```
//---------------------------------------------------------
-----------------

#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------
-----------------
USEFORM("UInduk.cpp", frminduk);
//---------------------------------------------------------
-----------------
WINAPI WinMain(HINSTANCE,
HINSTANCE, LPSTR, int)
{
	try
	{
		Application->Initialize();
		Application->Title = "PPM";
		Application-
>CreateForm(__classid(Tfrminduk),
&frminduk);
		Application->Run();
	}
	catch (Exception &exception)
	{
		Application-
>ShowException(&exception);
	}
	catch (...)
	{
		try
		{
			throw Exception("");
		}
		catch (Exception &exception)
		{
			Application-
>ShowException(&exception);
		}
	}
	return 0;
}
//---------------------------------------------------------
-----------------

//---------------------------------------------------------
-----------------

#ifndef UIndukH
#define UIndukH
//---------------------------------------------------------
-----------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "SUIForm.hpp"
#include "SUIMgr.hpp"
#include <ExtCtrls.hpp>
#include "SUIPageControl.hpp"
#include "SUIStatusBar.hpp"
#include "SUITabControl.hpp"
#include <ComCtrls.hpp>
#include "SUIGroupBox.hpp"
#include "SUIImagePanel.hpp"
#include "SUIEdit.hpp"
#include "SUIButton.hpp"
#include <NMFtp.hpp>
#include <Psock.hpp>
#include <Dialogs.hpp>
#include <Graphics.hpp>
#include "SUIListView.hpp"
#include "SUIGrid.hpp"
#include <Grids.hpp>
#include <NMEcho.hpp>


//---------------------------------------------------------
-----------------
class Tfrminduk : public TForm
{
__published:	// IDE-managed Components
	TsuiBuiltInFileTheme *sfiletheme;
	TsuiForm *sfrminduk;
	TsuiStatusBar *sbar;
	TsuiPageControl *spagecontrol;
	TsuiTabSheet *spclogin;
	TsuiGroupBox *suiGroupBox1;
	TLabel *Label1;
	TLabel *Label3;
	TLabel *Label4;
	TLabel *Label5;
	TLabel *Label7;
	TLabel *Label8;
	TsuiEdit *sehost;
	TsuiEdit *seuserid;
	TsuiEdit *sepassword;
	TsuiButton *sbtnconnect;
	TNMFTP *NMFTP1;
	TsuiGroupBox *suiGroupBox2;
	TsuiEdit *sefile1;
	TsuiTabSheet *spcdownload;
	TsuiButton *sbtnfile1;
	TLabel *Label9;
	TLabel *Label14;
	TOpenDialog *odfile;
	TsuiButton *sbtncompress;
	TsuiImagePanel *suiImagePanel1;
	TsuiImagePanel *suiImagePanel2;
	TLabel *Label10;
	TLabel *Label11;
	TsuiEdit *sefile2;
	TsuiButton *suiButton1;
```

```
TSaveDialog *sdfile;
TsuiImagePanel *suiImagePanel3;
TsuiEdit *sefiled1;
TsuiButton *sbtnfiled1;
TsuiButton *sbtndecompress;
TLabel *Label12;
TLabel *Label15;
TsuiTabSheet *suiTabSheet1;
TsuiTabSheet *suiTabSheet2;
TOpenDialog *odfiled;
TSaveDialog *sdfiled;
TsuiImagePanel *suiImagePanel4;
TsuiImagePanel *suiImagePanel5;
TLabel *Label17;
TLabel *Label18;
TsuiEdit *sefileupload;
TsuiButton *sbtnbrosfileupload;
TsuiButton *sbtnfileupload;
TLabel *Label19;
TLabel *Label20;
TLabel *Label21;
TLabel *Label22;
TsuiButton *sbtnfiledownload;
TsuiEdit *sefiledownload;
TLabel *Label23;
TLabel *Label24;
TsuiButton *sbtnbrosfiledownload;
TOpenDialog *odfileupload;
TOpenDialog *odfiledownload;
TsuiStringGrid *sstringgrid2;
TsuiButton *sbtnlist;
TsuiStringGrid *sstringgrid1;
TsuiButton *sbtnview;
TLabel *Label25;
TLabel *Label26;
TLabel *Label27;
TLabel *Label28;
TLabel *Label29;
TLabel *Label30;
TLabel *Label31;
TLabel *Label32;
TSaveDialog *sddownload;
TLabel *Label2;
TLabel *Label6;
TLabel *Label13;
TLabel *Label16;
TLabel *Label33;
TLabel *Label34;
TLabel *Label35;
TLabel *Label36;
TLabel *Label37;
TLabel *Label38;
TLabel *Label39;
TLabel *Label40;
TLabel *Label41;
TLabel *Label42;
TLabel *Label43;
TLabel *Label44;
TLabel *Label45;
TLabel *Label46;
TLabel *Label47;
TLabel *Label48;
TLabel *Label49;
TLabel *Label50;
TLabel *Label51;
TLabel *Label52;
TLabel *Label53;
TLabel *Label54;
TLabel *Label55;
TLabel *Label56;
    void __fastcall sbtnconnectClick(TObject
*Sender);
    void __fastcall NMFTP1Connect(TObject
*Sender);
    void __fastcall
NMFTP1Disconnect(TObject *Sender);
    void __fastcall sbtnfile1Click(TObject
*Sender);
    void __fastcall suiButton1Click(TObject
*Sender);
    void __fastcall sbtnfiled1Click(TObject
*Sender);
    void __fastcall sbtncompressClick(TObject
*Sender);
    void __fastcall
sbtndecompressClick(TObject *Sender);
    void __fastcall
sbtnbrosfileuploadClick(TObject *Sender);
    void __fastcall
sbtnbrosfiledownloadClick(TObject *Sender);
    void __fastcall sbtnlistClick(TObject
*Sender);
    void __fastcall
NMFTP1Success(TCmdType Trans_Type);
    void __fastcall sbtnviewClick(TObject
*Sender);
    void __fastcall sbtnfileuploadClick(TObject
*Sender);
    void __fastcall FormCreate(TObject
*Sender);
    void __fastcall
sstringgrid1SelectCell(TObject *Sender, int
ACol,
      int ARow, bool &CanSelect);
    void __fastcall
sbtnfiledownloadClick(TObject *Sender);
    void __fastcall
NMFTP1AuthenticationNeeded(bool
&Handled);
    void __fastcall
NMFTP1ConnectionRequired(bool &Handled);
private:  // User declarations
```

```
        void __fastcall clearText();

public:                 // User declarations
        __fastcall Tfrminduk(TComponent*
Owner);
};
//---------------------------------------------------------
-----------------
extern PACKAGE Tfrminduk *frminduk;
//---------------------------------------------------------
-----------------
#endif


//---------------------------------------------------------
-----------------

#include <vcl.h>
#pragma hdrstop

#include "UInduk.h"
#include "PPMd.h"
#include "Coder.hpp"
#include "SubAlloc.hpp"

#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
//---------------------------------------------------------
-----------------
#pragma package(smart_init)
#pragma link "SUIForm"
#pragma link "SUIMgr"
#pragma link "SUIPageControl"
#pragma link "SUIStatusBar"
#pragma link "SUITabControl"
#pragma link "SUIGroupBox"
#pragma link "SUIImagePanel"
#pragma link "SUIEdit"
#pragma link "SUIButton"
#pragma link "SUIListView"
#pragma link "SUIGrid"
#pragma resource "*.dfm"
Tfrminduk *frminduk;
AnsiString SFile, SUkuranFile, SUkuranFiled,
          SKecepatan;

enum { UP_FREQ=5, INT_BITS=7,
PERIOD_BITS=7,
TOT_BITS=INT_BITS+PERIOD_BITS,
   INTERVAL=1 << INT_BITS,
BIN_SCALE=1 << TOT_BITS, ROUND=16,
MAX_FREQ=124,
   O_BOUND=9 };

static _THREAD1 struct SEE2_CONTEXT { //
SEE-contexts for PPM-contexts with masked
symbols
   WORD Summ;
   BYTE Shift, Count;
   void init(UINT InitVal) { Summ=InitVal <<
(Shift=PERIOD_BITS-4); Count=7; }
   UINT getMean() {
      UINT RetVal=(Summ >> Shift);        Summ
-= RetVal;
      return RetVal+!RetVal;
   }
   void update() { if (--Count == 0)
setShift_rare(); }
   void setShift_rare();
} _THREAD SEE2Cont[23][32],
DummySEE2Cont;
#pragma pack(1)
static _THREAD1 struct PPM_CONTEXT {
struct STATE {
   _BYTE Symbol, Freq;
   _DWORD iSuccessor;
   _PAD_TO_64(Dummy)
   PPM_CONTEXT* getSucc() const { return
(PPM_CONTEXT*)Indx2Ptr(iSuccessor); }
};
   _BYTE NumStats, Flags;                // Notes:
   _WORD SummFreq;                       // 1.
