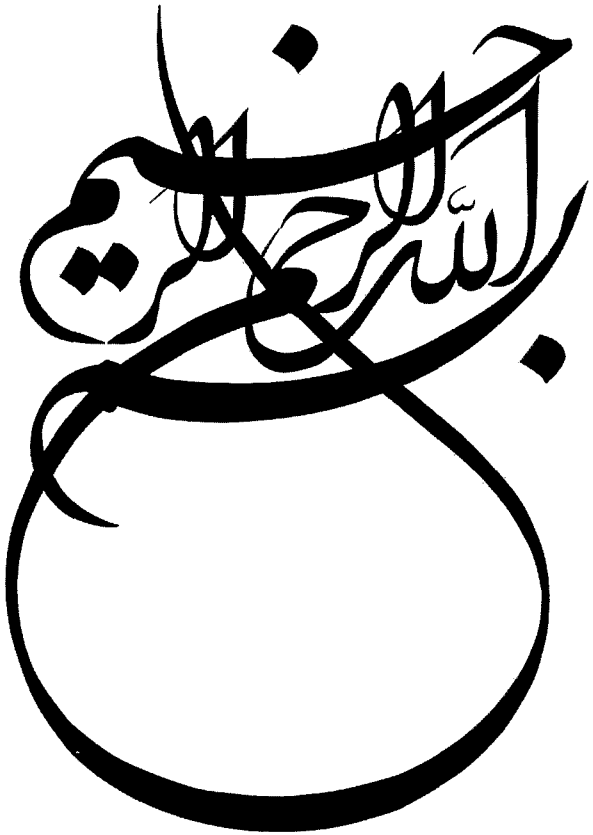
برنامه کامل

پروسه تبدیل اطلاعات اصطلاحنامه از یک فایل متنی به ساختار اطلاعاتی نرم‌افزار ساخت اصطلاحنامه "قاموس"

طراحی و اجرا: مهرداد نوروزی اقبالی


بسم الله الرحمن الرحيم

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

#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("Process.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
     try
     {
          Application->Initialize();
          Application->CreateForm(__classid(TForm1), &Form1);
          Application->Run();
     }
     catch (Exception &exception)
     {
          Application->ShowException(&exception);
     }
     catch (...)
     {
          try
          {
               throw Exception("");
          }
          catch (Exception &exception)
          {
               Application->ShowException(&exception);
          }
     }
     return 0;
}
//---------------------------------------------------------------------------
```

```
#include <vcl.h>
#include <ComCtrls.hpp>
#include <odbcinst.h>

#include "Functions.hpp"
#include "Unicode.hpp"

enum RV {
    RV_OK,                  // Okay, Success, True, whatever

    // -- Usage and environment errors --
    RV_USAGE,               // Invalid command line
    RV_NOT_ENOUGH_SPACE,    // Not enough space for backup
    RV_FILE_LOCKED,         // DB was locked by another process
    RV_OPEN_FILE,           // Invalid filename or could not open
    RV_CHANGE_DIRECTORY,    // Error chdiring to the location of the mdb
    RV_WORKING_DIR,         // Error chdiring to the specified working dir

    // -- Errors from SQLConfigDataSource() --
    RV_DB_COMPACTION,       // Database compaction failed
    RV_CREATE_DB,           // Failed to create DB
    RV_REPAIR_DB,           // Failed to repair DB
    RV_CREATE_DSN,          // Failed to create DSN

    // -- Technical errors --
    RV_GET_FILE_SIZE,       // Error getting file size
    RV_GET_DISK_SPACE,      // Error getting disk space
    RV_MOVE_FILE,           // Could not move database
    RV_MAKE_BACKUP_COPY,    // Could not create a backup copy
    RV_NO_SAFE_PATHNAME,    // Could not get a shortened form of a pathname
    RV_RESTORE_FILE,        // Could not move database back to original location when done

    RV_MAX
};

int __fastcall CreateAccessDatabase(char *FileName)
{
     AnsiString Attributes;
     char *Pos;
     char *Driver = "Microsoft Access Driver (*.mdb)";

     Attributes = Format("CREATE_DB=\"%s\" General;;", ARRAYOFCONST(((char *)FileName)));
     Pos = Attributes.c_str();

     do
     {
          if(*Pos == ';')
               *Pos = 0;
          Pos++;
     }while(*Pos);

     return SQLConfigDataSource(NULL, ODBC_CONFIG_SYS_DSN, Driver, Attributes.c_str());
}
```

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

bool IsDigitalString(const wchar_t *String)
{
     for(unsigned register i = 0; i < wcslen(String); i++)
          if(!iswdigit(String[i]))
               return false;

     return true;
}
//---------------------------------------------------------------------------

int ThesaurusSpecialCompare(WideString s1, WideString s2, unsigned len)
{
     int ReturnValue;

     if(IsDigitalString(s1) && IsDigitalString(s2) && !s1.IsEmpty() && !s2.IsEmpty())
          ReturnValue = StrToInt(s1) - StrToInt(s2);
     else
     {
          AnsiString Str1 = s1;
          AnsiString Str2 = s2;

          if(
             (*Str1.c_str() == '[' && *Str1.AnsiLastChar() == ']') ||
             (*Str1.c_str() == ']' && *Str1.AnsiLastChar() == '[')
            )
          {
               Str1.Delete(1, 1);
               Str1.Delete(Str1.Length(), 1);
          }

          if(
             (*Str2.c_str() == '[' && *Str2.AnsiLastChar() == ']') ||
             (*Str2.c_str() == ']' && *Str2.AnsiLastChar() == '[')
            )
          {
               Str2.Delete(1, 1);
               Str2.Delete(Str2.Length(), 1);
          }

          ReturnValue = Str1.AnsiCompareIC(Str2);
     }

     return ReturnValue;
}
//---------------------------------------------------------------------------

int AnsiStringCompare(WideString s1, WideString s2, unsigned len)
{
     return WideCompareStr(s1, s2);
}
//---------------------------------------------------------------------------
```

```
int AnsiStringCompareIC(WideString s1, WideString s2, unsigned len)
{
     return WideCompareText(s1, s2);
}
//---------------------------------------------------------------------------

void RemoveSpace(AnsiString *String)
{
     char ch = *String->c_str();

     while(ch == ' ' || ch == '\t' || ch == '\r' || ch == '\n')
     {
          String->Delete(1, 1);
          ch = *String->c_str();
     }

     if(!String->IsEmpty())
     {
          ch = *String->AnsiLastChar();
          while(ch == ' ' || ch == '\t' || ch == '\r' || ch == '\n')
          {
               String->SetLength(String->Length() - 1);
               if(!String->IsEmpty())
                    ch = *String->AnsiLastChar();
          }
     }
}
//---------------------------------------------------------------------------

void RemoveSpace(WideString *String)
{
     if(!String->IsEmpty())
     {
          wchar_t ch = *String->c_bstr();
          int Len;

          if(!ch)
               String->Empty();
          else
          {
               while(ch == ' ' || ch == '\t' || ch == '\r' || ch == '\n')
               {
                    String->Delete(1, 1);
                    if(!String->IsEmpty())
                         ch = *String->c_bstr();
                    else
                         break;
               }

               if(!String->IsEmpty())
               {
                    Len = String->Length();
```

```
                    ch = String->c_bstr()[Len - 1];
                    while(ch == ' ' || ch == '\t' || ch == '\r' || ch == '\n')
                    {
                         String->SetLength(--Len);
                         if(!String->IsEmpty())
                              ch = String->c_bstr()[Len - 1];
                    }
               }
          }
     }
}
//---------------------------------------------------------------------------

void RemoveSpace(char *Str)
{
     if(Str)
     {
          AnsiString String = Str;
          RemoveSpace(&String);
          StrCopy(Str, String.c_str());
     }
}
//---------------------------------------------------------------------------

void RemoveSpace(wchar_t *Str)
{
     if(Str)
     {
          WideString String(Str);
          RemoveSpace((WideString *)&String);
          if(!String.IsEmpty())
               wcscpy(Str, String);
          else
               *Str = 0;
     }
}
//---------------------------------------------------------------------------

int SearchTreeView(TTreeView *TreeView, AnsiString *String)
{
     int ReturnValue = -1;

     for(register i = 0; i < TreeView->Items->Count; i++)
          if(!String->AnsiCompareIC(TreeView->Items->Item[i]->Text))
          {
               ReturnValue = i;
               break;
          }

     return ReturnValue;
}
//---------------------------------------------------------------------------
```

```
int SearchTreeView(TTreeView *TreeView, WideString String)
{
     int ReturnValue = -1;
     AnsiString Str;

     for(register i = 0; i < TreeView->Items->Count; i++)
     {
          Str = TreeView->Items->Item[i]->Text;
          if(!StrICompW(String, (WideString)Str))
          {
               ReturnValue = i;
               break;
          }
     }

     return ReturnValue;
}
//---------------------------------------------------------------------------

int SearchStringList(TStringList *List, AnsiString *String)
{
     int ReturnValue = -1;

     for(register i = 0; i < List->Count; i++)
          if(!String->AnsiCompareIC(List->Strings[i]))
          {
               ReturnValue = i;
               break;
          }

     return ReturnValue;
}
//---------------------------------------------------------------------------

int SearchStringList(TWideStringList *List, WideString String)
{
     int ReturnValue = -1;

     for(register i = 0; i < List->Count; i++)
          if(!StrICompW(String, List->Strings[i]))
          {
               ReturnValue = i;
               break;
          }

     return ReturnValue;
}
//---------------------------------------------------------------------------

int SearchStrings(TStrings *List, AnsiString *String)
{
     int ReturnValue = -1;
```

```
     for(register i = 0; i < List->Count; i++)
          if(!String->AnsiCompareIC(List->Strings[i]))
          {
               ReturnValue = i;
               break;
          }

     return ReturnValue;
}
//---------------------------------------------------------------------------

int SearchStrings(TWideStrings *List, WideString *String)
{
     int ReturnValue = -1;

     for(register i = 0; i < List->Count; i++)
          if(!AnsiStringCompareIC(*String, List->Strings[i]))
          {
               ReturnValue = i;
               break;
          }

     return ReturnValue;
}
//---------------------------------------------------------------------------

void RemoveDuplicateSpaces(AnsiString *Str)
{
     int Pos;

     while((Pos = Str->AnsiPos("  ")) > 0)
          Str->Delete(Pos, 1);
}
//---------------------------------------------------------------------------

void RemoveDuplicateSpaces(char *Str)
{
     AnsiString String = Str;
     RemoveDuplicateSpaces(&String);
     StrCopy(Str, String.c_str());
}
//---------------------------------------------------------------------------

void RemoveDuplicateSpaces(wchar_t *Str)
{
     if(Str)
     {
          int Pos;

          WideString String(Str);

          while((Pos = String.Pos("  ")) > 0)
               String.Delete(Pos, 1);
```

```
          if(!String.IsEmpty())
               wcscpy(Str, String);
          else
               *Str = 0;
     }
}
//---------------------------------------------------------------------------

void Sort(TStringList *List, bool RemoveDuplicates, bool RemoveEmptyLines, bool RemoveAdditionalSpaces)
{
     int Pos;
     AnsiString Str;

     if(RemoveAdditionalSpaces)
     {
          for(register i = 0; i < List->Count; i++)
          {
               Str = List->Strings[i];
               RemoveSpace(&Str);
               RemoveDuplicateSpaces(&Str);
               List->Strings[i] = Str;
          }
     }

     List->Sort();

     if(RemoveDuplicates || RemoveEmptyLines)
          for(register i = 0; i < List->Count; i++)
          {
               if(RemoveDuplicates && i)
               {
                    if(!List->Strings[i].AnsiCompareIC(List->Strings[i - 1]))
                    {
                         List->Delete(i);
                         i--;
                    }
               }
               else
                    if(RemoveEmptyLines)
                    {
                         if(List->Strings[i].IsEmpty())
                         {
                              List->Delete(i);
                              i--;
                         }
                    }
          }
}
//---------------------------------------------------------------------------

void Sort(AnsiString *String, bool RemoveDuplicates, bool RemoveEmptyLines, bool RemoveAdditionalSpaces)
{
```

```
	TStringList *List = new TStringList;
	int Pos;

	List->Text = *String;
	Sort(List, RemoveDuplicates, RemoveEmptyLines, RemoveAdditionalSpaces);
	*String = List->Text;

	delete List;
}
//---------------------------------------------------------------------------

void Sort(TWideStringList *List, bool RemoveDuplicates, bool RemoveEmptyLines, bool RemoveAdditionalSpaces)
{
	int Pos;
	WideString Str;

	if(RemoveAdditionalSpaces)
	{
		for(register i = 0; i < List->Count; i++)
		{
			Str = List->Strings[i];
			RemoveSpace(Str);
			RemoveDuplicateSpaces(Str);
			List->Strings[i] = Str;
		}
	}

	List->Sort();

	if(RemoveDuplicates || RemoveEmptyLines)
		for(register i = 0; i < List->Count; i++)
		{
			if(RemoveDuplicates && i)
			{
				if(CompareStringW(LOCALE_SYSTEM_DEFAULT, NORM_IGNORECASE,
							List->Strings[i], -1, List->Strings[i - 1], -1) == 2)
				{
					List->Delete(i);
					i--;
				}
			}
			else
				if(RemoveEmptyLines)
				{
					if(List->Strings[i].IsEmpty())
					{
						List->Delete(i);
						i--;
					}
				}
		}
}
//---------------------------------------------------------------------------
```

```
void Sort(WideString *String, bool RemoveDuplicates, bool RemoveEmptyLines, bool RemoveAdditionalSpaces)
{
     TWideStringList *List = new TWideStringList;
     int Pos;

     List->Text = *String;
     Sort(List, RemoveDuplicates, RemoveEmptyLines, RemoveAdditionalSpaces);
     *String = List->Text;

     if(List->Count < 2)
     {
          RemoveSpace((WideString *)String);
          RemoveDuplicateSpaces(*String);
     }

     delete List;
}
//---------------------------------------------------------------------------

int __fastcall StringToInt(AnsiString Str)
{
     int ReturnValue = 0;
     if(!Str.IsEmpty())
          ReturnValue = StrToInt(Str);

     return ReturnValue;
}
//---------------------------------------------------------------------------

bool __fastcall IsInNumericRange(AnsiString LowValueStr, AnsiString HiValueStr, int CurrentValue)
{
     bool ReturnValue = false;
     int Low, High;
     int Swap;

     Low = StringToInt(LowValueStr);
     High = StringToInt(HiValueStr);

     if(High < Low && !HiValueStr.IsEmpty() && !LowValueStr.IsEmpty())
     {
          Swap = High;
          High = Low;
          Low = Swap;
     }

     if((CurrentValue <= High || HiValueStr.IsEmpty()) &&
        (CurrentValue >= Low || LowValueStr.IsEmpty()))
          ReturnValue = true;

     return ReturnValue;
}
//---------------------------------------------------------------------------
```

```
char * __fastcall AnsiStrPosIC(char * Str, char * SubStr)
{
     char *Str2 = new char[StrLen(Str) + 1], *SubStr2 = new char[StrLen(SubStr) + 1];
     char *Pos;
     char *ReturnValue = NULL;

     StrCopy(Str2, Str);
     StrCopy(SubStr2, SubStr);

     StrUpper(Str2);
     StrUpper(SubStr2);

     Pos = AnsiStrPos(Str2, SubStr2);

     if(Pos)
          ReturnValue = Str + int(Pos) - int(Str2);

     delete[] SubStr2;
     delete[] Str2;

     return ReturnValue;
}
//---------------------------------------------------------------------------

wchar_t * __fastcall AnsiStrPosICW(wchar_t * Str, wchar_t * SubStr)
{
     WideString Str2(Str), SubStr2(SubStr);
     int Pos;
     wchar_t *ReturnValue = NULL;

     Str2 = Sysutils::WideUpperCase(Str2);
     SubStr2 = Sysutils::WideUpperCase(SubStr2);

     Pos = Str2.Pos(SubStr2);

     if(Pos)
          ReturnValue = Str + Pos - 1;

     return ReturnValue;
}
//---------------------------------------------------------------------------

WideString WideCopy(WideString *dest, WideString *src, int Len)
{
     if(Len < 0)
          Len = src->Length();

     dest->Empty();
     if(Len)
          dest->Insert(*src, 0);
     dest->SetLength(Len);
```

```
     return *dest;
}
//---------------------------------------------------------------------------

WideString WideCopy(WideString *dest, AnsiString src, int Len)
{
     if(Len < 0)
          Len = src.Length();

     dest->Empty();
     if(Len)
          dest->Insert(src, 0);
     dest->SetLength(Len);

     return *dest;
}
//---------------------------------------------------------------------------

WideString WideCopy(WideString *dest, wchar_t *src, int Len)
{
     WideString Source(src, Len < 0 ? wcslen(src) : Len + 1);
     if(Len < 0)
          Len = Source.Length();

     dest->Empty();
     if(Len)
          dest->Insert(Source, 0);
     dest->SetLength(Len);

     return *dest;
}
//---------------------------------------------------------------------------

int __fastcall WriteUnicode(TStream *Stream, wchar_t *Buffer, int Count)
{
     AnsiString String = "";
     wchar_t *Pos = Buffer;
     int Number;
     char ch[2] = " ";

     for(int i = 0; i < Count; ++i)
     {
          Number = (short unsigned)*Pos;

          if(Number < 128 && Number != L'{' && Number != L'}')
          {
               ch[0] = Number;
               String += ch;
          }
          else
               String += "\\u" + IntToStr(Number) + "'";
          ++Pos;
     }
```

```
    return Stream->Write(String.c_str(), String.Length());
}
//---------------------------------------------------------------------------
```

```
#include <ComCtrls.hpp>
#include <Unicode.hpp>

int  AnsiStringCompare(WideString s1, WideString s2, unsigned len = 0);
int  AnsiStringCompareIC(WideString s1, WideString s2, unsigned len = 0);
char * __fastcall AnsiStrPosIC(char * Str, char * SubStr);
wchar_t * __fastcall AnsiStrPosICW(wchar_t * Str, wchar_t * SubStr);
int __fastcall CreateAccessDatabase(char *FileName);
bool IsDigitalString(const char *String);
bool IsDigitalString(const wchar_t *String);
bool __fastcall IsInNumericRange(AnsiString LowValueStr, AnsiString HiValueStr, int CurrentValue);
void RemoveDuplicateSpaces(AnsiString *Str);
void RemoveDuplicateSpaces(char *Str);
void RemoveDuplicateSpaces(wchar_t *Str);
void RemoveSpace(WideString *String);
void RemoveSpace(AnsiString *String);
void RemoveSpace(char *str);
void RemoveSpace(wchar_t *Str);
int  SearchStringList(TStringList *List, AnsiString *String);
int  SearchStringList(TWideStringList *List, WideString String);
int  SearchStrings(TStrings *List, AnsiString *String);
int  SearchStrings(TWideStrings *List, WideString *String);
int  SearchTreeView(TTreeView *TreeView, AnsiString *String);
int  SearchTreeView(TTreeView *TreeView, WideString String);
void Sort(AnsiString *String, bool RemoveDuplicates = true, bool RemoveEmptyLines = true, bool RemoveDuplicateSpaces = true);
void Sort(TStringList *List, bool RemoveDuplicates = true, bool RemoveEmptyLines = true, bool RemoveDuplicateSpaces = true);
void Sort(WideString *String, bool RemoveDuplicates = true, bool RemoveEmptyLines = true, bool RemoveDuplicateSpaces = true);
void Sort(TWideStringList *List, bool RemoveDuplicates = true, bool RemoveEmptyLines = true, bool RemoveDuplicateSpaces = true);
int __fastcall StringToInt(AnsiString Str);
int  ThesaurusSpecialCompare(WideString s1, WideString s2, unsigned len = 0);
WideString WideCopy(WideString *dest, WideString *src, int Len = -1);
WideString WideCopy(WideString *dest, AnsiString src, int Len = -1);
WideString WideCopy(WideString *dest, wchar_t *src, int Len = -1);
int  __fastcall WriteUnicode(TStream *Stream, wchar_t *Buffer, int Count);
```

