МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Российский государственный университет им. А.Н. Косыгина   
(Технологии. Дизайн. Искусство)»**

Институт ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ЦИФРОВЫЕ ТРАНСФОРМАЦИИ

**Отчет по лабораторной работе № 14**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Списки, стеки, очереди»**

Выполнил: Сидоров Д. С., группа ИТС-123

Проверил: к.т.н., доц. Семёнов А. А.

Москва, 2025г

**Списки, стеки, очереди**

Лабораторная работа посвящена изучению ДСД на C++.

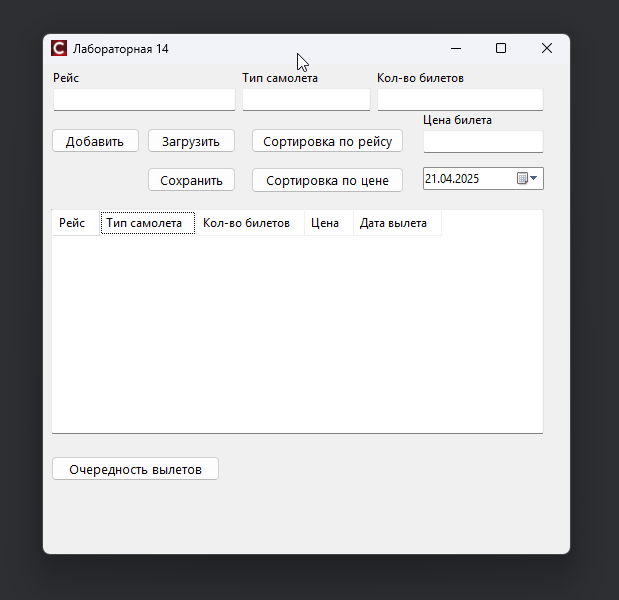
Задание:

Создать приложение, демонстрирующее работу списков, очередей и стеков.

1) Организовать считывание текстового файла с помощью структуры стек.

2) В л/р 7 нужно добавить поле дата вылета. Соответственно, при добавлении рейса необходимо формировать структуру очередь, согласно датам вылета самолетов. Нужно создать кнопку "Очередность вылетов", которая выведет созданную структуру.

*Рисунок 1. Пример кода для последовательного считывания файла.*



*Рисунок 2. Работающее приложение из п. 2*

**Листинг кода**

**14.1-program.cpp**

//-#include <fstream>

#include <iostream>

#include <string>

using namespace std;

struct StackNode {

string data;

StackNode\* next;

};

class TextStack {

private:

StackNode\* top;

public:

TextStack() : top(nullptr) {}

bool isEmpty() {

return top == nullptr;

}

void push(const string& line) {

StackNode\* newNode = new StackNode;

newNode->data = line;

newNode->next = top;

top = newNode;

}

string pop() {

if (isEmpty()) {

return "";

}

StackNode\* temp = top;

string popped = top->data;

top = top->next;

delete temp;

return popped;

}

string peek() {

if (isEmpty()) {

return "";

}

return top->data;

}

~TextStack() {

while (!isEmpty()) {

pop();

}

}

};

int main() {

TextStack fileStack;

string filename;

cout << "Введите имя файла для чтения: ";

cin >> filename;

ifstream inputFile(filename);

if (!inputFile.is\_open()) {

cerr << "Ошибка открытия файла!" << endl;

return 1;

}

string line;

while (getline(inputFile, line)) {

fileStack.push(line);

}

inputFile.close();

cout << "\nСодержимое файла (в обратном порядке):\n";

while (!fileStack.isEmpty()) {

cout << fileStack.pop() << endl;

}

return 0;

}

**Unit1.cpp**

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

#include <vcl.h>

#include <vector>

#include <algorithm>

#include <queue>

#pragma hdrstop

#include "Unit1.h"

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

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

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

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

: TForm(Owner)

{

}

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

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

{

StringGrid1->ColCount = 5;

StringGrid1->RowCount = 1;

StringGrid1->Cells[0][0] = L"Рейс";

StringGrid1->Cells[1][0] = L"Тип самолета";

