|  |  |
| --- | --- |
| https://lh7-us.googleusercontent.com/QuBaagabzZYLr7U3ZbYOtMusd5FbWWxIvMUTN8jrLHzSLMg534z9gXRTIG1Us4i_lOwmWlaBxKedNt-SQ26dm4WmyqwjGDmEO6z8GE3QrZosqvHM88J2EFeVf1u0GzyCZQlhWmp1Zeo85tKo4LJVXQ | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» |

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Отчёт по практике. Вариант №12  
по учебной дисциплине «Программирование»

на тему «Работа со структурами данных»

Выполнил  
студент группы М3О-119БВ-24

Нарзиев А.Т.

Приняла

Ст. преподаватель каф. 304

Давыдкина Елена Александровна

Москва  
 2024

# Содержание

[Содержание 2](#_Toc196439019)

[Задание 3](#_Toc196439020)

[Решение 4](#_Toc196439021)

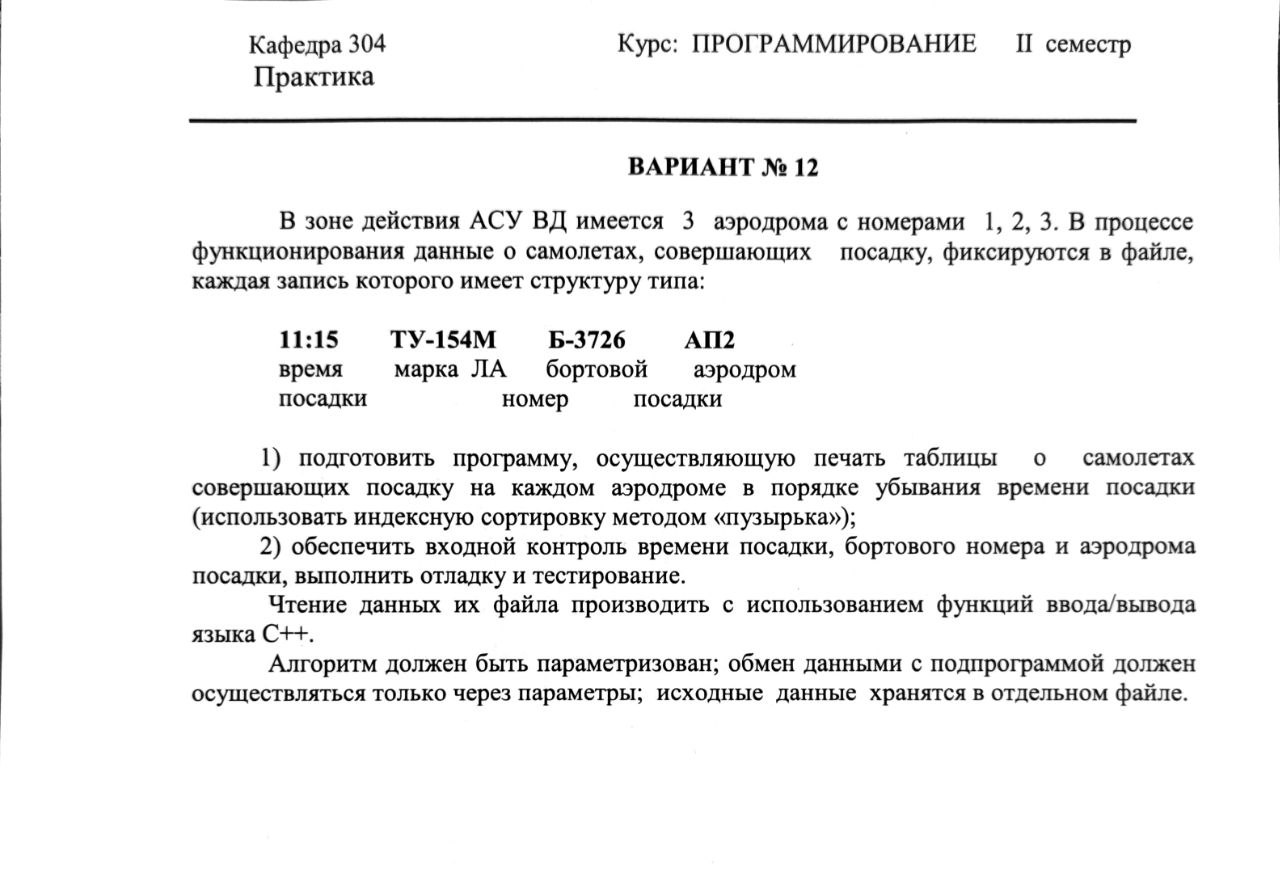
[1. Блок-схема: 4](#_Toc196439022)

