**Приложение Б**

Исходный код приложения

#include <vcl.h>

#include <stdio.h>

#include <DateUtils.hpp>

#pragma hdrstop

#include "main.h"

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

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

struct flight \*BeginFlights=NULL; // Pointer to beginning of flight list

struct client \*BeginCustomers=NULL; // Pointer to beginning list of clients

struct flight \*BUF\_Flights=NULL;

struct client \*BUF\_Customers=NULL;

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

/\*\*

\* Description structure "Flight"

\*/

struct flight {

TDateTime date; // Departure date

TDateTime hour; // Departure hours

unsigned int number\_flight; // Flight number

AnsiString airport\_departure; // Airport of departure

AnsiString airport\_arrival; // Airport arrival

int free\_places\_agency; // Number of free places for travel agencies

struct flight \*next\_flight; // Pointer to the next item in the list

};

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

/\*\*

\* Description structure "Client"

\*/

struct client {

AnsiString surname; // Surname of the client

AnsiString name; // Name of the client

TDateTime departure\_date; // Date of departure

AnsiString resting\_place; // Resting place

AnsiString airport\_arrival; // Arrival airport

int number\_participants; // Number of participants

struct flight \*pointer\_flight; // Pointer to the flight

struct client \*next\_client; // Pointer to the next item in the list

};

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

/\* Function prototypes \*/

void ReplacementInterface(bool, bool, bool); // Replacement UI main window

void ShowRecords(flight \*\*begin); // Display a list in StringGrid1

void ShowRecords(client \*\*begin); // Display a list in StringGrid2

bool ModifyingList(flight \*\*working);

bool ModifyingList(client \*\*working);

void RevertingChanges(flight \*\*working,flight \*\*begin);

void RevertingChanges(client \*\*working, client \*\*begin);

void RecordSelection(flight \*\*working, flight \*\*begin, int SelTable);

void RecordSelection(client \*\*working, client \*\*begin, int SelTable);

void flight\_sort(flight \*\*begin, unsigned int num\_field);

void client\_sort(client \*\*begin, unsigned int num\_field);

flight\* q\_flight\_sorted(struct flight\*\* a,unsigned int r,unsigned int n);

client\* q\_client\_sorted(struct client\*\* a,unsigned int r,unsigned int n);

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

\_\_fastcall TForm1::TForm1(TComponent\* Owner) : TForm(Owner){

}

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

void \_\_fastcall TForm1::FormCreate(TObject \*Sender) {

Form1->StringGrid1->ColCount = 6; // Set number cols in a list of flights

Form1->StringGrid2->ColCount = 7; // Set number cols in a list of clients

Form1->StringGrid1->RowCount = 2; // Set number rows in a list of flights

Form1->StringGrid2->RowCount = 2; // Set number rows in a list of clients

Form1->StringGrid1->Options << goRowSelect; // Selects entire row in table

Form1->StringGrid2->Options << goRowSelect;

Form1->StringGrid1->Cells[0][0] = "Departure date";

Form1->StringGrid1->Cells[1][0] = "Departure time";

Form1->StringGrid1->Cells[2][0] = "Flight number";

Form1->StringGrid1->Cells[3][0] = "Departure airport";

Form1->StringGrid1->Cells[4][0] = "Arrival airport";

Form1->StringGrid1->Cells[5][0] = "Places for agency";

Form1->LabeledEdit1->EditLabel->Caption = "Departure date";

Form1->LabeledEdit2->EditLabel->Caption = "Departure time";

Form1->LabeledEdit3->EditLabel->Caption = "Flight number";

Form1->LabeledEdit4->EditLabel->Caption = "Departure airport";

Form1->LabeledEdit5->EditLabel->Caption = "Arrival airport";

Form1->LabeledEdit6->EditLabel->Caption = "Places for agency";

Form1->StringGrid2->Cells[0][0] = "Client surname";

Form1->StringGrid2->Cells[1][0] = "Client name";

Form1->StringGrid2->Cells[2][0] = "Departure date";

Form1->StringGrid2->Cells[3][0] = "Resting place";

Form1->StringGrid2->Cells[4][0] = "Arrival airport";

Form1->StringGrid2->Cells[5][0] = "Number of participants";

Form1->StringGrid2->Cells[6][0] = "Flight number";

Form1->LabeledEdit7->EditLabel->Caption = "Client surname";

Form1->LabeledEdit8->EditLabel->Caption = "Client name";

Form1->LabeledEdit9->EditLabel->Caption = "Departure date";

Form1->LabeledEdit10->EditLabel->Caption = "Resting place";

Form1->LabeledEdit11->EditLabel->Caption = "Arrival airport";

Form1->LabeledEdit12->EditLabel->Caption = "Count participants";

Form1->LabeledEdit13->EditLabel->Caption = "Flight number";

Form1->LabeledEdit12->Enabled = false;

Form1->LabeledEdit13->Enabled = false;

Form1->Button1->Caption = "Add";

Form1->Button2->Caption = "Edit";

Form1->Button3->Caption = "Delete";

Form1->Button4->Caption = "Auto-assign";

Form1->Button5->Caption = "Exit";

Form1->Button6->Caption = "Save";

Form1->Button7->Caption = "Cancel";