StringGrid1->Cells[2][0] = L"Кол-во билетов";

StringGrid1->Cells[3][0] = L"Цена";

StringGrid1->Cells[4][0] = L"Дата вылета";

for (int col = 0; col < StringGrid1->ColCount; ++col) {

int maxWidth = StringGrid1->Canvas->TextWidth(StringGrid1->Cells[col][0]) + 20;

StringGrid1->ColWidths[col] = maxWidth;

}

SaveDialog1->Filter = L"Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*";

OpenDialog1->Filter = L"Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*";

}

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

void \_\_fastcall TForm1::ButtonAddClick(TObject \*Sender)

{

try {

Reis r(

LE\_Reis->Text,

LE\_Type->Text,

StrToInt(LE\_Kol->Text),

StrToFloat(LE\_Price->Text),

DateTimePicker1->Date

);

Reis::AddToGrid(StringGrid1, r);

flightQueue.Enqueue(r);

} catch (...) {

ShowMessage(L"Ошибка ввода данных!");

}

}

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

void \_\_fastcall TForm1::ButtonSaveClick(TObject \*Sender)

{

if (SaveDialog1->Execute()) {

Reis::SaveToFile(StringGrid1, SaveDialog1->FileName);

}

}

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

void Reis::AddToGrid(TStringGrid \*grid, const Reis &r)

{

int row = grid->RowCount;

grid->RowCount = row + 1;

grid->Cells[0][row] = r.name;

grid->Cells[1][row] = r.type;

grid->Cells[2][row] = IntToStr(r.kol);

grid->Cells[3][row] = FloatToStrF(r.price, ffFixed, 8, 2);

grid->Cells[4][row] = FormatDateTime("dd.mm.yyyy", r.date);

for (int col = 0; col < grid->ColCount; ++col) {

int textWidth = grid->Canvas->TextWidth(grid->Cells[col][row]) + 20;

if (textWidth > grid->ColWidths[col])

grid->ColWidths[col] = textWidth;

}

}

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

void Reis::SaveToFile(TStringGrid \*grid, String filename)

{

TStringList \*list = new TStringList();

for (int i = 1; i < grid->RowCount; i++) {

String line = grid->Cells[0][i] + "," +

grid->Cells[1][i] + "," +

grid->Cells[2][i] + "," +

grid->Cells[3][i] + "," +

grid->Cells[4][i];

list->Add(line);

}

list->SaveToFile(filename, TEncoding::UTF8);

delete list;

}

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

void \_\_fastcall TForm1::ButtonLoadClick(TObject \*Sender)

{

if (OpenDialog1->Execute()) {

Reis::LoadFromFile(StringGrid1, OpenDialog1->FileName, flightQueue);

}

}

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

void Reis::LoadFromFile(TStringGrid \*grid, String filename, FlightQueue& queue)

{

TStringList \*list = new TStringList();

list->LoadFromFile(filename, TEncoding::UTF8);

// Очищаем grid и очередь перед загрузкой

grid->RowCount = 1;

while (!queue.IsEmpty()) {

queue.Dequeue();

}

for (int i = 0; i < list->Count; ++i) {

TStringList \*cols = new TStringList();

cols->StrictDelimiter = true;

cols->Delimiter = ',';

cols->DelimitedText = list->Strings[i];

if (cols->Count >= 5) {

try {

TDate flightDate = StrToDate(cols->Strings[4]);

Reis r(

cols->Strings[0],

cols->Strings[1],

StrToInt(cols->Strings[2]),

StrToFloat(cols->Strings[3]),

flightDate

);

AddToGrid(grid, r);

queue.Enqueue(r);

}

catch (...) {

}

}

delete cols;

}

delete list;

}

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

void \_\_fastcall TForm1::ButtonSortByNameClick(TObject \*Sender)

{

Reis::SortByName(StringGrid1);

}

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

void Reis::SortByName(TStringGrid \*grid)

{

int rowCount = grid->RowCount;

std::vector<Reis> data;

for (int i = 1; i < rowCount; i++) {

TDate flightDate = StrToDate(grid->Cells[4][i]);

data.emplace\_back(

grid->Cells[0][i],

grid->Cells[1][i],

StrToInt(grid->Cells[2][i]),

StrToFloat(grid->Cells[3][i]),

flightDate

);