NumStats & NumMasked contain
   _DWORD iStats;                      //  number of
symbols minus 1
   _PAD_TO_64(Dummy1)                    // 2.
contexts example:
   _DWORD iSuffix;                      //
MaxOrder:
   _PAD_TO_64(Dummy2)                    //
ABCD    context
   inline void encodeBinSymbol(int symbol);//
BCD    suffix
   inline void   encodeSymbol1(int symbol);//
BCDE   successor
   inline void   encodeSymbol2(int symbol);//
other orders:
   inline void          decodeBinSymbol();//
BCD    context
   inline void           decodeSymbol1();//     CD
suffix
   inline void           decodeSymbol2();//
BCDE   successor
   inline void          update1(STATE* p);
   inline void          update2(STATE* p);
   inline SEE2_CONTEXT*     makeEscFreq2();
   void                    rescale();
   _DWORD                  cutOff(int Order);
```

```
   STATE&   oneState() const { return
(STATE&) SummFreq; }
   STATE*   getStats() const { return
(STATE*)Indx2Ptr(iStats); }
   PPM_CONTEXT* suff() const { return
(PPM_CONTEXT*)Indx2Ptr(iSuffix); }
} * _THREAD MaxContext;
#pragma pack()

static BYTE NS2BSIndx[256], QTable[260];
// constants
static _THREAD1 PPM_CONTEXT::STATE*
_THREAD FoundState; // found next state
transition
static _THREAD1 int _THREAD BSumm,
_THREAD OrderFall, _THREAD RunLength;
static _THREAD1 int _THREAD InitRL,
_THREAD MaxOrder;
static _THREAD1 BYTE _THREAD
CharMask[256], _THREAD NumMasked,
_THREAD PrevSuccess;
static _THREAD1 BYTE _THREAD EscCount,
_THREAD PrintCount;
static _THREAD1 WORD _THREAD
BinSumm[25][64];          // binary SEE-contexts
static _THREAD1 BOOL _THREAD CutOff;

inline void SWAP(PPM_CONTEXT::STATE&
s1,PPM_CONTEXT::STATE& s2) {
   _WORD t1=(_WORD&)s1;
_DWORD t2=s1.iSuccessor;
   (_WORD&)s1 = (_WORD&)s2;
s1.iSuccessor=s2.iSuccessor;
   (_WORD&)s2 = t1;
s2.iSuccessor=t2;
}
inline void
StateCpy(PPM_CONTEXT::STATE& s1,const
PPM_CONTEXT::STATE& s2) {
   (_WORD&)s1 = (_WORD&)s2;
s1.iSuccessor=s2.iSuccessor;
}
void SEE2_CONTEXT::setShift_rare()
{
   UINT i=Summ >> Shift;
   i=PERIOD_BITS-(i > 40)-(i > 280)-(i >
1020);
      if (i < Shift) { Summ >>= 1;     Shift--; }
   else if (i > Shift) { Summ <<= 1;     Shift++; }
   Count=6 << Shift;
}
static struct PPMD_STARTUP { inline
PPMD_STARTUP(); } const PPMd_StartUp;
inline PPMD_STARTUP::PPMD_STARTUP()
// constants initialization
{
   UINT i, k, m, Step;
   for (i=0,k=1;i < N1     ;i++,k += 1)
Indx2Units[i]=k;
   for (k++;i < N1+N2      ;i++,k += 2)
Indx2Units[i]=k;
   for (k++;i < N1+N2+N3   ;i++,k += 3)
Indx2Units[i]=k;
   for (k++;i < N1+N2+N3+N4;i++,k += 4)
Indx2Units[i]=k;
   for (k=i=0;k < 128;k++) {
      i += (Indx2Units[i] < k+1);
Units2Indx[k]=i;
   }
   NS2BSIndx[0]=2*0;
NS2BSIndx[1]=NS2BSIndx[2]=2*1;
   memset(NS2BSIndx+3,2*2,26);
memset(NS2BSIndx+29,2*3,256-29);
   for (i=0;i < UP_FREQ;i++)
QTable[i]=i;
   for (m=i=UP_FREQ, k=Step=1;i < 260;i++) {
      QTable[i]=m;
      if ( !--k ) { k = ++Step;          m++; }
   }
}
static void _FASTCALL StartModelRare(int
MaxOrder,BOOL CutOff)
{
   int i, k, s;
   BYTE i2f[25];
   OrderFall=::MaxOrder=MaxOrder;
::CutOff=CutOff;
   memset(CharMask,0,sizeof(CharMask));
EscCount=PrintCount=1;
   if (MaxOrder < 2) {                  // we are in
solid mode
      for (PPM_CONTEXT* pc=MaxContext;pc-
>iSuffix != 0;pc=pc->suff())
            OrderFall--;
      return;
   }
   InitSubAllocator();
   RunLength=InitRL=-((MaxOrder <
13)?MaxOrder:13);
   MaxContext =
(PPM_CONTEXT*)AllocContext();
   MaxContext->SummFreq=(MaxContext-
>NumStats=255)+2;
   MaxContext->iStats =
Ptr2Indx(AllocUnits(256/2));
   for (PrevSuccess=i=MaxContext-
>iSuffix=MaxContext->Flags=0;i < 256;i++) {
      MaxContext->getStats()[i].Symbol=i;
MaxContext->getStats()[i].Freq=1;
      MaxContext->getStats()[i].iSuccessor=0;
   }
   for (k=i=0;i < 25;i2f[i++]=k+1)
```

```
        while (QTable[k] == i)          k++;
static const signed char EscCoef[12]={16,-
10,1,51,14,89,23,35,64,26,-42,43};
   for (k=0;k < 64;k++) {
      for (s=i=0;i < 6;i++)             s +=
EscCoef[2*i+((k >> i) & 1)];
      s=128*CLAMP(s,32,256-32);
      for (i=0;i < 25;i++)
BinSumm[i][k]=BIN_SCALE-s/i2f[i];
   }
   for (i=0;i < 23;i++)
        for (k=0;k < 32;k++)
SEE2Cont[i][k].init(8*i+5);
}
inline void
AuxCutOff(PPM_CONTEXT::STATE* p,int
Order) {
   if (Order < MaxOrder) {
      PrefetchData(p->getSucc());
      p->iSuccessor=p->getSucc()-
>cutOff(Order+1);
   } else                           p->iSuccessor=0;
}
_DWORD PPM_CONTEXT::cutOff(int Order)
{
   int i, tmp, EscFreq, Scale;
   STATE* p, * p0;
   if ( !NumStats ) {
      if ((_BYTE*)(p=&oneState())->getSucc()
>= UnitsStart) {
         AuxCutOff(p,Order);