Form1->TabSheet1->Caption = "List of flights";

Form1->TabSheet2->Caption = "List of clients";

Form1->TabSheet3->Caption = "Edit flight";

Form1->TabSheet4->Caption = "Edit client";

Form1->Panel2->Visible = false; // Hide edit pane

Form1->TabSheet3->TabVisible = false; // Hide editing tab

Form1->TabSheet4->TabVisible = false; // Hide editing tab

Form1->StringGrid1->ColWidths[0]=80;

Form1->StringGrid1->ColWidths[1]=80;

Form1->StringGrid1->ColWidths[2]=80;

Form1->StringGrid1->ColWidths[3]=202;

Form1->StringGrid1->ColWidths[4]=202;

Form1->StringGrid1->ColWidths[5]=111;

Form1->StringGrid2->ColWidths[0]=109;

Form1->StringGrid2->ColWidths[1]=95;

Form1->StringGrid2->ColWidths[2]=80;

Form1->StringGrid2->ColWidths[3]=115;

Form1->StringGrid2->ColWidths[4]=170;

Form1->StringGrid2->ColWidths[5]=100;

Form1->StringGrid2->ColWidths[6]=85;

HMENU MenuHandle = GetSystemMenu(Handle, false); // Barring a reversal

if (MenuHandle) // of the window to

DeleteMenu(MenuHandle, SC\_MAXIMIZE, MF\_BYCOMMAND); // full screen

Constraints->MaxHeight=533; //

Constraints->MinHeight=533; // Ban window resizing

Constraints->MaxWidth=810; //

Constraints->MinWidth=810; //

}

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

/\*\*

\* Press 'Add' record

\*/

void \_\_fastcall TForm1::Button1Click(TObject \*Sender) {

Form1->Tag = 0; // Set the flag to add entries

switch (Form1->PageControl1->ActivePageIndex) {

case 0: // Selected fly list

{

ReplacementInterface(true, false, false);

struct flight \*tmp=BeginFlights;

if (tmp==NULL) { // If the list is empty

tmp=new flight; // Create the first instance of structure

BeginFlights=tmp; // Sets start address of first list pointer structure

BUF\_Flights=tmp; // Sets buffer pointer address of the first structure

} else { // If the list is not empty

while ((tmp->next\_flight)!=NULL)

tmp=tmp->next\_flight;

(tmp->next\_flight) = new flight; // Assign buffer pointer

BUF\_Flights=tmp->next\_flight; // address created memory

}

}

BUF\_Flights->next\_flight=NULL;

break;

case 1: // Selected client list

{

ReplacementInterface(true, true, false);

struct client \*tmp=BeginCustomers;

if (tmp==NULL) { // If the list is empty

tmp=new client; // Create the first instance of structure

BeginCustomers=tmp; // Sets start address of first list pointer structure

BUF\_Customers=tmp; // Sets buffer pointer address of the first structure

} else { // If the list is not empty

while ((tmp->next\_client)!=NULL)

tmp=tmp->next\_client;

(tmp->next\_client)=new client; // Assign buffer pointer

BUF\_Customers=tmp->next\_client; // address created memory

}

}

BUF\_Customers->next\_client=NULL;

BUF\_Customers->pointer\_flight=NULL;

BUF\_Customers->number\_participants=0;

break;

default:

ShowMessage("Unknown error has occurred the program!!!");

break;

}

}

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

/\*\*

\* Press 'Edit' record

\*/

void \_\_fastcall TForm1::Button2Click(TObject \*Sender) {

Form1->Tag = 1; // Set the flag to edit entries

switch (Form1->PageControl1->ActivePageIndex) {

case 0: // Selected list of flights

{

RecordSelection(&BUF\_Flights, &BeginFlights, Form1->StringGrid1->Row);

if (BUF\_Flights==NULL) {

ShowMessage("You cannot change selected record!!");

return;

}

ReplacementInterface(true, false, true);

}

break;

case 1: // Selected list of clients

{

RecordSelection(&BUF\_Customers, &BeginCustomers, Form1->StringGrid2->Row);

if (BUF\_Customers==NULL) {

ShowMessage("You cannot change selected record!!");

return;

}

ReplacementInterface(true, true, true);

}

break;

default:

ShowMessage("Unknown error has occurred the program!!!");

break;

}

}

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

/\*\*

\* Press 'Delete' record

\*/

void \_\_fastcall TForm1::Button3Click(TObject \*Sender) {

switch (Form1->PageControl1->ActivePageIndex) {

case 0: // Selected list of flights

{

RecordSelection(&BUF\_Flights, &BeginFlights, Form1->StringGrid1->Row);

if (BUF\_Flights==NULL) {

ShowMessage("You cannot delete selected record!!");

return;

}

if (MessageBox(0,"Are you sure you want to delete the selected entry?",

"Attention!", MB\_YESNO) == mrYes) {

struct client \*tmp=BeginCustomers; // Check list of clients,

while (tmp!=NULL) { // with a pointer

if (tmp->pointer\_flight==BUF\_Flights) // to this entry.

tmp->pointer\_flight=NULL; // Zeroing of these pointers.

