|  |  |
| --- | --- |
| 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)

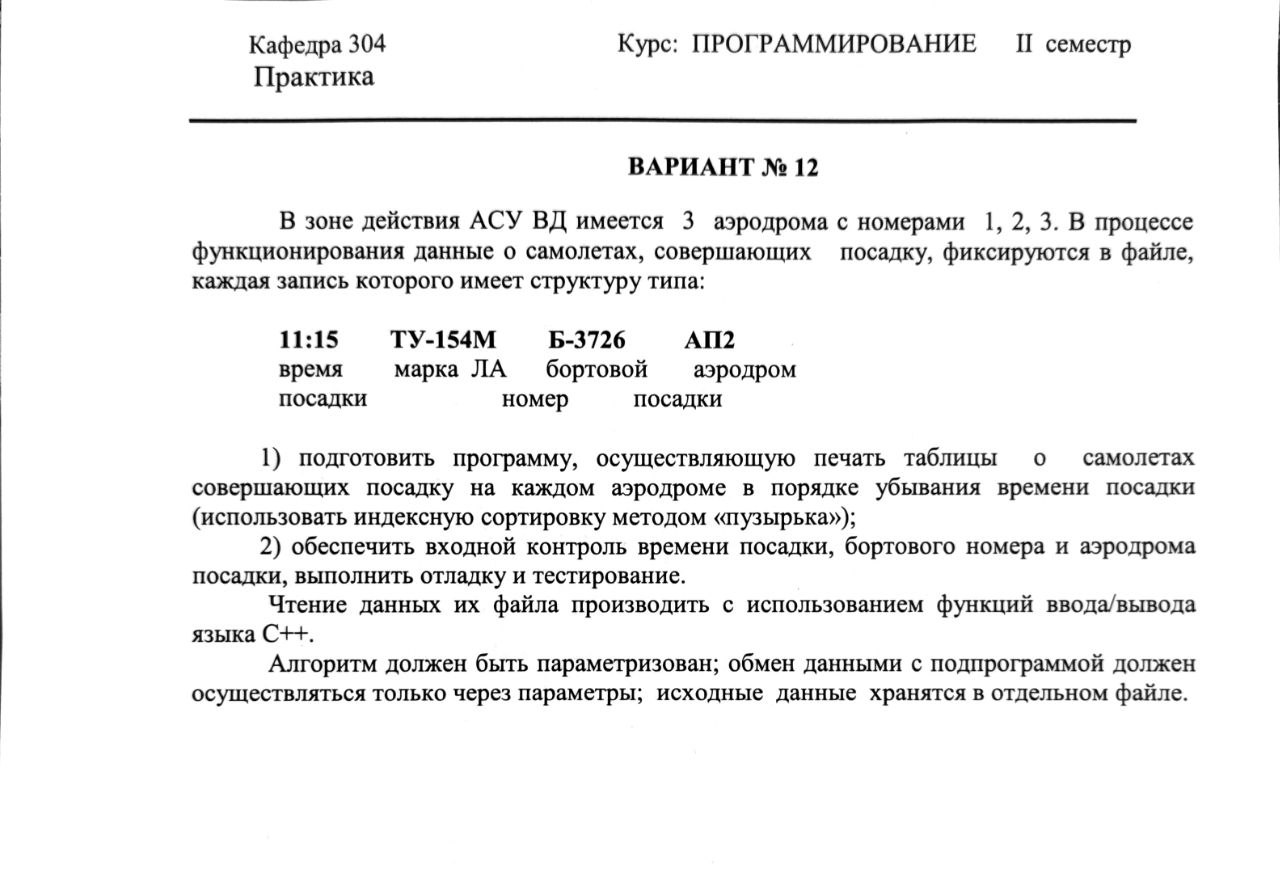
[1. Блок-схема: 4](#_Toc196439022)

[2. Код программы: 5](#_Toc196439023)

