen,2);
   for (i=0; i<256; i++)
   {
      if (!length[i]) continue;
      _write(wh,&i,1);
      _write(wh,&bytes[i],4);
      _write(wh,&length[i],2);
   }
 /* Ende Dateikopf */

   wrbyte=0; bytepos=7;
   while (!eof(rh))
   {
      _read(rh,&c,1);
      codepos=length[c]-1; code=bytes[c];
      while (codepos>-1)
      {
         wrbyte|=(((code>>codepos)&0x1)<<bytepos);
         bytepos--; codepos--;
         if (bytepos<0)
         {
            _write(wh,&wrbyte,1);
            wrbyte=0; bytepos=7;
         }
      }
   }
   _write(wh,&wrbyte,1);
}

/* Dieses kleine rekursive Funktiönchen erzeugt den Baum successive bei */
/* dem Versuch ein Element bezüglich seines Codes einzufügen. Sollte dabei */
/* Blattwerk notwendig werden, werden diese gleich mit erzeugt. */
void insertbaum(struct baum *b, unsigned char byte, unsigned long code, unsigned int len)
{
   unsigned char d;

   if (!len)
   {
      b->zeichen=byte;
```

```
      return;
   }
   d=(code>>(len-1))&0x1; /* Links- oder rechts?! */

   if (!d)
   {
      if (!b->nullast)
      {
         b->nullast=(struct baum *)malloc(sizeof(struct baum));
         b->nullast->zeichen=0;
         b->nullast->nullast=NULL;
         b->nullast->einsast=NULL;
      }
      insertbaum(b->nullast,byte,code,len-1);
   }
   else
   {
      if (!b->einsast)
      {
         b->einsast=(struct baum *)malloc(sizeof(struct baum));
         b->einsast->zeichen=0;
         b->einsast->nullast=NULL;
         b->einsast->einsast=NULL;
      }
      insertbaum(b->einsast,byte,code,len-1);
   }
}

/* Decodiere name1 und mache name2 daraus... */
void decode(char *name1, char *name2)
{
   int rh,wh,i,bytepos;
   struct baum *b;
   unsigned long filelen;
   unsigned char byte,d;

   setmem(length,sizeof(length),0);
   setmem(bytes,sizeof(bytes),0);

   rh=_open(name1,O_BINARY | O_RDONLY);
   if (rh<1)
   {
      puts("Datei nicht gefunden!");
      exit(0);
   }
   wh=_creat(name2,O_BINARY | O_WRONLY);

   /* Kopf einlesen! */
   baumroot=(struct baum *)malloc(sizeof(struct baum));
   baumroot->zeichen=0;
   baumroot->nullast=baumroot->einsast=NULL;
   _read(rh,&filelen,4);
   _read(rh,&codelen,2);
   for (i=0; i<codelen; i++)
   {
      _read(rh,&byte,1);
      _read(rh,&bytes[byte],4);
      _read(rh,&length[byte],2);
      insertbaum(baumroot,byte, bytes[byte], length[byte]);
   }
   /*ausgabe();*/

   b=baumroot;
   while (!eof(rh))
   {
      bytepos=7; _read(rh,&byte,1);
      while (bytepos>=0)
      {
         if (!b->nullast && !b->einsast) /* ist ein Blatt -> Ziel */
         {
            if (filelength(wh)==filelen) break;
            write(wh,&(b->zeichen),1);
            b=baumroot;
         }
         d=(byte>>bytepos)&0x1;
         if (!d)
            b=b->nullast;
         else
```

