|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  **«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**  **(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»** | | | | |
| **Журнал практики** | | | | |
| Институт № 3\_ | «\_Системы управления, информатика и электроэнергетика\_\_» | | | |
|  |  | | | |
| Кафедра | \_\_\_\_\_\_\_304\_\_\_\_\_ | | Учебная группа | \_М30-119БВ-24\_\_ |
|  |  | |  |  |
| ФИО обучающегося | | \_\_\_\_\_\_ Нарзиев Артемий Тимурович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
|  | |  | | |
| Направление подготовки/  специальность | | \_\_\_\_\_09.03.04\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Программная инженерия\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
|  | | *шифр, наименование направления подготовки/специальности* | | |
|  | |  | | |
| Вид практики | | \_\_\_\_\_\_ознакомительная\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
|  | | *учебная, производственная, преддипломная или другой вид практики* | | |
| Оценка за практику | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Давыдкина Е.А. | | |

Москва

2025

|  |  |  |
| --- | --- | --- |
| 1. **Место и сроки проведения практики:** | | |
| Наименование организации: | \_Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский авиационный институт (национальный исследовательский университет)» | |
| Сроки проведения практики |  | |
| дата начала практики: | \_\_\_\_\_\_\_\_\_10.02.2025\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
| дата окончания практики: | \_\_\_\_\_\_\_\_\_07.06.2025\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
| 1. **Инструктаж по технике безопасности:** | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_Е.А.Давыдкина\_\_/ | \_10\_\_ \_февраля\_ 2025г. |
| *подпись проводившего* | *расшифровка подписи* | *дата проведения* |
| 1. **Индивидуальное задание обучающегося:** | | |
|  | | |

|  |
| --- |
| 1. **План выполнения индивидуального задания обучающегося:** |

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Место проведения** | **Тема** | **Период выполнения** |
| 1 | МАИ | Вводное занятие и инструктаж по технике безопасности | 10.02.2025 |
| 2 |  | Разработка части кода ПО по тестированию исходных данных | 21.02.2025 |
| 3 |  | Контрольная встреча с руководителем практики для обсуждения программы | 28.02.2025 |
| 4 |  | Контрольная встреча с руководителем практики для обсуждения программы | 21.03.2025 |
| 5 |  | Разработка части кода ПО, сортирующую записи с использованием индексной сортировки | 28.03.2025 |
| 6 |  | Контрольная встреча с руководителем практики для обсуждения программы | 11.04.2025 |
| 7 |  | Контрольная встреча с руководителем практики для обсуждения программы | 25.04.2025 |
| 8 |  | Контрольная встреча с руководителем практики для обсуждения программы | 16.05.2025 |
| 9 |  | Подготовка отчета | 30.05.2025 |
| 10 |  | Контрольная встреча с руководителем практики для обсуждения отчета | 05.06.2025 |
| 11 |  | Сдача отчетов | 07.06.2025 |

|  |  |  |
| --- | --- | --- |
| **Утверждаю** | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_Е.А. Давыдкина\_\_/ | \_10\_ \_февраля\_ 2025г. |
| *подпись руководителя от МАИ* | *расшифровка подписи* | *дата утверждения\** |
|  |  |  |
|  |  |  |
| **Ознакомлен** | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_\_\_ А.Т.Нарзиев\_\_\_\_/ | \_10\_ \_февраля\_ 2025г. |
| *подпись обучающегося* | *расшифровка подписи* | *дата ознакомления\** |

*\*Дата утверждения и ознакомления – дата начала практики*

|  |  |  |
| --- | --- | --- |
| 1. **Отзыв руководителя практики от организации/предприятия:** | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_Е.А.Давыдкина\_\_\_/ | \_\_07\_ \_\_июня\_\_\_ 2025г. |
| *подпись руководителя от организации/предприятии* | *расшифровка подписи* | *дата* |

|  |
| --- |
| 1. **Отчет обучающего по практике:** |

# 

|  |  |  |
| --- | --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | /\_\_ А.Т.Нарзиев\_\_\_\_/ | \_07\_ \_\_\_июня\_\_\_ 2025 г. |
| *подпись обучающегося* | *расшифровка подписи* | *дата* |

# Содержание

[Содержание 6](#_Toc199882834)

[Задание 7](#_Toc199882835)

[Введение 8](#_Toc199882836)

[1. Основные теоретические аспекты 8](#_Toc199882837)

[1.1. Структуры в C++ 8](#_Toc199882838)

[1.2. Индексная сортировка 8](#_Toc199882839)

[1.3. Сортировка пузырьком 9](#_Toc199882840)

[1.4. Валидация данных 10](#_Toc199882841)

[1.5. Пример сортировки 10](#_Toc199882842)

[Решение 12](#_Toc199882843)

[2. Код программы: 12](#_Toc199882844)

[2.1. Главный файл – main.cpp: 12](#_Toc199882845)

[3. Описание функций 26](#_Toc199882846)

[3.1. Функция print\_error 26](#_Toc199882847)

[3.2. Функция read\_data 26](#_Toc199882848)

[3.3. Функция purify 30](#_Toc199882849)

[3.4. Функция check\_and\_convert\_time 31](#_Toc199882850)

[3.5. Функция check\_bort\_valid 33](#_Toc199882851)

[3.6. Функция check\_airport\_valid 35](#_Toc199882852)

[3.7. Функция bubble\_sort 37](#_Toc199882853)

[3.8. Функция print\_table 38](#_Toc199882854)

[3.9. Функция process\_airport 39](#_Toc199882855)

[3.10. Функция check\_model\_valid 40](#_Toc199882856)