tmp=tmp->next\_client; //

}

if (BUF\_Flights==BeginFlights) // If you delete the first element

BeginFlights=BeginFlights->next\_flight;

else {

struct flight \*tmp=BeginFlights;

while (tmp->next\_flight!=BUF\_Flights)

tmp=tmp->next\_flight;

tmp->next\_flight=BUF\_Flights->next\_flight;

}

delete BUF\_Flights;

BUF\_Flights=NULL;

ShowRecords(&BeginFlights);

ShowRecords(&BeginCustomers);

}

}

break;

case 1: // Selected list of clients

{

RecordSelection(&BUF\_Customers, &BeginCustomers, Form1->StringGrid2->Row);

if (BUF\_Customers==NULL) {

ShowMessage("You cannot delete selected record!!");

return;

}

if (MessageBox(0,"Are you sure you want to delete the selected entry?",

"Attention!", MB\_YESNO) == mrYes) {

if (BUF\_Customers==BeginCustomers) // If you delete the first element

BeginCustomers=BeginCustomers->next\_client;

else {

struct client \*tmp=BeginCustomers;

while (tmp->next\_client!=BUF\_Customers)

tmp=tmp->next\_client;

tmp->next\_client=BUF\_Customers->next\_client;

}

delete BUF\_Customers;

BUF\_Customers=NULL;

ShowRecords(&BeginCustomers);

}

}

break;

default:

ShowMessage("Unknown error has occurred the program!!!");

break;

}

}

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

/\*\*

\* Press 'Auto-assign' (Automatic assignment of flights)

\*/

void \_\_fastcall TForm1::Button4Click(TObject \*Sender) {

if ((BeginFlights==NULL)||(BeginCustomers==NULL)) {

(MessageDlg("One of the lists is empty!!\n(Do not perform auto-assign)",

mtError, TMsgDlgButtons() << mbYes,0));

return;

}

Form1->PageControl1->ActivePageIndex=1;

struct client \*cle=BeginCustomers; //

while(cle!=NULL) { // Clear the list client from the old

if (cle->pointer\_flight!=NULL) { // pointer values.

cle->pointer\_flight=NULL; //

cle->number\_participants=0; // And reset the value of the field

} // "Number of participants".

cle=cle->next\_client; //

}

struct client \*kl\_tmp=BeginCustomers;

while(kl\_tmp!=NULL) { // Until end of the list of clients

if (kl\_tmp->pointer\_flight==NULL) { // If client is not assigned a flight

struct flight \*fl\_tmp=BeginFlights;

while (fl\_tmp!=NULL) { // Until the end a list of flights

if ((fl\_tmp->date==kl\_tmp->departure\_date) &&

(fl\_tmp->airport\_arrival==kl\_tmp->airport\_arrival)) {

unsigned int ch=0;

// all the places to travel on this flight

unsigned int x = fl\_tmp->free\_places\_agency;

struct client \*kl\_tmp2=BeginCustomers;

while(kl\_tmp2!=NULL) {

if (fl\_tmp==kl\_tmp2->pointer\_flight)

ch++;

kl\_tmp2=kl\_tmp2->next\_client;

}

if (ch<x) {

kl\_tmp->pointer\_flight=fl\_tmp; // Fill in the field

struct client \*kl\_tmp3=BeginCustomers; //

while(kl\_tmp3!=NULL) { // Change the value of field

if (fl\_tmp==kl\_tmp3->pointer\_flight) // "number of participants"

kl\_tmp3->number\_participants=ch+1; // in the entire client list

kl\_tmp3=kl\_tmp3->next\_client; //

}

break;

}

}

fl\_tmp=fl\_tmp->next\_flight;

}

}

kl\_tmp=kl\_tmp->next\_client;

}

ShowRecords(&BeginCustomers);

}

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

/\*\*

\* Press 'Exit'

\*/

void \_\_fastcall TForm1::Button5Click(TObject \*Sender) {

Form1->Close();

}

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

/\*\*

\* Press 'Save' (Save the changes)

\*/

void \_\_fastcall TForm1::Button6Click(TObject \*Sender) {

switch (Form1->PageControl1->ActivePageIndex) {

case 2:

if (!ModifyingList(&BUF\_Flights)) // If you cannot change value of

return;

ReplacementInterface(false, false, false); // Change interface

ShowRecords(&BeginFlights); // Display the modified list

break;

case 3:

if (!ModifyingList(&BUF\_Customers)) // If you cannot change value of

return;

ReplacementInterface(false, true, false); // Change interface

ShowRecords(&BeginCustomers); // Display the modified list

break;

default:

ShowMessage("Unknown error has occurred the program!!!");

break;

}

}

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

/\*\*

\* Press 'Cancel' (Cancel changes)

\*/

void \_\_fastcall TForm1::Button7Click(TObject \*Sender) {

switch (Form1->PageControl1->ActivePageIndex) {

case 2:

if (Form1->Tag==0) { // If you added the record

RevertingChanges(&BUF\_Flights,&BeginFlights);

BUF\_Flights=NULL;

}

ReplacementInterface(false, false, false); // Change interface

break;

case 3:

if (Form1->Tag==0) { // If you added the record

RevertingChanges(&BUF\_Customers,&BeginCustomers);

BUF\_Customers=NULL;

}

ReplacementInterface(false, true, false); // Change interface

break;

default:

ShowMessage("Unknown error has occurred the program!!!");

break;

}

}

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

/\*\*

\* Replacement interface main window to edit entity or view a list entities

\*/

void ReplacementInterface(bool FSwitch, bool LSwitch, bool editing) {

if (FSwitch) { // If changing from viewing to editing

Form1->TabSheet1->TabVisible = false;

Form1->TabSheet2->TabVisible = false;

Form1->Panel2->Visible = true;

Form1->Panel1->Visible = false;

if (!LSwitch) { // If edit the list of "flying"

Form1->TabSheet3->TabVisible = true;

Form1->TabSheet4->TabVisible = false;

if (editing) { // If edit entity

Form1->LabeledEdit1->Text = DateToStr(BUF\_Flights->date);

Form1->LabeledEdit2->Text = TimeToStr(BUF\_Flights->hour);

Form1->LabeledEdit3->Text = IntToStr(BUF\_Flights->number\_flight);

Form1->LabeledEdit4->Text = BUF\_Flights->airport\_departure;

Form1->LabeledEdit5->Text = BUF\_Flights->airport\_arrival;

Form1->LabeledEdit6->Text = IntToStr(BUF\_Flights->free\_places\_agency);

}

Form1->LabeledEdit1->SetFocus();

} else { // If edit the list of "clients"

Form1->TabSheet3->TabVisible = false;

Form1->TabSheet4->TabVisible = true;

if (editing) { // If edit entity

Form1->LabeledEdit7->Text = BUF\_Customers->surname;

Form1->LabeledEdit8->Text = BUF\_Customers->name;

Form1->LabeledEdit9->Text = DateToStr(BUF\_Customers->departure\_date);

Form1->LabeledEdit10->Text = BUF\_Customers->resting\_place;

Form1->LabeledEdit11->Text = BUF\_Customers->airport\_arrival;

Form1->LabeledEdit12->Text = IntToStr(BUF\_Customers->number\_participants);

if (BUF\_Customers->pointer\_flight!=NULL)

Form1->LabeledEdit13->Text =

IntToStr(BUF\_Customers->pointer\_flight->number\_flight);

else

Form1->LabeledEdit13->Text = "<Not assigned>";

}

Form1->LabeledEdit7->SetFocus();

}

} else { // If changing from editing on preview

Form1->TabSheet1->TabVisible = true;

Form1->TabSheet2->TabVisible = true;

Form1->Panel2->Visible = false;

Form1->Panel1->Visible = true;

if (!LSwitch) { // If edit the list of "flying"

Form1->LabeledEdit1->Clear();

Form1->LabeledEdit2->Clear();

Form1->LabeledEdit3->Clear();

Form1->LabeledEdit4->Clear();

Form1->LabeledEdit5->Clear();

Form1->LabeledEdit6->Clear();

Form1->TabSheet3->TabVisible = false;

Form1->PageControl1->ActivePageIndex = 0;

} else { // If edit the list of "clients"

Form1->LabeledEdit7->Clear();

Form1->LabeledEdit8->Clear();

Form1->LabeledEdit9->Clear();

Form1->LabeledEdit10->Clear();

Form1->LabeledEdit11->Clear();

Form1->LabeledEdit12->Clear();

Form1->LabeledEdit13->Clear();

Form1->TabSheet4->TabVisible = false;

Form1->PageControl1->ActivePageIndex = 1;

}

}

}

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

/\*

\* Function list of flights display on the screen (in StringGrid1)

\*/

void ShowRecords(flight \*\*begin) {

for(int x=1; x<Form1->StringGrid1->RowCount; x++)

Form1->StringGrid1->Rows[x]->Clear();

Form1->StringGrid1->RowCount=2;

if (!begin) // If the entry does not exist, exit the function.

return;

struct flight \*tmp=NULL;

tmp=(\*begin);

for (int i=0, j=1; tmp!=NULL; j++, i=0) {

Form1->StringGrid1->Cells[i++][j]=tmp->date;

Form1->StringGrid1->Cells[i++][j]=tmp->hour;

Form1->StringGrid1->Cells[i++][j]=tmp->number\_flight;

Form1->StringGrid1->Cells[i++][j]=tmp->airport\_departure;

Form1->StringGrid1->Cells[i++][j]=tmp->airport\_arrival;

Form1->StringGrid1->Cells[i++][j]=tmp->free\_places\_agency;

tmp=tmp->next\_flight;

Form1->StringGrid1->RowCount++;

}

return;

}

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

/\*

\* Function list of clients display on the screen (in StringGrid2)

\*/

void ShowRecords(client \*\*begin) {

for(int x=1; x<Form1->StringGrid2->RowCount; x++)

Form1->StringGrid2->Rows[x]->Clear();

Form1->StringGrid2->RowCount=2;

if (!begin) // If the entry does not exist, exit the function.

return;

struct client \*tmp=NULL;

tmp=(\*begin);

for (int i=0, j=1; tmp!=NULL; j++, i=0) {

Form1->StringGrid2->Cells[i++][j]=tmp->surname;

Form1->StringGrid2->Cells[i++][j]=tmp->name;

Form1->StringGrid2->Cells[i++][j]=tmp->departure\_date;