```
            b=b->einsast;
         bytepos--;
      }
   }
   _close(rh);
   _close(wh);
   _chmod(name2,1,0x20);
   freebaum(baumroot);
}

void code(char *datname, char *codname)
{
   unsigned long rd,wd;
   int rh,wh;

   setmem(bytes,sizeof(bytes),0);

   rh=_open(datname,O_BINARY | O_RDONLY);
   if (rh<1)
   {
      puts("Datei nicht gefunden!");
      exit(0);
   }
   wh=_creat(codname,O_BINARY | O_WRONLY);

   count_bytes(rh); /* Zählt die Vorkommen von Bytes und errechnet */
                    /* Wahrscheinlichkeiten und generiert eine Liste. */

   generate_code(); /* erzeugt aus den Informationen den Code */

   /*ausgabe();*/

   write_packed_data(rh,wh);

   rd=filelength(rh);
   wd=filelength(wh);
   _close(rh);
   _close(wh);
   freeall();
   _chmod(codname,1,0x20);

   printf("\n%s -> %s : %ld%% (%ld%%)",datname,codname,wd*100L/rd,(wd-(codelen*7)+2)*100L/rd);
}

void main(int argi, char **args)
{
   if (argi<2) exit(0);

   if (toupper(*args[1])=='A') code(args[2],args[3]);
   if (toupper(*args[1])=='X') decode(args[2],args[3]);
}
```

**Die folgenden Prozentzahlen sind *mit*(*ohne)* Dateikopf zu interpretieren.**

Typische Ausgabe nach dem Packen einer Textdatei:

```
A(□):00110
D(□):00011
20( ):011
21(!):001111001010
22("):0101111011
23(#):00000111111
25(%):001111001001100
27('):0101101110
28(():0011110011
29()):0001011111
2A(*):01001101101
2B(+):00000111111001
2C(,):0011111
2D(-):0001010
2E(.):000000
2F(/):0000011110
30(0):0000101
31(1):010011111
32(2):000101110
33(3):010001101
34(4):0100111011
35(5):0101101111
36(6):0001011110
37(7):01011110101
38(8):0100011000
39(9):0100111010
3A(:):00000110
3B(;):0011110010110
3C(<):000100001
3D(=):0011110010010
3E(>):000001110
3F(?):001111001000
40(@):010011110
41(A):01011111
42(B):010011100
43(C):110001010
44(D):0100010
45(E):0000111
46(F):11000111
47(G):110001011
48(H):010110110
49(I):01000111
4A(J):01011110100
4B(K):0100110111
4C(L):010111100
4D(M):00010001
4E(N):01001100
4F(O):0100110101
50(P):00000100
51(Q):01001101100
52(R):11000100
53(S):100010
54(T):0000110
55(U):10001101
56(V):0100110100
57(W):001111000
58(X):000001111101
59(Y):001111001001101
5A(Z):100011000
5B([):00010011
5D(]):00000101
5E(^):00001000
5F(_):0011110010111100
61(a):11010
62(b):001110
63(c):010010
64(d):11001
65(e):101
66(f):1000111
67(g):110110
68(h):110111
69(i):1001
6A(j):0100011001
6B(k):0101100
6C(l):111111
6D(m):111110
6E(n):1110
6F(o):110000
70(p):00111101
71(q):00111100101110
72(r):0010
73(s):01010
74(t):11110
75(u):10000
76(v):11000110
77(w):0101110
78(x):0001000001
79(y):00010000001
7A(z):0100001
7B({):0011110010111101
7C(|):001111001011111
7D(}):0011110010011100
7E(~):00001001
81(□):01011010
84(„):00010010
8E(□):0000011111000
94(”):00010110
99(™):0011110010011101
9A(š):00010000000
E1(á):100011001
E6(æ):0011110010011110
F8(ø):0100000
FB(û):0011110010011111

datum.hlp -> datum.cod : 63% (62%)
```

Typische Ausgabe nach dem Packen von PACKER.EXE selbst:

```
0(□):011
1(□):101101
2(□):01011
3(□):1111111
4(□):000100
5(□):1100111
6(□):00111
7(□):1001111
8(□):110111
9(□):111011100
A(□):1100101
B(□):1001100
C(□):1010101
D(
□):101111110
E(□):0011011
F(□):110010010
10(□):01001110
11(□):001000000
12(□):000101000
13(□):1000101100
14(□):11110000
15(□):1000101000
16(□):10011010
17(□):1100010100
18(□):1110100100
19(□):0101010100
1A(□):10001011110
1B(□):11001100000
1C(□):0010010000
1D(□):11001100001
1E(-):0100000
1F():101001000
20( ):11111011
21(!):01000010
22("):1110100101
23(#):11111001100
24($):1001011100
25(%):00101001
26(&):111101
27('):00100000100
28(():1010010010
29()):10100100110
2A(*):01010100000
2B(+):111110010
2C(,):0101010001
2D(-):1011111110
2E(.):10010110
2F(/):100010111000
30(0):010010000
31(1):00101101000
32(2):1110100110
33(3):0010111
34(4):00101000000
35(5):01001000010
36(6):101011110
37(7):01001000011
38(8):10100100111
39(9):101111111110
3A(:):0010000011
3B(;):01010000
3C(<):1010111000
3D(=):00011000
3E(>):100011010
3F(?):1011101010
40(@):11001000
41(A):0010100001
42(B):1111001000
43(C):000101001
44(D):1100011100
45(E):0000110000
46(F):110000
47(G):0001101
48(H):1000101001
49(I):111110011101
4A(J):11000111010
4B(K):1100110001
4C(L):11000111011
4D(M):111010011110
4E(N):00100110
4F(O):1100011110
50(P):1010011
51(Q):111010011111
52(R):000011001
53(S):0000110001
54(T):101011111100
55(U):11001101
56(V):1110101
57(W):10111100
58(X):001010001
59(Y):11111010
5A(Z):0101000100
5B([):0000110100
5C(\):110001111110
5D(]):11110001
5E(^):110110
5F(_):00000000
60(`):101011111101
61(a):1010111111
62(b):00001110
63(c):0010011100
64(d):00000001
65(e):000011110
66(f):001101000
67(g):0010011101
68(h):001001111
69(i):010100011
6A(j):11000110
6B(k):110001111111
6C(l):000011011
6D(m):101010011100
6E(n):101110110
6F(o):000110010
70(p):1000101010
71(q):1000101111001
72(r):11110011
73(s):111011101
74(t):1011100
75(u):1110110
76(v):010001
77(w):00000010
78(x):000001000
79(y):0000110101
7A(z):1010010100
7B({):1010100111
7C(|):1001011101
7D(}):000110011
7E(~):0010001
7F(□):110010011
80(□):0001011
81(□):1000111
82(‚):10001011101000
83(ƒ):000111
84(„):100010111011
85(…):00100000101
86(†):01001001
87(‡):11000100
88(ˆ):010100100
89(‰):100000
8A(Š):0010101
8B(‹):11010
8C(Œ):10101000
8D(□):101110111
8E(□):00000011
8F(□):10111010110
90(□):100101111
91(‘):0101000101
92(’):10100101010
93(“):0000010010
94(”):01010100001
95(•):01010010100
96(–):000011111
97(—):000001010
98(˜):1001101100
99(™):10111010111
9A(š):1010110
9B(›):100001
9C(œ):0010010010
9D(□):101010011010
9E(□):101000100
9F(Ÿ):01010111110
A0( ):01010010101
A1(¡):010101110
A2(¢):00000100110
A3(£):10111111110
A4(¤):1000101110101
A5(¥):1000101101000
A6(¦):10101110100
A7(§):00000100111
A8(¨):0000010110
A9(©):10100101011
AA(ª):111100101
AB(«):11000101100
AC(¬):0000010111
AD(-):000010100
AE(®):0010010001
AF(¯):00101101010
B0(°):101000101
B1(±):1100010101
B2(²):1000101101001
B3(³):00001010100
B4(´):101000000
B5(µ):0101001011
B6(¶):000010101010
B7(·):0010010011
B8(¸):0100101
B9(¹):100110111
BA(º):001011011
BB(»):0000101011
BC(¼):100010110101
BD(½):000010101011
BE(¾):1111100111
BF(¿):10001011111
C0(À):11111100
C1(Á):101110100
C2(Â):010000110
C3(Ã):10100001
C4(Ä):101100
C5(Å):010000111
C6(Æ):00110101
C7(Ç):10111110
C8(È):000010110
C9(É):01010110
CA(Ê):010100110
CB(Ë):11101111
CC(Ì):1111001001
CD(Í):00001001
CE(Î):1010111001
CF(Ï):101011101101
D0(Ð):111111011
D1(Ñ):0000011
D2(Ò):10001001
D3(Ó):000010111
D4(Ô):00001000100
D5(Õ):10001011101001
D6(Ö):1001101101
D7(×):10001011011
D8(Ø):10001100
D9(Ù):1001000
DA(Ú):001101001
DB(Û):00100001
DC(Ü):0101010101
DD(Ý):00010101
DE(Þ):01001111
DF(ß):0101011111
E0(à):00100101
E1(á):11000101101
E2(â):010101001
E3(ã):10100011
E4(ä):010101011
E5(å):10001000
E6(æ):1000101011
E7(ç):00101101011
E8(è):1001001
E9(é):10111101
EA(ê):1100010111
EB(ë):001100
EC(ì):0100110
ED(í):1011111111111
EE(î):101001011
EF(ï):1010100111011
F0(ð):101000001
F1(ñ):00001000101
F2(ò):010100111
F3(ó):1010111011
F4(ô):000010000
F5(õ):0000100011
F6(ö):11111000
F7(÷):11101000
F8(ø):00101100
F9(ù):111111010
FA(ú):110011001
FB(û):100011011
FC(ü):1001110
FD(ý):101010010
FE(þ):1001010
FF(ÿ):11100
```