[2. Код программы: 5](#_Toc196439023)

[Тесты 10](#_Toc196439024)

[Вывод 11](#_Toc196439025)

# Задание



# Решение

## Блок-схема:

## Код программы:

### Главный файл – main.cpp:

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

\*                               Курс Информатика                              \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Project type  : Linux Console Application                                   \*

\* Project name  : Lab\_2                                                       \*

\* File name     : main.cpp                                                    \*

\* Language      : CPP                                                         \*

\* Programmers   : Шалаев Александр Максимович, Нарзиев Артемий Тимурович      \*

\* Modified By   :                                                             \*

\* Created       : 02.04.2025                                                  \*

\* Last Revision : 10.04.2025                                                  \*

\* Comment       : Обработка данных о посадках. Вариант: 6                     \*

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

#include <iostream>

#include <fstream>

#include <iomanip>

// Для локали на русском

#include <locale>

#include <codecvt>

#include <windows.h>

using namespace std;

// КОНСТАНТЫ

const int MAX\_PLANES = 100;

const int TIME\_LEN = 6;

const int MODEL\_LEN = 20;

const int BORT\_LEN = 7;

const int AIRPORT\_LEN = 4;

// СТРУКТУРА ДАННЫХ

struct Plane {

    char time[TIME\_LEN];       // HH:MM

    int minutes;               // время в минутах

    char model[MODEL\_LEN];     // модель самолета

    char bort[BORT\_LEN];       // бортовой номер

    char airport[AIRPORT\_LEN]; // аэродром

};

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

\*  ПРОТОТИПЫ ФУНКЦИЙ

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

void print\_error(int err, const char\* field, int line\_num);

int read\_data(const char \*filename, Plane \*planes, int \*count);

void process\_airport(Plane \*planes, int count, const char \*airport);

void purify(char\* field);

int is\_time\_valid(const char \*time, int line\_num);

int is\_bort\_valid(const char \*bort, int line\_num);

int is\_airport\_valid(const char \*airport, int line\_num);

void bubble\_sort(int \*indices, int size, Plane \*data);

void print\_table(int \*indices, int size, Plane \*data, const char \*airport);

//ТЕСТЫ

// Активный тест (для быстрой проверки)

const char \*FILE\_NAME = "tests/correct/test1.txt";

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

\*  ГЛАВНАЯ ФУНКЦИЯ

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

int main() {

    // Установка кодировки консоли для корректного отображения символов

    SetConsoleOutputCP(CP\_UTF8);

    setlocale(LC\_ALL, "en\_US.UTF-8");

    Plane planes[MAX\_PLANES];

    int count = 0;

    const char \*airports[] = {"AP1", "AP2", "AP3"};

    int err = read\_data(FILE\_NAME, planes, &count);

    if (err == -1) {

        print\_error(err, FILE\_NAME, 0);

        return 1;

    }

    // Всегда обрабатываем данные, если count > 0

    for (int i = 0; i < 3; i++) {

        process\_airport(planes, count, airports[i]);

    }

    return 0;

}

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

\*  ИНИЦИАЛИЗАЦИЯ ФУНКЦИЙ

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

/\*\*

 \* Выводит сообщение об ошибке с указанием типа и местоположения.