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

#include <vcl.h>
#pragma hdrstop

#include "Process.h"
#include "Functions.hpp"
#include "WFbtree.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "TntStdCtrls"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if(OpenDialog1->Execute())
    {
        LabeledEdit1->Text = OpenDialog1->FileName;
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
    if(SaveDialog1->Execute())
    {
        LabeledEdit2->Text = SaveDialog1->FileName;
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
    try
    {
        LoopEncountered = false;
        if(!LabeledEdit2->Text.IsEmpty())
        {
            Form1->Enabled = false;

            DeleteFile("~001data.tmp");
            DeleteFile("~002data.tmp");
            Tree = new Fbtree("~001data.tmp");
            Linear = new Fbtree("~002data.tmp");

            StatusBar1->SimpleText = "ساخت ایندکس...";
```

```
                FindAllTerms();
                StatusBar1->SimpleText = "بهينه‌سازي..."  ;
                Linear->Optimize();

                StatusBar1->SimpleText = "جستجوي واژه هاي ريشه..." ;
                FindTopTerms();
                StatusBar1->SimpleText = "بهينه‌سازي..."  ;
                Tree->Optimize();

                Tree->Go(TOP);
                StatusBar1->SimpleText = "ايجاد ساختار..." ;
                MakeThesaurusStructure();

                if(!LoopEncountered)
                {
                     StatusBar1->SimpleText = "لطفا صبر كنيد. ذخيره با فرمت اطلاعاتي MS-Access..."   ;
                     SendToMicrosoftAccess();
                }

                delete Linear;
                delete Tree;

                DeleteFile("~001data.tmp");
                DeleteFile("~002data.tmp");
          }
     }
     __finally
     {
           StatusBar1->SimpleText = "";
           Form1->Enabled = true;
           /*Button1->Enabled = true;
           Button2->Enabled = true;
           Button3->Enabled = true;*/
     }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FindAllTerms(void)
{
     if(!LabeledEdit1->Text.IsEmpty() && FileExists(LabeledEdit1->Text))
     {
          Record Rec;
          InputLines = new TWideStringList();

          InputLines->LoadFromFile(LabeledEdit1->Text);

          CountLines = 0;
          while(FindTextRecord(&Rec, CountLines))
          {
               SaveRecord(Linear, &Rec);
          }

          delete InputLines;
```

```
    }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FindTopTerms(void)
{
    if(!LabeledEdit1->Text.IsEmpty() && FileExists(LabeledEdit1->Text))
    {
        Record Rec;
        InputLines = new TWideStringList();

        InputLines->LoadFromFile(LabeledEdit1->Text);

        CountLines = 0;
        while(FindTextRecord(&Rec, CountLines, true))
        {
            if(Rec.USE.IsEmpty())
                SaveRecord(Tree, &Rec);
        }

        delete InputLines;
    }
}
//---------------------------------------------------------------------------

bool __fastcall TForm1::FindTextRecord(Record *Rec, int LineNumber, bool OnlyTopTerms)
{
    int  Counter = LineNumber, EndOfRecord;
    bool Done = false;
    bool ReturnValue = false;
    WideString String;
    int BTCounter;
    ThesaurusFields LastField;

    while(!Done && Counter < InputLines->Count)
    {
        ResetRecord(Rec);
        BTCounter = 0;

        Counter = CountLines = FindStartOfNextTextRecord(Counter - 1, OnlyTopTerms);
        EndOfRecord = FindStartOfNextTextRecord(Counter + 1);

        if(Counter < InputLines->Count)
        {
            String = InputLines->Strings[Counter++];

            LastField = MT;
            Rec->MT.Empty();
            if(RadioButton7->Checked)
            {
                if(Counter < InputLines->Count)
                {
                    Rec->MT = String;
```

```
                ReturnValue = true;
            }
        }
        else if(RadioButton6->Checked)
        {
            if(Counter < InputLines->Count - 1)
            {
                Rec->MT = InputLines->Strings[Counter++];
                ReturnValue = true;
            }
        }
        else if(RadioButton5->Checked)
        {
            if(Counter < InputLines->Count)
            {
                Rec->MT = String.SubString(TntEdit15->Text.Length() + 1, String.Length());
                ReturnValue = true;
            }
        }

        for(int i = Counter; i < EndOfRecord; ++i)
        {
            String = InputLines->Strings[Counter++];
            RemoveSpace((WideString *)&String);

            if(!IsIgnorableLine(String))
            {
                if(CheckBox1->Checked && IsStartedWith(String, TntEdit1->Text))//BT
                {
                    ADDWITHSEPARATOR(Rec->BT, String.SubString(TntEdit1->Text.Length() + 1, String.Length()));
                    ++BTCounter;
                    LastField = BT;
                }
                else if(CheckBox2->Checked && IsStartedWith(String, TntEdit2->Text))//NT
                {
                    ADDWITHSEPARATOR(Rec->NT, String.SubString(TntEdit2->Text.Length() + 1, String.Length()))
                    LastField = NT;
                }
                else if(CheckBox3->Checked && IsStartedWith(String, TntEdit3->Text))//RT
                {
                    ADDWITHSEPARATOR(Rec->RT, String.SubString(TntEdit3->Text.Length() + 1, String.Length()))
                    LastField = RT;
                }
                else if(CheckBox4->Checked && IsStartedWith(String, TntEdit4->Text))//USE
                {
                    ADDWITHSEPARATOR(Rec->USE, String.SubString(TntEdit4->Text.Length() + 1, String.Length()))
                    LastField = USE;
                }
                else if(CheckBox5->Checked && IsStartedWith(String, TntEdit5->Text))//UF
                {
                    ADDWITHSEPARATOR(Rec->UF, String.SubString(TntEdit5->Text.Length() + 1, String.Length()))
                    LastField = UF;
                }
```

```
else if(CheckBox6->Checked && IsStartedWith(String, TntEdit6->Text))//SN
{
     Rec->SN += String.SubString(TntEdit6->Text.Length() + 1, String.Length());
     LastField = SN;
}
else if(CheckBox7->Checked && IsStartedWith(String, TntEdit7->Text))//HN
{
     Rec->HN += String.SubString(TntEdit7->Text.Length() + 1, String.Length());
     LastField = HN;
}
else if(CheckBox8->Checked && IsStartedWith(String, TntEdit8->Text))//IND
{
     Rec->IND += String.SubString(TntEdit8->Text.Length() + 1, String.Length());
     LastField = IND;
}
else if(CheckBox9->Checked && IsStartedWith(String, TntEdit9->Text))//Frq
{
     Rec->Frq += String.SubString(TntEdit9->Text.Length() + 1, String.Length());
     LastField = Frq;
}
else if(CheckBox10->Checked && IsStartedWith(String, TntEdit10->Text))//SC
{
     Rec->SC += " " + String.SubString(TntEdit10->Text.Length() + 1, String.Length());
     LastField = SC;
}
else if(CheckBox11->Checked && IsStartedWith(String, TntEdit11->Text))//Translate 1
{
     Rec->T1 += String.SubString(TntEdit11->Text.Length() + 1, String.Length());
     LastField = T1;
}
else if(CheckBox12->Checked && IsStartedWith(String, TntEdit13->Text))//Translate 2
{
     Rec->T2 += String.SubString(TntEdit12->Text.Length() + 1, String.Length());
     LastField = T2;
}
else if(CheckBox13->Checked && IsStartedWith(String, TntEdit13->Text))//Translate 3
{
     Rec->T3 += String.SubString(TntEdit13->Text.Length() + 1, String.Length());
     LastField = T3;
}
else
{
     if(CheckBox1->Checked && LastField == BT)//BT
     {
          ADDWITHSEPARATOR(Rec->BT, String);
          ++BTCounter;
     }
     else if(CheckBox2->Checked && LastField == NT)//NT
          ADDWITHSEPARATOR(Rec->NT, String)
     else if(CheckBox3->Checked && LastField == RT)//RT
          ADDWITHSEPARATOR(Rec->RT, String)
     else if(CheckBox4->Checked && LastField == USE)//USE
          ADDWITHSEPARATOR(Rec->USE, String)
```

```
                    else if(CheckBox5->Checked && LastField == UF)//UF
                         ADDWITHSEPARATOR(Rec->UF, String)
                    else if(CheckBox10->Checked && LastField == SC)//SC
                         Rec->SC += " " + String;
               }
          }
     }

     if((OnlyTopTerms && !BTCounter) || !OnlyTopTerms || Counter >= InputLines->Count)
          Done = true;
   }
   else
     Done = true;
 }

