

```

/*****
/*
/* 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 kleinste Wahrscheinlichkeit! */
        while (n)

```

```

{
    if (n->p<kk1->p) kkl=n;
    else
        if (n->p<kl->p) kl=n;
    else
        if (kl==kk1) kl=n;
    n=n->next;
}
/* Fasse zusammen */
if (kl->p<=kk1->p) n=kl; else n=kk1; /* 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=kk1; 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<kk1->count; i++)
    kl->inhalt[kl->count+i]=kk1->inhalt[i];
kl->count+=kk1->count;
kl->p+=kk1->p;
/* kkl löschen! */
if (kk1->prev) (kk1->prev)->next=kk1->next;
if (kk1->next) (kk1->next)->prev=kk1->prev;
if (kk1==root) root=kk1->next;
free((struct liste *)kk1);
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>=0)
        {
            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 Funktionschen 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]);
}

```