 \*/

void print\_error(int err, const char\* field, int line\_num) {

    cout << "[31m";

    if (err == -1)      cout << "File \"" << field << "\" not found";

    else if (err == -2) cout << "Invalid time format in line " << line\_num;

    else if (err == -3) cout << "Invalid bort number in line " << line\_num;

    else if (err == -4) cout << "Invalid airport code in line " << line\_num;

    else if (err == -5) cout << "Empty model in line " << line\_num;

    else if (err == -6) cout << "Extra data in line " << line\_num;

    cout << "[0m\n";

}

/\*\*

 \* Читает данные о посадках из файла, парсит и валидирует их.

 \*/

int read\_data(const char \*filename, Plane \*planes, int \*count) {

    ifstream file(filename);

    if (!file.is\_open()) return -1;

    char line[100];

    int line\_num = 0;

    bool has\_errors = false;

    while (file.getline(line, 100) && \*count < MAX\_PLANES) {

        line\_num++;

        Plane p = {};

        int pos = 0;

        bool is\_valid = true;

        // ПАРСИНГ И ВАЛИДАЦИЯ ПОЛЕЙ

        for (int i = 0; i < 4; i++) {

            char\* dest = i == 0 ? p.time : (i == 1 ? p.model : (i == 2 ? p.bort : p.airport));

            int max\_len = i == 0 ? TIME\_LEN-1 : (i == 1 ? MODEL\_LEN-1 : (i == 2 ? BORT\_LEN-1 : AIRPORT\_LEN-1));

            int j = 0;

            while (line[pos] && line[pos] != ',' && j < max\_len) {

                if (line[pos] != ' ') dest[j++] = line[pos];

                pos++;

            }

            dest[j] = '\0';

            while (line[pos] == ',' || line[pos] == ' ') pos++;

        }

        // ПРОВЕРКА НА ЛИШНИЕ ДАННЫЕ

        if (line[pos] != '\0') {

            print\_error(-6, filename, line\_num);

            has\_errors = true;

            is\_valid = false;

        }

        // НОРМАЛИЗАЦИЯ

        purify(p.time);

        purify(p.model);

        purify(p.bort);

        purify(p.airport);

        // ПОСЛЕДОВАТЕЛЬНЫЕ ПРОВЕРКИ

        if (p.model[0] == '\0' && is\_valid) {

            print\_error(-5, filename, line\_num);

            is\_valid = false;

        }

        if (is\_time\_valid(p.time, line\_num) != 0 && is\_valid) {

            print\_error(-2, filename, line\_num);

            is\_valid = false;

        }

        if (is\_bort\_valid(p.bort, line\_num) != 0 && is\_valid) {

            print\_error(-3, filename, line\_num);

            is\_valid = false;

        }

        if (is\_airport\_valid(p.airport, line\_num) != 0 && is\_valid) {

            print\_error(-4, filename, line\_num);

            is\_valid = false;

        }

        if (!is\_valid) {

            has\_errors = true;

            continue;

        }

        // КОНВЕРТАЦИЯ ВРЕМЕНИ

        p.minutes = ((p.time[0]-'0')\*10 + (p.time[1]-'0'))\*60 +

                    ((p.time[3]-'0')\*10 + (p.time[4]-'0'));

        planes[(\*count)++] = p;

    }

    file.close();

    return has\_errors ? -7 : 0; // Код -7: есть ошибки, но данные обработаны

}

/\*\*

 \* Нормализует поле: удаляет пробелы и переводит символы в верхний регистр.

 \*/

void purify(char\* field) {

    char\* dst = field;

    for (; \*field; field++) {

        if (\*field != ' ' && \*field != '\t') \*dst++ = toupper(\*field);

    }

    \*dst = '\0';

}

/\*\*

 \* Проверяет корректность формата времени (HH:MM).

 \*/

int is\_time\_valid(const char \*time, int line\_num) {

    if (time[2] != ':' || time[5] != '\0') return -2;

    for (int i = 0; i < 5; i++) {

        if (i == 2) continue;

        if (time[i] < '0' || time[i] > '9') return -2;

    }

    int h = (time[0]-'0')\*10 + (time[1]-'0');

    int m = (time[3]-'0')\*10 + (time[4]-'0');

    return (h >= 0 && h < 24 && m >= 0 && m < 60) ? 0 : -2;

}

/\*\*

 \* Проверяет формат бортового номера (X-XXXX, где X — буква/цифра).