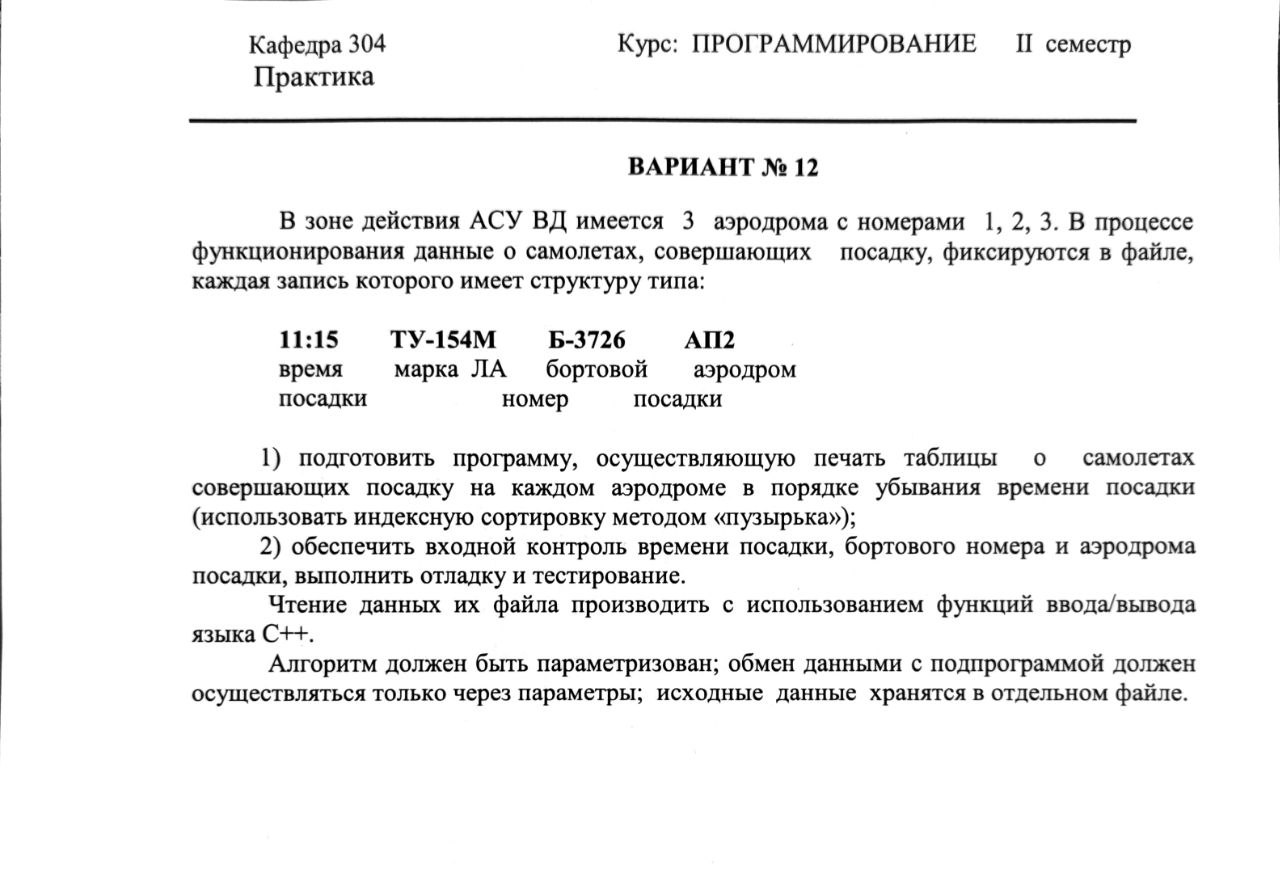
[Тесты 41](#_Toc199882857)

[1. Некорректные тесты 41](#_Toc199882858)

[2. Корректные тесты 47](#_Toc199882859)

[Вывод 55](#_Toc199882860)

# 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

## 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>

#include <cstring>

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;

const int MAX\_ERRORS = 1000; // Максимальное количество ошибок

// Ожидаемая длина полей

const int EXPECTED\_TIME\_LEN = 5;    // HH:MM

const int EXPECTED\_BORT\_LEN = 6;    // X-XXXX

const int EXPECTED\_AIRPORT\_LEN = 3; // APX

const int ALLOWED\_FIELDS = 4;

// СТРУКТУРА ДАННЫХ

struct Plane {

    char time[TIME\_LEN];

    int minutes;

    char model[MODEL\_LEN];

    char bort[BORT\_LEN];

    char airport[AIRPORT\_LEN];

};

// Структура для хранения ошибок

struct ErrorInfo {

    int line\_num;

    char field\_name[50]; // Увеличен буфер для имени поля

    char message[250];   // Увеличен буфер для сообщения

};

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

\*  ПРОТОТИПЫ ФУНКЦИЙ

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

void print\_error(const ErrorInfo& error);

int read\_data(const char \*filename, Plane \*planes, int \*count, ErrorInfo \*errors, int \*error\_count);

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

void purify(char\* field);

int check\_and\_convert\_time(const char \*time, int line\_num, ErrorInfo \*errors, int \*error\_count);

void check\_model\_valid(const char \*model, int line\_num, ErrorInfo \*errors, int \*error\_count);

void check\_bort\_valid(const char \*bort, int line\_num, ErrorInfo \*errors, int \*error\_count);

void check\_airport\_valid(const char \*airport, int line\_num, ErrorInfo \*errors, int \*error\_count);

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

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

// Тестовый файл

const char \*FILE\_NAME = "tests/correct/test2.txt";

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

\*  ГЛАВНАЯ ФУНКЦИЯ

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

int main()

{

    Plane planes[MAX\_PLANES];

    int count = 0;

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

    ErrorInfo errors[MAX\_ERRORS];

    int error\_count = 0;

    // Эхо-печать имени файла

    cout << "Обрабатываем файл: " << FILE\_NAME << "\n\n";

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

    // Вывод всех ошибок

    if (error\_count > 0) {

        cout << "\nНайдены ошибки:\n";

        for (int i = 0; i < error\_count; i++) {

            print\_error(errors[i]);

        }

    }

    if (err == -1) {

        return 1;

    }

    // Эхо-печать количества успешно обработанных записей

    cout << "\nУспешно обработано записей: " << count << "\n\n";

    for (int i = 0; i < 3; i++) {

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

    }

    return 0;

}

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

\*  ИНИЦИАЛИЗАЦИЯ ФУНКЦИЙ

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

void print\_error(const ErrorInfo& error)

{

    cout << "[31m";

    cout << "Строка " << error.line\_num << ", поле '" << error.field\_name

         << "': " << error.message;

    cout << "[0m\n";

}

int read\_data(const char \*filename, Plane \*planes, int \*count, ErrorInfo \*errors, int \*error\_count)

{

    ifstream file(filename);

    if (!file.is\_open())

    {

        strncpy(errors[\*error\_count].field\_name, "Файл", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, "Файл не найден", sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = 0;

        (\*error\_count)++;

        return -1;

    }

    char line[100];

    int line\_num = 0;

    bool has\_errors = false;

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

    {

        line\_num++;

        // Эхо-печать обрабатываемой строки

        cout << "Обработка строки " << line\_num << ": " << line << endl;

        Plane p = {};

        int pos = 0;

        bool is\_valid = true;

        bool is\_empty = true;

        int comma\_count = 0;

        // Проверка на пустую строку

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

        {

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

            {

                is\_empty = false;

                break;