[Тесты 10](#_Toc196439024)

[Вывод 11](#_Toc196439025)

# Задание



# Введение

Программа предназначена для обработки данных о посадках самолетов на трех аэродромах (AP1, AP2, AP3). Она выполняет следующие функции:  
- Чтение данных из файла в формате: Время,Модель,Бортовой\_номер,Аэродром  
- Валидацию данных (проверка корректности времени, бортового номера и кода аэродрома)  
- Сортировку записей по времени посадки в порядке убывания  
- Форматированный вывод результатов для каждого аэродрома

Программа написана на C++ с использованием структур, функций, файлового ввода-вывода и алгоритмов сортировки.

## **Основные теоретические аспекты**

### Структуры в C++

Структура — это составной тип данных, который объединяет несколько переменных разных типов под одним именем.

###### Объявление структуры

struct Plane {  
 char time[6]; // Время посадки (HH:MM)  
 int minutes; // Время в минутах для сортировки  
 char model[20]; // Модель самолета  
 char bort[7]; // Бортовой номер (формат: X-XXXX)  
 char airport[4]; // Код аэродрома (AP1, AP2, AP3)  
};

* Каждый элемент структуры называется **полем**.
* Поля могут быть разных типов (char[], int и т. д.).

###### Доступ к полям структуры

Plane p; // Создаем объект структуры  
strcpy(p.time, "12:30"); // Записываем время  
cout << p.time; // Читаем время

### Индексная сортировка

Индексная сортировка — это метод сортировки, при котором вместо перемещения самих данных сортируются их индексы.

###### Преимущества:

* Экономит память (не нужно копировать большие структуры)
* Сохраняет исходный порядок данных

###### Пример работы:

Допустим, есть массив структур Plane planes[3] с временами посадки:  
1. planes[0].minutes = 125  
2. planes[1].minutes = 80  
3. planes[2].minutes = 300

Шаги индексной сортировки:

1. Создаем массив индексов: int indices[] = {0, 1, 2}  
2. Сортируем индексы по planes[indices[i]].minutes  
3. После сортировки: indices = {2, 0, 1} (по убыванию)

###### Реализация в коде:

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) {  
 swap(indices[j], indices[j + 1]); // Меняем индексы местами  
 }  
 }  
 }  
}

### Сортировка пузырьком

Алгоритм:  
1. Последовательно сравниваются соседние элементы.  
2. Если порядок неправильный — они меняются местами.  
3. Процесс повторяется, пока массив не будет отсортирован.

**Пример:**  
Исходный массив: [5, 3, 8, 1]  
1. **Первый проход:**  
- 5 > 3 → меняем → [3, 5, 8, 1]  
- 5 < 8 → не меняем  
- 8 > 1 → меняем → [3, 5, 1, 8]  
2. **Второй проход:**  
- 3 < 5 → не меняем  
- 5 > 1 → меняем → [3, 1, 5, 8]  
3. **Третий проход:**  
- 3 > 1 → меняем → [1, 3, 5, 8]

**Время работы:**  
- Худший случай: **O(n²)**  
- Лучший случай (уже отсортирован): **O(n)**

### Валидация данных

Программа проверяет:  
1. **Время посадки** (HH:MM):  
- Должно быть в формате ЧЧ:ММ (например, 23:59)  
- Часы: 00-23, минуты: 00-59  
2. **Бортовой номер** (X-XXXX):  
- Первый символ — буква (A-Z)  
- Затем дефис и 4 цифры (0-9)  
3. **Аэродром** (AP1, AP2, AP3):  
- Только эти три варианта

**Пример проверки времени:**

bool is\_time\_valid(const char \*time) {  
 if (strlen(time) != 5 || time[2] != ':') return false;  
 int hh = (time[0] - '0') \* 10 + (time[1] - '0');  
 int mm = (time[3] - '0') \* 10 + (time[4] - '0');  
 return (hh >= 0 && hh < 24 && mm >= 0 && mm < 60);  
}

