Содержание

[**Теория** 2](#_Toc136595649)

[Структуры 2](#_Toc136595650)

[Сортировка «пузырьком» 2](#_Toc136595651)

[**Задание** 3](#_Toc136595652)

[**Блок-схемы алгоритмов** 4](#_Toc136595653)

[Функция «readPlanesFromFile» 5](#_Toc136595654)

[Функция «ErrorHandler» 6](#_Toc136595655)

[Функция «BubbleSort» 7](#_Toc136595656)

[**Код программы** 8](#_Toc136595657)

[Laba3.cpp 8](#_Toc136595658)

[**Тестирование программы** 12](#_Toc136595659)

[1. Тестирование некорректных режимов 12](#_Toc136595660)

[2. Тестирование корректных режимов 13](#_Toc136595661)

[**Вывод** 14](#_Toc136595662)

**Теория**

Структуры

Структуры в C++ — это пользовательские типы данных, которые могут содержать различные переменные разных типов внутри себя. Они позволяют объединять данные, относящиеся к одному объекту или концепции в единый блок. Структуры могут содержать поля и функции-члены, которые могут работать с этими полями.

Сортировка «пузырьком»

Сортировка пузырьком — это простой алгоритм сортировки, который последовательно сравнивает и меняет местами соседние элементы в массиве до тех пор, пока все элементы не будут упорядочены.

Принцип работы сортировки пузырьком:

1. Начинается сравнение первого и второго элементов массива. Если первый элемент больше второго, то они меняются местами.
2. Затем сравниваются второй и третий элементы, и так далее, до конца массива. Если в результате сравнения элементы меняются местами, то считается, что была выполнена одна итерация.
3. После первой итерации на последнем месте окажется наибольший элемент массива.
4. Процесс повторяется для оставшихся элементов массива (от первого до n-1), пока все элементы не будут упорядочены.

Пример кода:

void bubbleSort(int arr[], int size) {

for (int i = 0; i < size - 1; i++) {

// Внутренний цикл для сравнения и перестановки соседних элементов

for (int j = 0; j < size - i - 1; j++) {

// Сравниваем два соседних элемента

if (arr[j] > arr[j + 1]) {

// Если левый элемент больше правого, меняем их местами

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

**Задание**

**ВАРИАНТ № 10**

В процессе функционирования АСУ ВД в файле фиксируются данные о самолетах, совершивших посадку. Каждая запись имеет структуру типа:

**473 ТУ-154М Б-3726 3**

номер марка ЛА бортовой пункт

рейса номер прибытия

1) подготовить программу, сортирующую записи с использованием индексной сортировки методом «пузырька» в порядке возрастания номеров рейсов; результаты печатать в виде таблицы;

2) обеспечить входной контроль бортового номера, номера рейса и пункта прибытия, выполнить отладку и тестирование.

Чтение данных их файла производить с использованием функций ввода/вывода языка С++.

Алгоритм должен быть параметризован; обмен данными с подпрограммой должен осуществляться только через параметры; исходные данные хранятся в отдельном файле.

**Блок-схемы алгоритмов**

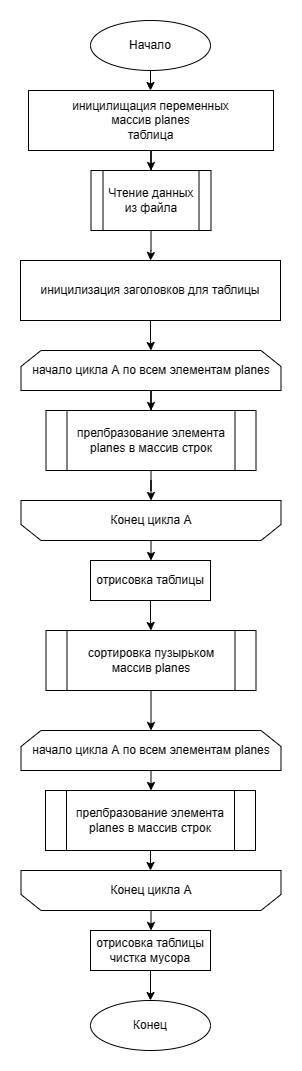


Рис. 1 - Блок-схема функции "laba3"