 CountLines = Counter - 1;

 return ReturnValue;
}
//---------------------------------------------------------------------------

int __fastcall TForm1::FindStartOfNextTextRecord(int LineNumber, bool OnlyTopTerms)
{
 int Counter = LineNumber;
 int ReturnValue = InputLines->Count;

 if(Counter < 0)
   Counter = 0;

 if(Counter < InputLines->Count)
 {
   WideString String;

   String = InputLines->Strings[Counter];
   RemoveSpace((WideString *)&String);

   if(RadioButton7->Checked)
   {
     while((!String.IsEmpty() && Counter) && Counter < InputLines->Count - 1)
     {
       String = InputLines->Strings[++Counter];
       RemoveSpace((WideString *)&String);
     }
     while(String.IsEmpty() && Counter < InputLines->Count - 1)
     {
       String = InputLines->Strings[++Counter];
       RemoveSpace((WideString *)&String);
     }
     ReturnValue = Counter;
   }
   else if(RadioButton6->Checked)
   {
     while(WideCompareText(TntEdit14->Text, String) && Counter < InputLines->Count - 1)
```

```
                {
                    /*if(Counter >= InputLines->Count - 1)
                    {
                        ++Counter;
                        break;
                    }*/
                    String = InputLines->Strings[++Counter];
                    RemoveSpace((WideString *)&String);
                }
                ReturnValue = Counter;
            }
            else if(RadioButton5->Checked)
            {
                while(!IsStartedWith(String, TntEdit15->Text) && Counter < InputLines->Count - 1)
                {
                    String = InputLines->Strings[++Counter];
                    RemoveSpace((WideString *)&String);
                }
                ReturnValue = Counter;
            }
        }

        return ReturnValue;
}
//---------------------------------------------------------------------------

bool __fastcall TForm1::IsStartedWith(WideString String, WideString SearchString)
{
        if(AnsiStrPosICW(String, SearchString))
            return true;
        else
            return false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ResetRecord(Record *Rec)
{
        Rec->MT.Empty();
        Rec->BT.Empty();
        Rec->NT.Empty();
        Rec->RT.Empty();
        Rec->USE.Empty();
        Rec->UF.Empty();
        Rec->SN.Empty();
        Rec->HN.Empty();
        Rec->IND.Empty();
        Rec->Frq.Empty();
        Rec->SC.Empty();
        Rec->T1.Empty();
        Rec->T2.Empty();
        Rec->T3.Empty();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::SaveRecord(Fbtree *TreeFile, Record *Rec, bool AddToSub)
{
     WideString Buffer;
     char *FileBuffer;
     RecordHeader RecHeader;

     RemoveSpace((WideString *)&Rec->MT);
     RemoveSpace((WideString *)&Rec->BT);
     RemoveSpace((WideString *)&Rec->NT);
     RemoveSpace((WideString *)&Rec->RT);
     RemoveSpace((WideString *)&Rec->USE);
     RemoveSpace((WideString *)&Rec->UF);
     RemoveSpace((WideString *)&Rec->SN);
     RemoveSpace((WideString *)&Rec->HN);
     RemoveSpace((WideString *)&Rec->IND);
     RemoveSpace((WideString *)&Rec->Frq);
     RemoveSpace((WideString *)&Rec->SC);
     RemoveSpace((WideString *)&Rec->T1);
     RemoveSpace((WideString *)&Rec->T2);
     RemoveSpace((WideString *)&Rec->T3);

     RecHeader.MTLen = Rec->MT.Length();
     RecHeader.BTLen = Rec->BT.Length();
     RecHeader.NTLen = Rec->NT.Length();
     RecHeader.RTLen = Rec->RT.Length();
     RecHeader.USELen = Rec->USE.Length();
     RecHeader.UFLen = Rec->UF.Length();
     RecHeader.SNLen = Rec->SN.Length();
     RecHeader.HNLen = Rec->HN.Length();
     RecHeader.INDLen = Rec->IND.Length();
     RecHeader.FrqLen = Rec->Frq.Length();
     RecHeader.SCLen = Rec->SC.Length();
     RecHeader.T1Len = Rec->T1.Length();
     RecHeader.T2Len = Rec->T2.Length();
     RecHeader.T3Len = Rec->T3.Length();

     *RecHeader.MT = 0;
     if(!Rec->MT.IsEmpty())
          wcscpy(RecHeader.MT, Rec->MT);

     Buffer.Empty();
     Buffer.Insert(Rec->MT, Buffer.Length() + 1);
     Buffer.Insert(Rec->BT, Buffer.Length() + 1);
     Buffer.Insert(Rec->NT, Buffer.Length() + 1);
     Buffer.Insert(Rec->RT, Buffer.Length() + 1);
     Buffer.Insert(Rec->USE, Buffer.Length() + 1);
     Buffer.Insert(Rec->UF, Buffer.Length() + 1);
     Buffer.Insert(Rec->SN, Buffer.Length() + 1);
     Buffer.Insert(Rec->HN, Buffer.Length() + 1);
     Buffer.Insert(Rec->IND, Buffer.Length() + 1);
     Buffer.Insert(Rec->Frq, Buffer.Length() + 1);
     Buffer.Insert(Rec->SC, Buffer.Length() + 1);
```

```
     Buffer.Insert(Rec->T1, Buffer.Length() + 1);
     Buffer.Insert(Rec->T2, Buffer.Length() + 1);
     Buffer.Insert(Rec->T3, Buffer.Length() + 1);

     FileBuffer = new char[Buffer.Length() * sizeof(wchar_t) + sizeof(RecordHeader)];
     memmove(FileBuffer, &RecHeader, sizeof(RecordHeader));
     memmove(FileBuffer + sizeof(RecordHeader), Buffer.c_bstr(), Buffer.Length() * sizeof(wchar_t));

     if(AddToSub)
          TreeFile->AddSubNode(FileBuffer, Buffer.Length() * sizeof(wchar_t) + sizeof(RecordHeader),
                               (char *)FileBuffer, 200);
     else
          TreeFile->AddNode(FileBuffer, Buffer.Length() * sizeof(wchar_t) + sizeof(RecordHeader),
                            (char *)FileBuffer, 200);

     delete[] FileBuffer;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::LoadRecord(Fbtree *TreeFile, Record *Rec)
{
     wchar_t *FileBuffer;
     wchar_t *Buffer;
     wchar_t *Pos;
     RecordHeader RecHeader;

     ResetRecord(Rec);
     FileBuffer = (wchar_t *)TreeFile->AllocGetCurrentNode(FileBuffer);

     memmove(&RecHeader, FileBuffer, sizeof(RecordHeader));
     (char *)Pos = (char *)FileBuffer + sizeof(RecordHeader);

     if(RecHeader.MTLen)
     {
          WideCopy((WideString *)&Rec->MT, Pos, RecHeader.MTLen);
          Pos += RecHeader.MTLen;
     }

     if(RecHeader.BTLen)
     {
          WideCopy((WideString *)&Rec->BT, Pos, RecHeader.BTLen);
          Pos += RecHeader.BTLen;
     }

     if(RecHeader.NTLen)
     {
          WideCopy((WideString *)&Rec->NT, Pos, RecHeader.NTLen);
          Pos += RecHeader.NTLen;
     }

     if(RecHeader.RTLen)
     {
          WideCopy((WideString *)&Rec->RT, Pos, RecHeader.RTLen);
```

```
        Pos += RecHeader.RTLen;
    }

    if(RecHeader.USELen)
    {
        WideCopy((WideString *)&Rec->USE, Pos, RecHeader.USELen);
        Pos += RecHeader.USELen;
    }

    if(RecHeader.UFLen)
    {
        WideCopy((WideString *)&Rec->UF, Pos, RecHeader.UFLen);
        Pos += RecHeader.UFLen;
    }

    if(RecHeader.SNLen)
    {
        WideCopy((WideString *)&Rec->SN, Pos, RecHeader.SNLen);
        Pos += RecHeader.SNLen;
    }

    if(RecHeader.HNLen)
    {
        WideCopy((WideString *)&Rec->HN, Pos, RecHeader.HNLen);
        Pos += RecHeader.HNLen;
    }

    if(RecHeader.INDLen)
    {
        WideCopy((WideString *)&Rec->IND, Pos, RecHeader.INDLen);
        Pos += RecHeader.INDLen;
    }

    if(RecHeader.FrqLen)
    {
        WideCopy((WideString *)&Rec->Frq, Pos, RecHeader.FrqLen);
        Pos += RecHeader.FrqLen;
    }


    if(RecHeader.SCLen)
    {
        WideCopy((WideString *)&Rec->SC, Pos, RecHeader.SCLen);
        Pos += RecHeader.SCLen;
    }

    if(RecHeader.T1Len)
    {
        WideCopy((WideString *)&Rec->T1, Pos, RecHeader.T1Len);
        Pos += RecHeader.T1Len;
    }

    if(RecHeader.T2Len)
```

```
     {
          WideCopy((WideString *)&Rec->T2, Pos, RecHeader.T2Len);
          Pos += RecHeader.T2Len;
     }

     if(RecHeader.T3Len)
     {
          WideCopy((WideString *)&Rec->T3, Pos, RecHeader.T3Len);
          Pos += RecHeader.T3Len;
     }

     delete[] FileBuffer;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::MakeThesaurusStructure(void)
{
     Tree->Go(TOP);
     Record Rec;
     void *Result = Tree->FreeSearch(1, NULL, NULL, NULL, NULL);

     while(Result && !LoopEncountered)
     {
          LoadRecord(Tree, &Rec);
          Level = 0;
          SearchForNarrowerTerms(Rec);

          Result = Tree->FreeSearch(1, NULL, NULL, NULL, NULL, NEXT);
     }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::SearchForNarrowerTerms(Record Rec)
{
     TWideStringList *NTList = new TWideStringList;
     int LastPosition = Tree->CurrentNodeAddress;
     Record NewRecord;
     WideString String;

     ++Level;

     if(Level > 200)
     {
          LoopEncountered = true;
          MessageDlg("The Term " + Rec.MT + " or one of its broader terms encountered an unlimited " \
                     "loop! Correct it and then try again.", mtError, TMsgDlgButtons() << mbOK, 0);
     }
     else
     {
          NTList->Text = Rec.NT;

          for(int i = 0; i < NTList->Count && !LoopEncountered; ++i)
          {
```

```
                String = NTList->Strings[i];
                RemoveSpace((WideString *)&String);
                Linear->Go(TOP);
                if(Linear->BSearch(String))
                {
                     LoadRecord(Linear, &NewRecord);
                     if(!i)
                          SaveRecord(Tree, &NewRecord, true);
                     else
                     {
                          Tree->Go(SUBTOP);
                          SaveRecord(Tree, &NewRecord);
                     }

                     SearchForNarrowerTerms(NewRecord);
                }
           }
     }

     Tree->Go(LastPosition);
     delete NTList;

     --Level;
}
//---------------------------------------------------------------------------

bool __fastcall TForm1::IsIgnorableLine(WideString String)
{
     WideString SubString;

     for(int i = 0; i < TntMemo1->Lines->Count; ++i)
     {
          SubString = TntMemo1->Lines->Strings[i];
          RemoveSpace((WideString *)&SubString);
          if(IsStartedWith(String, SubString))
               return true;
     }

     return false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::SendToMicrosoftAccess(void)
{
     AnsiString ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
                                   SaveDialog1->FileName + ";Persist Security Info=False";
     AnsiString Fields = "LineNum Integer, "\
                         "TermLevel Integer, "\
                         "Term Char(100), "\
                         "RT Memo, " \
                         "SN Memo, " \
                         "UF Memo, " \
                         "USE Memo, " \
```

```
                    "Translate1 Char(100), " \
                    "Translate2 Char(100), " \
                    "Translate3 Char(100), " \
                    "Frequency Integer, " \
                    "Description Memo, " \
                    "InputDate Char(30), " \
                    "SC Char(30), " \
                    "HN Char(150), " \
                    "Primary Key (LineNum)";

    Application->ProcessMessages();

    ADOQuery1->Close();
    ADOQuery1->ConnectionString = "";
    ADOTable1->Active = false;
    ADOTable1->ConnectionString = "";
    Application->ProcessMessages();

    DeleteFile(SaveDialog1->FileName);
    CreateAccessDatabase(SaveDialog1->FileName.c_str());
    Application->ProcessMessages();

    ADOQuery1->ConnectionString = ConnectionString;

    ADOQuery1->SQL->Clear();
    ADOQuery1->SQL->Add("Create Table Thesaurus (" + Fields + ");");
    ADOQuery1->ExecSQL();

    ADOQuery1->SQL->Clear();
    ADOQuery1->SQL->Add("Create Table NPTerms (" + Fields + ");");
    ADOQuery1->ExecSQL();

    ADOQuery1->Close();
    ADOQuery1->ConnectionString = "";
    ADOTable1->ConnectionString = ConnectionString;

    Application->ProcessMessages();
    Tree->Go(TOP);
    AccessCounter = AccessLevel = 0;

    if(!LoopEncountered)
        ExportToAccess();

    ADOTable1->ConnectionString = "";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ExportToAccess(void)
{
    void *Result;
    Record Rec;

    if(!AccessLevel)
```

```
        {
                ADOTable1->TableName = "Thesaurus";
                ADOTable1->Active = true;
        }

        ++AccessLevel;

        Tree->Go(SUBTOP);

        Result = Tree->FreeSearch(AccessLevel, NULL, NULL, NULL, NULL);

        while(Result)
        {
                LoadRecord(Tree, &Rec);
                ReplaceWithTranslation(&Rec);

                ADOTable1->Append();
                ADOTable1->FieldByName("LineNum")->AsInteger = ++AccessCounter;
                ADOTable1->FieldByName("TermLevel")->AsInteger = AccessLevel - 1;
                ADOTable1->FieldByName("Term")->Value = Rec.MT;
                ADOTable1->FieldByName("RT")->Value = Rec.RT;
                ADOTable1->FieldByName("SN")->Value = Rec.SN;
                ADOTable1->FieldByName("UF")->Value = Rec.UF;
                ADOTable1->FieldByName("USE")->Value = Rec.USE;
                ADOTable1->FieldByName("Translate1")->Value = Rec.T1;
                ADOTable1->FieldByName("Translate2")->Value = Rec.T2;
                ADOTable1->FieldByName("Translate3")->Value = Rec.T3;
                ADOTable1->FieldByName("InputDate")->Value = Rec.IND;
                ADOTable1->FieldByName("SC")->Value = Rec.SC;
                ADOTable1->FieldByName("HN")->Value = Rec.HN;
                ADOTable1->FieldByName("Frequency")->AsInteger = StringToInt(Rec.Frq);
                ADOTable1->Post();

                if(Tree->Go(SUB))
                        ExportToAccess();