 \*/

int is\_bort\_valid(const char \*bort, int line\_num) {

    if (bort[1] != '-' || bort[6] != '\0') return -3;

    if (bort[0] < 'A' || bort[0] > 'Z') return -3;

    for (int i = 2; i < 6; i++) {

        if (bort[i] < '0' || bort[i] > '9') return -3;

    }

    return 0;

}

/\*\*

 \* Проверяет код аэродрома (допустимые значения: AP1, AP2, AP3).

 \*/

int is\_airport\_valid(const char \*airport, int line\_num) {

    const char valid[3][4] = {"AP1", "AP2", "AP3"};

    for (int i = 0; i < 3; i++) {

        bool match = true;

        for (int j = 0; j < 3; j++) {

            if (airport[j] != valid[i][j]) match = false;

        }

        if (match) return 0;

    }

    return -4;

}

/\*\*

 \* Сортирует индексы записей по времени посадки (от поздних к ранним).

 \*/

void bubble\_sort(int \*indices, int size, Plane \*data) {

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

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

            if (data[indices[j]].minutes < data[indices[j+1]].minutes) {

                int temp = indices[j];

                indices[j] = indices[j+1];

                indices[j+1] = temp;

            }

        }

    }

}

/\*\*

 \* Форматирует и выводит таблицу с данными для указанного аэродрома.

 \*/

void print\_table(int \*indices, int size, Plane \*data, const char \*airport) {

    if (size == 0) {

        cout << "Airport " << airport << ": no landings\n";

        return;

    }

    cout << "\nAirport " << airport << ":\n";

    cout << "┌────────────┬───────────────┬────────────────┬────────────────┐\n";

    cout << "│ Time       │ Model         │ Bort Number    │ Airport        │\n";

    cout << "├────────────┼───────────────┼────────────────┼────────────────┤\n";

    for (int i = 0; i < size; i++) {

        const Plane &p = data[indices[i]];

        cout << "│ " << left << setw(11) << p.time << "│ "

             << setw(14) << p.model << "│ "

             << setw(15) << p.bort << "│ "

             << setw(15) << p.airport << "│\n";

    }

    cout << "└────────────┴───────────────┴────────────────┴────────────────┘\n";

}

/\*\*

 \* Обрабатывает данные для конкретного аэродрома: фильтрация, сортировка, вывод.

 \* @param planes Массив структур Plane.

 \* @param count Общее количество записей.

 \* @param airport Код аэродрома.

 \*/

void process\_airport(Plane \*planes, int count, const char \*airport) {

    int indices[MAX\_PLANES];

    int size = 0;

    for (int i = 0; i < count; i++) {

        bool match = true;

        for (int j = 0; j < 3; j++) {

            if (planes[i].airport[j] != airport[j]) match = false;

        }

        if (match) indices[size++] = i;

    }

    bubble\_sort(indices, size, planes);

    print\_table(indices, size, planes, airport);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* КОНЕЦ main.cpp \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## Описание функций

### Функция print\_error

Назначение

Выводит сообщение об ошибке с указанием типа и местоположения.

Прототип

void print\_error(int err, const char\* field, int line\_num);

Обращение к функции:

print\_error(-1, "data.txt", 0);

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| err | int | Код ошибки (от -1 до -6). | Входной |
| field | const char\* | Название файла или проблемное поле. | Входной |
| line\_num | int | Номер строки с ошибкой. | Входной |

### Функция read\_data

Назначение

Читает данные о посадках из файла, парсит и валидирует их.

Прототип

int read\_data(const char \*filename, Plane \*planes, int \*count);

Обращение к функции:

Plane planes[MAX\_PLANES];

int count = 0;

int err = read\_data("tests/correct/test1.txt", planes, &count);

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| filename | const char\* | Имя файла с данными. | Входной |
| planes | Plane\* | Массив структур Plane для заполнения. | Выходной |
| count | int\* | Указатель на количество успешно считанных записей. | Входной/Выходной |

### Функция purify

Назначение

Нормализует поле: удаляет пробелы и переводит символы в верхний регистр.