            }

        }

        if (is\_empty)

        {

            strncpy(errors[\*error\_count].field\_name, "Строка", sizeof(errors[\*error\_count].field\_name) - 1);

            strncpy(errors[\*error\_count].message, "Пустая строка", sizeof(errors[\*error\_count].message) - 1);

            errors[\*error\_count].line\_num = line\_num;

            (\*error\_count)++;

            has\_errors = true;

            continue;

        }

        // Проверка количества запятых

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

        {

            if (line[i] == ',') comma\_count++;

        }

        if (comma\_count < 3)

        {

            strncpy(errors[\*error\_count].field\_name, "Строка", sizeof(errors[\*error\_count].field\_name) - 1);

            strncpy(errors[\*error\_count].message, "Неправильное построение строки (недостаточно запятых)", sizeof(errors[\*error\_count].message) - 1);

            errors[\*error\_count].line\_num = line\_num;

            (\*error\_count)++;

            has\_errors = true;

            continue;

        }

        // ПАРСИНГ ПОЛЕЙ

        for (int field\_num = 0; field\_num < ALLOWED\_FIELDS; field\_num++)

        {

            char\* dest = NULL;

            int max\_len = 0;

            int j = 0;

            bool field\_too\_long = false;

            char temp\_field[100] = {0}; // Временный буфер для хранения поля

            int temp\_index = 0;

            if (field\_num == 0)

            {

                dest = p.time;

                max\_len = TIME\_LEN;

            } else if (field\_num == 1)

            {

                dest = p.model;

                max\_len = MODEL\_LEN;

            } else if (field\_num == 2)

            {

                dest = p.bort;

                max\_len = BORT\_LEN;

            } else

            {

                dest = p.airport;

                max\_len = AIRPORT\_LEN;

            }

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

            // Читаем символы до запятой или конца строки во временный буфер

            while (line[pos] && line[pos] != ',' && temp\_index < 99)

            {

                temp\_field[temp\_index++] = line[pos++];

            }

            temp\_field[temp\_index] = '\0';

            // Очищаем поле от пробелов

            purify(temp\_field);

            int purified\_len = strlen(temp\_field);

            // Проверяем длину очищенного поля

            if (purified\_len > max\_len - 1)

            {

                field\_too\_long = true;

                is\_valid = false;

            }

            // Копируем очищенное поле в целевую переменную

            strncpy(dest, temp\_field, max\_len - 1);

            dest[max\_len - 1] = '\0';

            // Если поле было слишком длинным, добавляем ошибку

            if (field\_too\_long)

            {

                const char\* field\_name = "";

                switch (field\_num)

                {

                    case 0: field\_name = "Время"; break;

                    case 1: field\_name = "Модель"; break;

                    case 2: field\_name = "Бортовой номер"; break;

                    case 3: field\_name = "Аэродром"; break;

                }

                char msg[250];

                snprintf(msg, sizeof(msg), "Поле слишком длинное (максимум %d символов, получено %d)",

                         max\_len - 1, purified\_len);

                strncpy(errors[\*error\_count].field\_name, field\_name, sizeof(errors[\*error\_count].field\_name) - 1);

                strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

                errors[\*error\_count].line\_num = line\_num;

                (\*error\_count)++;

            }

            if (line[pos] == ',')

            {

                pos++;

            }

        }

        // Пропускаем пробелы после последнего поля

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

        // Запоминаем количество ошибок перед валидацией

        int prev\_error\_count = \*error\_count;

        // ПРОВЕРКА ВРЕМЕНИ (первой, чтобы ошибки шли в правильном порядке)

        p.minutes = check\_and\_convert\_time(p.time, line\_num, errors, error\_count);

        if (p.minutes < 0)

        {

            is\_valid = false;

        }

        // ВАЛИДАЦИЯ ОСТАЛЬНЫХ ПОЛЕЙ

        check\_model\_valid(p.model, line\_num, errors, error\_count);

        check\_bort\_valid(p.bort, line\_num, errors, error\_count);

        check\_airport\_valid(p.airport, line\_num, errors, error\_count);

        // Проверка на лишние данные

        if (line[pos] != '\0')

        {

            strncpy(errors[\*error\_count].field\_name, "Строка", sizeof(errors[\*error\_count].field\_name) - 1);

            strncpy(errors[\*error\_count].message, "Лишние данные в конце строки", sizeof(errors[\*error\_count].message) - 1);

            errors[\*error\_count].line\_num = line\_num;

            (\*error\_count)++;

            is\_valid = false;

        }

        // Если были ошибки валидации, не добавляем запись

        if (\*error\_count > prev\_error\_count)

        {

            is\_valid = false;

        }

        // Добавляем запись, если она валидна

        if (is\_valid)

        {

            planes[(\*count)++] = p;

        } else {

            has\_errors = true;

        }

    }

    file.close();

    return has\_errors ? -7 : 0;

}

void purify(char\* field)

{

    char\* read\_ptr = field;

    char\* write\_ptr = field;

    while (\*read\_ptr)

    {

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

            \*write\_ptr = \*read\_ptr;

            write\_ptr++;

        }

        read\_ptr++;

    }

    \*write\_ptr = '\0';

}

int check\_and\_convert\_time(const char \*time, int line\_num, ErrorInfo \*errors, int \*error\_count)

{

    bool has\_errors = false;

    // Проверка длины

    if (strlen(time) != EXPECTED\_TIME\_LEN)

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Неправильная длина (должно быть %d символов, получено %zu)",

                 EXPECTED\_TIME\_LEN, strlen(time));

        strncpy(errors[\*error\_count].field\_name, "Время", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        return -1;

    }

    // Проверка разделителя

    if (time[2] != ':')

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Неправильный разделитель (должен быть ':', а получен '%c')", time[2]);

        strncpy(errors[\*error\_count].field\_name, "Время", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        has\_errors = true;

    }

    // Проверка цифровых символов

    for (int i = 0; i < EXPECTED\_TIME\_LEN; i++)

    {

        if (i == 2) continue; // Пропускаем разделитель

        if (time[i] < '0' || time[i] > '9')

        {

            char msg[250];

            snprintf(msg, sizeof(msg), "Недопустимый символ '%c' в позиции %d", time[i], i);

            strncpy(errors[\*error\_count].field\_name, "Время", sizeof(errors[\*error\_count].field\_name) - 1);

            strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

            errors[\*error\_count].line\_num = line\_num;

            (\*error\_count)++;

            has\_errors = true;