### Пример сортировки

Рассмотрим массив времен посадки в минутах:

Исходные данные: [125, 80, 300, 45]

Индексы: [0, 1, 2, 3]

###### Первая итерация:

* + Сравниваем 0 и 1: 125 > 80 - порядок верный
  + Сравниваем 1 и 2: 80 < 300 - меняем местами
  + Массив: [125, 300, 80, 45]

###### Вторая итерация:

* + Сравниваем 0 и 1: 125 < 300 - меняем местами
  + Сравниваем 1 и 2: 300 > 80 - порядок верный
  + Массив: [300, 125, 80, 45]

###### Третья итерация:

* + Сравниваем 0 и 1: 300 > 125 - порядок верный
  + Массив остается без изменений

###### Результат:

Отсортированные индексы: [2, 0, 1, 3]

Соответствующие времена: [300, 125, 80, 45]

# 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;    // Максимальное количество записей о самолетах

const int TIME\_LEN = 6;        // Длина строки времени (HH:MM + нуль-терминатор)

const int MODEL\_LEN = 20;      // Максимальная длина модели самолета

const int BORT\_LEN = 7;        // Длина бортового номера (X-XXXX + нуль-терминатор)

const int AIRPORT\_LEN = 4;     // Длина кода аэродрома (APX + нуль-терминатор)

// СТРУКТУРА ДАННЫХ

struct Plane {

    char time[TIME\_LEN];       // Время посадки в формате HH:MM

    int minutes;               // Время в минутах для сортировки

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

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

    char airport[AIRPORT\_LEN]; // Код аэродрома

};

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

\*  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

const char \*FILE\_NAME = "tests/correct/test3.txt";    // 23:59,CONCORDE,C-9999,AP3

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

\*  ГЛАВНАЯ ФУНКЦИЯ

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

int main() {

    // Установка кодировки консоли для корректного отображения символов

    SetConsoleOutputCP(CP\_UTF8);

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

    // Основные переменные программы

    Plane planes[MAX\_PLANES];  // Массив для хранения данных о самолетах

    int count = 0;             // Счетчик успешно загруженных записей

    const char \*airports[] = {"AP1", "AP2", "AP3"}; // Список аэродромов для обработки

    // Чтение данных из файла

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

    if (err == -1) {

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

        return 1;

    }

    // Обработка данных для каждого аэродрома

    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.

 \* @param err Код ошибки (от -1 до -6).

 \* @param field Название файла или проблемное поле.

 \* @param line\_num Номер строки с ошибкой.

 \* @note Цвет сообщения: красный (использует ANSI-коды).

 \*/

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 << " (skipped)";

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

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

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

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

    else if (err == -8) cout << "Empty line " << line\_num << " (skipped)";

    cout << "[0m\n";

}

/\*\*

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

 \* @param filename Имя файла с данными.

 \* @param planes Массив структур Plane для заполнения.

 \* @param count Указатель на количество успешно считанных записей.

 \* @return 0 при успехе, -1 если файл не найден, -7 при наличии ошибок в данных.

 \* @note Формат строки: "HH:MM,Модель,Бортовой\_номер,Аэродром".

 \*/

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

    // Открытие файла

    ifstream file(filename);

    if (!file.is\_open()) return -1;

    // Переменные для обработки файла

    char line[100];         // Буфер для чтения строки

    int line\_num = 0;       // Счетчик строк

    bool has\_errors = false; // Флаг наличия ошибок

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

        line\_num++;

        Plane p = {};       // Временная структура для хранения данных

        int pos = 0;       // Позиция в строке

        bool is\_valid = true; // Флаг валидности строки

        // Проверка на пустую строку

        bool is\_empty = true;

        for (int i = 0; line[i] != '\0'; i++) {

            if (line[i] != ' ' && line[i] != '\t') {

                is\_empty = false;

                break;