Form1->StringGrid2->Cells[i++][j]=tmp->resting\_place;

Form1->StringGrid2->Cells[i++][j]=tmp->airport\_arrival;

Form1->StringGrid2->Cells[i++][j]=tmp->number\_participants;

if (tmp->pointer\_flight==NULL)

Form1->StringGrid2->Cells[i++][j] = "<Not assigned>";

else

Form1->StringGrid2->Cells[i++][j] =

IntToStr(tmp->pointer\_flight->number\_flight);

tmp=tmp->next\_client;

Form1->StringGrid2->RowCount++;

}

return;

}

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

/\*

\* Changing the record list of flights

\*/

bool ModifyingList(flight \*\*working) {

try {

StrToDate(Form1->LabeledEdit1->Text); //

StrToTime(Form1->LabeledEdit2->Text); // Checking the data entered

StrToInt(Form1->LabeledEdit3->Text); //

StrToInt(Form1->LabeledEdit6->Text); //

} catch ( ... ) {

(MessageDlg("Missing or incorrect data in the fields!!",

mtError, TMsgDlgButtons() << mbYes,0));

return false;

}

(\*working)->date = StrToDate(Form1->LabeledEdit1->Text);

(\*working)->hour = StrToTime(Form1->LabeledEdit2->Text);

(\*working)->number\_flight = StrToInt(Form1->LabeledEdit3->Text);

(\*working)->airport\_departure = Form1->LabeledEdit4->Text;

(\*working)->airport\_arrival = Form1->LabeledEdit5->Text;

(\*working)->free\_places\_agency = StrToInt(Form1->LabeledEdit6->Text);

return true;

}

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

/\*

\* Changing the record list of clients

\*/

bool ModifyingList(client \*\*working) {

try {

StrToDate(Form1->LabeledEdit9->Text); // Checking the data entered

} catch ( ... ) {

(MessageDlg("Missing or incorrect data in the fields!!",

mtError, TMsgDlgButtons() << mbYes,0));

return false;

}

(\*working)->surname = Form1->LabeledEdit7->Text;

(\*working)->name = Form1->LabeledEdit8->Text;

(\*working)->departure\_date = StrToDate(Form1->LabeledEdit9->Text);

(\*working)->resting\_place = Form1->LabeledEdit10->Text;

(\*working)->airport\_arrival = Form1->LabeledEdit11->Text;

return true;

}

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

/\*

\* Delete created at the end of the list flight structure (If cancel changes)

\*/

void RevertingChanges(flight \*\*working, flight \*\*begin) {

struct flight \*tmp=(\*begin);

if (tmp==(\*working)) { // If only the first record was created

delete BeginFlights;

BeginFlights=NULL;

return;

}

while ((tmp->next\_flight)!=(\*working))

tmp=tmp->next\_flight;

delete tmp->next\_flight;

tmp->next\_flight=NULL;

}

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

/\*

\* Delete created at the end of the list client structure (If cancel changes)

\*/

void RevertingChanges(client \*\*working, client \*\*begin) {

struct client \*tmp=(\*begin);

if (tmp==(\*working)) { // If only the first record was created

delete BeginCustomers;

BeginCustomers=NULL;

return;

}

while ((tmp->next\_client)!=(\*working))

tmp=tmp->next\_client;

delete tmp->next\_client;

tmp->next\_client=NULL;

}

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

/\*

\* Insert in buffer selected at StringGrid1 records (flight)

\*/

void RecordSelection(flight \*\*working, flight \*\*begin, int SelTable) {

(\*working)=(\*begin);

if (\*begin==NULL)

return;

for (int i=1;i<SelTable;i++)

(\*working)=(\*working)->next\_flight;

}

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

/\*

\* Insert in buffer selected at StringGrid2 records (client)

\*/

void RecordSelection(client \*\*working, client \*\*begin, int SelTable) {

(\*working)=(\*begin);

if (\*begin==NULL)

return;

for (int i=1;i<SelTable;i++)

(\*working)=(\*working)->next\_client;

}

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

/\*

\* Move input focus to the next component

\* (For all components of the labelEdit (and one ComboBox))

\*/

void \_\_fastcall TForm1::LabeledEdit1KeyPress(TObject \*Sender, char &Key) {

if (Key==13)

Form1->SelectNext(Form1->ActiveControl,true,true);

}

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

/\*

\* Save a backup copy in file (Press 'Menu'->'File'->'Save list copy in file')

\*/

void \_\_fastcall TForm1::N2Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

if (MessageBox(0,"Are you sure you want to save all the entries to a file?",

"Attention!", MB\_YESNO) == mrNo)

return;

if (BeginFlights==NULL) // If a record dont exist

(MessageDlg("Not exist entities in the list of flights!!\n"

"Fly list is not saved.", mtError, TMsgDlgButtons() << mbYes,0));

else {

FILE \*f=NULL;

f = fopen("d\_struct\_fl.dat","wb");

if (f==NULL) { // If you can not create a file

fclose(f);

(MessageDlg("Error creating file!!\nFly list is not saved.",

mtError, TMsgDlgButtons() << mbYes,0));