        }

    }

    // Извлечение часов и минут

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

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

    // Проверка диапазонов

    if (hours < 0 || hours >= 24)

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Часы должны быть в диапазоне 00-23 (получено %02d)", hours);

        strncpy(errors[\*error\_count].field\_name, "Время", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        has\_errors = true;

    }

    if (minutes < 0 || minutes >= 60)

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Минуты должны быть в диапазоне 00-59 (получено %02d)", minutes);

        strncpy(errors[\*error\_count].field\_name, "Время", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        has\_errors = true;

    }

    return has\_errors ? -1 : hours \* 60 + minutes;

}

void check\_model\_valid(const char \*model, int line\_num, ErrorInfo \*errors, int \*error\_count)

{

    if (strlen(model) == 0)

    {

        strncpy(errors[\*error\_count].field\_name, "Модель", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, "Пустое поле", sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        return;

    }

    if (strlen(model) >= MODEL\_LEN - 1) {

        char msg[250];

        snprintf(msg, sizeof(msg), "Слишком длинное название модели (максимум %d символов)", MODEL\_LEN - 2);

        strncpy(errors[\*error\_count].field\_name, "Модель", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

    }

}

void check\_bort\_valid(const char \*bort, int line\_num, ErrorInfo \*errors, int \*error\_count)

{

    int len = strlen(bort);

    // Проверка на пустое поле

    if (len == 0)

    {

        strncpy(errors[\*error\_count].field\_name, "Бортовой номер", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, "Пустое поле", sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        return;

    }

    if (len > EXPECTED\_BORT\_LEN)

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Слишком длинный бортовой номер (максимум %d символов, получено %d)",

                 EXPECTED\_BORT\_LEN, len);

        strncpy(errors[\*error\_count].field\_name, "Бортовой номер", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        return;

    }

    if (len < EXPECTED\_BORT\_LEN)

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Слишком короткий бортовой номер (должно быть %d символов, получено %d)",

                 EXPECTED\_BORT\_LEN, len);

        strncpy(errors[\*error\_count].field\_name, "Бортовой номер", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        return;

    }

    // Остальные проверки

    if (bort[1] != '-')

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Неправильный разделитель (должен быть '-', а получен '%c')", bort[1]);

        strncpy(errors[\*error\_count].field\_name, "Бортовой номер", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

    }

    if (bort[0] < 'A' || bort[0] > 'Z')

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Первым символом должна быть заглавная буква A-Z (получено '%c')", bort[0]);

        strncpy(errors[\*error\_count].field\_name, "Бортовой номер", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

    }

    for (int i = 2; i < EXPECTED\_BORT\_LEN; i++)

    {

        if (bort[i] < '0' || bort[i] > '9')

        {

            char msg[250];

            snprintf(msg, sizeof(msg), "Недопустимый символ '%c' в позиции %d (ожидалась цифра)", bort[i], i);

            strncpy(errors[\*error\_count].field\_name, "Бортовой номер", sizeof(errors[\*error\_count].field\_name) - 1);

            strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

            errors[\*error\_count].line\_num = line\_num;

            (\*error\_count)++;

        }

    }

}

void check\_airport\_valid(const char \*airport, int line\_num, ErrorInfo \*errors, int \*error\_count)

{

    if (strlen(airport) != EXPECTED\_AIRPORT\_LEN)

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Неправильная длина (должно быть %d символа, получено %zu)",

                 EXPECTED\_AIRPORT\_LEN, strlen(airport));

        strncpy(errors[\*error\_count].field\_name, "Аэродром", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

        return;

    }

    if (airport[0] != 'A' || airport[1] != 'P')

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Код должен начинаться с 'AP' (получено '%.2s')", airport);

        strncpy(errors[\*error\_count].field\_name, "Аэродром", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

    }

    if (airport[2] < '1' || airport[2] > '3')

    {

        char msg[250];

        snprintf(msg, sizeof(msg), "Последний символ должен быть цифрой 1-3 (получено '%c')", airport[2]);

        strncpy(errors[\*error\_count].field\_name, "Аэродром", sizeof(errors[\*error\_count].field\_name) - 1);

        strncpy(errors[\*error\_count].message, msg, sizeof(errors[\*error\_count].message) - 1);

        errors[\*error\_count].line\_num = line\_num;

        (\*error\_count)++;

    }

}

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;

            }

        }

    }

}

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";

}

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

{

    int indices[MAX\_PLANES];

    int size = 0;

    for (int i = 0; i < count; i++)

    {

        if (strncmp(planes[i].airport, airport, EXPECTED\_AIRPORT\_LEN) == 0)

        {

            indices[size++] = i;

        }

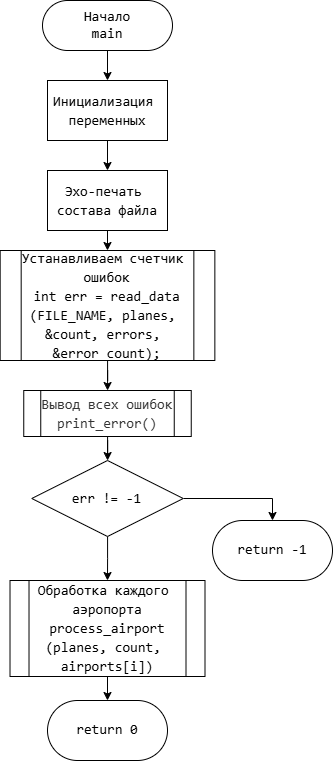
    }

    bubble\_sort(indices, size, planes);

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

}

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

****

## Описание функций

### Функция print\_error

**Назначение**Выводит форматированное сообщение об ошибке с указанием номера строки, имени поля и текста ошибки.  
**Прототип**void print\_error(const ErrorInfo& error);  
**Обращение к функции:**print\_error(errors[i]);  
**Описание параметров:**

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной/Выходной |
| error | const ErrorInfo& | Структура с информацией об ошибке | Входной |