                Result = Tree->FreeSearch(AccessLevel, NULL, NULL, NULL, NULL, NEXT);
                Application->ProcessMessages();
        }

        Tree->Go(MOTHER);
        --AccessLevel;

        if(!AccessLevel)
                ADOTable1->Active = false;
}
//---------------------------------------------------------------------------

WideString __fastcall TForm1::TranslationOf(WideString String)
{
        WideString ReturnValue(String);

        if(!String.IsEmpty())
```

```
     {
          Record Rec;
          WideString Translation;
          RemoveSpace((WideString *)&String);

          Linear->Go(TOP);
          if(Linear->BSearch(String))
          {
               LoadRecord(Linear, &Rec);

               if(RadioButton2->Checked)
                    Translation = Rec.T1;
               else if(RadioButton3->Checked)
                    Translation = Rec.T2;
               else if(RadioButton4->Checked)
                    Translation = Rec.T3;

               if(!Translation.IsEmpty())
                    ReturnValue = Translation;
          }
     }

     return ReturnValue;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ReplaceWithTranslation(Record *Rec)
{
     WideString MTBackup(Rec->MT);
     TWideStringList *List = new TWideStringList();

     if(!RadioButton1->Checked)
     {
          Rec->MT = TranslationOf(Rec->MT);
          if(WideCompareText(Rec->MT, MTBackup))
          {
               if(RadioButton2->Checked)
                    Rec->T1 = MTBackup;
               else if(RadioButton3->Checked)
                    Rec->T2 = MTBackup;
               else if(RadioButton4->Checked)
                    Rec->T3 = MTBackup;
          }

          List->Text = Rec->RT;
          for(int i = 0; i < List->Count; ++i)
               List->Strings[i] = TranslationOf(List->Strings[i]);
          Rec->RT = List->Text;

          List->Text = Rec->USE;
          for(int i = 0; i < List->Count; ++i)
               List->Strings[i] = TranslationOf(List->Strings[i]);
          Rec->USE = List->Text;
```

```
          List->Text = Rec->UF;
          for(int i = 0; i < List->Count; ++i)
               List->Strings[i] = TranslationOf(List->Strings[i]);
          Rec->UF = List->Text;
     }

     delete List;
}
//---------------------------------------------------------------------------
```

```
object Form1: TForm1
  Left = 292
  Top = 330
  BiDiMode = bdRightToLeft
  BorderIcons = [biSystemMenu, biMinimize]
  BorderStyle = bsSingle
  Caption = 'تبدیل متن اصطلاحنامه'
  ClientHeight = 375
  ClientWidth = 674
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Tahoma'
  Font.Style = []
  OldCreateOrder = False
  ParentBiDiMode = False
  Position = poScreenCenter
  PixelsPerInch = 96
  TextHeight = 13
  object Label1: TLabel
    Left = 70
    Top = 80
    Width = 164
    Height = 13
    Caption = 'نادیده گرفتن خطوط با نشانه‌های زیر:'
  end
  object GroupBox1: TGroupBox
    Left = 409
    Top = 8
    Width = 257
    Height = 345
    Caption = 'نشانه فیلدها'
    TabOrder = 0
    object CheckBox1: TCheckBox
      Left = 144
      Top = 24
      Width = 97
      Height = 17
      Caption = 'عنوان اعم'
      Checked = True
      Enabled = False
      State = cbChecked
      TabOrder = 0
    end
    object CheckBox2: TCheckBox
      Left = 144
      Top = 48
      Width = 97
      Height = 17
      Caption = 'عنوان اخص'
      Checked = True
      Enabled = False
```

```
    State = cbChecked
    TabOrder = 1
  end
  object CheckBox3: TCheckBox
    Left = 144
    Top = 72
    Width = 97
    Height = 17
    Caption = 'عنوان وابسته'
    Checked = True
    State = cbChecked
    TabOrder = 2
  end
  object CheckBox4: TCheckBox
    Left = 144
    Top = 96
    Width = 97
    Height = 17
    Caption = 'بکار برید'
    Checked = True
    State = cbChecked
    TabOrder = 3
  end
  object CheckBox5: TCheckBox
    Left = 144
    Top = 120
    Width = 97
    Height = 17
    Caption = 'مجاز'
    Checked = True
    State = cbChecked
    TabOrder = 4
  end
  object CheckBox6: TCheckBox
    Left = 144
    Top = 144
    Width = 97
    Height = 17
    Caption = 'یادداشت دامنه'
    Checked = True
    State = cbChecked
    TabOrder = 5
  end
  object CheckBox7: TCheckBox
    Left = 144
    Top = 168
    Width = 97
    Height = 17
    Caption = 'تاریخچه'
    Checked = True
    State = cbChecked
    TabOrder = 6
  end
```

```
  object CheckBox8: TCheckBox
    Left = 144
    Top = 192
    Width = 97
    Height = 17
    Caption = 'تاريخ ورود'
    Checked = True
    State = cbChecked
    TabOrder = 7
  end
  object CheckBox9: TCheckBox
    Left = 144
    Top = 216
    Width = 97
    Height = 17
    Caption = 'بسامد'
    Checked = True
    State = cbChecked
    TabOrder = 8
  end
  object CheckBox10: TCheckBox
    Left = 136
    Top = 240
    Width = 105
    Height = 17
    Caption = 'رشته موضوعي'
    Checked = True
    State = cbChecked
    TabOrder = 9
  end
  object CheckBox11: TCheckBox
    Left = 144
    Top = 264
    Width = 97
    Height = 17
    Caption = 'ترجمه ١'
    TabOrder = 10
  end
  object CheckBox12: TCheckBox
    Left = 144
    Top = 288
    Width = 97
    Height = 17
    Caption = 'ترجمه ٢'
    TabOrder = 11
  end
  object CheckBox13: TCheckBox
    Left = 144
    Top = 312
    Width = 97
    Height = 17
    Caption = 'ترجمه ٣'
    TabOrder = 12
```

```
    end
    object TntEdit1: TTntEdit
      Left = 16
      Top = 24
      Width = 121
      Height = 21
      TabOrder = 13
      Text = 'BT:'
    end
    object TntEdit2: TTntEdit
      Left = 16
      Top = 48
      Width = 121
      Height = 21
      TabOrder = 14
      Text = 'NT:'
    end
    object TntEdit3: TTntEdit
      Left = 16
      Top = 72
      Width = 121
      Height = 21
      TabOrder = 15
      Text = 'RT:'
    end
    object TntEdit4: TTntEdit
      Left = 16
      Top = 96
      Width = 121
      Height = 21
      TabOrder = 16
      Text = 'USE:'
    end
    object TntEdit5: TTntEdit
      Left = 16
      Top = 120
      Width = 121
      Height = 21
      TabOrder = 17
      Text = 'UF:'
    end
    object TntEdit6: TTntEdit
      Left = 16
      Top = 144
      Width = 121
      Height = 21
      TabOrder = 18
      Text = 'SN:'
    end
    object TntEdit7: TTntEdit
      Left = 16
      Top = 168
      Width = 121
```

```
      Height = 21
      TabOrder = 19
      Text = 'HN:'
    end
    object TntEdit8: TTntEdit
      Left = 16
      Top = 192
      Width = 121
      Height = 21
      TabOrder = 20
      Text = 'IND:'
    end
    object TntEdit9: TTntEdit
      Left = 16
      Top = 216
      Width = 121
      Height = 21
      TabOrder = 21
      Text = 'Frq:'
    end
    object TntEdit10: TTntEdit
      Left = 16
      Top = 240
      Width = 121
      Height = 21
      TabOrder = 22
      Text = 'SC:'
    end
    object TntEdit11: TTntEdit
      Left = 16
      Top = 264
      Width = 121
      Height = 21
      TabOrder = 23
    end
    object TntEdit13: TTntEdit
      Left = 16
      Top = 312
      Width = 121
      Height = 21
      TabOrder = 24
    end
    object TntEdit12: TTntEdit
      Left = 16
      Top = 288
      Width = 121
      Height = 21
      TabOrder = 25
    end
  end
  object LabeledEdit1: TLabeledEdit
    Left = 88
    Top = 16
```

```
    Width = 234
    Height = 21
    EditLabel.Width = 70
    EditLabel.Height = 13
    EditLabel.BiDiMode = bdRightToLeft
    EditLabel.Caption = 'نـام فـايـل ورودي '
    EditLabel.ParentBiDiMode = False
    LabelPosition = lpRight
    LabelSpacing = 10
    TabOrder = 1
  end
  object Button1: TButton
    Left = 7
    Top = 16
    Width = 75
    Height = 25
    Caption = 'ديسك...'
    TabOrder = 2
    OnClick = Button1Click
  end
  object GroupBox2: TGroupBox
    Left = 241
    Top = 80
    Width = 161
    Height = 153
    Caption = 'نـشانـه آغاز يـك ركـورد'
    TabOrder = 5
    object TntEdit14: TTntEdit
      Left = 8
      Top = 64
      Width = 121
      Height = 21
      TabOrder = 2
      Text = 'Record:'
    end
    object TntEdit15: TTntEdit
      Left = 8
      Top = 112
      Width = 121
      Height = 21
      TabOrder = 4
      Text = 'MT:'
    end
    object RadioButton5: TRadioButton
      Left = 16
      Top = 94
      Width = 137
      Height = 17
      Caption = 'نـشانـه واژه اصلي:'
      TabOrder = 3
    end
    object RadioButton6: TRadioButton
      Left = 40
```

```
      Top = 46
      Width = 113
      Height = 17
      Caption = 'خطي شامل:'
      TabOrder = 1
    end
    object RadioButton7: TRadioButton
      Left = 40
      Top = 22
      Width = 113
      Height = 17
      Caption = 'خط خالي'
      Checked = True
      TabOrder = 0
      TabStop = True
    end
  end
  object LabeledEdit2: TLabeledEdit
    Left = 88
    Top = 48
    Width = 234
    Height = 21
    EditLabel.Width = 73
    EditLabel.Height = 13
    EditLabel.BiDiMode = bdRightToLeft
    EditLabel.Caption = 'نام فايل خروجي'
    EditLabel.ParentBiDiMode = False
    LabelPosition = lpRight
    LabelSpacing = 3
    TabOrder = 3
  end
  object Button2: TButton
    Left = 7
    Top = 48
    Width = 75
    Height = 25
    Caption = 'ديسك...'
    TabOrder = 4
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 7
    Top = 328
    Width = 226
    Height = 25
    Caption = 'تبديل'
    TabOrder = 8
    OnClick = Button3Click
  end
  object TntMemo1: TTntMemo
    Left = 8
    Top = 96
    Width = 226
```

```
    Height = 225
    TabOrder = 7
  end
  object GroupBox3: TGroupBox
    Left = 241
    Top = 248
    Width = 161
    Height = 105
    Caption = 'جايگزيي زبان اصلي با'
    TabOrder = 6
    object RadioButton1: TRadioButton
      Left = 32
      Top = 24
      Width = 113
      Height = 17
      Caption = 'عدم جايگزيي'
      Checked = True
      TabOrder = 0
      TabStop = True
    end
    object RadioButton2: TRadioButton
      Left = 32
      Top = 40
      Width = 113
      Height = 17
      Caption = 'ترجمه ١'
      TabOrder = 1
    end
    object RadioButton3: TRadioButton
      Left = 32
      Top = 56
      Width = 113
      Height = 17
      Caption = 'ترجمه ٢'
      TabOrder = 2
    end
    object RadioButton4: TRadioButton
      Left = 32
      Top = 72
      Width = 113
      Height = 17
      Caption = 'ترجمه ٣'
      TabOrder = 3
    end
  end
  object StatusBar1: TStatusBar
    Left = 0
    Top = 356
    Width = 674
    Height = 19
    Panels = <>
    SimplePanel = True
    SimpleText = #9#9#9#9
```