} else {

struct flight \*tmp=BeginFlights;

unsigned short sym,fl=0;

do {

sym=DateToStr(tmp->date).Length()+1; //Сalculate length of chars "Date"

fwrite(&sym,sizeof(unsigned short),1,f); // Write length "Date"

fwrite((DateToStr(tmp->date)).c\_str(),sizeof(char)\*sym,1,f); //Write "Date"

sym=TimeToStr(tmp->hour).Length()+1; //Сalculate length of chars "Hour"

fwrite(&sym,sizeof(unsigned short),1,f); // Write length "Hour"

fwrite((TimeToStr(tmp->hour)).c\_str(),sizeof(char)\*sym,1,f); //Write "Hour"

fwrite(&tmp->number\_flight,sizeof(unsigned int),1,f); //Write "Flight number"

sym=tmp->airport\_departure.Length()+1; //Сalculate length "Departure Airport"

fwrite(&sym,sizeof(unsigned short),1,f); // Write length "Departure Airport"

fwrite((tmp->airport\_departure).c\_str(),sizeof(char)\*sym,1,f); //Write "D.A."

sym=tmp->airport\_arrival.Length()+1; //Сalculate length "Arrival Airport"

fwrite(&sym,sizeof(unsigned short),1,f); //Write length "Arrival Airport"

fwrite((tmp->airport\_arrival).c\_str(),sizeof(char)\*sym,1,f); //Write "A.A."

fwrite(&tmp->free\_places\_agency,sizeof(int),1,f);//Write "free places agency"

tmp=tmp->next\_flight;

if (tmp==NULL)

fl=1;

fwrite(&fl,sizeof(unsigned short),1,f);

} while (tmp!=NULL);

fclose(f);

}

}

if (BeginCustomers==NULL) // If a record dont exist

(MessageDlg("Not exist entities in the list of clients!!\n"

"Client list is not saved.", mtError, TMsgDlgButtons() << mbYes,0));

else {

FILE \*f=NULL;

f = fopen("d\_struct\_kl.dat","wb");

if (f==NULL) { // If you can not create a file

fclose(f);

(MessageDlg("Error creating file!!\nClient list is not saved.",

mtError, TMsgDlgButtons() << mbYes,0));

} else {

struct client \*tmp=BeginCustomers;

unsigned short sym,fl=0;

do {

sym=tmp->surname.Length()+1; //Сalculate length of chars "Surname client"

fwrite(&sym,sizeof(unsigned short),1,f); // Write length "Surname client"

fwrite((tmp->surname).c\_str(),sizeof(char)\*sym,1,f); //Write "Surname client"

sym=tmp->name.Length()+1; //Сalculate length of chars "Name client"

fwrite(&sym,sizeof(unsigned short),1,f); // Write length "Name client"

fwrite((tmp->name).c\_str(),sym,1,f); // Write "Name client"

sym=DateToStr(tmp->departure\_date).Length()+1;//Сalc. length "Departure date"

fwrite(&sym,sizeof(unsigned short),1,f); // Write length "Departure date"

// Write "Departure date"

fwrite((DateToStr(tmp->departure\_date)).c\_str(),sizeof(char)\*sym,1,f);

sym=tmp->resting\_place.Length()+1; // Сalculate length chars "Resting place"

fwrite(&sym,sizeof(unsigned short),1,f); // Write length "Resting place"

fwrite((tmp->resting\_place).c\_str(),sizeof(char)\*sym,1,f); //Write "R. place"

sym=tmp->airport\_arrival.Length()+1; // Сalculate length "Arrival airport"

fwrite(&sym,sizeof(unsigned short),1,f); // Write length "Arrival airport"

fwrite((tmp->airport\_arrival).c\_str(),sizeof(char)\*sym,1,f); // Write "A. a."

tmp=tmp->next\_client;

if (tmp==NULL)

fl=1;

fwrite(&fl,sizeof(unsigned short),1,f);

} while (tmp!=NULL);

fclose(f);

}

}

}

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

/\*

\* Load a backup copy from file (Press 'Menu'->'File'->'Load list copy from file')

\*/

void \_\_fastcall TForm1::N5Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

if (MessageBox(0,"Do you really want to restore the flight list from a file?"

"\n(All current records will be deleted)",

"Attention!", MB\_YESNO) == mrYes) {

FILE \*f=NULL;

f = fopen("d\_struct\_fl.dat","rb");

if (f==NULL) { // If you cannot open the file

fclose(f);

(MessageDlg("Error opening file!\nList of flights is not restored.",

mtError, TMsgDlgButtons() << mbYes,0));

} else {

struct flight \*tmp=BeginFlights, \*buf=NULL;

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

if (tmp!=NULL) { //

while ((tmp->next\_flight)!=NULL) { //

buf=tmp->next\_flight; //

delete tmp; //

tmp=buf; // Delete all entries from RAM

} //

delete tmp; //

BeginFlights=NULL; //

} //

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

unsigned short fl=1;

unsigned short k;

char s[100];

unsigned short sym;

while(!feof(f)) { // Until the end of the file

if (fl) { // If record - first

tmp=new flight;

BeginFlights=tmp;

fl=0;

} else {

tmp->next\_flight=new flight;

tmp=tmp->next\_flight;

}

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->date=StrToDate(AnsiString(s));

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->hour=StrToTime(AnsiString(s));

fread(&tmp->number\_flight,sizeof(unsigned int),1,f);

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->airport\_departure=AnsiString(s);

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->airport\_arrival=AnsiString(s);

fread(&tmp->free\_places\_agency,sizeof(int),1,f);

tmp->next\_flight=NULL;

fread(&k,sizeof(unsigned short),1,f);

if (k) // End-of-file flag

break;

}

fclose(f);

ShowRecords(&BeginFlights);

}

}

if (MessageBox(0,"Do you really want to restore the clients list from file?"