Функция «readPlanesFromFile»

1. Назначение

Считывание данных из файла и заполнение массива записей полетов

1. Прототип функции:

char readPlanesFromFile(Plane\*& planes, int& size)

1. Обращение:

ret = readPlanesFromFile(planes, size);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной\выходной |
| planes | Plane\*& | Ссылка на цказатель самолетов | Входной |
| size | int& | Сылка на размер массива | Входной |
| result | char | Результат работы функции | выходной |

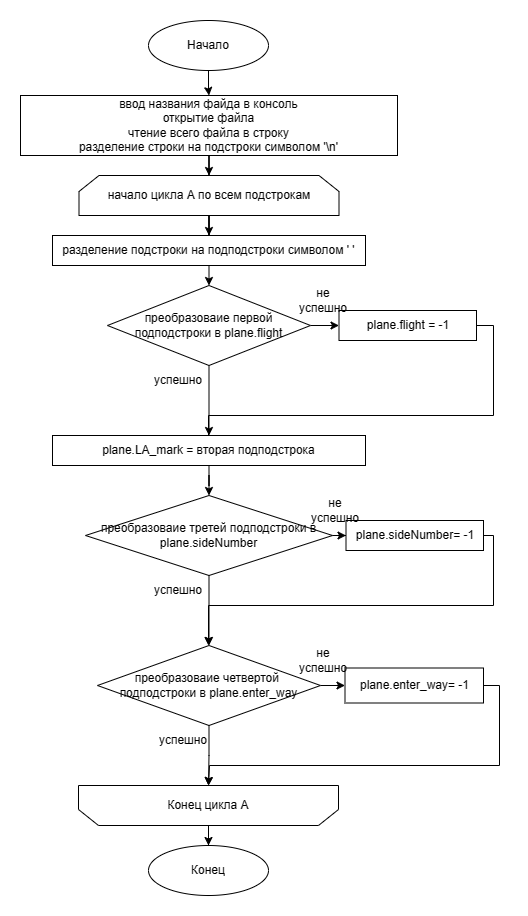


Рис. 2 - Блок-схема функции "readPlanesFromFile"

Функция «ErrorHandler»

1. Назначение

Преобразует самолёт в массив строк

1. Прототип функции

char errorHandler(Plane\*& planes, int& real\_size, String\*\* planesStr, int size,

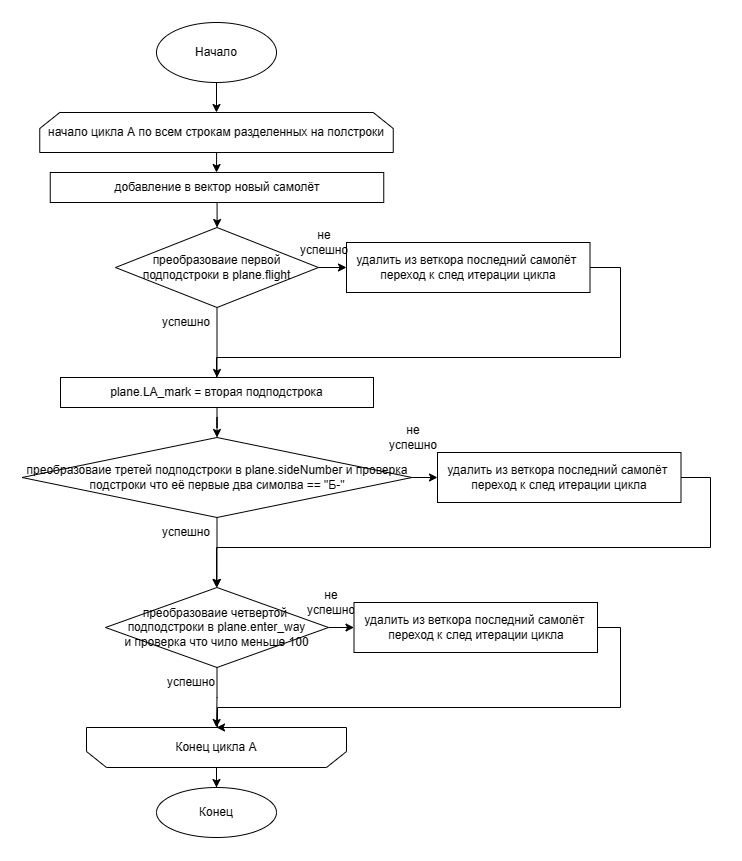
int\* cell\_size)