         if (p->iSuccessor || Order < O_BOUND)
              goto AT_RETURN;
      }
REMOVE: FreeUnit(this);                  return 0;
   }
   iStats =
Ptr2Indx(p0=(STATE*)MoveUnitsUp(getStats()
,tmp=(NumStats+2) >> 1));
   for (p=p0+(i=NumStats);p >= p0;p--)
        if ((_BYTE*)p->getSucc() < UnitsStart)
{
            p->iSuccessor=0;
SWAP(*p,p0[i--]);
        } else
AuxCutOff(p,Order);
   if (i != NumStats && Order) {
      NumStats=i;                    p=p0;
      if (i < 0) { FreeUnits(p,tmp);     goto
REMOVE; }
      else if (i == 0) {
         Flags=(Flags & 0x10)+0x08*(p->Symbol
>= 0x40);
         p->Freq=1+(2*(p->Freq-1))/(SummFreq-
p->Freq);
         StateCpy(oneState(),*p);
FreeUnits(p,tmp);
      } else {
         iStats =
Ptr2Indx(p=(STATE*)ShrinkUnits(p0,tmp,(i+2)
>> 1));
         Scale=(SummFreq > 16*i);
EscFreq=SummFreq-p->Freq;
         Flags=(Flags &
(0x10+0x04*Scale))+0x08*(p->Symbol >=
0x40);
         SummFreq=p->Freq=(p->Freq+Scale)
>> Scale;
         do {
            EscFreq -= (++p)->Freq;
            SummFreq += (p->Freq=(p-
>Freq+Scale) >> Scale);
            Flags |= 0x08*(p->Symbol >= 0x40);
         } while ( --i );
         SummFreq +=
(EscFreq=(EscFreq+Scale) >> Scale);
      }
   }
AT_RETURN:
   if ((_BYTE*)this == UnitsStart) {
      UnitsCpy(AuxUnit,this,1);          return
Ptr2Indx(AuxUnit);
   } else if((_BYTE*)suff() == UnitsStart)
iSuffix=Ptr2Indx(AuxUnit);
   return Ptr2Indx(this);
}
static void
RestoreModelRare(PPM_CONTEXT* pc)
{
   pText=HeapStart;
   if (!CutOff || GetUsedMemory() <
(SubAllocatorSize >> 2)) {
      StartModelRare(MaxOrder,CutOff);
PrintCount=0xFF;
      EscCount=0;                    return;
   }
   for (PPM_CONTEXT::STATE*
p;MaxContext->NumStats == 1 &&
MaxContext != pc &&
         (_BYTE*)(p=MaxContext-
>getStats())[1].getSucc() < UnitsStart;
         MaxContext=MaxContext->suff()) {
      MaxContext->Flags=(MaxContext->Flags
& 0x10)+0x08*(p->Symbol >= 0x40);
      p->Freq=(p->Freq+1) >> 1;
StateCpy(MaxContext->oneState(),*p);
      MaxContext->NumStats=0;
FreeUnits(p,1);
   }
   while ( MaxContext->iSuffix )
MaxContext=MaxContext->suff();
```

```
   AuxUnit=UnitsStart;
ExpandTextArea();
   do {
      PrepareTextArea();              MaxContext-
>cutOff(0);
      ExpandTextArea();
   } while (GetUsedMemory() >
3*(SubAllocatorSize >> 2));
   GlueCount=GlueCount1=0;
OrderFall=MaxOrder;
}
static _DWORD _FASTCALL
CreateSuccessors(BOOL
Skip,PPM_CONTEXT::STATE*
p,PPM_CONTEXT* pc);
inline _DWORD
ReduceOrder(PPM_CONTEXT::STATE*
p,PPM_CONTEXT* pc)
{
   PPM_CONTEXT::STATE* p1;
   PPM_CONTEXT* pc1=pc;
   _DWORD iUpBranch = FoundState-
>iSuccessor = Ptr2Indx(pText);
   BYTE tmp, sym=FoundState->Symbol;
OrderFall++;
   if ( p ) { pc=pc->suff();          goto
LOOP_ENTRY; }
   for ( ; ; ) {
      if ( !pc->iSuffix )             return
Ptr2Indx(pc);
      pc=pc->suff();
      if ( pc->NumStats ) {
         if ((p=pc->getStats())->Symbol != sym)
               do { tmp=p[1].Symbol;   p++; }
while (tmp != sym);
         tmp=2*(p->Freq < MAX_FREQ-3);
         p->Freq += tmp;              pc-
>SummFreq += tmp;
      } else { p=&(pc->oneState());      p->Freq
+= (p->Freq < 11); }
LOOP_ENTRY:
      if ( p->iSuccessor )            break;
      p->iSuccessor=iUpBranch;
OrderFall++;
   }
   if (p->iSuccessor <= iUpBranch) {
      p1=FoundState;                  FoundState=p;
      p-
>iSuccessor=CreateSuccessors(FALSE,NULL,p
c);
      FoundState=p1;
   }
   if (OrderFall == 1 && pc1 == MaxContext) {
      FoundState->iSuccessor=p->iSuccessor;
      pText--;
   }
   return p->iSuccessor;
}
void PPM_CONTEXT::rescale()
{
   UINT f0, sf, EscFreq, a=(OrderFall != 0),
i=NumStats;
   STATE tmp, * p1, * p;              Flags &=
0x14;
   for (p=FoundState;p != getStats();p--)
SWAP(p[0],p[-1]);
   f0=p->Freq;                        sf=SummFreq;
   EscFreq=SummFreq-p->Freq;
SummFreq=p->Freq=(p->Freq+a) >> 1;
   do {
      EscFreq -= (++p)->Freq;
      SummFreq += (p->Freq=(p->Freq+a) >>
1);
      if ( p->Freq )                  Flags |=
0x08*(p->Symbol >= 0x40);
      if (p[0].Freq > p[-1].Freq) {
         StateCpy(tmp,*(p1=p));
         do { StateCpy(p1[0],p1[-1]); } while
(tmp.Freq > (--p1)[-1].Freq);
         StateCpy(*p1,tmp);
      }
   } while ( --i );
   if (p->Freq == 0) {
      do { i++; } while ((--p)->Freq == 0);
      EscFreq += i;
a=(NumStats+2) >> 1;
      if ((NumStats -= i) == 0) {
         StateCpy(tmp,*getStats());     Flags &=
0x18;
         tmp.Freq=(2*tmp.Freq+EscFreq-
1)/EscFreq;
         if (tmp.Freq > MAX_FREQ/3)
tmp.Freq=MAX_FREQ/3;
         FreeUnits(getStats(),a);
StateCpy(oneState(),tmp);