```
  end
  object OpenDialog1: TOpenDialog
    DefaultExt = 'txt'
    Filter = 'Text files (*.txt)|*.txt|All files (*.*)|*.*'
    Options = [ofReadOnly, ofPathMustExist, ofFileMustExist, ofEnableSizing]
    Left = 197
    Top = 112
  end
  object SaveDialog1: TSaveDialog
    DefaultExt = 'mdb'
    Filter = 'MS-Access Database (*.mdb)|*.mdb'
    Options = [ofOverwritePrompt, ofHideReadOnly, ofPathMustExist, ofEnableSizing]
    Left = 165
    Top = 112
  end
  object ADOQuery1: TADOQuery
    CursorLocation = clUseServer
    Parameters = <>
    Left = 165
    Top = 144
  end
  object ADOTable1: TADOTable
    Left = 197
    Top = 144
  end
end
```

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

#ifndef ProcessH
#define ProcessH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include "TntStdCtrls.hpp"
#include <ExtCtrls.hpp>
#include <Dialogs.hpp>
#include "WFbtree.h"
#include <ADODB.hpp>
#include <DB.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------

#define ADDWITHSEPARATOR(a, b) {                                    \
                                    if(!(a).IsEmpty())              \
                                         (a) += "\r\n";             \
                                    (a) += (b);                     \
                               }

typedef enum {MT = 1, BT, NT, RT, USE, UF, SN, HN, IND, Frq, SC, T1, T2, T3} ThesaurusFields;

struct RecordHeader
{
     wchar_t MT[101];
     int MTLen;
     int BTLen;
     int NTLen;
     int RTLen;
     int USELen;
     int UFLen;
     int SNLen;
     int HNLen;
     int INDLen;
     int FrqLen;
     int SCLen;
     int T1Len;
     int T2Len;
     int T3Len;
};

struct Record
{
     WideString MT;
     WideString BT;
     WideString NT;
     WideString RT;
     WideString USE;
     WideString UF;
```

```
        WideString SN;
        WideString HN;
        WideString IND;
        WideString Frq;
        WideString SC;
        WideString T1;
        WideString T2;
        WideString T3;
};


class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TGroupBox *GroupBox1;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TCheckBox *CheckBox9;
        TCheckBox *CheckBox10;
        TCheckBox *CheckBox11;
        TCheckBox *CheckBox12;
        TCheckBox *CheckBox13;
        TTntEdit *TntEdit1;
        TTntEdit *TntEdit2;
        TTntEdit *TntEdit3;
        TTntEdit *TntEdit4;
        TTntEdit *TntEdit5;
        TTntEdit *TntEdit6;
        TTntEdit *TntEdit7;
        TTntEdit *TntEdit8;
        TTntEdit *TntEdit9;
        TTntEdit *TntEdit10;
        TTntEdit *TntEdit11;
        TTntEdit *TntEdit13;
        TLabeledEdit *LabeledEdit1;
        TButton *Button1;
        TGroupBox *GroupBox2;
        TTntEdit *TntEdit14;
        TTntEdit *TntEdit15;
        TLabeledEdit *LabeledEdit2;
        TButton *Button2;
        TOpenDialog *OpenDialog1;
        TSaveDialog *SaveDialog1;
        TTntEdit *TntEdit12;
        TButton *Button3;
        TTntMemo *TntMemo1;
        TLabel *Label1;
```

```
    TGroupBox *GroupBox3;
    TRadioButton *RadioButton1;
    TRadioButton *RadioButton2;
    TRadioButton *RadioButton3;
    TRadioButton *RadioButton4;
    TADOQuery *ADOQuery1;
    TADOTable *ADOTable1;
    TRadioButton *RadioButton5;
    TRadioButton *RadioButton6;
    TRadioButton *RadioButton7;
    TStatusBar *StatusBar1;
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
    void __fastcall Button3Click(TObject *Sender);
private:  // User declarations
    TWideStringList *InputLines;
    int CountLines;
    Fbtree *Tree;
    Fbtree *Linear;
    int Level;
    int AccessLevel;
    int AccessCounter;
    bool LoopEncountered;

    void __fastcall FindTopTerms(void);
    void __fastcall FindAllTerms(void);
    void __fastcall MakeThesaurusStructure(void);
    bool __fastcall FindTextRecord(Record *Rec, int LineNumber = 0, bool OnlyTopTerms = false);
    int  __fastcall FindStartOfNextTextRecord(int LineNumber = 0, bool OnlyTopTerms = false);
    bool __fastcall IsStartedWith(WideString String, WideString SearchString);
    void __fastcall ResetRecord(Record *Rec);
    void __fastcall SaveRecord(Fbtree *TreeFile, Record *Rec, bool AddToSub = false);
    void __fastcall LoadRecord(Fbtree *TreeFile, Record *Rec);
    void __fastcall SearchForNarrowerTerms(Record Rec);
    bool __fastcall IsIgnorableLine(WideString String);
    void __fastcall SendToMicrosoftAccess(void);
    void __fastcall ExportToAccess(void);
    void __fastcall ReplaceWithTranslation(Record *Rec);
    WideString __fastcall TranslationOf(WideString String);

public:         // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

```
#include <vcl.h>
#pragma hdrstop

#include "stdio.h"
#include "WFBtree.h"

#pragma package(smart_init)
//---------------------------------------------------------------------------

struct DateSpec
{
     int year;
     char month, day;
     char dayofweek;
};

char fbtree_currentindex_sign;
AnsiString ExecutionPath;
//---------------------------------------------------------------------------

long date_unique_number(DateSpec *date)
{
     return (long)date->year * 372 + (date->month - 1) * 31 + (date->day - 1);
}
//---------------------------------------------------------------------------

bool __fastcall _DeleteFile(const AnsiString FileName, bool UseExecutionPath)
{
     AnsiString FileNamePath = FileName;
     bool returnval;

     if(UseExecutionPath)
          FileNamePath.Insert(ExecutionPath, 1);

     if(FileExists(FileNamePath))
          returnval = DeleteFile(FileNamePath);
     else
          returnval = true;

     return returnval;
}
//---------------------------------------------------------------------------

bool __fastcall _FileExists(const AnsiString FileName, bool UseExecutionPath)
{
     AnsiString FileNamePath = FileName;
     bool returnval;

     FileNamePath.Insert(ExecutionPath, 1);
     returnval = FileExists(FileNamePath);

     return returnval;
}
```

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

Fbtree::Fbtree(char *Filename, bool change_header, bool casesens, int (*comp)(WideString s1, WideString s2, unsigned len), FbtreeIndex
 *f_index, bool use_execpath, bool OpenIndexes)
{
     register i;

     UseExecutionPath = use_execpath;
     ExecutionPath = UseExecutionPath ? (GetExecutionPath() + "\\") : String("");

     record_preview.LeftAddress = record_preview.RightAddress = record_preview.SubNodeAddress = record_preview.MotherNodeAddress =
 record_preview.PreviousNodeAddress = NULL;
     CaseSensitive = casesens;

     for(register i = 0; i <= MAXFBTREELEVEL; i++)
     {
          LastSearchedAddress[i] = NULL;
          Stack[i] = new int[1];
          StackPosition[i] = 0;
     }

     CompareFunction = comp;
     OpenAllIndexes = OpenIndexes;

     FbtreeHeader head = {"HUF Data File", 26, "ʕœ'’–ɕ", "ʕœ'’–ɕ", 0, {0}, 0.0, {0}, {0}};

     if(change_header == YES)
          *head.Security1 = *head.Security2 = 0;

     Header = head;

     StrCopy(FbtreeFilename, Filename);

     OptimizeActive = NO;
     CurrentFileLevel = 1;
     OpenIndexFlag = 0;

     Found = _FileExists(Filename, UseExecutionPath);

     if(Found)
          Mode = fmOpenReadWrite;
     else
          Mode = fmCreate;

     File = new TFileStream(ExecutionPath + Filename, Mode);

     if(File->Handle > 0)
     {
          if(Found == false)
               WriteHeader();
     }

     for(i = 0; i < MAXFBTREEINDEX; i++)
```

```
          Index[i] = NULL;

     fbtree_index = NULL;
     IndexCounter = 0;
     if(f_index)
     {
          while(f_index[IndexCounter].Filename && IndexCounter < MAXFBTREEINDEX)
               IndexCounter++;

          if(IndexCounter)
               fbtree_index = new FbtreeIndex[IndexCounter + 1];

          for(i = 0; i < IndexCounter; i++)
          {
               fbtree_index[i].Filename = new char[strlen(f_index[i].Filename) + 1];
               StrCopy(fbtree_index[i].Filename, f_index[i].Filename);

               fbtree_index[i].Level = f_index[i].Level;
               fbtree_index[i].StartKey = f_index[i].StartKey;
               fbtree_index[i].Len = f_index[i].Len;
               fbtree_index[i].Comp = f_index[i].Comp;
               fbtree_index[i].CompareKind = f_index[i].Comp ? fbtFREECOMPARE : f_index[i].CompareKind;
               fbtree_index[i].Sign = f_index[i].Sign;
          }
          fbtree_index[IndexCounter].Filename = NULL;
     }

     if(File->Handle > 0)
     {
          ReadHeader();
          GetNextNewNodeAddress();
          CurrentNodeAddress = Header.FirstNodeAddress;
     }

     if(OpenAllIndexes)
          OpenIndexFiles();

     ProgressBar = NULL;
}
//---------------------------------------------------------------------------

Fbtree::~Fbtree(void)
{
     CloseIndexFiles(true);

     if(IndexCounter)
     {
          for(register i = IndexCounter - 1; i >= 0; i--)
               delete[] fbtree_index[i].Filename;

          delete[] fbtree_index;
     }
```

```
     delete File;

     for(register i = MAXFBTREELEVEL; i >= 0; i--)
          if(Stack[i])
               delete[] Stack[i];
}
//---------------------------------------------------------------------------

void Fbtree::GetNextNewNodeAddress(void)
{
     NewNodeAddress = File->Size;
}
//---------------------------------------------------------------------------

void Fbtree::WriteNewRecord(void *nodestruct)
{
     if(File->Handle > 0)
     {
          File->Seek(NewNodeAddress, soFromBeginning);
          File->Write(&record_preview, sizeof(RecordPreview));
          File->Write(nodestruct, record_preview.RecordLen);
          CurrentNodeAddress = NewNodeAddress;

          GetNextNewNodeAddress();
     }
}
//---------------------------------------------------------------------------

void Fbtree::WriteHeader(char *source, int start, int size)
{
     if(File->Handle > 0)
     {
          if(source)
          {
               char *ptr = (char *)&Header;

               ptr += start;

               for(register i = 0; i < size; i++)
                    *ptr++ = *(source + i);
          }

          File->Seek(0, soFromBeginning);
          File->Write(&Header, sizeof(FbtreeHeader));
     }
}
//---------------------------------------------------------------------------

void Fbtree::ReadHeader(char *dest, int start, int size)
{
     if(File->Handle > 0)
     {
          File->Seek(0, soFromBeginning);
```

```
          File->Read(&Header, sizeof(FbtreeHeader));

          if(dest)
          {
               char *ptr = (char *)&Header;

               ptr += start;

               for(register i = 0; i < size; i++)
                    *dest++ = *(ptr + i);
          }
     }
}
//---------------------------------------------------------------------------

void Fbtree::ReadRecord(void *nodestruct, long nodeaddress)
{
     if(File->Handle > 0)
     {
          if(nodeaddress == -1)
               nodeaddress = CurrentNodeAddress;

          File->Seek(nodeaddress, soFromBeginning);
          File->Read(&record_preview, sizeof(RecordPreview));
          File->Read(nodestruct, record_preview.RecordLen);
     }
}
//---------------------------------------------------------------------------

void *Fbtree::AllocGetCurrentNode(void *nodestruct)
{
     return nodestruct = (char *)AllocReadRecord(nodestruct, CurrentNodeAddress);
}
//---------------------------------------------------------------------------

void *Fbtree::AllocReadRecord(void *nodestruct, long nodeaddress)
{
     void *ReturnValue = NULL;

     if(File->Handle > 0)
     {
          if(nodeaddress == -1)
               nodeaddress = CurrentNodeAddress;

          File->Seek(nodeaddress, soFromBeginning);
          File->Read(&record_preview, sizeof(RecordPreview));

          ReturnValue = nodestruct = new char[record_preview.RecordLen + 2];

          if(nodestruct)
               File->Read(nodestruct, record_preview.RecordLen);
          *((char *)nodestruct + record_preview.RecordLen) = 0;
     }
```

```
     return ReturnValue;
}
//---------------------------------------------------------------------------

int Fbtree::Compare(void *comp1, void *comp2, unsigned size, char CompareKind)
{
     int ReturnValue;

     if(CompareKind == fbtFREECOMPARE)
          if(CompareFunction)
               ReturnValue = CompareFunction((wchar_t *)comp1, (wchar_t *)comp2, size);
          else
               CompareKind = fbtSTRING;

     if(CompareKind != fbtFREECOMPARE)
          switch(CompareKind)
          {
               case fbtSTRING:
                    if(CaseSensitive)
                         ReturnValue = wcsncmp((wchar_t *)comp1, (wchar_t *)comp2, size);
                    else
                         ReturnValue = _wcsnicmp((wchar_t *)comp1, (wchar_t *)comp2, size);
                    break;

               case fbtCHAR:
                    ReturnValue = *(wchar_t *)comp1 > *(wchar_t *)comp2 ? 1 : *(wchar_t *)comp1 == *(wchar_t *)comp2 ? 0 : -1;
                    break;

               case fbtINTEGER:
                    ReturnValue = *(int *)comp1 > *(int *)comp2 ? 1 : *(int *)comp1 == *(int *)comp2 ? 0 : -1;
                    break;

               case fbtLONG:
                    ReturnValue = *(long *)comp1 > *(long *)comp2 ? 1 : *(long *)comp1 == *(long *)comp2 ? 0 : -1;
                    break;

               case fbtFLOAT:
                    ReturnValue = *(float *)comp1 > *(float *)comp2 ? 1 : *(float *)comp1 == *(float *)comp2 ? 0 : -1;
                    break;

               case fbtDOUBLE:
                    ReturnValue = *(double *)comp1 > *(double *)comp2 ? 1 : *(double *)comp1 == *(double *)comp2 ? 0 : -1;
                    break;

               case fbtDATE:
                    ReturnValue = date_unique_number((DateSpec *)comp1) > date_unique_number((DateSpec *)comp2) ? 1 :
 date_unique_number((DateSpec *)comp1) == date_unique_number((DateSpec*)comp2) ? 0 : -1;
                    break;
          }

     return ReturnValue;
}
```

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

char Fbtree::AddNode(void *nodestruct, int nodestructsize, void *startnodekey, int keysize, char CompareKind, int nodeaddress)
{
     wchar_t *readnode = NULL, *readnodebak;
     char ReturnValue = 0;
     int comparevalue;

     if(File->Handle > 0)
     {
          OpenIndexFiles();

          record_preview.RecordLen = (unsigned)nodestructsize;
          record_preview.KeyStartAddress = (unsigned)((char *)startnodekey - (char *)nodestruct);
          record_preview.KeyLen = (unsigned)keysize;
          record_preview.CompareKind = CompareKind;

          if(nodeaddress > 0)
          {
               File->Seek(nodeaddress, soFromBeginning);
               CurrentNodeAddress = nodeaddress;
          }

          if(!Header.FirstNodeAddress)
          {
               record_preview.LeftAddress =
               record_preview.RightAddress =
               record_preview.SubNodeAddress =
               record_preview.MotherNodeAddress =
               record_preview.PreviousNodeAddress = 0;

               Header.FirstNodeAddress = CurrentNodeAddress = NewNodeAddress;
               WriteHeader();
               WriteNewRecord(nodestruct);

               WriteIndex(nodestruct);
          }
          else
          {
               PreviousNodeAddress = CurrentNodeAddress;
               if(!nodeaddress)
               {
                    record_preview.LeftAddress = record_preview.RightAddress = record_preview.SubNodeAddress = NULL;
                    record_preview.PreviousNodeAddress = PreviousNodeAddress;
                    WriteNewRecord(nodestruct);
                    WriteIndex(nodestruct);