            }

        }

        if (is\_empty) {

            print\_error(-8, filename, line\_num);

            has\_errors = true;

            continue;

        }

        // ПАРСИНГ ПОЛЕЙ

        for (int field\_num = 0; field\_num < 4 && is\_valid; field\_num++) {

            // Определение целевого поля и его максимальной длины

            char\* dest = NULL;

            int max\_len = 0;

            int j = 0;

            if (field\_num == 0) {

                dest = p.time;

                max\_len = TIME\_LEN-1;

            } else if (field\_num == 1) {

                dest = p.model;

                max\_len = MODEL\_LEN-1;

            } else if (field\_num == 2) {

                dest = p.bort;

                max\_len = BORT\_LEN-1;

            } else {

                dest = p.airport;

                max\_len = AIRPORT\_LEN-1;

            }

            // Пропуск пробелов перед полем

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

            // Чтение поля до запятой или конца строки

            j = 0;

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

                dest[j++] = line[pos++];

            }

            dest[j] = '\0';

            // Пропуск запятой и пробелов после поля

            if (line[pos] == ',') {

                pos++;

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

            }

        }

        // НОРМАЛИЗАЦИЯ

        purify(p.time);

        purify(p.model);

        purify(p.bort);

        purify(p.airport);

        // ВАЛИДАЦИЯ ПОЛЕЙ

        if (p.model[0] == '\0') {

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

            is\_valid = false;

        }

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

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

            is\_valid = false;

        }

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

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

            is\_valid = false;

        }

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

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

            is\_valid = false;

        }

        else {

            // Проверка на лишние данные только если все поля валидны

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

            if (line[pos] != '\0') {

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

                is\_valid = false;

            }

        }

        if (!is\_valid) {

            has\_errors = true;

            continue;

        }

        // КОНВЕРТАЦИЯ ВРЕМЕНИ И СОХРАНЕНИЕ

        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;

}

/\*\*

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

 \* @param field Указатель на строку для обработки.

 \* @note Изменяет исходную строку.

 \*/

void purify(char\* field) {

    // Указатели для чтения и записи

    char\* read\_ptr = field;

    char\* write\_ptr = field;

    // Обработка каждого символа

    while (\*read\_ptr) {

        if (\*read\_ptr != ' ' && \*read\_ptr != '\t') {

            \*write\_ptr = toupper(\*read\_ptr);

            write\_ptr++;

        }

        read\_ptr++;

    }

    \*write\_ptr = '\0';

}

/\*\*

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

 \* @param time Строка времени.

 \* @param line\_num Номер строки (для вывода ошибки).

 \* @return 0 при корректном формате, -2 при ошибке.

 \*/

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

    // Проверка формата

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

    // Проверка цифр

    for (int i = 0; i < 5; i++) {

        if (i == 2) continue;

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

    }

    // Проверка диапазонов часов и минут

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

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

    return (hours >= 0 && hours < 24 && minutes >= 0 && minutes < 60) ? 0 : -2;

}

/\*\*

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

 \* @param bort Бортовой номер.

 \* @param line\_num Номер строки (для вывода ошибки).

 \* @return 0 при корректном формате, -3 при ошибке.

 \*/

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

    // Проверка длины и формата

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

    // Проверка первой буквы

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

    // Проверка цифр

    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).

 \* @param airport Код аэродрома.

 \* @param line\_num Номер строки (для вывода ошибки).

 \* @return 0 при корректном коде, -4 при ошибке.

 \*/

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

    // Объявление всех переменных в начале

    int len = 0; // Длина поля аэропоорта

    int i, j; // Счетчики

    bool match; // Флаг для добавления самолетов в соответств. аэропорт

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

    // Проверка длины

    while (airport[len] != '\0' && len < 4) len++;

    if (len != 3) return -4;

    // Проверка допустимых значений

    for (i = 0; i < 3; i++) {

        match = true;

        for (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).