```
packer.exe -> packer.cod : 94% (83%)
```

Nun eine Wavedatei:

```
0(□):0100
1(□):10010
2(□):111111
3(□):111001
4(□):101111
5(□):001110
6(□):001001
7(□):100000
8(□):1111101
9(□):1111010
A(□):1101111
B(□):0111111
C(□):0011010
D(
□):0010101
E(□):1000010
F(□):0001111
10(□):0001000
11(□):11101101
12(□):11100011
13(□):11010111
14(□):11010011
15(□):10111000
16(□):01111010
17(□):01101111
18(□):01101011
19(□):10001111
1A(□):10000111
1B(□):10001011
1C(□):00011101
1D(□):00010111
1E(←):111011111
1F(▲):00010011
20( ):111011101
21(!):111101111
22("):111100111
23(#):111100011
24($):111000101
25(%):110111011
26(&):110111001
27('):110101011
28(():101110111
29()):110100101
2A(*):101110011
2B(+):101101111
2C(,):101110010
2D(-):011110111
2E(.):101101101
2F(/):011100011
30(0):011101111
31(1):011111010
32(2):011111011
33(3):101101110
34(4):110011001
35(5):011111001
36(6):011101110
37(7):011111000
38(8):101011110
39(9):011010011
3A(:):101101010
3B(;):101011010
3C(<):011101101
3D(=):101010010
3E(>):011100110
3F(?):101001110
40(@):101010100
41(A):011100010
42(B):010111100
43(C):101011111
44(D):011010101
45(E):110001100
46(F):110011101
47(G):001111100
48(H):101010001
49(I):101001101
4A(J):110011111
4B(K):010111111
4C(L):101000011
4D(M):101011011
4E(N):001011101
4F(O):100111110
50(P):011001000
51(Q):001111111
52(R):001100110
53(S):101010111
54(T):101011101
55(U):001011110
56(V):001101110
57(W):110000000
58(X):001111001
59(Y):010110111
5A(Z):110001000
5B([):001111010
5C(\):011010010
5D(]):011001010
5E(^):011001111
5F(_):011100001
60(`):001101101
61(a):001111000
62(b):101000000
63(c):011011001
64(d):110010101
65(e):011011010
66(f):100111011
67(g):110011000
68(h):101001111
69(i):001011011
6A(j):001100111
6B(k):110010110
6C(l):100111001
6D(m):110001101
6E(n):010111110
6F(o):101000001
70(p):011000100
71(q):011000011
72(r):101010110
73(s):011000000
74(t):110000101
75(u):101000110
76(v):011001101
77(w):110010111
78(x):001111110
79(y):110011111
7A(z):110001110
7B({):011000010
7C(|):110000001
7D(}):100111010
7E(~):001111011
7F(□):011010100
80(□):010110011
81(□):010111010
82(‚):101010000
83(ƒ):110011100
84(„):110010000
85(…):011000001
86(†):011000101
87(‡):001011001
88(ˆ):011100111
89(‰):010111101
8A(Š):011001011
8B(‹):100111111
8C(Œ):010111011
8D(□):011001100
8E(□):101010101
8F(□):011001110
90(□):001010010
91(‘):010111001
92(’):110001011
93(“):001010011
94(”):100111000
95(•):001101111
96(–):110000110
97(—):001101100
98(˜):110010011
99(™):011010001
9A(š):101001100
9B(›):110000111
9C(œ):001011000
9D(□):010110101
9E(□):101001010
9F(Ÿ):101001000
A0( ):001111101
A1(¡):101000100
A2(¢):011001001
A3(£):101011100
A4(¤):010110110
A5(¥):110000100
A6(¦):001011010
A7(§):110000011
A8(¨):110001001
A9(©):011000110
AA(ª):010111000
AB(«):011010000
AC(¬):011100000
AD(-):101000101
AE(®):100111101
AF(¯):011011101
B0(°):010110010
B1(±):101000010
B2(²):011011100
B3(³):110000010
B4(´):100111100
B5(µ):110010001
B6(¶):001011100
B7(·):011011000
B8(¸):101010011
B9(¹):011000111
BA(º):010110100
BB(»):101001011
BC(¼):101101000
BD(½):011101000
BE(¾):101000111
BF(¿):001011111
C0(À):110010100
C1(Á):011101001
C2(Â):110010010
C3(Ã):101011000
C4(Ä):101011001
C5(Å):110011011
C6(Æ):011011011
C7(Ç):110001010
C8(È):110011010
C9(É):101001001
CA(Ê):011101011
CB(Ë):110011110
CC(Ì):101101011
CD(Í):011101010
CE(Î):011101100
CF(Ï):101101001
D0(Ð):101110110
D1(Ñ):011110110
D2(Ò):101101100
D3(Ó):110100100
D4(Ô):110101010
D5(Õ):110101100
D6(Ö):110111000
D7(×):110111010
D8(Ø):110101101
D9(Ù):111000100
DA(Ú):111000000
DB(Û):111100010
DC(Ü):111100001
DD(Ý):111100110
DE(Þ):111101110
DF(ß):111011100
E0(à):111011110
E1(á):00010010
E2(â):00010110
E3(ã):00011100
E4(ä):10001010
E5(å):10000110
E6(æ):10001110
E7(ç):00101000
E8(è):01011000
E9(é):00110010
EA(ê):01110010
EB(ë):10111010
EC(ì):11010100
ED(í):11110010
EE(î):11101100
EF(ï):11110110
F0(ð):0001010
F1(ñ):1000100
F2(ò):1000110
F3(ó):0011000
F4(ô):0111100
F5(õ):1101000
F6(ö):1110000
F7(÷):1111100
F8(ø):000110
F9(ù):001000
FA(ú):100110
FB(û):101100
FC(ü):110110
FD(ý):111010
FE(þ):01010
FF(ÿ):0000
```

```
viclose.wav -> viclose.cod : 91% (91%)
```

Nun noch ein Bild (BMP):