1. Обращение

ret = errorHandler(planeVector, planeVectorSize, string\_table, size, cells\_size)

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной\выходной |
| planeVector | Plane& | Ссылка на массив самолёт | Входной |
| planeVectorSize | int& | Ссылка на размер самолётов | Входной |
| String\_table | String\*\* | Массив массивов строк | Входной |
| size | int | Размер пролого массива | Входной |
| cells\_size | int\* | Размер всех массиво внтури массива | Входной |
| result | char | Результат функции | Выходной |

**

*Рис. 3 - Блок-схема функции "*ErrorHandler*"*

Функция «BubbleSort»

1. Назначение

Сортирует индексный массив по высотным эшелонам методом пузырька

1. Прототип функции

void BubbleSort(indexItem \*sortArray, int amountCorrectFlights);

1. Обращение

BubbleSort(sortArray, amountCorrectFlights);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной\выходной |
| sortArray | indexItem[] | Индексный массив | Выходной |
| size | int | Размер массива | Входной |



Рис. 4 - Блок-схема функции "BubbleSort"

**Код программы**

Laba3.cpp

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* КАФЕДРА № 304 1 КУРС \*

\*---------------------------------------------------------------\*

\* Project Type : Win32 Console Application \*

\* Project Name : Laba3 \*

\* File Name : Laba3.cpp \*

\* Language : C/C++ \*

\* Programmer(s) : Закусилов Л.З., \*

\* Modifyed By : \*

\* Lit source : \*

\* Created : 29/03/23 \*

\* Last Revision : 01/06/23 \*

\* Comment(s) : сортировка собсвенныхз структур \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "Laba3.h"

#include "String.h"

#include "Table.h"

#include <stdio.h>

#include <fcntl.h>

#include <io.h>

//структура самолёта

struct Plane

{

int flight = 0; //номер рейса

String LA\_mark = ""; //марка ЛА

int side\_number = 0; //бортовой номер

int enter\_way = 0; //пункт прибытия

};

//конец стурктуры

String\* PlaneToString(Plane& plane)

{

String\* result = new String[4];

if (plane.flight == -1) {

result[0] = "ERR";

}

else {

String::IntegerToString(result[0], plane.flight);

}

result[1] = plane.LA\_mark;

if (plane.side\_number == -1) {

result[2] = "ERR";

}

else {

String::IntegerToString(result[2], plane.side\_number);

result[2].add(0, 1041);

result[2].add(1, '-');

}

if (plane.enter\_way == -1) {

result[3] = "ERR";

}

else {

String::IntegerToString(result[3], plane.enter\_way);

}

return result;

}

//Функция чтения данных из файла

char readPlanesFromFile(Plane\*& planes, int& size)

{

int ret = 0;

String file\_name = "laba3.txt";

//clear planes

if (planes != nullptr) {

delete[] planes;

}

//input file name

printf(" write file name: ");

ret = writeText(file\_name);

if (ret) {

return 1;

}

//read data from file and echo print

String buffer;

ret = readFromFile(buffer, file\_name.c\_str());

if (ret) {

return 1;

}

//prepare data

String\* lines = nullptr;

ret = buffer.split(lines, size, '\n');

planes = new Plane[size];

String\* line = nullptr;

int number\_of\_components = 0;

//calculate every line

for (int i = 0; i < size; i++)

{

//split line

String str(lines[i]);

ret = str.split(line, number\_of\_components, ' ');

//check for number of componetnts

if (number\_of\_components < 4)

{

printf("too few arguments in line: '%d'\n", i + 1);

}

if (number\_of\_components > 4)

{

printf("too many arguments in line: '%d'\n", i + 1);