 \* @param indices Массив индексов для сортировки.

 \* @param size Размер массива.

 \* @param data Массив структур Plane с данными.

 \*/

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

    int temp; // Временная переменная для обмена

    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) {

                temp = indices[j];

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

                indices[j+1] = temp;

            }

        }

    }

}

/\*\*

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

 \* @param indices Массив индексов записей.

 \* @param size Количество записей.

 \* @param data Массив структур Plane.

 \* @param airport Код аэродрома.

 \* @note Если записей нет, выводит "no landings".

 \*/

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

    // Проверка на отсутствие данных

    if (size == 0) {

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

        return;

    }

    // Вывод заголовка таблицы

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

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

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

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

    // Вывод данных

    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";

    }

    // Вывод нижней границы таблицы

    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) {

    // Массив для хранения индексов подходящих записей

    int indices[MAX\_PLANES];

    int size = 0; // Количество подходящих записей

    bool match; // Флаг для добавления самолетов в соответств. аэропорт

    // Фильтрация записей по аэродрому

    for (int i = 0; i < count; i++) {

        match = true;

        for (int j = 0; j < 3; j++) {

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

        }

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

    }

    // Сортировка и вывод

    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

## Некорректные тесты

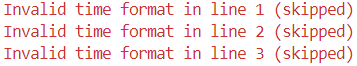
Вот полностью оформленные тесты в требуемом формате:

##### Тест 1

**Цель:** Проверить реакцию на некорректный формат времени  
**Исходные данные:**  
Файл: test1.txt

25:30,BOEING-747,A-1234,AP1  
12:60,AIRBUS-A320,B-5678,AP2  
12-30,CESSNA-172,C-9999,AP3

**Ожидаемый результат:**  
Invalid time format in line 1  
Invalid time format in line 2  
Invalid time format in line 3

**Полученный результат:**  


**Вывод:** Все ошибки формата времени корректно обнаружены. Тест пройден.

##### Тест 2

**Цель:** Проверить валидацию бортового номера  
**Исходные данные:**  
Файл: test2.txt

12:30,BOEING-747,1234-56,AP1  
08:15,AIRBUS-A320,B-ABCD,AP2  
23:59,CESSNA-172,C-999,AP3

**Ожидаемый результат:**  
Invalid bort number in line 1  
Invalid bort number in line 2  
Invalid bort number in line 3

**Полученный результат:**  

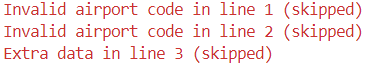

**Вывод:** Все ошибки в бортовых номерах корректно обнаружены. Тест пройден.

##### Тест 3

**Цель:** Проверить обработку неверных кодов аэродрома  
**Исходные данные:**  
Файл: test3.txt

12:30,BOEING-747,A-1234,APX  
08:15,AIRBUS-A320,B-5678,AP  
23:59,CESSNA-172,C-9999,AP33

**Ожидаемый результат:**  
Invalid airport code in line 1  
Invalid airport code in line 2  
Invalid airport code in line 3

**Полученный результат:**  


**Вывод:** Все ошибки в кодах аэродромов корректно обнаружены. Тест пройден.

##### Тест 4

**Цель:** Проверить обработку отсутствия модели самолета  
**Исходные данные:**  
Файл: test4.txt

12:30,,A-1234,AP1  
08:15, ,B-5678,AP2  
23:59,CESSNA-172,C-9999,AP3

**Ожидаемый результат:**  
Empty model in line 1  
Empty model in line 2

**Полученный результат:**  


**Вывод:** Пустые модели корректно обнаружены. Тест пройден.

##### Тест 5

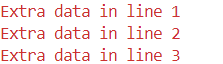
**Цель:** Проверить обработку строк с избыточными данными  
**Исходные данные:**  
Файл: test5.txt   
20:59, TUNG-172 , C-9999, AP3 ,EXTRA\_DATA