"\n(All current records will be deleted)",

"Attention!", MB\_YESNO) == mrYes) {

FILE \*f=NULL;

f = fopen("d\_struct\_kl.dat","rb");

if (f==NULL) { // If you cannot open the file

fclose(f);

(MessageDlg("Error opening the file!\nClients list are not restored.",

mtError, TMsgDlgButtons() << mbYes,0));

} else {

struct client \*tmp=BeginCustomers, \*buf=NULL;

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

if (tmp!=NULL) { //

while ((tmp->next\_client)!=NULL) { //

buf=tmp->next\_client; //

delete tmp; //

tmp=buf; // Delete all entries from RAM

} //

delete tmp; //

BeginCustomers=NULL; //

} //

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

unsigned short fl=1,k;

char s[100];

unsigned short sym;

while(!feof(f)) { // Until the end of the file

if (fl) { // If record - first

tmp=new client;

BeginCustomers=tmp;

fl=0;

} else {

tmp->next\_client=new client;

tmp=tmp->next\_client;

}

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->surname=AnsiString(s);

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->name=AnsiString(s);

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->departure\_date=StrToDate(AnsiString(s));

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->resting\_place=AnsiString(s);

fread(&sym,sizeof(unsigned short),1,f);

fread(s,sizeof(char)\*sym,1,f);

tmp->airport\_arrival=AnsiString(s);

tmp->number\_participants=0;

tmp->next\_client=NULL;

tmp->pointer\_flight=NULL;

fread(&k,sizeof(unsigned short),1,f);

if (k) // End-of-file flag

break;

}

fclose(f);

ShowRecords(&BeginCustomers);

}

}

}

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

/\*

\* Press 'Menu'->'Sorting'->'List of flights'->'Departure date'

\*/

void \_\_fastcall TForm1::N8Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

flight\_sort(&BeginFlights,1);

ShowRecords(&BeginFlights);

Form1->PageControl1->ActivePageIndex=0;

}

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

/\*

\* Press 'Menu'->'Sorting'->'List of flights'->'Flight number'

\*/

void \_\_fastcall TForm1::N10Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

flight\_sort(&BeginFlights,2);

ShowRecords(&BeginFlights);

Form1->PageControl1->ActivePageIndex=0;

}

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

/\*

\* Press 'Menu'->'Sorting'->'List of flights'->'Departure airport'

\*/

void \_\_fastcall TForm1::N11Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

flight\_sort(&BeginFlights,3);

ShowRecords(&BeginFlights);

Form1->PageControl1->ActivePageIndex=0;

}

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

/\*

\* Press 'Menu'->'Sorting'->'List of flights'->'Arrival airport'

\*/

void \_\_fastcall TForm1::N12Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

flight\_sort(&BeginFlights,4);

ShowRecords(&BeginFlights);

Form1->PageControl1->ActivePageIndex=0;

}

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

/\*

\* Press 'Menu'->'Sorting'->'List of clients'->'Client surname'

\*/

void \_\_fastcall TForm1::N13Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

client\_sort(&BeginCustomers,1);

ShowRecords(&BeginCustomers);

Form1->PageControl1->ActivePageIndex=1;

}

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

/\*

\* Press 'Menu'->'Sorting'->'List of clients'->'Departure date'

\*/

void \_\_fastcall TForm1::N14Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

client\_sort(&BeginCustomers,2);

ShowRecords(&BeginCustomers);

Form1->PageControl1->ActivePageIndex=1;

}

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

/\*

\* Press 'Menu'->'Sorting'->'List of clients'->'Resting place'

\*/

void \_\_fastcall TForm1::N15Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

client\_sort(&BeginCustomers,3);

ShowRecords(&BeginCustomers);

Form1->PageControl1->ActivePageIndex=1;

}

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

/\*

\* Press 'Menu'->'Sorting'->'List of clients'->'Arrival airport'

\*/

void \_\_fastcall TForm1::N16Click(TObject \*Sender) {

if ((Form1->PageControl1->ActivePageIndex==2)||

(Form1->PageControl1->ActivePageIndex==3)) {

MessageDlg("Please, you must exit edit mode recording!",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

client\_sort(&BeginCustomers,4);

ShowRecords(&BeginCustomers);

Form1->PageControl1->ActivePageIndex=1;

}

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

/\*

\* Press 'Menu'->'About'

\*/

void \_\_fastcall TForm1::N3Click(TObject \*Sender) {

MessageDlg("Practical application to the course project on the theme:"

"\n\"Programming technique using dynamic lists\"."