}

//check flight

int temp;

ret = String(line[0]).toInteger(temp);

planes[i].flight = temp;

if (ret) {

planes[i].flight = -1;

printf("'flight' corrupted on line '%d'\n", i + 1);

}

//check LA\_mark

planes[i].LA\_mark = line[1];

//check side\_number

String sideNumber = line[2];

if (line[2][0] != 1041 || line[2][1] != '-')

{

planes[i].side\_number = -1;

printf("'side number' corrupted on line '%d'\n", i + 1);

}

else {

sideNumber.remove(0);

sideNumber.remove(0);

ret = sideNumber.toInteger(temp);

planes[i].side\_number = temp;

if (ret) {

planes[i].side\_number = -1;

printf("'side number' corrupted on line '%d'\n", i + 1);

}

}

//check enter\_way

ret = String(line[3]).toInteger(temp);

planes[i].enter\_way = temp;

if (ret) {

planes[i].enter\_way = -1;

printf("'enter way' corrupted on line '%d'\n", i + 1);

}

}

return 0;

}

//Конец функции

//сортировка пузыркём

char bubbleSort(Plane\* planes, unsigned int start, unsigned int end)

{

char flag = 1;

for (unsigned int i = start; i < end - 1; i++)

{

flag = 1;

for (unsigned int j = start; j < end - i - 1; j++)

{

if (planes[j].flight > planes[j + 1].flight)

{

Plane temp = planes[j + 1];

planes[j + 1] = planes[j];

planes[j] = temp;

flag = 0;

}

}

if (flag)

{

break;

}

}

return 0;

}

//Конец сортировки пузыркём

//начало алгоритма

int laba3()

{

//иницилизация переменных

char ret = 0; //переменная для опознования ошибок

int size = 0; //переменнная хранящие колисечтво структур в файле

Plane\* planes = nullptr; //динамичесуий массив структур

String\*\* t; //массив массивов строк для таблицы

//конец иницилизации переменных

//чтнение дыннх из файла

ret = readPlanesFromFile(planes, size);

if (ret) {

return -1;

}

t = new String\*[size + 1];

t[0] = new String[4];

t[0][0] = L"Номер рейса";

t[0][1] = L"Марка ЛА";

t[0][2] = L"Бортовой номер";

t[0][3] = L"Пункт прибытия";

//эчо печать

for (int i = 0; i < size; i++)

{

t[i + 1] = PlaneToString(planes[i]);

}

//конец эхо печати

Table a(t, 4, size + 1);

a.drawTable();

//вызов функции сортирвоки

ret = insertionSort(planes, 0, size);

if (ret) {

return -1;

}

//печать отсортированых структур

for (int i = 0; i < size; i++)

{

t[i + 1] = PlaneToString(planes[i]);

}

a.drawTable();

//конец печати

//чистка мусора

if (planes != nullptr)

{

delete[] planes;

}

//конец чистки мусора

return 0;

}//конец программы

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of main.cpp file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Тестирование программы**

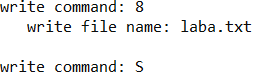
1. Тестирование некорректных режимов

1. Проверить работу программы, если файла с исходными данными не существует

Ожидаемый результат:

Продолжение работы программы

Результат:

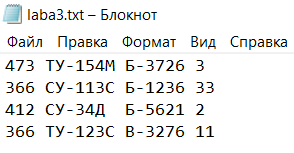


1. Проверить работу программы, если в файле даны некорректные данные

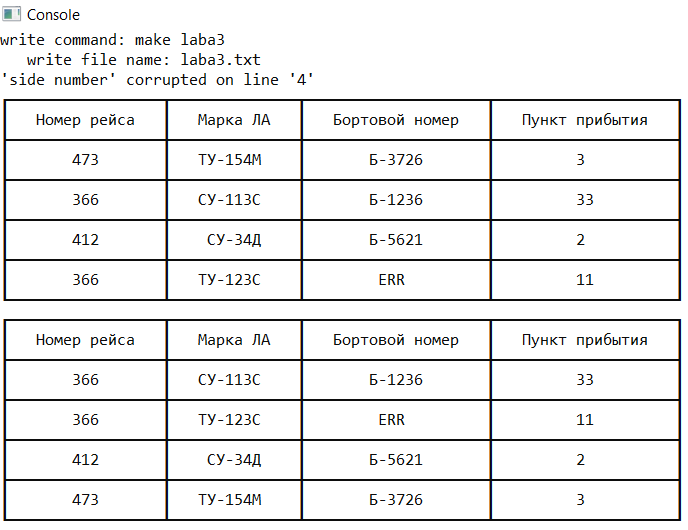
Ожидаемый результат:

Вывод сообщения об ошибке

Исходные данные:

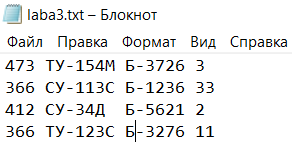
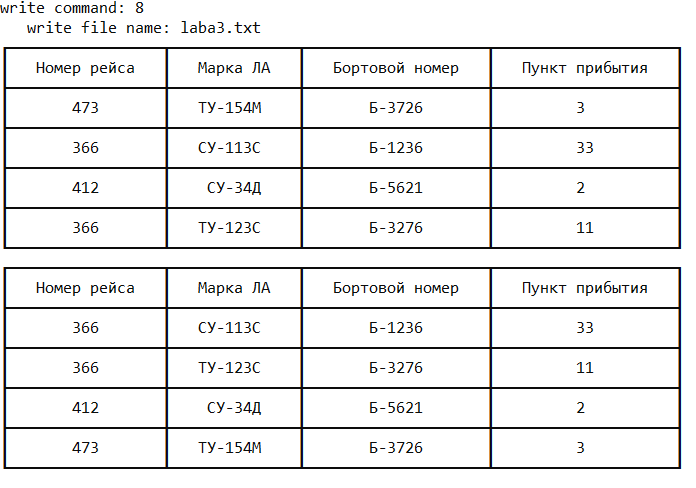


Результат:



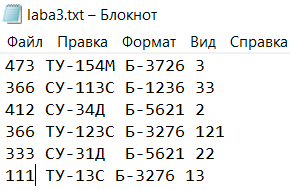
2. Тестирование корректных режимов

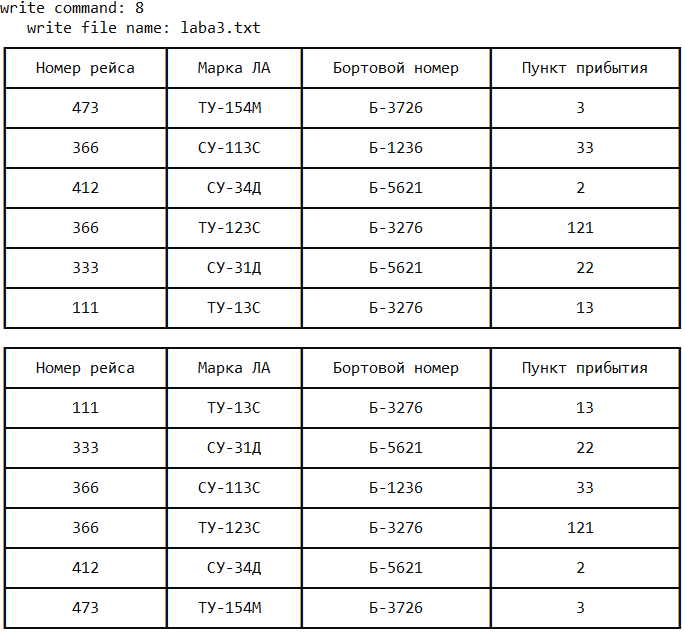
1. Исходные данные:



Результат:

1. Исходные данные





Результат:

**Вывод**

Написал программу, сортирующую записи с использованием индексной сортировки методом пузырька, а также обеспечил входной контроль номера рейса и высоты летательного аппарата.

Чтение данных из файла произвел с использованием функций ввода/вывода языка С++.

Алгоритм параметризован; обмен данными с подпрограммой осуществил через параметры; каждый из наборов исходных данных хранится в отдельном файле.