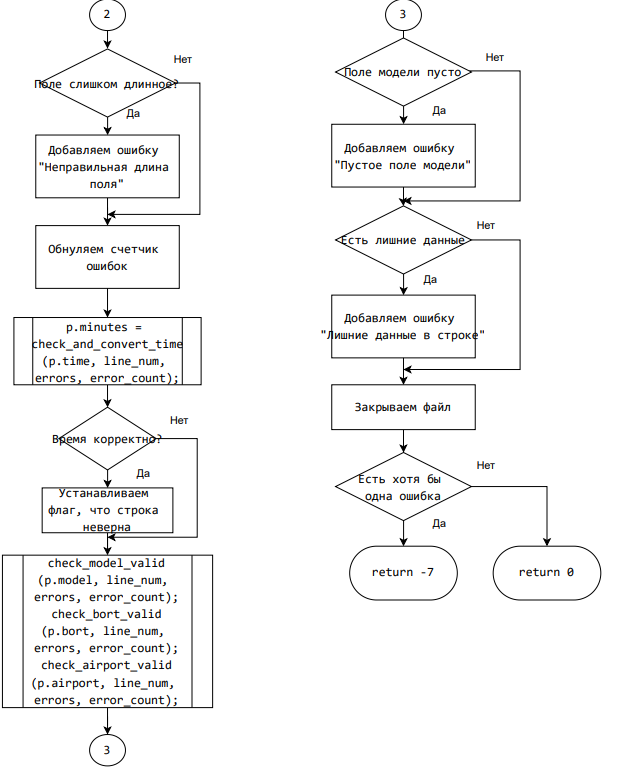
### print_error.drawio.png

### Функция read\_data

**Назначение**Читает данные из файла, парсит строки, валидирует поля и заполняет массив структур Plane.  
**Прототип**int read\_data(const char \*filename, Plane \*planes, int \*count, ErrorInfo \*errors, int \*error\_count);  
**Обращение к функции:**read\_data(FILE\_NAME, planes, &count, errors, &error\_count);  
**Описание параметров:**

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной/Выходной |
| filename | const char\* | Имя файла с данными | Входной |
| planes | Plane\* | Массив структур Plane | Выходной |
| count | int\* | Указатель на количество записей | Входной/Выходной |
| errors | ErrorInfo\* | Массив структур с ошибками | Выходной |
| error\_count | int\* | Указатель на количество ошибок | Входной/Выходной |

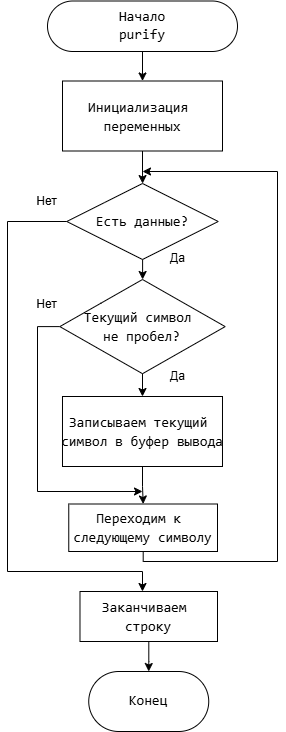
### 



### Функция purify

**Назначение**Нормализует поле: удаляет пробелы и переводит символы в верхний регистр.  
**Прототип**void purify(char\* field);  
**Обращение к функции:**purify(p.time);  
**Описание параметров:**

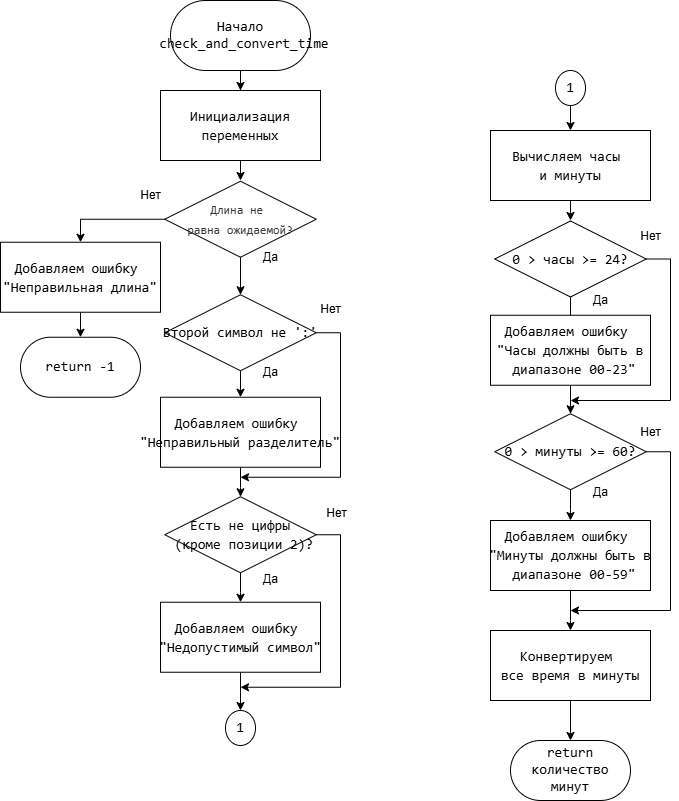
|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной/Выходной |
| field | char\* | Указатель на строку для обработки | Входной/Выходной |



### Функция check\_and\_convert\_time

**Назначение**Проверяет корректность формата времени (HH:MM) и конвертирует его в минуты.  
**Прототип**int check\_and\_convert\_time(const char \*time, int line\_num, ErrorInfo \*errors, int \*error\_count);  
**Обращение к функции:**check\_and\_convert\_time(p.time, line\_num, errors, error\_count);  
**Описание параметров:**

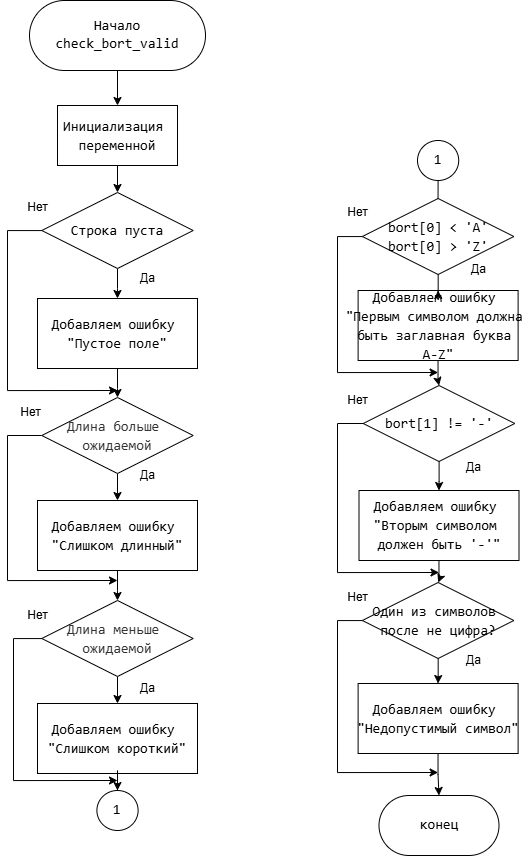
|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной/Выходной |
| time | const char\* | Строка времени для проверки | Входной |
| line\_num | int | Номер строки | Входной |
| errors | ErrorInfo\* | Массив ошибок | Входной/Выходной |
| error\_count | int\* | Указатель на счетчик ошибок | Входной/Выходной |