         FoundState=&oneState();       return;
      }
      iStats =
Ptr2Indx(ShrinkUnits(getStats(),a,(NumStats+2)
>> 1));
   }
   SummFreq += (EscFreq+1) >> 1;
   if (OrderFall || (Flags & 0x04) == 0) {
      a=(sf -= EscFreq)-f0;
      a=CLAMP(UINT((f0*SummFreq-
sf*getStats()->Freq+a-
1)/a),2U,MAX_FREQ/2U-18U);
   } else                             a=2;
   (FoundState=getStats())->Freq += a;
SummFreq += a;
   Flags |= 0x04;
}
```

```
static _DWORD _FASTCALL
CreateSuccessors(BOOL
Skip,PPM_CONTEXT::STATE*
p,PPM_CONTEXT* pc)
{
   PPM_CONTEXT ct;
   _DWORD iUpBranch = FoundState-
>iSuccessor;
   PPM_CONTEXT::STATE* ps[MAX_O], **
pps=ps;
   UINT cf, s0;
   BYTE tmp, sym=FoundState->Symbol;
   if ( !Skip ) {
      *pps++ = FoundState;
      if ( !pc->iSuffix )              goto
NO_LOOP;
   }
   if ( p ) { pc=pc->suff();           goto
LOOP_ENTRY; }
   do {
      pc=pc->suff();
      if ( pc->NumStats ) {
         if ((p=pc->getStats())->Symbol != sym)
               do { tmp=p[1].Symbol;   p++; }
while (tmp != sym);
         tmp=(p->Freq < MAX_FREQ);
         p->Freq += tmp;              pc-
>SummFreq += tmp;
      } else {
         p=&(pc->oneState());
         p->Freq += (!pc->suff()->NumStats &
(p->Freq < 11));
      }
LOOP_ENTRY:
      if (p->iSuccessor != iUpBranch) {
         pc=p->getSucc();             break;
      }
      *pps++ = p;
   } while ( pc->iSuffix );
NO_LOOP:
   if (pps == ps)                     return
Ptr2Indx(pc);
   ct.NumStats=0;
ct.Flags=0x10*(sym >= 0x40);

ct.oneState().Symbol=sym=*(_BYTE*)Indx2Ptr
(iUpBranch);

ct.oneState().iSuccessor=Ptr2Indx((_BYTE*)Ind
x2Ptr(iUpBranch)+1);
   ct.Flags |= 0x08*(sym >= 0x40);
   if ( pc->NumStats ) {
      if ((p=pc->getStats())->Symbol != sym)
            do { tmp=p[1].Symbol;      p++; }
while (tmp != sym);
      s0=pc->SummFreq-pc->NumStats-(cf=p-
>Freq-1);
      cf=1+((2*cf <= s0)?(12*cf >
s0):((cf+2*s0)/s0));
      ct.oneState().Freq=(cf < 7)?(cf):(7);
   } else
         ct.oneState().Freq=pc->oneState().Freq;
   do {
      PPM_CONTEXT* pc1 =
(PPM_CONTEXT*)AllocContext();
      if ( !pc1 )                    return 0;
      ((DWORD*)pc1)[0]=((DWORD*)&ct)[0];
((DWORD*)pc1)[1]=((DWORD*)&ct)[1];
#if defined(_32_EXOTIC) ||
defined(_64_EXOTIC)
      ((DWORD*)pc1)[2]=((DWORD*)&ct)[2];
((DWORD*)pc1)[3]=((DWORD*)&ct)[3];
#endif /* defined(_32_EXOTIC) ||
defined(_64_EXOTIC) */
      pc1->iSuffix=Ptr2Indx(pc);
      (*--pps)->iSuccessor=Ptr2Indx(pc=pc1);
   } while (pps != ps);
   return Ptr2Indx(pc);
}
// Tabulated escapes for exponential symbol
distribution
static const BYTE
ExpEscape[16]={51,43,18,12,11,9,8,7,6,5,4,3,3,
2,2,2};
static void _FASTCALL
UpdateModel(PPM_CONTEXT* MinContext)
{
   BYTE Flag, sym, FSymbol=FoundState-
>Symbol;
   UINT ns1, ns, cf, sf, s0, FFreq=FoundState-
>Freq;
   _DWORD iSuccessor,
iFSuccessor=FoundState->iSuccessor;
   PPM_CONTEXT* pc;
   PPM_CONTEXT::STATE* p=NULL;
   if ( MinContext->iSuffix ) {
      pc=MinContext->suff();
      if ( pc->NumStats ) {
         if ((p=pc->getStats())->Symbol !=
FSymbol) {
            do { sym=p[1].Symbol;      p++; }
while (sym != FSymbol);
            if (p[0].Freq >= p[-1].Freq) {
               SWAP(p[0],p[-1]);       p--;
            }
         }
         if (p->Freq < MAX_FREQ) {
            cf=1+(FFreq < 4*8);
            p->Freq += cf;             pc->SummFreq
+= cf;
         }
```

```
      } else { p=&(pc->oneState());       p->Freq
+= (p->Freq < 11); }
   }
   pc=MaxContext;
   if (!OrderFall && iFSuccessor) {
      FoundState-
>iSuccessor=CreateSuccessors(TRUE,p,MinCon
text);
      if ( !FoundState->iSuccessor )      goto
RESTART_MODEL;
      MaxContext=FoundState->getSucc();
return;
   }
   *pText++ = FSymbol;                  iSuccessor
= Ptr2Indx(pText);
   if (pText >= UnitsStart)             goto
RESTART_MODEL;
   if ( iFSuccessor ) {
      if ((_BYTE*)Indx2Ptr(iFSuccessor) <
UnitsStart)

iFSuccessor=CreateSuccessors(FALSE,p,MinCo
ntext);
      else
PrefetchData(Indx2Ptr(iFSuccessor));
   } else

iFSuccessor=ReduceOrder(p,MinContext);
   if ( !iFSuccessor )                  goto
RESTART_MODEL;
   if ( !--OrderFall ) {
      iSuccessor=iFSuccessor;           pText -=
(MaxContext != MinContext);
   }
   s0=MinContext->SummFreq-FFreq;
ns=MinContext->NumStats;
   Flag=0x08*(FSymbol >= 0x40);
   for ( ;pc != MinContext;pc=pc->suff()) {
      if ((ns1=pc->NumStats) != 0) {
         if ((ns1 & 1) != 0) {

p=(PPM_CONTEXT::STATE*)ExpandUnits(pc
->getStats(),(ns1+1) >> 1);
            if ( !p )                 goto
RESTART_MODEL;
            pc->iStats=Ptr2Indx(p);
         }
         pc->SummFreq += QTable[ns+4] >> 3;
      } else {

p=(PPM_CONTEXT::STATE*)AllocUnits(1);