00:59, TUNG-192 , C-1000, AP2 ,EXTRA\_DATA

23:59, Sahur-911 , b-1000, AP3 ,EXTRA\_DATA

**Ожидаемый результат:**  
Extra data in line 1

Extra data in line 2  
Extra data in line 3

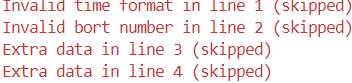
**Полученный результат:**  


**Вывод:** Лишние данные корректно обнаружены. Тест пройден.

##### Тест 6

**Цель:** Проверить обработку файла с комбинацией ошибок  
**Исходные данные:**  
Файл: test6.txt  
25:30,BOEING-747,A-1234,AP1  
12:30,,A-1234,AP1  
08:15,AIRBUS-A320,1234-56,APX  
23:59,CESSNA-172,C-9999,AP3,EXTRA

**Ожидаемый результат:**  
Invalid time format in line 1  
Empty model in line 2  
Invalid bort number in line 3  
Invalid airport code in line 3  
Extra data in line 4

**Полученный результат:**  


**Вывод:** Все типы ошибок корректно обработаны. Тест пройден.

##### Тест 7

**Цель:** Проверить обработку граничных значений времени  
**Исходные данные:**  
Файл: test7.txt  
00:00,MINI-JET,A-0000,AP1  
23:59,JUMBO-JET,A-9999,AP3  
24:00,GHOST-PLANE,X-0000,AP2

**Ожидаемый результат:**  
Invalid time format in line 3

**Полученный результат:**  


**Вывод:** Граничные значения времени корректно проверяются. Тест пройден.

##### Тест 8

**Цель:** Проверить обработку пустых строк  
**Исходные данные:**  
Файл: test9.txt  
12:30,BOEING-747,A-1234,AP1

23:59,CESSNA-172,C-9999,AP3

**Ожидаемый результат:**  
Пустые строки игнорируются но с уведомлением.

**Полученный результат:**  


**Вывод:** Пустые строки корректно игнорируются. Тест пройден.

## 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 |

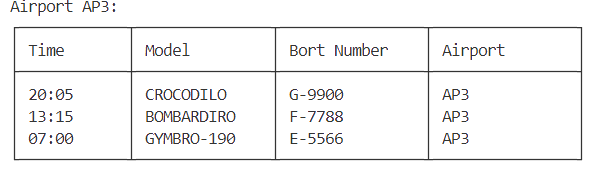
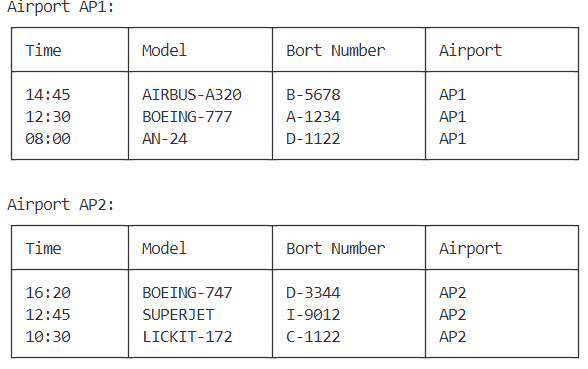
Аэродром AP2

| 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 |

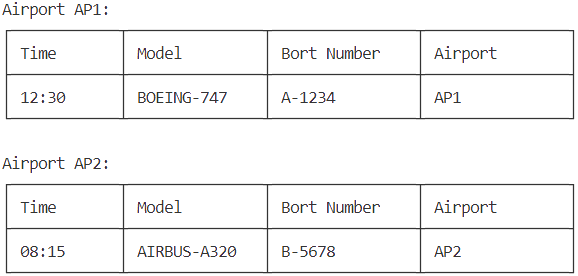
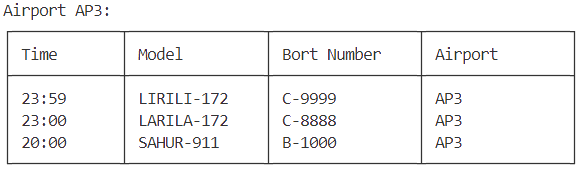
Аэродром AP2