```
0(□):00011
1(□):011100010
2(□):00110010
3(□):0111111
4(□):00111
5(□):10101
6(□):000011
7(□):1110010
8(□):101111
9(□):001010
A(□):1111111
B(□):1110111
C(□):1111000
D(
□):1110011
E(□):0110010
F(□):0101100
10(□):1111110
11(□):000101
12(□):100111
13(□):011011
14(□):011101
15(□):010100
16(□):1101111
17(□):1101101
18(□):100101
19(□):100110
1A(□):100010
1B(□):1111011
1C(□):1111100
1D(□):0010011
1E(-):0101110
1F():0111110
20( ):1110001
21(!):1110000
22("):1110101
23(#):1100100
24($):1011011
25(%):0000011
26(&):0000001
27('):0111101
28(():1100001
29()):0110011
2A(*):1100000
2B(+):0110100
2C(,):0010111
2D(-):0010000
2E(.):0000101
2F(/):0010010
30(0):0010110
31(1):0100010
32(2):1001001
33(3):0100000
34(4):0001001
35(5):11110010
36(6):01111000
37(7):11101101
38(8):11101000
39(9):11011101
3A(:):11110011
3B(;):11101001
3C(<):01111001
3D(=):11010100
3E(>):11010111
3F(?):11010101
40(@):11010011
41(A):11011001
42(B):11001111
43(C):11110100
44(D):10100110
45(E):01101010
46(F):10100000
47(G):11001011
48(H):11000110
49(I):11001100
4A(J):11011000
4B(K):11000111
4C(L):10100100
4D(M):01100000
4E(N):01011110
4F(O):10111001
50(P):01011111
51(Q):10111011
52(R):11001010
53(S):01000110
54(T):01001100
55(U):01011011
56(V):01001111
57(W):10111010
58(X):11010010
59(Y):01110001
5A(Z):01010110
5B([):01010111
5C(\):01110011
5D(]):11001101
5E(^):10110000
5F(_):11010001
60(`):10110100
61(a):11010000
62(b):01011010
63(c):01001101
64(d):10111000
65(e):10110011
66(f):01110000
67(g):01101011
68(h):11000101
69(i):11000100
6A(j):00110000
6B(k):11001110
6C(l):10100010
6D(m):10100011
6E(n):10110101
6F(o):10100101
70(p):01010101
71(q):00110110
72(r):01001110
73(s):01100010
74(t):10000110
75(u):10100111
76(v):01001010
77(w):00110011
78(x):00110001
79(y):10000111
7A(z):01100001
7B({):10001111
7C(|):10000101
7D(}):10000011
7E(~):10000010
7F(□):00010000
80(□):01001011
81(□):10000100
82(‚):00110111
83(ƒ):00110101
84(„):111101010
85(…):10010001
86(†):10000001
87(‡):00010001
88(ˆ):10001100
89(‰):01001000
8A(Š):01010100
8B(‹):00100011
8C(Œ):00110100
8D(□):00001001
8E(□):00000001
8F(□):10000000
90(□):10001101
91(‘):00100010
92(’):01000011
93(“):01000111
94(”):111101011
95(•):111110110
96(–):10001110