                    File->Seek(PreviousNodeAddress, soFromBeginning);
                    File->Read(&record_preview, sizeof(RecordPreview));

                    switch(PreviousPath)
                    {
                         case LEFT:
```

```
                              record_preview.LeftAddress = CurrentNodeAddress;
                              break;

                         case RIGHT:
                              record_preview.RightAddress = CurrentNodeAddress;
                              break;
                    }

                    File->Seek(PreviousNodeAddress, soFromBeginning);
                    File->Write(&record_preview, sizeof(RecordPreview));

                    ReturnValue = 1;
               }
               else
               {
                    readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);
                    readnodebak = readnode;
                    (char *)readnode += (unsigned)((char *)startnodekey - (char *)nodestruct);

                    comparevalue = Compare(startnodekey, readnode, keysize, CompareKind);

                    if(readnode)
                    {
                         readnode = readnodebak;
                         delete[] readnode;
                    }

                    if(comparevalue <= 0)
                    {
                         PreviousPath = LEFT;
                         ReturnValue = AddNode(nodestruct, nodestructsize, startnodekey, keysize, CompareKind, record_preview.LeftAddress);
                    }
                    else
                    {
                         PreviousPath = RIGHT;
                         ReturnValue = AddNode(nodestruct, nodestructsize, startnodekey, keysize, CompareKind, record_preview.RightAddress);
                    }
               }
          }

          CloseIndexFiles();
     }

     return ReturnValue;
}
//---------------------------------------------------------------------------

char Fbtree::AddSubNode(void *nodestruct, int nodestructsize, void *startnodekey, int keysize, char CompareKind, int nodeaddress)
{
     wchar_t *readnode = new wchar_t[nodestructsize + 1], *readnodebak = readnode;
     char ReturnValue = 0;

     if(readnode && File->Handle > 0)
```

```
    {
        if(Header.FirstNodeAddress)
        {
            CurrentFileLevel++;

            OpenIndexFiles();

            File->Seek(CurrentNodeAddress, soFromBeginning);
            File->Read(&record_preview, sizeof(RecordPreview));

            if(record_preview.SubNodeAddress)
                ReturnValue = AddNode(nodestruct, nodestructsize, startnodekey, keysize, CompareKind, record_preview.SubNodeAddress);
            else
            {
                ReturnValue = 1;

                record_preview.RecordLen = (unsigned)nodestructsize;
                record_preview.KeyStartAddress = (unsigned)((char *)startnodekey - (char *)nodestruct);
                record_preview.KeyLen = (unsigned)keysize;
                record_preview.CompareKind = CompareKind;

                if(nodeaddress > 0)
                {
                    File->Seek(nodeaddress, soFromBeginning);
                    CurrentNodeAddress = nodeaddress;
                }

                PreviousNodeAddress = CurrentNodeAddress;

                record_preview.LeftAddress = record_preview.RightAddress = record_preview.SubNodeAddress =
record_preview.PreviousNodeAddress = NULL;
                record_preview.MotherNodeAddress = PreviousNodeAddress;
                WriteNewRecord(nodestruct);
                WriteIndex(nodestruct);

                File->Seek(PreviousNodeAddress, soFromBeginning);
                File->Read(&record_preview, sizeof(RecordPreview));

                record_preview.SubNodeAddress = CurrentNodeAddress;

                File->Seek(PreviousNodeAddress, soFromBeginning);
                File->Write(&record_preview, sizeof(RecordPreview));

            }

            CloseIndexFiles();
        }
        else
            ReturnValue = AddNode(nodestruct, nodestructsize, startnodekey, keysize, CompareKind, nodeaddress);
    }

    if(readnode)
    {
```

```
          readnode = readnodebak;
          delete[] readnode;
     }

     return ReturnValue;
}
//---------------------------------------------------------------------------

char Fbtree::GetCurrentNode(void *nodestruct)
{
     char ReturnValue = 0;

     if(Header.FirstNodeAddress)
     {
          ReadRecord(nodestruct);
          ReturnValue = 1;
     }

     return ReturnValue;
}
//---------------------------------------------------------------------------

void *Fbtree::BSearch(void *searchstring, int startaddress)
{
     void *ReturnValue = NULL;
     int comparevalue;

     if(Header.FirstNodeAddress && File->Handle > 0)
     {
          if(startaddress == -1)
          {
               startaddress = CurrentNodeAddress;
          }

          wchar_t *readnode = NULL, *readnodebak;
          readnode = (wchar_t *)AllocReadRecord(readnode, startaddress);
          readnodebak = readnode;
          (char *)readnode += record_preview.KeyStartAddress;

          comparevalue = Compare(searchstring, readnode, record_preview.KeyLen, record_preview.CompareKind);

          if(!comparevalue)
          {
               CurrentNodeAddress = startaddress;
               ReturnValue = readnodebak;
          }
          else
          {
               readnode = readnodebak;
               delete[] readnode;
               readnode = NULL;

               if(comparevalue > 0)
```

```
                {
                    if(record_preview.RightAddress)
                        ReturnValue = BSearch(searchstring, record_preview.RightAddress);
                }
                else
                {
                    if(record_preview.LeftAddress)
                        ReturnValue = BSearch(searchstring, record_preview.LeftAddress);
                }
            }

            if(readnode)
            {
                readnode = readnodebak;
                delete[] readnode;
            }
        }

        return ReturnValue;
}
//---------------------------------------------------------------------------

void *Fbtree::FreeSearch(int searchlevel, void *searchstring, void *nodestruct, void *compareaddress, unsigned comparelen, char firstornext,
 char CompareKind)
{
        searchlevel--;

        void *ReturnValue = NULL;
        int  comparevalue;
        bool done = false;
        char first_next = firstornext;
        int PopAddress, CurrentNodeAddressbak = CurrentNodeAddress;
        RecordPreview rpsave;

        if(File->Handle > 0 && Header.FirstNodeAddress && (firstornext == FIRST || (firstornext == NEXT && LastSearchedAddress[searchlevel])))
        {
            if(firstornext == NEXT)
                Go(LastSearchedAddress[searchlevel]);
            else
                StackPosition[searchlevel] = 0;

            File->Seek(CurrentNodeAddress, soFromBeginning);
            File->Read(&record_preview, sizeof(RecordPreview));

            wchar_t *readnode = NULL, *readnodebak;

            readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);
            readnodebak = readnode;

            rpsave = record_preview;

            while(!done && readnode)
            {
```

```
if(rpsave.LeftAddress && first_next == FIRST)
{
     if(!Push(CurrentNodeAddress, searchlevel))
          break;
     Go(LEFT);
     readnode = readnodebak;
     delete[] readnode;
     readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);
     readnodebak = readnode;
     rpsave = record_preview;
}
else
{
     if(first_next == FIRST)
     {
          if(searchstring)
          {
               (char *)readnode += (unsigned)((char *)compareaddress - (char *)nodestruct);
               comparevalue = Compare(searchstring, readnode, comparelen, CompareKind);
          }
          else
               comparevalue = 0;

          if(!comparevalue)
          {
               ReturnValue = readnodebak;
               LastSearchedAddress[searchlevel] = CurrentNodeAddress;
               break;
          }
          else
          {
               if(Go(RIGHT))
               {
                    readnode = readnodebak;
                    delete readnode;
                    readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);
                    readnodebak = readnode;
                    rpsave = record_preview;
               }
               else
               {
                    PopAddress = Pop(searchlevel);

                    if(!PopAddress)
                         break;

                    Go(PopAddress);
                    readnode = readnodebak;
                    delete[] readnode;
                    readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);
                    readnodebak = readnode;
                    rpsave = record_preview;
                    rpsave.LeftAddress = 0;
```

```
                    }
                }
            }
            else
            {
                first_next = FIRST;

                if(Go(RIGHT))
                {
                    readnode = readnodebak;
                    delete[] readnode;
                    readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);
                    readnodebak = readnode;
                    rpsave = record_preview;
                }
                else
                {
                    PopAddress = Pop(searchlevel);

                    if(!PopAddress)
                        break;

                    Go(PopAddress);
                    readnode = readnodebak;
                    delete[] readnode;
                    readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);
                    readnodebak = readnode;
                    rpsave = record_preview;
                    rpsave.LeftAddress = 0;
                }
            }
        }
    }

    if(readnode)
    {
        readnode = readnodebak;
        delete[] readnode;
    }
}

if(!ReturnValue)
    CurrentNodeAddress = CurrentNodeAddressbak;

return ReturnValue;
}
//---------------------------------------------------------------------------

int Fbtree::Push(int address, int searchlevel)
{
    int ReturnValue = 1;

    StackPosition[searchlevel]++;
```

```
     StackBak = new int[StackPosition[searchlevel] + 1];
     if(!StackBak)
          ReturnValue = 0;
     else
     {
          for(register i = 0; i < StackPosition[searchlevel] - 1; i++)
               StackBak[i] = Stack[searchlevel][i];

          delete[] Stack[searchlevel];
          Stack[searchlevel] = StackBak;

          StackBak[StackPosition[searchlevel] - 1] = address;
     }

     return ReturnValue;
}
//---------------------------------------------------------------------------

int Fbtree::Pop(int searchlevel)
{
     int ReturnValue = 0;
     if(StackPosition[searchlevel])
          ReturnValue = Stack[searchlevel][--StackPosition[searchlevel]];

     StackBak = new int[StackPosition[searchlevel] + 1];
     if(!StackBak)
          ReturnValue = 0;
     else
     {
          for(register i = 0; i < StackPosition[searchlevel]; i++)
               StackBak[i] = Stack[searchlevel][i];

          delete[] Stack[searchlevel];
          Stack[searchlevel] = StackBak;
     }

     return ReturnValue;
}
//---------------------------------------------------------------------------

char Fbtree::Go(int where)
{
     char ReturnValue = 0;
     int sw = where < 41 ? (int)where : ADDRESS;

     if(Header.FirstNodeAddress && File->Handle > 0)
     {
          File->Seek(CurrentNodeAddress, soFromBeginning);
          File->Read(&record_preview, sizeof(RecordPreview));

          switch(sw)
          {
```

```
case LEFT:
     if(record_preview.LeftAddress)
     {
          ReturnValue = 1;
          CurrentNodeAddress = record_preview.LeftAddress;
     }
     break;

case RIGHT:
     if(record_preview.RightAddress)
     {
          ReturnValue = 1;
          CurrentNodeAddress = record_preview.RightAddress;
     }
     break;

case SUB:
     if(record_preview.SubNodeAddress)
     {
          ReturnValue = 1;
          CurrentNodeAddress = record_preview.SubNodeAddress;
          CurrentFileLevel++;
     }
     break;

case PREVIOUS:
     if(record_preview.PreviousNodeAddress)
     {
          ReturnValue = 1;
          CurrentNodeAddress = record_preview.PreviousNodeAddress;
     }
     break;

case MOTHER:
     if(record_preview.MotherNodeAddress)
     {
          ReturnValue = 1;
          CurrentNodeAddress = record_preview.MotherNodeAddress;
          CurrentFileLevel--;
     }
     break;

case TOP:
     while(record_preview.MotherNodeAddress)
     {
          ReturnValue = 1;
          Go(MOTHER);
     }

     Go(SUBTOP);
     CurrentFileLevel = 1;
     break;
```

```
                case SUBTOP:
                    while(record_preview.PreviousNodeAddress)
                    {
                        ReturnValue = 1;
                        Go(PREVIOUS);
                    }
                    break;

                case ADDRESS:
                    if(where)
                    {
                        CurrentNodeAddress = where;

                        if(IndexCounter)
                        {
                            int counter = 1;

                            while(Go(MOTHER))
                                counter++;

                            CurrentNodeAddress = where;
                            CurrentFileLevel = counter;
                        }

                        ReturnValue = 1;
                    }
                    break;
            }
        }

    return ReturnValue;
}
//---------------------------------------------------------------------------

char Fbtree::DeleteNode(void)
{
    RecordPreview rpsave;
    int savenodeaddress = CurrentNodeAddress;
    int othernodeaddress = 0;
    char ReturnValue = 0;

    if(Header.FirstNodeAddress && File->Handle > 0)
    {
        ReturnValue = PREVIOUS;
        OpenIndexFiles();

        File->Seek(CurrentNodeAddress, soFromBeginning);
        File->Read(&rpsave, sizeof(RecordPreview));

        void *buffer = AllocReadRecord(buffer, CurrentNodeAddress);
        RemoveAddressIndex(CurrentNodeAddress);
        delete[] buffer;
```

```
        if(Go(PREVIOUS))
        {
             File->Seek(CurrentNodeAddress, soFromBeginning);
             File->Read(&record_preview, sizeof(RecordPreview));

             if(!rpsave.LeftAddress || !rpsave.RightAddress)
             {
                  if(!rpsave.LeftAddress)
                  {
                       if(record_preview.LeftAddress == savenodeaddress)
                            record_preview.LeftAddress = rpsave.RightAddress;
                       else
                            record_preview.RightAddress = rpsave.RightAddress;

                       File->Seek(CurrentNodeAddress, soFromBeginning);
                       File->Write(&record_preview, sizeof(RecordPreview));

                       if(rpsave.RightAddress)
                            othernodeaddress = rpsave.RightAddress;
                  }
                  else
                  {
                       if(record_preview.LeftAddress == savenodeaddress)
                            record_preview.LeftAddress = rpsave.LeftAddress;
                       else
                            record_preview.RightAddress = rpsave.LeftAddress;

                       File->Seek(CurrentNodeAddress, soFromBeginning);
                       File->Write(&record_preview, sizeof(RecordPreview));

                       if(rpsave.LeftAddress)
                            othernodeaddress = rpsave.LeftAddress;
                  }

                  if(othernodeaddress)
                  {
                       File->Seek(othernodeaddress, soFromBeginning);
                       File->Read(&record_preview, sizeof(RecordPreview));
                       record_preview.PreviousNodeAddress = rpsave.PreviousNodeAddress;
                       File->Seek(othernodeaddress, soFromBeginning);
                       File->Write(&record_preview, sizeof(RecordPreview));
                  }
             }
             else
             {
                  if(record_preview.LeftAddress == savenodeaddress)
                       record_preview.LeftAddress = rpsave.LeftAddress;
                  else
                       record_preview.RightAddress = rpsave.LeftAddress;