### Функция check\_bort\_valid

**Назначение**Проверяет формат бортового номера (X-XXXX).  
**Прототип**void check\_bort\_valid(const char \*bort, int line\_num, ErrorInfo \*errors, int \*error\_count);  
**Обращение к функции:**check\_bort\_valid(p.bort, line\_num, errors, error\_count);  
**Описание параметров:**

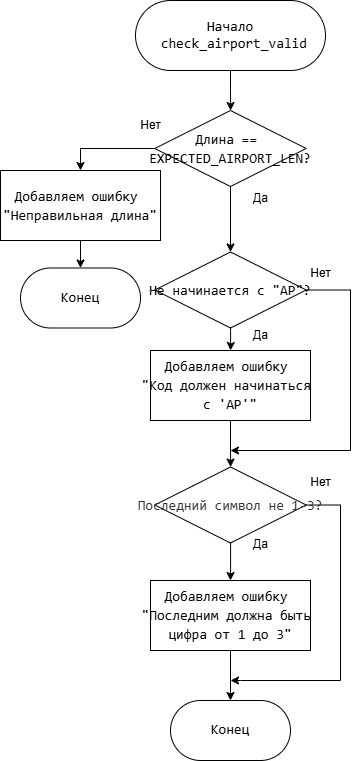
|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной/Выходной |
| bort | const char\* | Бортовой номер для проверки | Входной |
| line\_num | int | Номер строки в файле | Входной |
| errors | ErrorInfo\* | Массив ошибок | Входной/Выходной |
| error\_count | int\* | Указатель на счетчик ошибок | Входной/Выходной |



### Функция check\_airport\_valid

**Назначение**  
Проверяет код аэродрома (AP1, AP2, AP3).  
**Прототип**  
void check\_airport\_valid(const char \*airport, int line\_num, ErrorInfo \*errors, int \*error\_count);  
**Обращение к функции:**  
check\_airport\_valid(p.airport, line\_num, errors, error\_count);  
**Описание параметров:**

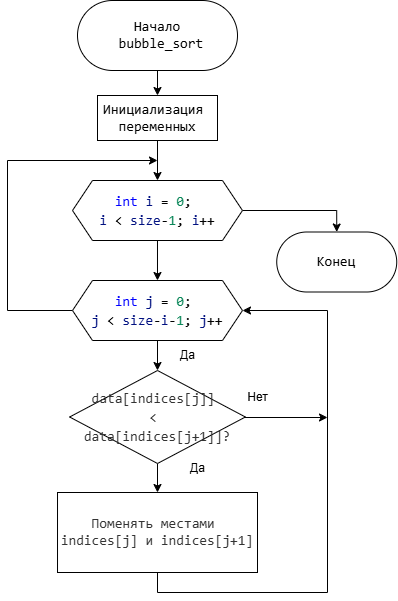
|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной/Выходной |
| airport | const char\* | Код аэродрома для проверки | Входной |
| line\_num | int | Номер строки в файле | Входной |
| errors | ErrorInfo\* | Массив ошибок | Входной/Выходной |
| error\_count | int\* | Указатель на счетчик ошибок | Входной/Выходной |



### Функция bubble\_sort

**Назначение**  
Сортирует индексы записей по времени посадки(в порядке убывания).  
**Прототип**  
void bubble\_sort(int \*indices, int size, Plane \*data);  
**Обращение к функции:**  
bubble\_sort(indices, size, planes);  
**Описание параметров:**

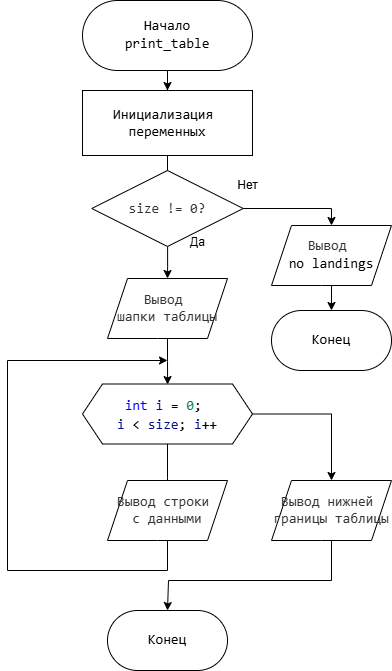
|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной/Выходной |
| indices | int\* | Массив индексов | Входной/Выходной |
| size | int | Размер массива | Входной |
| data | Plane\* | Массив структур Plane | Входной |



### Функция print\_table

**Назначение**  
Форматирует и выводит таблицу с данными для аэродрома.  
**Прототип**  
void print\_table(int \*indices, int size, Plane \*data, const char \*airport);  
**Обращение к функции:**  
print\_table(indices, size, planes, airport);  
**Описание параметров:**

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Назначение | Входной/Выходной |
| indices | int\* | Массив индексов | 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.  
**Прототип**  
void process\_airport(Plane \*planes, int count, const char \*airport);  
**Обращение к функции:**  
process\_airport(planes, count, airports[i]);  
**Описание параметров:**

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| planes | Plane\* | Массив структур с данными | Входной |
| count | int | Количество записей в массиве | Входной |
| airport | const char\* | Код обрабатываемого аэродрома | Входной |

### process_airport.drawio.png

### Функция check\_model\_valid

Назначение  
Проверяет валидность поля “Модель самолета”.