97(—):00000101
98(˜):110111000
99(™):110111001
9A(š):011000111
9B(›):101000010
9C(œ):100100000
9D(□):100100001
9E(□):1111101111
9F(Ÿ):1111101110
A0( ):1110110010
A1(¡):000000000
A2(¢):0111000111
A3(£):1101011010
A4(¤):000010000
A5(¥):1011001011
A6(¦):1110110001
A7(§):1011001000
A8(¨):0111001011
A9(©):1101011011
AA(ª):0111000110
AB(«):0100001001
AC(¬):0100100110
AD(-):1111101011
AE(®):0000000011
AF(¯):0000010010
B0(°):0000000010
B1(±):1010000111
B2(²):1011001001
B3(³):11101100110
B4(´):11101100111
B5(µ):1011001010
B6(¶):0100001000
B7(·):0100100111
B8(¸):11111010000
B9(¹):0000010011
BA(º):11111010100
BB(»):11111010101
BC(¼):10100001100
BD(½):10100001101
BE(¾):11101100000
BF(¿):01110010011
C0(À):01110010010
C1(Á):11111010011
C2(Â):11010110010
C3(Ã):00001000100
C4(Ä):01000010110
C5(Å):01100011000
C6(Æ):11010110011
C7(Ç):01000010111
C8(È):00000100000
C9(É):01001001001
CA(Ê):01001001010
CB(Ë):01110010000
CC(Ì):00001000111
CD(Í):00001000110
CE(Î):01100011011
CF(Ï):011100100010
D0(Ð):111110100100
D1(Ñ):110101100000
D2(Ò):00000100001
D3(Ó):010010010110
D4(Ô):110101100001
D5(Õ):011000110010
D6(Ö):011100101010
D7(×):011100101000
D8(Ø):010010010111
D9(Ù):1111101000100
DA(Ú):010000101010
DB(Û):010000101000
DC(Ü):1111101000110
DD(Ý):000010001010
DE(Þ):011100101011
DF(ß):011000110011
E0(à):011000110101
E1(á):000010001011
E2(â):1110110000100
E3(ã):1110110000101
E4(ä):110101100010
E5(å):010010010000
E6(æ):010010010001
E7(ç):010000101001
E8(è):1101011000110
E9(é):0111001000110
EA(ê):0111001000111
EB(ë):1110110000110
EC(ì):1101011000111
ED(í):0110001101000
EE(î):11111010001110
EF(ï):11111010001111
F0(ð):0100001010110
F1(ñ):11101100001110
F2(ò):01110010100100
F3(ó):01110010100101
F4(ô):11111010010100
F5(õ):11111010010101
F6(ö):11111010001010
F7(÷):11111010001011
F8(ø):0100001010111
F9(ù):11111010010110
FA(ú):01100011010010
FB(û):11101100001111
FC(ü):11111010010111
FD(ý):01100011010011
FE(þ):0111001010011
FF(ÿ):0000010001
```

pups.bmp -> pups.cod : 94% (90%)

Zu guter letzt wollen wir noch untersuchen, wie sich mehrfaches Packen auf die Dateigröße auswirkt. Verwenden wir dazu wieder unsere Textdatei und bennen wir jede gepackte Datei mit der Endung C0-C4:

```
 Datenträger in Laufwerk C: heißt WIN95 WD
 Seriennummer des Datenträgers: 3935-1DD6
 Verzeichnis von C:\TURBOC\USER

DATUM    HLP        52.593  19.10.98  17:36 DATUM.HLP
DATUM    C0         33.514  03.11.98  12:15 DATUM.C0
DATUM    C1         34.720  03.11.98  12:16 DATUM.C1
DATUM    C2         36.456  03.11.98  12:16 DATUM.C2
DATUM    C3         38.990  03.11.98  12:17 DATUM.C3
DATUM    C4         40.385  03.11.98  12:17 DATUM.C4
         6 Datei(en)              236.658 Bytes
         0 Verzeichnis(se)     935.723.008 Bytes frei
```

Wie man leicht erkennt, ist die optimale Packung nur das erste mal in DATUM.C0 gelungen. Alle weiteren Packversuche vergrößern die Datei nur noch. Es wäre auch schlimm, wenn es anders wäre. Sobald man nachweisen könnte, daß $\lim_{n\to\infty}$ Dateigröße(nach Packen $_n$) = 1 Bit wäre, gäbe es logischerweise auch nur noch zwei Programm auf der ganzen Welt, denn alle anderen ließen sich ja auf diese zurückführen. Da dem nicht so ist, kann man daraus schließen, daß sich bei permanenter Anwengung *eines* Algorithmus nach einer bestimmten Grenze oder Packrate die Datei wieder größer wird.

***Anmerkungen:***

Da die ASCII-Tabellen von DOS und Windows bei allen Zeichen > 127 nicht kompatibel sind, stehen natürlich falsche Zeichen in den Klammern. Aber trotzdem kann man durchaus erkennen, wie der Huffman Code zustande kommt.
Der verwendete Algorithmus ist nicht sehr schnell, da er ohne Puffer arbeitet. Es wäre eine stake Verbesserung, wenn nicht nur ein Byte gelesen würde, sondern immer ein ganzer Bereich. Damit würde man aber das didaktische Ziel verfälschen, da die Leseoperationen des Hauptaugenmerks bedürften, so daß ich diese Verbesserung einem Tüftler überlasse.
Selbstverständlich ist der Huffman Code kein optimales Verfahren, um Dateien zu packen. Eine auf Bits basierenden Lauflängencodierung in Kombination mit dem Huffman Code wäre die Lösung. Dazu müßte bei der Huffman Codierung der Begriff Zeichen auf Bitfolgen und nicht auf starre 8-Bit-Folgen angewendet werden, um die Lösung zur Optimierung. So müßte der Begriff Zeichen für den Huffman abstrahiert werden und man von einer Variablen Bitlänge ausgehen. Nach dem Motto „Paare willkommen“. Folgen zwei oder mehr Einsen auf einander, so ist das ein Zeichen, genauso wie zwei oder mehr Nullen. Es gilt zu testen, was daraus würde...