| 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 |

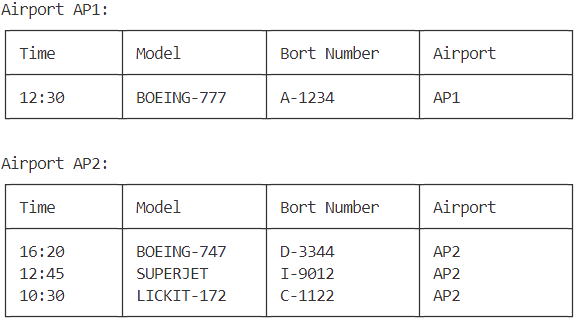
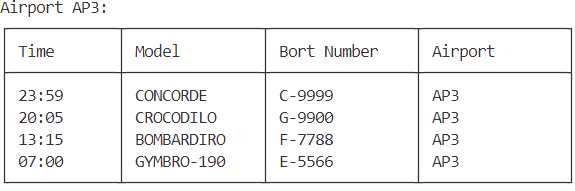
Аэродром AP2

| 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

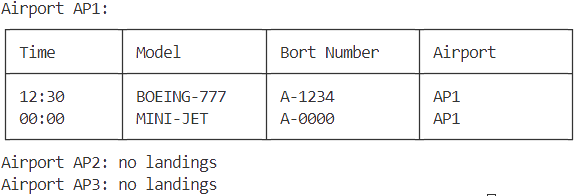
00:00,MINI-JET,A-0000,AP1

###### 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”

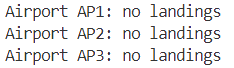
###### Ожидаемый результат:

Airport AP1: no landings

Airport AP2: no landings

Airport AP3: no landings

###### Полученный результат:



Вывод: Программа корректно обработала отсутствие данных. Тест пройден.

##### Тест №6

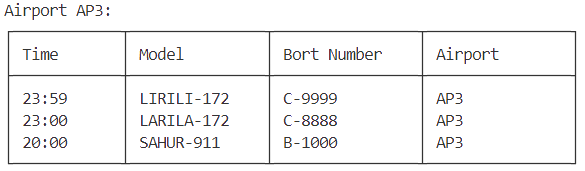
Цель: Нормализация значений с дефисами и лишними пробелами.  
Исходные данные:  
Файл: test6.txt

23:59, LIRILI-172 , C-9999, AP3   
23:00, LARILA-172 , C-8888, AP3   
20:00, Sahur-911 , b-1000, AP3

###### Ожидаемый результат:

| 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: Данные с дефисами в модели и бортовом номере корректно нормализованы. Тест пройден.

##### Тест №7

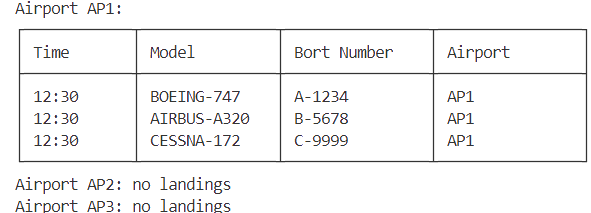
Цель: Проверить сортировку при одинаковом времени посадки.  
Исходные данные:  
Файл: test7.txt

12:30,BOEING-747,A-1234,AP1   
12:30,AIRBUS-A320,B-5678,AP1   
12:30,CESSNA-172,C-9999,AP1

###### Ожидаемый результат:

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 12:30 | BOEING-747 | A-1234 | AP1 |
| 12:30 | AIRBUS-A320 | B-5678 | AP1 |
| 12:30 | CESSNA-172 | C-9999 | AP1 |

Полученный результат:



Вывод: При одинаковом времени записи выводятся в порядке их следования в файле. Тест пройден.