Прототип

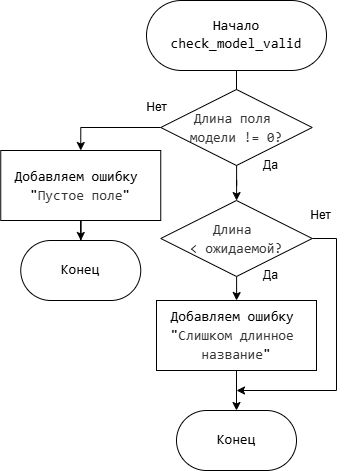
void check\_model\_valid(const char \*model, int line\_num, ErrorInfo \*errors, int \*error\_count);

Обращение к функции

check\_model\_valid(p.model, line\_num, errors, error\_count);

Описание параметров

| Идентификатор | Тип | Назначение | Входной/Выходной |
| --- | --- | --- | --- |
| model | const char\* | Строка модели для проверки | Входной |
| line\_num | int | Номер строки в файле | Входной |
| errors | ErrorInfo\* | Массив для записи ошибок | Выходной |
| error\_count | int\* | Указатель на счетчик ошибок | Входной/Выходной |



# Тесты

## Некорректные тесты

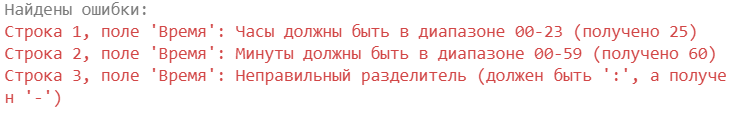
##### Тест 1

**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:**

Строка 1, поле 'Время': Часы должны быть в диапазоне 00-23  
Строка 2, поле 'Время': Минуты должны быть в диапазоне 00-59  
Строка 3, поле 'Время': Неправильный разделитель (должен быть ':', а получен '-')

**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:**

Строка 1, поле 'Бортовой номер': Поле слишком длинное (максимум 6 символов, получено 7)

Строка 1, поле 'Бортовой номер': Неправильный разделитель (должен быть '-', а получен '2')

Строка 1, поле 'Бортовой номер': Первым символом должна быть заглавная буква A-Z (получено '1')

Строка 1, поле 'Бортовой номер': Недопустимый символ '-' в позиции 4 (ожидалась цифра)

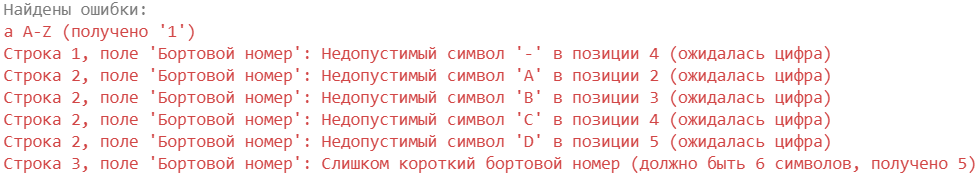
Строка 2, поле 'Бортовой номер': Недопустимый символ 'A' в позиции 2 (ожидалась цифра)

Строка 2, поле 'Бортовой номер': Недопустимый символ 'B' в позиции 3 (ожидалась цифра)

Строка 2, поле 'Бортовой номер': Недопустимый символ 'C' в позиции 4 (ожидалась цифра)

Строка 2, поле 'Бортовой номер': Недопустимый символ 'D' в позиции 5 (ожидалась цифра)

Строка 3, поле 'Бортовой номер': Слишком короткий бортовой номер (должно быть 6 символов, получено 5)

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

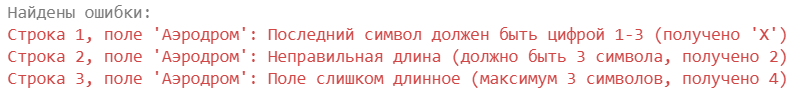
##### Тест 3

**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:**

Строка 1, поле 'Аэродром': Последний символ должен быть цифрой 1-3 (получено 'X')  
Строка 2, поле 'Аэродром': Неправильная длина (должно быть 3 символа)  
Строка 3, поле 'Аэродром': Неправильная длина (должно быть 3 символа)

**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

**Ожидаемый результат:**

Строка 1, поле 'Модель': Пустое поле  
Строка 2, поле 'Модель': Пустое поле

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

##### Тест 5

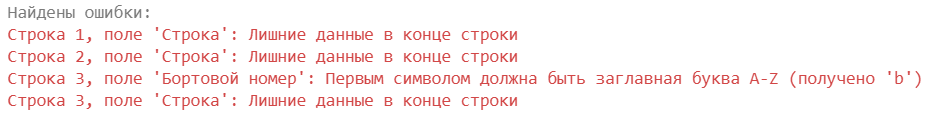
**Цель:** Проверить обработку строк с избыточными данными  
**Исходные данные:**  
Файл: test5.txt

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:**

Строка 1, поле 'Строка': Лишние данные в конце строки  
Строка 2, поле 'Строка': Лишние данные в конце строки

Строка 3, поле 'Бортовой номер': Первым символом должна быть заглавная буква A-Z (получено 'b')  
Строка 3, поле 'Строка': Лишние данные в конце строки

**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:**

Строка 1, поле 'Время': Часы должны быть в диапазоне 00-23 (получено 25)

Строка 2, поле 'Модель': Пустое поле

Строка 3, поле 'Бортовой номер': Поле слишком длинное (максимум 6 символов, получено 7)

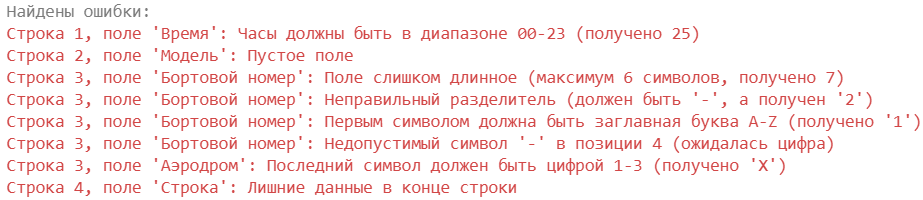
Строка 3, поле 'Бортовой номер': Неправильный разделитель (должен быть '-', а получен '2')

Строка 3, поле 'Бортовой номер': Первым символом должна быть заглавная буква A-Z (получено '1')

Строка 3, поле 'Бортовой номер': Недопустимый символ '-' в позиции 4 (ожидалась цифра)

Строка 3, поле 'Аэродром': Последний символ должен быть цифрой 1-3 (получено 'X')

Строка 4, поле 'Строка': Лишние данные в конце строки

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

##### Тест 7

**Цель:** Проверить обработку граничных значений времени  
**Исходные данные:**  
Файл: test7.txt

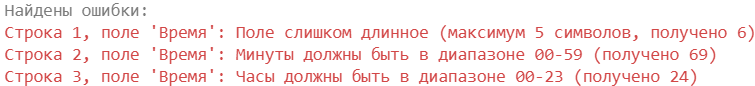
00:100,MINI-JET,A-0000,AP1  
23:69,JUMBO-JET,A-9999,AP3  
24:00,GHOST-PLANE,X-0000,AP2

**Ожидаемый результат:**

Строка 1, поле 'Время': Поле слишком длинное (максимум 5 символов, получено 6)

Строка 2, поле 'Время': Минуты должны быть в диапазоне 00-59 (получено 69)

Строка 3, поле 'Время': Часы должны быть в диапазоне 00-23 (получено 24)