"\n\nThe program is a database of travel agency"

"\nwith two simply connected dynamic lists:\n"

"list of flights and list of clients.\n\n"

"Completed: student group POIT-31;\n"

"Baranov V.", mtInformation, TMsgDlgButtons() << mbYes,0);

}

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

/\*

\* Press 'Menu'->'Exit'

\*/

void \_\_fastcall TForm1::N4Click(TObject \*Sender) {

Form1->Close();

}

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

/\*

\* Sorting records fly list

\*/

void flight\_sort(flight \*\*begin, unsigned int num\_field) {

if ((\*begin)==NULL) {

MessageDlg("Not exist enities in the list flights!!\n"

"(List sorting is not executed)", mtError, TMsgDlgButtons() << mbYes,0);

return;

}

if ((\*begin)->next\_flight==NULL) {

MessageDlg("In fly list only one entity!\n(List sorting is not executed)",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

struct flight \*tmp=(\*begin);

unsigned int n=0,sym;

while (tmp->next\_flight!=NULL) { // Calculating size array

tmp=tmp->next\_flight;

n++;

}

tmp=(\*begin);

struct flight\*\* ar = new struct flight\*[n+1]; // Create array pointers

for (unsigned int i=0;i<n+1;i++,tmp=tmp->next\_flight)

ar[i]=tmp;

\*ar=q\_flight\_sorted(ar,num\_field,n); // Sort by selected field

(\*begin)=ar[0];

tmp=ar[0];

for (unsigned int i=1;i<n+1;i++) {

tmp->next\_flight=ar[i];

ar[i]=NULL; // <--Do not touch it - this is the magic.

tmp=tmp->next\_flight; // Without it does not work

}

tmp->next\_flight=NULL;

delete []ar;

}

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

/\*

\* Sorting records client list

\*/

void client\_sort(client \*\*begin, unsigned int num\_field) {

if ((\*begin)==NULL) {

MessageDlg("Not exist enities in the list clients!!\n"

"(List sorting is not executed)", mtError, TMsgDlgButtons() << mbYes,0);

return;

}

if ((\*begin)->next\_client==NULL) {

MessageDlg("In client list only one entity!\n(List sorting is not executed)",

mtError, TMsgDlgButtons() << mbYes,0);

return;

}

struct client \*tmp=(\*begin);

unsigned int n=0,sym;

while (tmp->next\_client!=NULL) { // Calculating size array

tmp=tmp->next\_client;

n++;

}

tmp=(\*begin);

struct client\*\* art = new struct client\*[n+1]; // Create array pointers

for (unsigned int i=0;i<n+1;i++,tmp=tmp->next\_client)

art[i]=tmp;

\*art=q\_client\_sorted(art,num\_field,n); // Sort by selected field

(\*begin)=art[0];

tmp=art[0];

for (unsigned int i=1;i<n+1;i++) {

tmp->next\_client=art[i];

art[i]=NULL; // <--Do not touch it - this is the magic.

tmp=tmp->next\_client; // Without it does not work

}

tmp->next\_client=NULL;

delete []art;

}

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

/\*

\* Algorithm sorting array [flights] (standard exchange method (bubble))

\*/

flight\* q\_flight\_sorted(struct flight\*\* a,unsigned int r,unsigned int n) {

flight\* buf=NULL;

for (unsigned int i=n;i>0;i--)

for (unsigned int j=0;j<i;j++) {

switch(r) {

case 1:

if (CompareDateTime(a[j]->date,a[j+1]->date)>0) {

buf=a[j];

a[j]=a[j+1];

a[j+1]=buf;

};

break;

case 2:

if ((a[j]->number\_flight)>(a[j+1]->number\_flight)) {

buf=a[j];

a[j]=a[j+1];

a[j+1]=buf;

};

break;

case 3:

if ((a[j]->airport\_departure).AnsiCompareIC((a[j+1]->

airport\_departure))>0) {

buf=a[j];

a[j]=a[j+1];

a[j+1]=buf;

};

break;

case 4:

if ((a[j]->airport\_arrival).AnsiCompareIC((a[j+1]->

airport\_arrival))>0) {

buf=a[j];

a[j]=a[j+1];

a[j+1]=buf;

};

break;

}

}

return \*a;

}

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

/\*

\* Algorithm sorting array [clients] (standard exchange method (bubble))

\*/

client\* q\_client\_sorted(struct client\*\* at,unsigned int r,unsigned int n) {

client\* buf=NULL;

for (unsigned int i=n;i>0;i--)

for (unsigned int j=0;j<i;j++) {

switch(r) {

case 1:

if ((at[j]->surname).AnsiCompareIC((at[j+1]->surname))>0) {

buf=at[j];

at[j]=at[j+1];

at[j+1]=buf;

};

break;

case 2:

if (CompareDateTime(at[j]->departure\_date,at[j+1]->departure\_date)>0) {

buf=at[j];

at[j]=at[j+1];

at[j+1]=buf;

};

break;

case 3:

if ((at[j]->resting\_place).AnsiCompareIC((at[j+1]->resting\_place))>0) {

buf=at[j];

at[j]=at[j+1];

at[j+1]=buf;

};

break;

case 4:

if ((at[j]->airport\_arrival).AnsiCompareIC((at[j+1]->

airport\_arrival))>0) {

buf=at[j];

at[j]=at[j+1];

at[j+1]=buf;

};

break;

}

}

return \*at;

}