         if ( !p )                    goto
RESTART_MODEL;
         StateCpy(*p,pc->oneState());    pc-
>iStats=Ptr2Indx(p);
         p->Freq=(p->Freq <=
MAX_FREQ/3)?(2*p->Freq-1):(MAX_FREQ-
15);
         pc->SummFreq=p->Freq+(ns >
1)+ExpEscape[QTable[BSumm >> 8]];
      }
      cf=2*FFreq*(pc->SummFreq+4);
sf=s0+pc->SummFreq;
      if (cf <= 6*sf) {
         cf=1+(cf > sf)+(cf > 3*sf);     pc-
>SummFreq += 4;
      } else
            pc->SummFreq += (cf=4+(cf >
8*sf)+(cf > 10*sf)+(cf > 13*sf));
      p=pc->getStats()+(++pc->NumStats);  p-
>iSuccessor=iSuccessor;
      p->Symbol = FSymbol;              p->Freq =
cf;
      pc->Flags |= Flag;
   }
   MaxContext =
(PPM_CONTEXT*)Indx2Ptr(iFSuccessor);
   return;
RESTART_MODEL:
   RestoreModelRare(pc);
}
#define GET_MEAN(SUMM,SHIFT)
((SUMM+ROUND) >> SHIFT)
inline void
PPM_CONTEXT::encodeBinSymbol(int
symbol)
{
   STATE& rs=oneState();
   WORD& bs=BinSumm[QTable[rs.Freq-
1]][NS2BSIndx[suff()-
>NumStats]+PrevSuccess+
         Flags+((RunLength >> 26) & 0x20)];
   UINT
tmp=rcBinStart(BSumm=bs,TOT_BITS); bs -=
GET_MEAN(BSumm,PERIOD_BITS);
   if (rs.Symbol == symbol) {
      bs += INTERVAL;
rcBinCorrect0(tmp);
      FoundState=&rs;                   rs.Freq +=
(rs.Freq < 196);
      RunLength++;
PrevSuccess=1;
   } else {
      rcBinCorrect1(tmp,BIN_SCALE-BSumm);
CharMask[rs.Symbol]=EscCount;
      NumMasked=PrevSuccess=0;
FoundState=NULL;
   }
}
inline void
PPM_CONTEXT::decodeBinSymbol()
```

```
{
   STATE& rs=oneState();
   WORD& bs=BinSumm[QTable[rs.Freq-
1]][NS2BSIndx[suff()-
>NumStats]+PrevSuccess+
         Flags+((RunLength >> 26) & 0x20)];
   UINT
tmp=rcBinStart(BSumm=bs,TOT_BITS); bs -=
GET_MEAN(BSumm,PERIOD_BITS);
   if ( !rcBinDecode(tmp) ) {
      bs += INTERVAL;
rcBinCorrect0(tmp);
      FoundState=&rs;                rs.Freq +=
(rs.Freq < 196);
      RunLength++;
PrevSuccess=1;
   } else {
      rcBinCorrect1(tmp,BIN_SCALE-BSumm);
CharMask[rs.Symbol]=EscCount;
      NumMasked=PrevSuccess=0;
FoundState=NULL;
   }
}
inline void PPM_CONTEXT::update1(STATE*
p)
{
   (FoundState=p)->Freq += 4;
SummFreq += 4;
   if (p[0].Freq > p[-1].Freq) {
      SWAP(p[0],p[-1]);
FoundState=--p;
      if (p->Freq > MAX_FREQ)
rescale();
   }
}
inline void
PPM_CONTEXT::encodeSymbol1(int symbol)
{
   STATE* p=getStats();
   UINT i=p->Symbol, LoCnt=p->Freq;
Range.scale=SummFreq;
   if (i == symbol) {
      PrevSuccess=(2*(Range.high=LoCnt) >
Range.scale);
      (FoundState=p)->Freq=(LoCnt += 4);
SummFreq += 4;
      if (LoCnt > MAX_FREQ)
rescale();
      Range.low=0;                   return;
   }
   PrefetchData(p+(i=NumStats));
PrevSuccess=0;
   while ((++p)->Symbol != symbol) {
      LoCnt += p->Freq;
      if (--i == 0) {
         if ( iSuffix )
PrefetchData(suff());
         Range.low=LoCnt;
CharMask[p->Symbol]=EscCount;
         i=NumMasked=NumStats;
FoundState=NULL;
         do { CharMask[(--p)-
>Symbol]=EscCount; } while ( --i );
         Range.high=Range.scale;         return;
      }
   }
   Range.high=(Range.low=LoCnt)+p->Freq;
update1(p);
}
inline void PPM_CONTEXT::decodeSymbol1()
{
   STATE* p=getStats();
   UINT i, count, HiCnt=p->Freq;
Range.scale=SummFreq;
   if ((count=rcGetCurrentCount()) < HiCnt) {
      PrevSuccess=(2*(Range.high=HiCnt) >
Range.scale);
      (FoundState=p)->Freq=(HiCnt += 4);
SummFreq += 4;
      if (HiCnt > MAX_FREQ)
rescale();
      Range.low=0;                   return;
   }
   PrefetchData(p+(i=NumStats));
PrevSuccess=0;
   while ((HiCnt += (++p)->Freq) <= count)
      if (--i == 0) {
         if ( iSuffix )
PrefetchData(suff());
         Range.low=HiCnt;
CharMask[p->Symbol]=EscCount;
         i=NumMasked=NumStats;
FoundState=NULL;
         do { CharMask[(--p)-
>Symbol]=EscCount; } while ( --i );
         Range.high=Range.scale;         return;
      }
   Range.low=(Range.high=HiCnt)-p->Freq;
update1(p);
}
inline void PPM_CONTEXT::update2(STATE*
p)
{
   (FoundState=p)->Freq += 4;
SummFreq += 4;
   if (p->Freq > MAX_FREQ)
rescale();
   EscCount++;
RunLength=InitRL;
}
```

```
inline SEE2_CONTEXT*
PPM_CONTEXT::makeEscFreq2()
{
   SEE2_CONTEXT* psee2c;
PrefetchData(getStats());
   if (NumStats != 0xFF) {
      psee2c=SEE2Cont[QTable[NumStats+3]-
4]+(SummFreq > 10*(NumStats+1))+
            2*(2*NumStats < suff()-
>NumStats+NumMasked)+Flags;
      Range.scale=psee2c->getMean();
   } else { psee2c=&DummySEE2Cont;
Range.scale=1; }
   PrefetchData(getStats()+NumStats);     return
psee2c;
}
inline void
PPM_CONTEXT::encodeSymbol2(int symbol)
{
   SEE2_CONTEXT* psee2c=makeEscFreq2();
   UINT Sym, LoCnt=0, i=NumStats-
NumMasked;
   STATE* p1, * p=getStats()-1;
   do {
      do { Sym=p[1].Symbol;   p++; } while