                  File->Seek(CurrentNodeAddress, soFromBeginning);
                  File->Write(&record_preview, sizeof(RecordPreview));
```

```
                File->Seek(rpsave.LeftAddress, soFromBeginning);
                File->Read(&record_preview, sizeof(RecordPreview));
                record_preview.PreviousNodeAddress = rpsave.PreviousNodeAddress;
                File->Seek(rpsave.LeftAddress, soFromBeginning);
                File->Write(&record_preview, sizeof(RecordPreview));

                CurrentNodeAddress = rpsave.LeftAddress;
                while(Go(RIGHT));
                File->Seek(CurrentNodeAddress, soFromBeginning);
                File->Read(&record_preview, sizeof(RecordPreview));

                record_preview.RightAddress = rpsave.RightAddress;
                File->Seek(CurrentNodeAddress, soFromBeginning);
                File->Write(&record_preview, sizeof(RecordPreview));

                savenodeaddress = CurrentNodeAddress;

                File->Seek(rpsave.RightAddress, soFromBeginning);
                File->Read(&record_preview, sizeof(RecordPreview));
                record_preview.PreviousNodeAddress = savenodeaddress;
                File->Seek(rpsave.RightAddress, soFromBeginning);
                File->Write(&record_preview, sizeof(RecordPreview));
           }

           CurrentNodeAddress = rpsave.PreviousNodeAddress;
      }
      else
      {
           if(rpsave.LeftAddress || rpsave.RightAddress)
           {
                if(!rpsave.LeftAddress || !rpsave.RightAddress)
                {
                     if(!rpsave.LeftAddress)
                     {
                          Go(RIGHT);
                          ReturnValue = RIGHT;
                     }
                     else
                     {
                          Go(LEFT);
                          ReturnValue = LEFT;
                     }

                     File->Seek(CurrentNodeAddress, soFromBeginning);
                     File->Read(&record_preview, sizeof(RecordPreview));
                     record_preview.PreviousNodeAddress = 0;
                     record_preview.MotherNodeAddress = rpsave.MotherNodeAddress;
                     File->Seek(CurrentNodeAddress, soFromBeginning);
                     File->Write(&record_preview, sizeof(RecordPreview));

                     savenodeaddress = CurrentNodeAddress;

                     if(Go(MOTHER))
```

```
                {
                    File->Seek(CurrentNodeAddress, soFromBeginning);
                    File->Read(&record_preview, sizeof(RecordPreview));
                    record_preview.SubNodeAddress = savenodeaddress;
                    File->Seek(CurrentNodeAddress, soFromBeginning);
                    File->Write(&record_preview, sizeof(RecordPreview));
                }
                else
                {
                    Header.FirstNodeAddress = savenodeaddress;
                    WriteHeader();
                }

                CurrentNodeAddress = savenodeaddress;
            }
            else
            {
                if(Go(MOTHER))
                {
                    File->Seek(CurrentNodeAddress, soFromBeginning);
                    File->Read(&record_preview, sizeof(RecordPreview));
                    record_preview.SubNodeAddress = rpsave.LeftAddress;
                    File->Seek(CurrentNodeAddress, soFromBeginning);
                    File->Write(&record_preview, sizeof(RecordPreview));
                }
                else
                {
                    Header.FirstNodeAddress = rpsave.LeftAddress;
                    WriteHeader();
                }

                File->Seek(rpsave.LeftAddress, soFromBeginning);
                File->Read(&record_preview, sizeof(RecordPreview));
                record_preview.PreviousNodeAddress = rpsave.PreviousNodeAddress;
                record_preview.MotherNodeAddress = rpsave.MotherNodeAddress;
                File->Seek(rpsave.LeftAddress, soFromBeginning);
                File->Write(&record_preview, sizeof(RecordPreview));

                CurrentNodeAddress = rpsave.LeftAddress;
                while(Go(RIGHT));
                File->Seek(CurrentNodeAddress, soFromBeginning);
                File->Read(&record_preview, sizeof(RecordPreview));

                record_preview.RightAddress = rpsave.RightAddress;
                File->Seek(CurrentNodeAddress, soFromBeginning);
                File->Write(&record_preview, sizeof(RecordPreview));

                savenodeaddress = CurrentNodeAddress;

                File->Seek(rpsave.RightAddress, soFromBeginning);
                File->Read(&record_preview, sizeof(RecordPreview));
                record_preview.PreviousNodeAddress = savenodeaddress;
                File->Seek(rpsave.RightAddress, soFromBeginning);
```

```
                    File->Write(&record_preview, sizeof(RecordPreview));

                    CurrentNodeAddress = rpsave.LeftAddress;
                }
            }
            else
            {
                if(Go(MOTHER))
                {
                    ReturnValue = MOTHER;
                    File->Seek(CurrentNodeAddress, soFromBeginning);
                    File->Read(&record_preview, sizeof(RecordPreview));
                    record_preview.SubNodeAddress = 0;
                    File->Seek(CurrentNodeAddress, soFromBeginning);
                    File->Write(&record_preview, sizeof(RecordPreview));
                }
                else
                {
                    ReturnValue = NULL;
                    Header.FirstNodeAddress = 0;
                    WriteHeader();
                }
            }
        }

        CloseIndexFiles();
    }

    return ReturnValue;
}
//---------------------------------------------------------------------------

void Fbtree::ReplaceNode(void *nodestruct, int nodestructsize)
{
    RecordPreview rpsave;
    long savenodeaddress;
    char deletevalue;

    if(Header.FirstNodeAddress && File->Handle > 0)
    {
        File->Seek(CurrentNodeAddress, soFromBeginning);
        File->Read(&rpsave, sizeof(RecordPreview));

        deletevalue = DeleteNode();
        if(deletevalue != MOTHER)
        {
            Go(SUBTOP);
            AddNode(nodestruct, nodestructsize, (char *)nodestruct + rpsave.KeyStartAddress, rpsave.KeyLen, rpsave.CompareKind);
        }
        else
            AddSubNode(nodestruct, nodestructsize, (char *)nodestruct + rpsave.KeyStartAddress, rpsave.KeyLen, rpsave.CompareKind);

        File->Seek(CurrentNodeAddress, soFromBeginning);
```

```
          File->Read(&record_preview, sizeof(RecordPreview));
          record_preview.SubNodeAddress = rpsave.SubNodeAddress;
          File->Seek(CurrentNodeAddress, soFromBeginning);
          File->Write(&record_preview, sizeof(RecordPreview));

          if(rpsave.SubNodeAddress)
          {
               long counter = 0;
               void *result;

               savenodeaddress = CurrentNodeAddress;
               Go(SUB);
               do
               {
                    if(!counter)
                    {
                         result = FreeSearch(MAXFBTREELEVEL, NULL, NULL, NULL, NULL);
                         counter++;
                    }
                    else
                         result = FreeSearch(MAXFBTREELEVEL, NULL, NULL, NULL, NULL, NEXT);

                    if(result)
                    {
                         File->Seek(CurrentNodeAddress, soFromBeginning);
                         File->Read(&record_preview, sizeof(RecordPreview));
                         record_preview.MotherNodeAddress = savenodeaddress;
                         File->Seek(CurrentNodeAddress, soFromBeginning);
                         File->Write(&record_preview, sizeof(RecordPreview));
                    }
               }while(result);

               Go(savenodeaddress);
          }

          if(deletevalue == MOTHER || (rpsave.MotherNodeAddress && (deletevalue == LEFT || deletevalue == RIGHT)))
          {
               Go(SUBTOP);

               File->Seek(rpsave.MotherNodeAddress, soFromBeginning);
               File->Read(&record_preview, sizeof(RecordPreview));
               record_preview.SubNodeAddress = CurrentNodeAddress;
               File->Seek(rpsave.MotherNodeAddress, soFromBeginning);
               File->Write(&record_preview, sizeof(RecordPreview));
          }
     }
}
//---------------------------------------------------------------------------

int Fbtree::MoveSubUp(void)
{
////////////////////////////////////////////////////////////////////////////////
     //NOTICE:
```

```
    //When the sub node is became move up, it loses
    //its left and right node addresses.
    //If this occured, ReturnValue sets to 1, otherwise sets to 0;

    //If current node has no sub node, it will be deleted;
////////////////////////////////////////////////////////////////////////////

    int ReturnValue = 0;
    RecordPreview subrpsave = {NULL};
    long savenodeaddress = CurrentNodeAddress;
    wchar_t *readnode = NULL;

    if(Go(SUB))
    {
        File->Seek(CurrentNodeAddress, soFromBeginning);
        File->Read(&subrpsave, sizeof(RecordPreview));
        readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);
    }

    Go(savenodeaddress);

    DeleteNode();
    if(subrpsave.MotherNodeAddress)
    {
        Go(SUBTOP);
        AddNode(readnode, subrpsave.RecordLen, readnode + subrpsave.KeyStartAddress, subrpsave.KeyLen, subrpsave.CompareKind);

        File->Seek(CurrentNodeAddress, soFromBeginning);
        File->Read(&record_preview, sizeof(RecordPreview));

        record_preview.SubNodeAddress = subrpsave.SubNodeAddress;
        File->Seek(CurrentNodeAddress, soFromBeginning);
        File->Write(&record_preview, sizeof(RecordPreview));

        savenodeaddress = CurrentNodeAddress;
        if(Go(SUB))
        {
            File->Seek(CurrentNodeAddress, soFromBeginning);
            File->Read(&record_preview, sizeof(RecordPreview));
            record_preview.MotherNodeAddress = savenodeaddress;
            File->Seek(CurrentNodeAddress, soFromBeginning);
            File->Write(&record_preview, sizeof(RecordPreview));
        }
    }

    if(readnode)
        delete readnode;

    Go(CurrentNodeAddress);

    return ReturnValue;
}
//---------------------------------------------------------------------------
```

```
char Fbtree::Optimize(void)
{
     char optimizefilename[128] = "OPTIMIZE.DAT";
     char ReturnValue = 0;

     bool del = _DeleteFile(optimizefilename, UseExecutionPath);

     if(del == true && File->Handle > 0)
     {
          OptimizeLevel = 0;
          CloseIndexFiles(true);

          Fbtree *opt_result = new Fbtree(optimizefilename, YES, CaseSensitive, CompareFunction, fbtree_index, UseExecutionPath, false);

          if(opt_result)
          {
               opt_result->OptimizeActive = YES;

               opt_result->OpenIndexFiles();

               Go(TOP);

               if(OptProccess(opt_result))
               {
                    FbtreeHeader oldheader = Header;

                    File->Size = sizeof(Header);
                    File->Seek(sizeof(Header), soFromBeginning);
                    opt_result->File->Seek(sizeof(Header), soFromBeginning);

                    File->CopyFrom(opt_result->File, opt_result->File->Size - sizeof(Header));

                    File->Seek(0, soFromBeginning);
                    opt_result->File->Seek(0, soFromBeginning);

                    memmove(&Header, &opt_result->Header, (int)(&Header.UserActive[0] - &Header.Header[0]));

                    StrCopy(Header.Security1, oldheader.Security1);
                    StrCopy(Header.Security2, oldheader.Security2);
                    StrCopy(Header.Provider, oldheader.Provider);
                    Header.Version = oldheader.Version;

                    WriteHeader();
                    CurrentNodeAddress = Header.FirstNodeAddress;
                    ReturnValue = 1;
               }

               opt_result->CloseIndexFiles(true);
               delete opt_result;

               _DeleteFile(optimizefilename, UseExecutionPath);
          }
```

```
          GetNextNewNodeAddress();

          if(OpenAllIndexes)
               OpenIndexFiles();
     }



     if(ProgressBar)
          ProgressBar->Position = ProgressBar->Min;

     return ReturnValue;
}
//---------------------------------------------------------------------------

char Fbtree::OptProccess(Fbtree *opt_result)
{
     int i;
     char ReturnValue = 0;
     int address_handle;
     int record_counter = 0, address;
     char opt_filename[256];
     wchar_t *readnode = NULL;

     OptimizeLevel++;

     if(UseExecutionPath)
          sprintf(opt_filename, "%s\\address.%d", ExecutionPath, OptimizeLevel);
     else
          sprintf(opt_filename, "address.%d", OptimizeLevel);

     TFileStream *AddressFile = new TFileStream(opt_filename, fmCreate);

     if(AddressFile->Handle > 0)
     {
          ReturnValue = 1;

          if(FreeSearch(MAXFBTREELEVEL, NULL, NULL, NULL, NULL))
          {
               AddressFile->Write(&CurrentNodeAddress, sizeof(long));
               record_counter++;

               while(FreeSearch(MAXFBTREELEVEL, NULL, NULL, NULL, NULL, NEXT))
               {
                    AddressFile->Write(&CurrentNodeAddress, sizeof(long));
                    record_counter++;
               }
          }

          if(ProgressBar && OptimizeLevel == 1)
          {
               ProgressBar->Min = 0;
```

```
        ProgressBar->Max = record_counter;
        ProgressBar->Position = 0;
    }

    if(record_counter > 2)
    {
        int random1 = 0, random2 = 0;
        int address1, address2;
        unsigned randomblock1, randomblock2, maxblock;

        randomize();
        maxblock = unsigned(record_counter / 32000);
        for(i = 0; i < record_counter; i++)
        {
            randomblock1 = random(maxblock + 1);
            randomblock2 = random(maxblock + 1);

            while(!random1)
                random2 = random1 = randomblock1 == maxblock ? random((unsigned)(record_counter % 32000)) : random(32000);

            while(random1 == random2 || !random2)
                random2 = randomblock2 == maxblock ? random((unsigned)(record_counter % 32000)) : random(32000);

            AddressFile->Seek((randomblock1 * 32000 + random1) * sizeof(long), soFromBeginning);
            AddressFile->Read(&address1, sizeof(long));
            AddressFile->Seek((randomblock2 * 32000 + random2) * sizeof(long), soFromBeginning);
            AddressFile->Read(&address2, sizeof(long));