Прототип

void purify(char\* field);

Обращение к функции:

char model[MODEL\_LEN] = " Boeing-747 ";

purify(model);

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| field | char\* | Указатель на строку для обработки. | Входной/Выходной |

### Функция is\_time\_valid

Назначение

Проверяет корректность формата времени (HH:MM).

Прототип

int is\_time\_valid(const char \*time, int line\_num);

Обращение к функции:

const char \*time = "25:70";

int valid = is\_time\_valid(time, 1);

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| time | const char\* | Строка времени. | Входной |
| line\_num | int | Номер строки (для вывода ошибки). | Входной |

### Функция is\_bort\_valid

Назначение

Проверяет формат бортового номера (X-XXXX, где X — буква/цифра).

Прототип

int is\_bort\_valid(const char \*bort, int line\_num);

Обращение к функции:

const char \*bort = "XYZ-12";

int valid = is\_bort\_valid(bort, 2);

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| bort | const char\* | Бортовой номер. | Входной |
| line\_num | int | Номер строки (для вывода ошибки). | Входной |

### Функция is\_airport\_valid

Назначение

Проверяет код аэродрома (AP1, AP2, AP3).

Прототип

int is\_airport\_valid(const char \*airport, int line\_num);

Обращение к функции:

const char \*airport = "INV";

int valid = is\_airport\_valid(airport, 3);

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| airport | const char\* | Код аэродрома. | Входной |
| line\_num | int | Номер строки (для вывода ошибки). | Входной |

### Функция bubble\_sort

Назначение

Сортирует индексы записей по времени посадки (от поздних к ранним).

Прототип

void bubble\_sort(int \*indices, int size, Plane \*data);

Обращение к функции:

int indices[MAX\_PLANES] = {0, 1, 2};

int size = 3;

bubble\_sort(indices, size, planes);

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| indices | int\* | Массив индексов для сортировки. | Входной/Выходной |
| size | int | Размер массива. | Входной |
| data | Plane\* | Массив структур Plane с данными. | Входной |

### Функция print\_table

Назначение

Форматирует и выводит таблицу с данными для указанного аэродрома.

Прототип

void print\_table(int \*indices, int size, Plane \*data, const char \*airport);

Обращение к функции:

int indices[MAX\_PLANES] = {0, 1, 2};

int size = 3;

print\_table(indices, size, planes, "AP1");

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| indices | int\* | Массив индексов записей. | Входной |
| size | int | Количество записей. | Входной |
| data | Plane\* | Массив структур Plane. | Входной |
| airport | const char\* | Код аэродрома. | Входной |

### Функция process\_airport

Назначение

Обрабатывает данные для конкретного аэродрома: фильтрация, сортировка, вывод.

Прототип

void process\_airport(Plane \*planes, int count, const char \*airport);

Обращение к функции:

process\_airport(planes, count, "AP2");

Описание параметров:

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| planes | Plane\* | Массив структур Plane. | Входной |
| count | int | Общее количество записей. | Входной |
| airport | const char\* | Код аэродрома. | Входной |

# Тесты

## Корректные тесты

## Корректные тесты

##### Тест №1

Цель теста:Проверить обработку данных с корректным форматом и заполнением.

###### Исходные данные:

Файл: test1.txt

08:00,AN-24,d-1122,AP1

14:45,Airbus-A320,B-5678,AP1

12:30,Boeing-777,A-1234,AP1

10:30,LickIt-172,C-1122,AP2

16:20,Boeing-747,D-3344,AP2

12 :45,SuperJet,I -9012,AP2

07:00,Gymbro-190,E-5566,AP3

13:15,Bombardiro,F-7788,AP3

20:05,Crocodilo,G-9900,AP3

###### Ожидаемый результат:

Аэродром AP1

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 14:45 | AIRBUS-A320 | B-5678 | AP1 |
| 12:30 | BOEING-777 | A-1234 | AP1 |
| 08:00 | AN-24 | D-1122 | AP1 |