(CharMask[Sym] == EscCount);
      CharMask[Sym]=EscCount;
      if (Sym == symbol)                goto
SYMBOL_FOUND;
      LoCnt += p->Freq;
   } while ( --i );
   Range.high=(Range.scale +=
(Range.low=LoCnt));
   psee2c->Summ += Range.scale;
NumMasked = NumStats;
   return;
SYMBOL_FOUND:
   Range.low=LoCnt;
Range.high=(LoCnt += p->Freq);
   for (p1=p; --i ; ) {
      do { Sym=p1[1].Symbol;  p1++; } while
(CharMask[Sym] == EscCount);
      LoCnt += p1->Freq;
   }
   Range.scale += LoCnt;
   psee2c->update();                  update2(p);
}
inline void PPM_CONTEXT::decodeSymbol2()
{
   SEE2_CONTEXT* psee2c=makeEscFreq2();
   UINT Sym, count, HiCnt=0, i=NumStats-
NumMasked;
   STATE* ps[256], ** pps=ps, * p=getStats()-1;
   do {
      do { Sym=p[1].Symbol;   p++; } while
(CharMask[Sym] == EscCount);
      HiCnt += p->Freq;              *pps++ = p;
   } while ( --i );
   Range.scale += HiCnt;
count=rcGetCurrentCount();
   p=*(pps=ps);
   if (count < HiCnt) {
      HiCnt=0;
      while ((HiCnt += p->Freq) <= count)
p=*++pps;
      Range.low = (Range.high=HiCnt)-p->Freq;
      psee2c->update();              update2(p);
   } else {
      Range.low=HiCnt;
Range.high=Range.scale;
      i=NumStats-NumMasked;
NumMasked = NumStats;
      do { CharMask[(*pps)-
>Symbol]=EscCount; pps++; } while ( --i );
      psee2c->Summ += Range.scale;
   }
}
inline void ClearMask(_PPMD_FILE*
EncodedFile,_PPMD_FILE* DecodedFile)
{
   EscCount=1;
memset(CharMask,0,sizeof(CharMask));
   if (++PrintCount == 0)
PrintInfo(DecodedFile,EncodedFile);
}
void _STDCALL EncodeFile(_PPMD_FILE*
EncodedFile,_PPMD_FILE* DecodedFile,
                  int MaxOrder,BOOL CutOff)
{
   rcInitEncoder();
StartModelRare(MaxOrder,CutOff);
   for (PPM_CONTEXT*
MinContext=MaxContext; ; ) {
      int c = _PPMD_E_GETC(DecodedFile);
      if ( MinContext->NumStats ) {
         MinContext->encodeSymbol1(c);
rcEncodeSymbol();
      } else                    MinContext-
>encodeBinSymbol(c);
      while ( !FoundState ) {
         RC_ENC_NORMALIZE(EncodedFile);
         do {
            if ( !MinContext->iSuffix ) goto
STOP_ENCODING;
            OrderFall++;
MinContext=MinContext->suff();
         } while (MinContext->NumStats ==
NumMasked);
         MinContext->encodeSymbol2(c);
rcEncodeSymbol();
      }
```

```
      if (!OrderFall && (_BYTE*)FoundState-
>getSucc() >= UnitsStart)

PrefetchData(MaxContext=FoundState-
>getSucc());
      else {
         UpdateModel(MinContext);
         if (EscCount == 0)
ClearMask(EncodedFile,DecodedFile);
      }
      RC_ENC_NORMALIZE(EncodedFile);
MinContext=MaxContext;
   }
STOP_ENCODING:
   rcFlushEncoder(EncodedFile);
PrintInfo(DecodedFile,EncodedFile);
}
void _STDCALL DecodeFile(_PPMD_FILE*
DecodedFile,_PPMD_FILE* EncodedFile,
                  int MaxOrder,BOOL CutOff)
{
   rcInitDecoder(EncodedFile);
StartModelRare(MaxOrder,CutOff);
   for (PPM_CONTEXT*
MinContext=MaxContext; ; ) {
      if ( MinContext->NumStats ) {
         MinContext->decodeSymbol1();
rcRemoveSubrange();
      } else                        MinContext-
>decodeBinSymbol();
      while ( !FoundState ) {
         RC_DEC_NORMALIZE(EncodedFile);
         do {
            if ( !MinContext->iSuffix ) goto
STOP_DECODING;
            OrderFall++;
MinContext=MinContext->suff();
         } while (MinContext->NumStats ==
NumMasked);
         MinContext->decodeSymbol2();
rcRemoveSubrange();
      }
      _PPMD_D_PUTC(FoundState-
>Symbol,DecodedFile);
      if (!OrderFall && (_BYTE*)FoundState-
>getSucc() >= UnitsStart)

PrefetchData(MaxContext=FoundState-
>getSucc());
      else {
         UpdateModel(MinContext);
         if (EscCount == 0)
ClearMask(EncodedFile,DecodedFile);
      }
      RC_DEC_NORMALIZE(EncodedFile);
MinContext=MaxContext;
   }
STOP_DECODING:
   PrintInfo(DecodedFile,EncodedFile);
}
#if !defined(NDEBUG)
#pragma option -w-ccc -w-rch
BOOL TestCompilation()
{
/*****************  testing on proper sizes of
data types  ******************/
   if (2*sizeof(_BYTE) != sizeof(_WORD) ||
12*sizeof(_BYTE) != sizeof(PPM_CONTEXT))
         return FALSE;
/*************  testing on proper alignment of
our data types  **************/
   PPM_CONTEXT tmp;
   if (2*sizeof(PPM_CONTEXT::STATE) !=
sizeof(PPM_CONTEXT) || &tmp.iSuffix !=
(_DWORD*)(&tmp.oneState()+1))
         return FALSE;
   return TRUE;
}
#endif /* !defined(NDEBUG) */

#define BACKSLASH '\\'
#define USAGE_STR "Usage: PPMd <e|d>
[switches] <FileName[s] | Wildcard[s]>\n"
static const char* pFName;
static DWORD StartFilePosition;
static BOOL EncodeFlag;
static clock_t StartClock;
static struct ARC_INFO { // FileLength & CRC?
Hmm, maybe in another times...