}

std::sort(data.begin(), data.end(), [](const Reis &a, const Reis &b) {

return a.name < b.name;

});

grid->RowCount = 1;

for (const auto &r : data)

AddToGrid(grid, r);

}

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

void \_\_fastcall TForm1::ButtonSortByPriceClick(TObject \*Sender)

{

Reis::SortByPrice(StringGrid1);

}

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

void Reis::SortByPrice(TStringGrid \*grid)

{

int rowCount = grid->RowCount;

std::vector<Reis> data;

for (int i = 1; i < rowCount; i++) {

TDate flightDate = StrToDate(grid->Cells[4][i]);

data.emplace\_back(

grid->Cells[0][i],

grid->Cells[1][i],

StrToInt(grid->Cells[2][i]),

StrToFloat(grid->Cells[3][i]),

flightDate

);

}

std::sort(data.begin(), data.end(), [](const Reis &a, const Reis &b) {

return a.price < b.price;

});

grid->RowCount = 1;

for (const auto &r : data)

AddToGrid(grid, r);

}

void \_\_fastcall TForm1::ButtonDepartureClick(TObject \*Sender)

{

if (flightQueue.IsEmpty()) {

ShowMessage(L"Очередь вылетов пуста!");

return;

}

TForm\* queueForm = new TForm(this);

queueForm->Caption = L"Очередь вылетов (отсортировано по дате)";

queueForm->Width = 850;

queueForm->Height = 400;

TStringGrid\* queueGrid = new TStringGrid(queueForm);

queueGrid->Parent = queueForm;

queueGrid->Align = alClient;

queueGrid->ScrollBars = ssBoth;

queueGrid->Options << goColSizing << goRowSelect;

queueGrid->ColCount = 5;

queueGrid->Cells[0][0] = L"Рейс";

queueGrid->Cells[1][0] = L"Тип самолета";

queueGrid->Cells[2][0] = L"Кол-во билетов";

queueGrid->Cells[3][0] = L"Цена";

queueGrid->Cells[4][0] = L"Дата вылета";

flightQueue.DisplayQueue(queueGrid, true);

for (int col = 0; col < queueGrid->ColCount; col++) {

int maxWidth = queueGrid->Canvas->TextWidth(queueGrid->Cells[col][0]) + 20;

for (int row = 1; row < queueGrid->RowCount; row++) {

int textWidth = queueGrid->Canvas->TextWidth(queueGrid->Cells[col][row]) + 20;

if (textWidth > maxWidth) {

maxWidth = textWidth;

}

}

queueGrid->ColWidths[col] = maxWidth;

}

queueForm->ShowModal();

delete queueForm;

}

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

**Unit1.h**

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

#ifndef Unit1H

#define Unit1H

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

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

#include <Vcl.Dialogs.hpp>

#include <Vcl.ExtCtrls.hpp>

#include <Vcl.Grids.hpp>

#include <Vcl.Mask.hpp>

#include <Vcl.WinXPickers.hpp>

#include <Vcl.ComCtrls.hpp>

#include <vector>

#include <algorithm>

#include <queue>

class FlightQueue;

class Reis {

public:

String name;

String type;

int kol;

float price;

TDate date;

Reis(String n, String t, int k, float p, TDate d)

: name(n), type(t), kol(k), price(p), date(d) {}

static void SortByName(TStringGrid \*grid);

static void SortByPrice(TStringGrid \*grid);

static void AddToGrid(TStringGrid \*grid, const Reis &r);

static void SaveToFile(TStringGrid \*grid, String filename);

static void LoadFromFile(TStringGrid \*grid, String filename, FlightQueue& queue);

};

class FlightQueue {

private:

std::queue<Reis> flights;

void FillGridRow(TStringGrid\* grid, int row, const Reis& flight) {

grid->Cells[0][row] = flight.name;

grid->Cells[1][row] = flight.type;

grid->Cells[2][row] = IntToStr(flight.kol);

grid->Cells[3][row] = FloatToStrF(flight.price, ffFixed, 8, 2);

grid->Cells[4][row] = FormatDateTime("dd.mm.yyyy", flight.date);