##### Тест №8

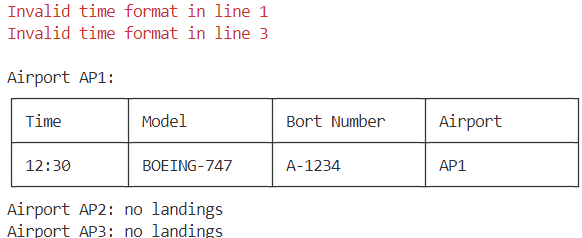
Цель: Проверить обработку файла с одной корректной записью среди ошибок.  
Исходные данные:  
Файл: test8.txt

INVALID\_DATA   
12:30,BOEING-747,A-1234,AP1   
EMPTY\_LINE

###### Ожидаемый результат:

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

Полученный результат:



Вывод: Корректная запись обработана, ошибки проигнорированы с выводом соответствующих сообщений. Тест пройден.

##### Тест №9

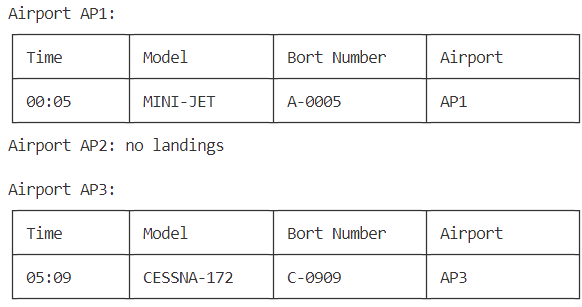
Цель: Проверить обработку времени с ведущими нулями.  
Исходные данные:  
Файл: test9.txt

00:05,MINI-JET,A-0005,AP1   
05:09,CESSNA-172,C-0909,AP3

###### Ожидаемый результат:

| Time | Model | Bort Number | Airport |
| --- | --- | --- | --- |
| 05:09 | CESSNA-172 | C-0909 | AP3 |
| 00:05 | MINI-JET | A-0005 | AP1 |

Полученный результат:

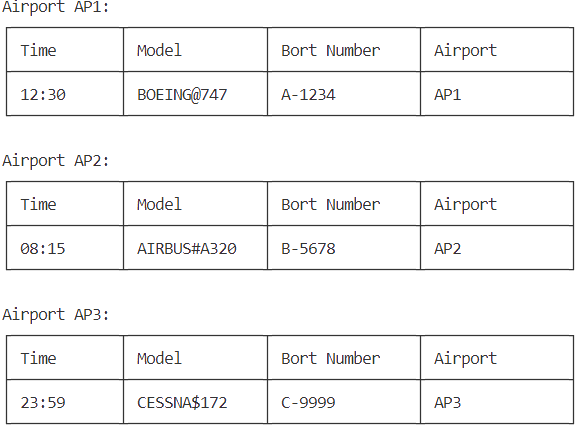


Вывод: Время с ведущими нулями обработано корректно. Тест пройден.

##### Тест 10

**Цель:** Проверить обработку специальных символов  
**Исходные данные:**  
Файл: test8.txt  
12:30,BOEING@747,A-1234,AP1  
08:15,AIRBUS#A320,B-5678,AP2  
23:59,CESSNA$172,C-9999,AP3

**Ожидаемый результат:**  
Все строки должны быть обработаны корректно после нормализации

**Полученный результат:**  


**Вывод:** Специальные символы корректно нормализованы. Тест пройден.

# Вывод

Программа успешно реализует обработку данных о посадках самолетов с полной валидацией входных данных. Использование индексной сортировки методом пузырька обеспечивает правильный порядок вывода записей по времени посадки. Программа параметризована и может быть легко адаптирована для работы с другими наборами данных.