      DWORD signature,attrib;
      WORD  info,FNLen,time,date;
} ai;

#if defined(_WIN32_ENVIRONMENT_)
#include <conio.h>

inline void EnvSetNormAttr(const char*
FName) {
SetFileAttributes(FName,FILE_ATTRIBUTE_N
ORMAL); }
inline int                    EnvGetCh() { return
getch(); }
inline void          EnvGetCWD(char* CurDir) {
GetCurrentDirectory(260,CurDir); }
inline void EnvSetDateTimeAttr(const char*
WrkStr)
{
   FILETIME ft;
   DosDateTimeToFileTime(ai.date,ai.time,&ft);
   HANDLE
hndl=CreateFile(WrkStr,GENERIC_WRITE,0,N
ULL,OPEN_EXISTING,0,0);
```

```
   SetFileTime(hndl,&ft,NULL,&ft);
CloseHandle(hndl);
   SetFileAttributes(WrkStr,ai.attrib);
}
struct ENV_FIND_RESULT: public
WIN32_FIND_DATA {
      const char*  getFName() const { return
cFileName; }
      void copyDateTimeAttr() const {
            ai.attrib=dwFileAttributes;

FileTimeToDosDateTime(&ftLastWriteTime,&a
i.date,&ai.time);
      }
};
struct ENV_FILE_FINDER {
      HANDLE hndl;
      ENV_FIND_RESULT Rslt;
      ENV_FIND_RESULT getResult() { return
Rslt; }
      BOOL isFileValid() {
            return ((Rslt.dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY) == 0);
      }
      BOOL findFirst(const char* Pattern) {
            return
((hndl=FindFirstFile(Pattern,&Rslt)) !=
INVALID_HANDLE_VALUE);
      }
      BOOL findNext() { return
FindNextFile(hndl,&Rslt); }
      void findStop() { FindClose(hndl); }
};

#elif defined(_DOS32_ENVIRONMENT_)
#include <conio.h>
#include <dos.h>
#include <direct.h>
#if defined(__DJGPP__)
#include <unistd.h>
#include <crt0.h>
#undef BACKSLASH
#define BACKSLASH '/'
char **__crt0_glob_function (char *arg) { return
0; }
void   __crt0_load_environment_file (char
*progname) { }
#endif /* defined(__DJGPP__) */

inline void EnvSetNormAttr(const char*
FName) {
_dos_setfileattr(FName,_A_NORMAL); }
inline int                    EnvGetCh() { return
getch(); }
inline void          EnvGetCWD(char* CurDir) {
getcwd(CurDir,260); }
inline void EnvSetDateTimeAttr(const char*
WrkStr)
{
   FILE* fpOut = fopen(WrkStr,"a+b");
   _dos_setftime(fileno(fpOut),ai.date,ai.time);
   fclose(fpOut);
   _dos_setfileattr(WrkStr,ai.attrib);
}
struct ENV_FIND_RESULT: public find_t {
   const char*  getFName() const { return name;
}
   void copyDateTimeAttr() const {
      ai.attrib=attrib;
      ai.time=wr_time;
ai.date=wr_date;
   }
};
struct ENV_FILE_FINDER {
   ENV_FIND_RESULT Rslt;
   ENV_FIND_RESULT getResult() { return
Rslt; }
   BOOL isFileValid() { return TRUE; }
   BOOL findFirst(const char* Pattern) {
      return
(_dos_findfirst(Pattern,_A_RDONLY |
_A_HIDDEN | _A_SYSTEM |
_A_ARCH,&Rslt) == 0);
   }
   BOOL findNext() { return
(_dos_findnext(&Rslt) == 0); }
   void findStop() { }
};

#elif defined(_POSIX_ENVIRONMENT_)
#undef BACKSLASH
#define BACKSLASH '/'
#include <sys/stat.h>
#include <time.h>
#include <utime.h>
#include <dirent.h>
#include <unistd.h>
#include <fnmatch.h>

inline void EnvSetNormAttr(const char*
FName) { chmod(FName,S_IWUSR|S_IRUSR);
}
inline int                    EnvGetCh() { return
getchar(); }
inline void          EnvGetCWD(char* CurDir) {
getcwd(CurDir,260); }
inline void EnvSetDateTimeAttr(const char*
WrkStr)
{
   struct utimbuf t;
   t.actime=t.modtime=(ai.date << 16)+ai.time;
```

```
   utime(WrkStr,&t);
chmod(WrkStr,ai.attrib);
}
struct ENV_FIND_RESULT {
   dirent de;
   struct stat st;
   const char*  getFName() const { return
de.d_name; }
   void copyDateTimeAttr() const {
      ai.attrib=st.st_mode;
      ai.time=st.st_mtime & 0xFFFF;
ai.date=st.st_mtime >> 16;
   }
};
struct ENV_FILE_FINDER {
   const char* pPattern;
   DIR* dir;
   dirent* de;
   struct stat st;
   ENV_FIND_RESULT getResult() {
      ENV_FIND_RESULT Rslt;
      Rslt.de=*de;                    Rslt.st=st;
      return Rslt;
   }
   BOOL isFileValid() {
      return (fnmatch(pPattern,de-
>d_name,FNM_NOESCAPE) == 0 &&
            stat(de->d_name,&st) == 0 &&
(st.st_mode & S_IRUSR) != 0 &&
            st.st_nlink == 1);
   }
   BOOL findFirst(const char* Pattern) {
      pPattern=Pattern;
      return ((dir=opendir(".")) &&
(de=readdir(dir)) != NULL);
   }
   BOOL findNext() { return ((de=readdir(dir))
!= NULL); }
   void findStop() { closedir(dir); }
};

#else /* _UNKNOWN_ENVIRONMENT_ */
#pragma message ("unknown environment:")
#pragma message ("    1. _fastcall and _stdcall
keywords are disabled")
#pragma message ("    2. wildcards and file
attributes are disabled")

#undef  USAGE_STR
#define USAGE_STR "Usage: PPMd <e|d>
[switches] FileName[s]\n"
inline void     EnvSetNormAttr(const char* ) {}
inline int                  EnvGetCh() { return
getchar(); }
inline void         EnvGetCWD(char* CurDir) {
CurDir[0]=0; }
inline void EnvSetDateTimeAttr(const char* ) {}
struct ENV_FIND_RESULT {
   char FName[260];
   const char*  getFName() const { return
FName; }
   void copyDateTimeAttr() const {}
};
struct ENV_FILE_FINDER {
   const char* pPattern;
   ENV_FIND_RESULT getResult() {
      ENV_FIND_RESULT Rslt;
strcpy(Rslt.FName,pPattern);
      return Rslt;
   }
   BOOL isFileValid() { return TRUE; }
   BOOL findFirst(const char* Pattern) {
      pPattern=Pattern;                return TRUE;
   }
   BOOL        findNext() { return FALSE; }
   void findStop() {}
};
#endif /* defined(__WIN32_ENVIRONMENT)
*/

static const char* const MTxt[] = { "Can`t open
file %s\n",
   "read/write error for files %s/%s\n", "Out of
memory!\n",
   "User break\n", "unknown command: %s\n",
"unknown switch: %s\n",
   "designed and written by Dmitry Shkarin
<dmitry.shkarin@mtu-net.ru>\n"
   USAGE_STR
   "Switches (for encoding only):\n"
   "\t-d     - delete file[s] after processing,
default: disabled\n"
   "\t-fName - set output file name to Name\n"