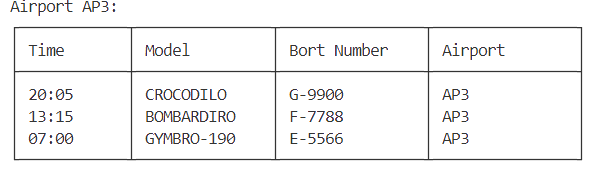
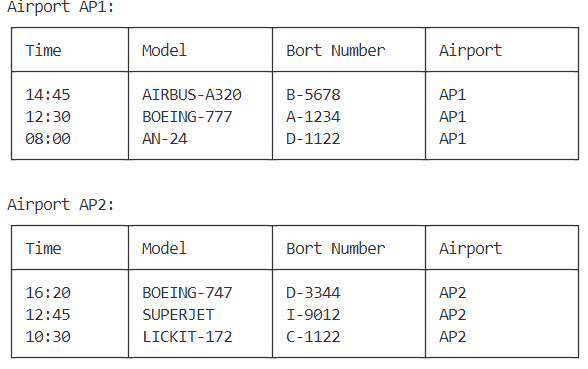
Аэродром AP2

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 16:20 | BOEING-747 | D-3344 | AP2 |
| 12:45 | SUPERJET | I-9012 | AP2 |
| 10:30 | LICKIT-172 | C-1122 | AP2 |

Аэродром AP3

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 20:05 | CROCODILO | G-9900 | AP3 |
| 13:15 | BOMBARDIRO | F-7788 | AP3 |
| 07:00 | GYMBRO-190 | E-5566 | AP3 |

###### Полученный результат



Вывод: Программа обработала данные корректно. Тест пройден.

##### Тест №2

Цель: Проверить обработку данных с лишними пробелами в полях.

###### Исходные данные:

Файл: test2.txt

###### 12:30,BOEING-747,A-1234,AP1

###### 08:15,AIRBUS-A320,B-5678,AP2

###### 23:59, LIRILI-172 , C-9999, AP3

###### 23:00, LARILA-172 , C-8888, AP3

###### 20:00, Sahur-911 , b-1000, AP3

###### Ожидаемый результат:

Аэродром AP1

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 12:30 | BOEING-747 | A-1234 | AP1 |

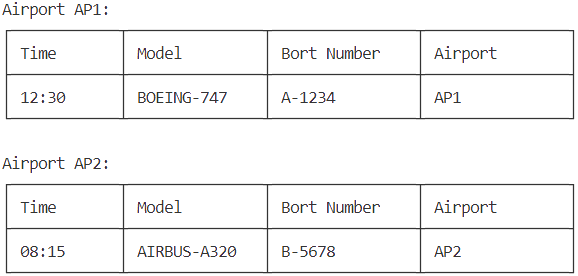
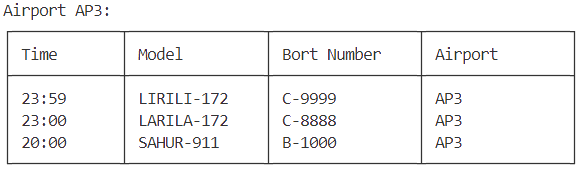
Аэродром AP2

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 08:15 | AIRBUS-A320 | B-5678 | AP2 |

Аэродром AP3

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 23:59 | LIRILI-172 | C-9999 | AP3 |
| 23:00 | LARILA-172 | C-8888 | AP3 |
| 20:00 | SAHUR-911 | B-1000 | AP3 |

###### Полученный результат:

Вывод: Программа корректно обработала данные, игнорируя лишние поля. Тест пройден.

Тест №3

Цель: Проверить обработку времени 23:59.