}

public:

void Enqueue(const Reis& r) {

flights.push(r);

}

Reis Dequeue() {

Reis flight = flights.front();

flights.pop();

return flight;

}

bool IsEmpty() const {

return flights.empty();

}

Reis Peek() const {

return flights.front();

}

void ForEach(std::function<void(const Reis&)> action) const {

std::queue<Reis> temp = flights;

while (!temp.empty()) {

action(temp.front());

temp.pop();

}

}

void DisplayQueue(TStringGrid\* grid, bool sortByDate = false) {

grid->RowCount = 1;

if (sortByDate) {

std::vector<Reis> sortedFlights;

auto temp = flights;

while (!temp.empty()) {

sortedFlights.push\_back(temp.front());

temp.pop();

}

std::sort(sortedFlights.begin(), sortedFlights.end(),

[](const Reis& a, const Reis& b) {

return a.date < b.date;

});

for (size\_t i = 0; i < sortedFlights.size(); i++) {

int row = i + 1;

grid->RowCount = row + 1;

FillGridRow(grid, row, sortedFlights[i]);

}

} else {

auto temp = flights;

int row = 1;

while (!temp.empty()) {

grid->RowCount = row + 1;

FillGridRow(grid, row, temp.front());

temp.pop();

row++;

}

}

}

};

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

class TForm1 : public TForm

{

\_\_published: // IDE-managed Components

TStringGrid \*StringGrid1;

TButton \*ButtonAdd;

TButton \*ButtonSave;

TButton \*ButtonLoad;

TButton \*ButtonSortByName;

TButton \*ButtonSortByPrice;

TLabeledEdit \*LE\_Reis;

TLabeledEdit \*LE\_Type;

TLabeledEdit \*LE\_Kol;

TLabeledEdit \*LE\_Price;

TOpenDialog \*OpenDialog1;

TSaveDialog \*SaveDialog1;

TButton \*ButtonDeparture;

TDateTimePicker \*DateTimePicker1;

void \_\_fastcall FormCreate(TObject \*Sender);

void \_\_fastcall ButtonAddClick(TObject \*Sender);

void \_\_fastcall ButtonSaveClick(TObject \*Sender);

void \_\_fastcall ButtonLoadClick(TObject \*Sender);

void \_\_fastcall ButtonSortByNameClick(TObject \*Sender);

void \_\_fastcall ButtonSortByPriceClick(TObject \*Sender);

void \_\_fastcall ButtonDepartureClick(TObject \*Sender);

private:

FlightQueue flightQueue; // User declarations

public: // User declarations

\_\_fastcall TForm1(TComponent\* Owner);

};

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

extern PACKAGE TForm1 \*Form1;

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

#endif

**Вывод:** В ходе выполнения данной лабораторной работы были успешно реализованы основные динамические структуры данных (ДСД): стек, очередь и список, что позволило на практике изучить их функциональные особенности и области применения. Первая часть задания была выполнена путем создания механизма для считывания текстового файла с использованием структуры стек, что продемонстрировало принцип LIFO (Last In, First Out) и позволило эффективно обрабатывать содержимое файла. Во второй части работы класс Reis из лабораторной работы №7 был модифицирован добавлением поля "дата вылета", что расширило функциональность исходного приложения. На основе этих данных была успешно реализована структура очереди, которая автоматически формируется при добавлении новых рейсов с учетом хронологического порядка дат вылета. Созданная функциональность кнопки "Очередность вылетов" позволяет наглядно отобразить сформированную структуру, демонстрируя принцип FIFO (First In, First Out), характерный для очередей. Практическая реализация данных структур дала возможность глубже понять механизмы работы с динамической памятью в C++, включая выделение и освобождение памяти, организацию связей между элементами структур и алгоритмы обработки данных в различных типах ДСД. Полученные в ходе работы навыки являются фундаментальными для дальнейшего изучения более сложных алгоритмов и структур данных в программировании.