   "\t-mN    - use N MB memory - [1,256],
default: %d\n"
   "\t-oN    - set model order to N - [2,%d],
default: %d\n"
   "\t-rN    - set method of model restoration at
memory insufficiency:\n"
   "\t\t-r0 - restart model from scratch
(default)\n"
   "\t\t-r1 - cut off model (slow)\n"
};

void _STDCALL PrintInfo(_PPMD_FILE*
DecodedFile,_PPMD_FILE* EncodedFile)
{
   //Tfrminduk *ind;

   char WrkStr[320];
   UINT NDec=ftell(DecodedFile);
   NDec += (NDec == 0);
```

```
   UINT NEnc=ftell(EncodedFile)-
StartFilePosition;
   UINT n1=(8U*NEnc)/NDec;
   UINT n2=(100U*(8U*NEnc-
NDec*n1)+NDec/2U)/NDec;
   if (n2 == 100) { n1++;                n2=0; }
   int RunTime=((clock()-StartClock) <<
10)/int(CLOCKS_PER_SEC);
   UINT Speed=NDec/(RunTime+(RunTime ==
0));
   UINT UsedMemory=GetUsedMemory() >>
10;
   UINT m1=UsedMemory >> 10;
   UINT m2=(10U*(UsedMemory-(m1 <<
10))+(1 << 9)) >> 10;
   if (m2 == 10) { m1++;                m2=0; }
   if ( !EncodeFlag )
SWAP(NDec,NEnc);
   sprintf(/*WrkStr,*/"%14s :%7d > %7d,
%1d.%02d bpb, used:%3d.%1dMB, speed: %d
KB/sec",

pFName,NEnc,NDec,n1,n2,m1,m2,Speed);
   SFile.sprintf("%14s", pFName);
   SUkuranFile.sprintf("%7d", NEnc);
   SUkuranFiled.sprintf("%7d", NDec);
   SKecepatan.sprintf("%d", Speed);

   printf("%-79.79s\r",WrkStr);
fflush(stdout);

   //ind->Label25->Caption = S;

}
static char* _STDCALL ChangeExtRare(const
char* In,char* Out,const char* Ext)
{
   char* RetVal=Out;
   const char* p=strrchr(In,'.');
   if (!p || strrchr(In,BACKSLASH) > p)
p=In+strlen(In);
   do { *Out++ = *In++; } while (In != p);
   *Out++='.';
   while((*Out++ = *Ext++) != 0)          ;
   return RetVal;
}
inline BOOL RemoveFile(const char* FName)
{
   EnvSetNormAttr(FName);                return
(remove(FName) == 0);
}
static BOOL _STDCALL TestAccessRare(const
char* FName)
{
static BOOL YesToAll=TRUE;
   FILE* fp=fopen(FName,"rb");
   if ( !fp )                        return TRUE;
   fclose(fp);
   if ( YesToAll )                   return
RemoveFile(FName);
   printf("%s already exists, overwrite?: <Y>es,
<N>o, <A>ll, <Q>uit?",FName);
   for ( ; ; )
      switch ( toupper(EnvGetCh()) ) {
         case 'A':                   YesToAll=TRUE;
         case '\r': case 'Y':
YesToAll=TRUE;//return RemoveFile(FName);
         case 0x1B: case 'Q':
printf(MTxt[3]); exit(-1);
         case 'N':
YesToAll=TRUE;//return FALSE;
      }
}
static FILE* FOpen(const char* FName,const
char* mode)
{
   FILE* fp=fopen(FName,mode);
   if ( !fp ) { printf(MTxt[0],FName);    exit(-1);
}
   setvbuf(fp,NULL,_IOFBF,64*1024);
return fp;
}
inline void PrepareCoding(int SASize,FILE* fp)
{
   if ( !StartSubAllocator(SASize) ) {
      printf(MTxt[2]);                 exit(-1);
   }
   StartClock=clock();
StartFilePosition=ftell(fp);
}
inline void EncodeFile(const
ENV_FIND_RESULT& efr,int MaxOrder,int
SASize,BOOL CutOff,const char* ArcName)
{
   char WrkStr[260];
   strcpy(WrkStr,ArcName);
   if (!WrkStr[0] &&
!TestAccessRare(ChangeExtRare(efr.getFName(
),WrkStr,"pmd")))
            return;
   FILE* fpIn = FOpen(efr.getFName(),"rb"), *
fpOut = FOpen(WrkStr,"a+b");

pFName=strrchr(efr.getFName(),BACKSLASH)
;
   pFName=( pFName
)?(pFName+1):(efr.getFName());
   efr.copyDateTimeAttr();
   ai.signature=PPMdSignature;
ai.FNLen=strlen(pFName)+(CutOff << 14);
   ai.info=(MaxOrder-1) | ((SASize-1) << 4) |
((PROG_VAR-'A') << 12);
```

```
   fwrite(&ai,sizeof(ai),1,fpOut);
fwrite(pFName,ai.FNLen & 0x1FF,1,fpOut);
   PrepareCoding(SASize,fpOut);
EncodeFile(fpOut,fpIn,MaxOrder,CutOff);
   putchar('\n');
   if (ferror(fpOut) || ferror(fpIn)) {
      printf(MTxt[1],efr.getFName(),WrkStr);
      exit(-1);
   }
   fclose(fpIn);                    fclose(fpOut);
}
inline BOOL DecodeOneFile(FILE* fpIn)
{
   char WrkStr[260];
   int MaxOrder, SASize;
   BOOL CutOff;
   if ( !fread(&ai,sizeof(ai),1,fpIn) )    return
FALSE;
   CutOff=ai.FNLen >> 14;
   ai.FNLen=CLAMP(int(ai.FNLen &
0x1FF),1,260-1);
   fread(WrkStr,ai.FNLen,1,fpIn);
WrkStr[ai.FNLen]=0;
   if ( !TestAccessRare(WrkStr) )          return
FALSE;
   FILE* fpOut =
FOpen(pFName=WrkStr,"wb");
   //FILE* fpOut = FOpen(otfile, "wb");

   MaxOrder=(ai.info & 0x0F)+1;
SASize=((ai.info >> 4) & 0xFF)+1;
   DWORD Variant=(ai.info >> 12)+'A';
   if (ai.signature != PPMdSignature || Variant !=
PROG_VAR) {
      printf(MTxt[0],WrkStr);            exit(-1);
   }
   PrepareCoding(SASize,fpIn);
DecodeFile(fpOut,fpIn,MaxOrder,CutOff);
   putchar('\n');
   if (ferror(fpOut) || ferror(fpIn) || feof(fpIn)) {
      printf(MTxt[1],WrkStr,WrkStr);      exit(-1);
   }
   fclose(fpOut);
EnvSetDateTimeAttr(WrkStr);
   return TRUE;
}
inline void DecodeFile(const
ENV_FIND_RESULT& efr)
{
   FILE* fpIn=FOpen(efr.getFName(),"rb");
   while ( DecodeOneFile(fpIn) );
   fclose(fpIn);

}
inline void TestArchive(char* ArcName,const
char* Pattern)
{
   if ( !Pattern[0] ) {
      char CurDir[260];
      EnvGetCWD(CurDir);
      const char*
p=strrchr(CurDir,BACKSLASH);
      p = (p &&
strlen(p+1))?(p+1):("PPMdFile");
      ChangeExtRare(p,ArcName,"pmd");
   } else
strcpy(ArcName,Pattern);
   FILE* fp = fopen(ArcName,"rb");
   if ( fp ) {
      if (!fread(&ai,sizeof(ai),1,fp) || ai.signature
!= PPMdSignature ||
                                 (ai.info >> 12)+'A' !=
PROG_VAR) {
         printf(MTxt[0],ArcName);       exit(-1);
      }
      fclose(fp);
   }
}
struct FILE_LIST_NODE {
   FILE_LIST_NODE* next;
   ENV_FIND_RESULT efr;
   FILE_LIST_NODE(const
ENV_FIND_RESULT&
Data,FILE_LIST_NODE** PrevNode) {
      efr=Data;                  next=*PrevNode;
      *PrevNode=this;
   }
   void destroy(FILE_LIST_NODE**
PrevNode) {
      *PrevNode=next;                delete this;
   }
};



//-------------------------------------------------------
----------
```