            AddressFile->Seek((randomblock1 * 32000 + random1) * sizeof(long), soFromBeginning);
            AddressFile->Write(&address2, sizeof(long));
            AddressFile->Seek((randomblock2 * 32000 + random2) * sizeof(long), soFromBeginning);
            AddressFile->Write(&address1, sizeof(long));

            random1 = random2 = 0;
        }
    }

    AddressFile->Seek(0, soFromBeginning);

    for(i = 0; i < record_counter; i++)
    {
        if(ProgressBar && OptimizeLevel == 1)
        {
            ProgressBar->Position = record_counter;

            if(ProgressBar->Max > 100)
            {
                if(!(record_counter%(ProgressBar->Max / 100)))
                    Application->ProcessMessages();
            }
            else
                Application->ProcessMessages();
        }
```

```
                AddressFile->Read(&address, sizeof(long));
                Go(address);

                readnode = (wchar_t *)AllocReadRecord(readnode, CurrentNodeAddress);

                if(!i)
                     opt_result->AddSubNode((void *)readnode, record_preview.RecordLen, (char *)readnode + record_preview.KeyStartAddress,
 record_preview.KeyLen, record_preview.CompareKind);
                else
                {
                     opt_result->Go(SUBTOP);
                     opt_result->AddNode((void *)readnode, record_preview.RecordLen, (char *)readnode + record_preview.KeyStartAddress,
 record_preview.KeyLen, record_preview.CompareKind);
                }

                if(readnode)
                     delete[] readnode;

                if(record_preview.SubNodeAddress)
                {
                     Go(SUB);

                     ReturnValue = OptProccess(opt_result);
                     opt_result->Go(MOTHER);
                     Go(MOTHER);
                }
           }
           delete AddressFile;

           _DeleteFile(opt_filename, false/*UseExecutionPath*/);
      }

      OptimizeLevel--;

      return ReturnValue;
}
//---------------------------------------------------------------------------

int Fbtree::PreIndex(int index_number)
{
      int ReturnValue = index_number;

      for(register i = 0; i < index_number; i++)
           if(!strcmp(fbtree_index[index_number].Filename, fbtree_index[i].Filename) && Index[i])
           {
                ReturnValue = i;
                break;
           }

      return ReturnValue;
}
//---------------------------------------------------------------------------
```

```
void Fbtree::OpenIndexFiles(void)
{
     register i;

     if(!OpenIndexFlag)
          for(i = 0; i < IndexCounter; i++)
               if((OpenAllIndexes || OptimizeActive == YES || CurrentFileLevel == fbtree_index[i].Level || (CurrentFileLevel >=
 fbtree_index[i].Level && fbtree_index[i].IncludeSubNodes)) && PreIndex(i) == i)
               {
                    if(OptimizeActive == YES)
                         _DeleteFile(fbtree_index[i].Filename, UseExecutionPath);

                    Index[i] = new Fbtree(fbtree_index[i].Filename, YES, CaseSensitive, fbtree_index[i].Comp, NULL, UseExecutionPath);
               }

     OpenIndexFlag++;
}
//---------------------------------------------------------------------------

void Fbtree::CloseIndexFiles(bool ForceClose)
{
     register i;

     OpenIndexFlag--;

     if(!OpenIndexFlag || ForceClose)
     {
          for(i = IndexCounter - 1; i >= 0; i--)
               if(Index[i])
               {
                    delete Index[i];
                    Index[i] = NULL;
               }

          OpenIndexFlag = 0;
     }
}
//---------------------------------------------------------------------------

void Fbtree::WriteIndex(void *nodestruct, unsigned delete_address)
{
     void *result;
     int PreIndex_number;

     for(register i = 0; i < IndexCounter; i++)
          if(CurrentFileLevel == fbtree_index[i].Level || (CurrentFileLevel >= fbtree_index[i].Level && fbtree_index[i].IncludeSubNodes))
     {
               wchar_t ch = 1;
               char addsub = 1;
               char *buffer = new char[fbtree_index[i].Len + 2];

               PreIndex_number = PreIndex(i);
```

```
                memmove(buffer, (char *)nodestruct + fbtree_index[i].StartKey, fbtree_index[i].Len);
                buffer[fbtree_index[i].Len] = 0;

                Index[PreIndex_number]->Go(TOP);

                fbtree_currentindex_sign = fbtree_index[i].Sign;

                result = Index[PreIndex_number]->BSearch(&ch);
                if(!result)
                     Index[PreIndex_number]->AddNode(&ch, sizeof(wchar_t), &ch, sizeof(wchar_t), fbtCHAR);
                else
                     if(Index[PreIndex_number]->Go(SUB))
                     {
                          result = Index[PreIndex_number]->BSearch(buffer);
                          addsub = 0;
                     }
                     else
                          result = NULL;

                if(!result)
                {
                     if(addsub)
                          Index[PreIndex_number]->AddSubNode(buffer, wcslen((wchar_t *)buffer) * sizeof(wchar_t) + 2, buffer,
fbtree_index[i].Len, fbtree_index[i].CompareKind);
                     else
                          Index[PreIndex_number]->AddNode(buffer, wcslen((wchar_t *)buffer) * sizeof(wchar_t) + 2, buffer,
fbtree_index[i].Len, fbtree_index[i].CompareKind);

                     Index[PreIndex_number]->AddSubNode(&CurrentNodeAddress, sizeof(int), &CurrentNodeAddress, sizeof(int), fbtLONG);
                }
                else
                {
                     if(Index[PreIndex_number]->Go(SUB))
                     {
                          result = NULL;

                          if(delete_address)
                               result = Index[PreIndex_number]->BSearch(&delete_address);

                          if(result)
                          {
                               if(Index[PreIndex_number]->DeleteNode() == MOTHER)
                                    Index[PreIndex_number]->AddSubNode(&CurrentNodeAddress, sizeof(long), &CurrentNodeAddress, sizeof(long),
fbtLONG);
                               else
                                    Index[PreIndex_number]->AddNode(&CurrentNodeAddress, sizeof(long), &CurrentNodeAddress, sizeof(long),
fbtLONG);
                          }
                          else
                               Index[PreIndex_number]->AddNode(&CurrentNodeAddress, sizeof(long), &CurrentNodeAddress, sizeof(long),
fbtLONG);
                     }
```

```
                    else
                        Index[PreIndex_number]->AddSubNode(&CurrentNodeAddress, sizeof(long), &CurrentNodeAddress, sizeof(long), fbtLONG);
                }

                delete[] buffer;
            }
}
//---------------------------------------------------------------------------

void Fbtree::RemoveAddressIndex(unsigned delete_address)
{
    void *result;
    int PreIndex_number;
    wchar_t ch = 0;

    for(register i = 0; i < IndexCounter; i++)
        if(CurrentFileLevel == fbtree_index[i].Level || (CurrentFileLevel >= fbtree_index[i].Level && fbtree_index[i].IncludeSubNodes))
        {
            PreIndex_number = PreIndex(i);

            Index[PreIndex_number]->Go(TOP);

            fbtree_currentindex_sign = fbtree_index[i].Sign;

            result = Index[PreIndex_number]->BSearch(&ch);

            if(!result)
                Index[PreIndex_number]->AddNode(&ch, sizeof(wchar_t), &ch, sizeof(wchar_t), fbtCHAR);

            Index[PreIndex_number]->AddSubNode(&delete_address, sizeof(long), &delete_address, sizeof(long), fbtLONG);
        }
}
//---------------------------------------------------------------------------

unsigned Fbtree::GoIndexAddress(char *filename, void *searchbuffer, int occurrence_number)
{
    unsigned ReturnValue = 0;
    wchar_t ch;

    OpenIndexFiles();
    for(register i = 0; i < IndexCounter; i++)
        if(!strcmp(fbtree_index[i].Filename, filename))
        {
            ch = 1;
            Fbtree *index_file = Index[i];

            if(index_file)
            {
                index_file->Go(TOP);
                if(index_file->BSearch(&ch))
                    if(index_file->Go(SUB))
                    {
                        if(index_file->BSearch(searchbuffer))
```

```
                    {
                        if(index_file->Go(SUB))
                        {
                            void *result;
                            unsigned long find_address = 0;

                            result = index_file->FreeSearch(MAXFBTREELEVEL, NULL, NULL, NULL, NULL);

                            for(register j = 1; j < occurrence_number; j++)
                                result = index_file->FreeSearch(MAXFBTREELEVEL, NULL, NULL, NULL, NULL, NEXT);

                            if(result)
                            {
                                index_file->GetCurrentNode(&find_address);

                                ReturnValue = find_address;

                                index_file->Go(TOP);
                                ch = 0;

                                if(index_file->BSearch(&ch))
                                    if(index_file->Go(SUB))
                                        if(index_file->BSearch(&find_address))
                                            ReturnValue = find_address = DELETED_INDEX;

                                if(ReturnValue)
                                    Go(find_address);
                            }
                        }
                    }
                }
            }
        }

    CloseIndexFiles();
    return ReturnValue;
}
//---------------------------------------------------------------------------

void Fbtree::SetHeaderProviderVersion(char *Provider, float Version)
{
    StrLCopy(Header.Provider, Provider, 49);
    Header.Version = Version;
    WriteHeader();
}
//---------------------------------------------------------------------------

bool Fbtree::CheckProviderVersion(char *Provider, float Version)
{
    bool ReturnValue = true;

    if(StrComp(Header.Provider, Provider))
        ReturnValue = false;
```

```
    if(Header.Version != Version)
        ReturnValue = false;

    return ReturnValue;
}
//---------------------------------------------------------------------------

AnsiString __fastcall Fbtree::GetExecutionPath(void)
{
    return ExtractFileDir(ParamStr(0));
}
//---------------------------------------------------------------------------
```

```
#ifndef H_FBTREE
#define H_FBTREE

#include "ComCtrls.hpp"

#define MAXFBTREELEVEL  100
#define MAXFBTREEINDEX  10


#define fbtSTRING            1
#define fbtINTEGER           2
#define fbtLONG              3
#define fbtFLOAT             4
#define fbtDOUBLE            5
#define fbtDATE              6
#define fbtCHAR              7
#define fbtFREECOMPARE       8

#define YES                 true
#define NO                  false

#define DELETED_INDEX       4294967295L

#define LEFT                1
#define RIGHT               2
#define SUB                 3
#define MOTHER              4
#define TOP                 5
#define SUBTOP              6
#define ADDRESS             7
#define CENTER              8
#define TOPLEFT             9
#define TOPRIGHT            10
#define SEARCH              11
#define NEXT                12
#define PREVIOUS            13
#define ALL                 14
#define FIRST               15
#define CURRENT             16
#define LAST                17

struct FbtreeIndex
{
     char *Filename;
     int Level;
     int StartKey;
     int Len;
     int (*Comp)(WideString s1, WideString s2, unsigned len);
     char CompareKind;
     char Sign;
     bool IncludeSubNodes;
};
```

```
struct RecordPreview
{
     long LeftAddress, RightAddress;
     long SubNodeAddress, MotherNodeAddress;
     long PreviousNodeAddress;
     unsigned RecordLen, KeyStartAddress, KeyLen;
     char CompareKind;
};


struct FbtreeHeader
{
     char Header[17];
     char EndofFile;
     char Security1[11];
     char Security2[11];
     unsigned long FirstNodeAddress;
     char Provider[50];
     float Version;
     char Reserved[30];
     char UserActive[512];
};


class Fbtree
{
     int Access;
     unsigned Mode;
     long NewNodeAddress, PreviousNodeAddress;
     long LastSearchedAddress[MAXFBTREELEVEL + 1];
     char PreviousPath;
     bool CaseSensitive;
     bool OpenAllIndexes;
     int  *Stack[MAXFBTREELEVEL + 1], *StackBak;
     int  StackPosition[MAXFBTREELEVEL + 1];
     int  OptimizeLevel;
     bool OptimizeActive;
     int  OpenIndexFlag;
     int  CurrentFileLevel;
     char FbtreeFilename[128];
     int (*CompareFunction)(WideString s1, WideString s2, unsigned len);

     FbtreeIndex *fbtree_index;
     Fbtree *Index[MAXFBTREEINDEX];
     int IndexCounter;

     void GetNextNewNodeAddress(void);
     void WriteNewRecord(void *nodestruct);
     void ReadRecord(void *nodestruct, long nodeaddress = -1);
     void *AllocReadRecord(void *nodestruct, long nodeaddress);
     int  Push(int address, int searchlevel);
     int  Pop(int searchlevel);
```

```
    char OptProccess(Fbtree *opt_result);
    int  Compare(void *comp1, void *comp2, unsigned size, char comparekind);
    void WriteIndex(void *nodestruct, unsigned delete_address = NULL);
    void RemoveAddressIndex(unsigned delete_address);
    int  PreIndex(int index_number);
    bool UseExecutionPath;

public:
    TFileStream *File;
    bool Found;
    int  CurrentNodeAddress;

    FbtreeHeader Header;
    RecordPreview record_preview;
    TProgressBar *ProgressBar;

    void WriteHeader(char *source = NULL, int start = 0, int size = 1);
    void ReadHeader(char *dest = NULL, int start = 0, int size = 1);
    char AddNode(void *nodestruct, int nodestructsize, void *startnodekey, int keysize, char comparekind = fbtSTRING, int nodeaddress =
-1);
    char AddSubNode(void *nodestruct, int nodestructsize, void *stratnodekey, int keysize, char comparekind = fbtSTRING, int nodeaddress =
-1);
    void *BSearch(void *searchstring, int startaddress = -1);
    void *FreeSearch(int searchlevel, void *searchstring, void *nodestruct, void *compareaddress, unsigned comparelen, char firstornext =
FIRST, char comparekind = fbtSTRING);
    char GetCurrentNode(void *nodestruct);
    char Go(int where);
    void OpenIndexFiles(void);
    void CloseIndexFiles(bool ForceClose = false);
    char DeleteNode(void);
    void ReplaceNode(void *nodestruct, int nodestructsize);
    int  MoveSubUp(void);
    char Optimize(void);
    unsigned GoIndexAddress(char *filename, void *searchbuffer, int occurrence_number = 1);
    void *AllocGetCurrentNode(void *nodestruct);
    void SetHeaderProviderVersion(char *Provider, float Version);
    bool CheckProviderVersion(char *Provider, float Version);
    AnsiString __fastcall GetExecutionPath(void);

    Fbtree(char *filename, bool change_header = YES, bool casesens = NO, int (*comp)(WideString s1, WideString s2, unsigned len) = NULL,
FbtreeIndex *f_index = NULL, bool use_execpath = YES, bool OpenIndexes = YES);
    ~Fbtree(void);
};

#endif
```