###### Исходные данные:

Файл: test3.txt

23:59,CONCORDE,C-9999,AP3

12:30,Boeing-777,A-1234,AP1

###### Ожидаемый результат:

Аэродром AP1

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 12:30 | BOEING-777 | A-1234 | AP1 |

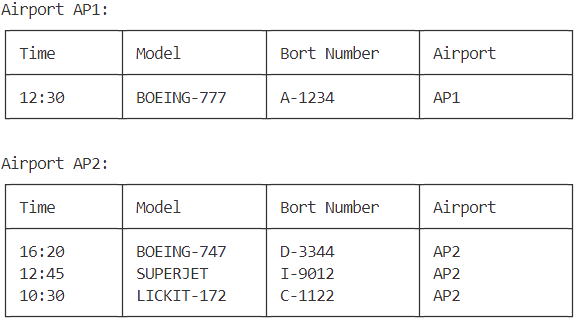
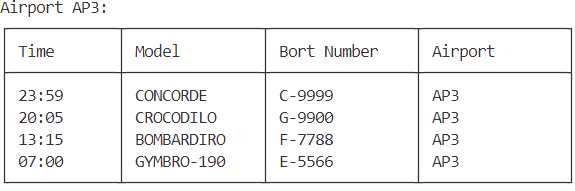
Аэродром AP2

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 16:20 | BOEING-747 | D-3344 | AP2 |
| 12:45 | SUPERJET | I-9012 | AP2 |
| 10:30 | LICKIT-172 | C-1122 | AP2 |

Аэродром AP3

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 23:59 | CONCORDE | C-9999 | AP3 |
| 20:05 | CROCODILO | G-9900 | AP3 |
| 13:15 | BOMBARDIRO | F-7788 | AP3 |
| 07:00 | GYMBRO-190 | E-5566 | AP3 |

###### Полученный результат:

Вывод: Граничное время обработано корректно. Тест пройден.

##### Тест №4

Цель: Проверить обработку времени 00:00 и минимальных данных.

###### Исходные данные:

Файл: test4.txt

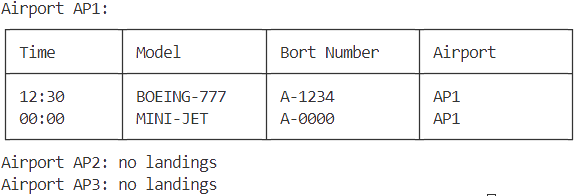
00:00,MINI-JET,A-0000,AP1

###### Ожидаемый результат:

Аэродром AP1

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 12:30 | BOEING-777 | A-1234 | AP1 |
| 00:00 | MINI-JET | A-0000 | AP1 |

###### Полученный результат:



Вывод: Минимальные значения обработаны без ошибок. Тест пройден.

##### Тест №5

Цель: Проверить вывод сообщения no landings для аэродрома без данных.

###### Исходные данные:

Файл: test5.txt

“Пустой Файл”

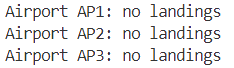
###### Ожидаемый результат:

Airport AP1: no landings

Airport AP2: no landings

Airport AP3: no landings

###### Полученный результат:



Вывод: Программа корректно обработала отсутствие данных. Тест пройден.

##### Тест №6

Цель: Проверить вывод значений в таблице с дефисами.

###### Исходные данные:

Файл: test5.txt

###### 10:00,BOEING-747-MAX,C-9999,AP3

###### 15:30,SUPERJET-100,D-8888,AP2

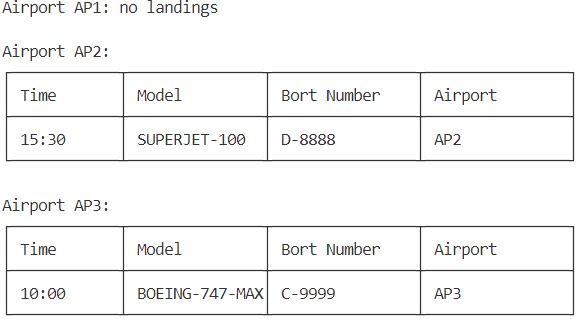
###### Ожидаемый результат:

Airport AP1: no landings

Airport AP2: no landings

Airport AP3: no landings

###### Полученный результат:



Вывод: Программа корректно обработала отсутствие данных. Тест пройден.

# Вывод