**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:**

Строка 2, поле 'Строка': Пустая строка

Airport AP1:

┌────────────┬───────────────┬────────────────┬────────────────┐

│ Time │ Model │ Bort Number │ Airport │

├────────────┼───────────────┼────────────────┼────────────────┤

│ 12:30 │ BOEING-747 │ A-1234 │ AP1 │

└────────────┴───────────────┴────────────────┴────────────────┘

Airport AP2: no landings

Airport AP3:

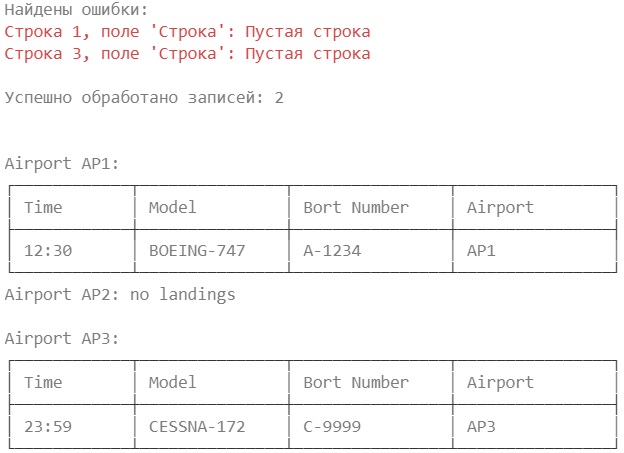
┌────────────┬───────────────┬────────────────┬────────────────┐

│ Time │ Model │ Bort Number │ Airport │

├────────────┼───────────────┼────────────────┼────────────────┤

│ 23:59 │ CESSNA-172 │ C-9999 │ AP3 │

└────────────┴───────────────┴────────────────┴────────────────┘

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

##### Тест 9

**Цель:** Проверить обработку смешанных ошибок в одной строке  
**Исходные данные:**  
Файл: test2.txt

12:30,AIRBUS-A320,A-1233,AP1

08\*651,AIRBUS-A320,,AP2

24:59, LIRILI-172 , C-99999, AP5

23:77, LARILA-172 , C-8s88, AP3

20:00, Sahur-911 , b-1000, AP1

**Ожидаемый результат:**

Строка 2, поле 'Время': Поле слишком длинное (максимум 5 символов, получено 6)

Строка 2, поле 'Время': Неправильный разделитель (должен быть ':', а получен '\*')

Строка 2, поле 'Бортовой номер': Пустое поле

Строка 3, поле 'Бортовой номер': Поле слишком длинное (максимум 6 символов, получено 7)

Строка 3, поле 'Время': Часы должны быть в диапазоне 00-23 (получено 24)

Строка 3, поле 'Аэродром': Последний символ должен быть цифрой 1-3 (получено '5')

Строка 4, поле 'Время': Минуты должны быть в диапазоне 00-59 (получено 77)

Строка 4, поле 'Бортовой номер': Недопустимый символ 's' в позиции 3 (ожидалась цифра)

Строка 5, поле 'Аэродром': Поле слишком длинное (максимум 3 символов, получено 3)

Строка 5, поле 'Бортовой номер': Первым символом должна быть заглавная буква A-Z (получено 'b')

Airport AP1:

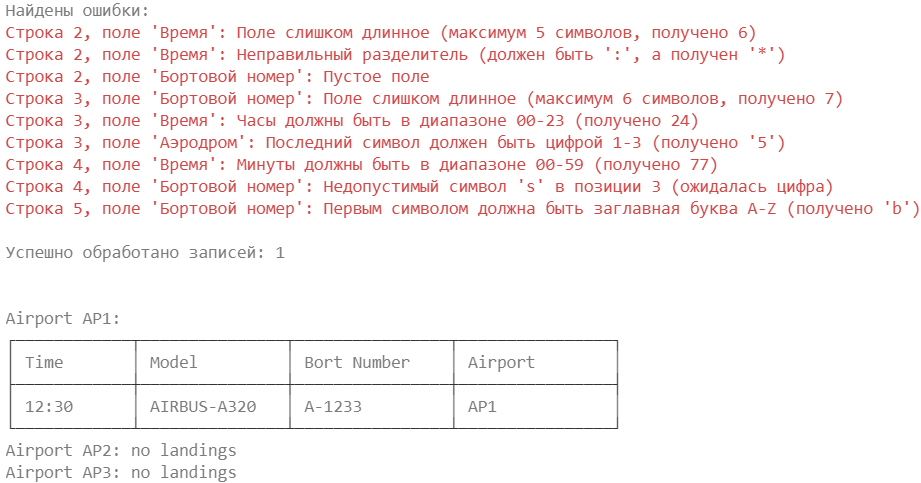
┌────────────┬───────────────┬────────────────┬────────────────┐

│ Time │ Model │ Bort Number │ Airport │

├────────────┼───────────────┼────────────────┼────────────────┤

│ 12:30 │ AIRBUS-A320 │ A-1233 │ AP1 │

└────────────┴───────────────┴────────────────┴────────────────┘

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

## 2. Корректные тесты

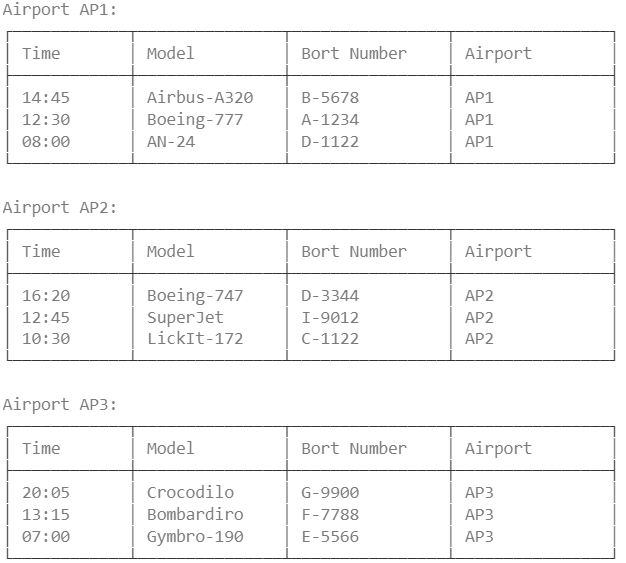
##### Тест №1

**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:**

Airport 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 │  
└────────────┴───────────────┴────────────────┴────────────────┘  
  
Airport 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 │  
└────────────┴───────────────┴────────────────┴────────────────┘  
  
Airport 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.

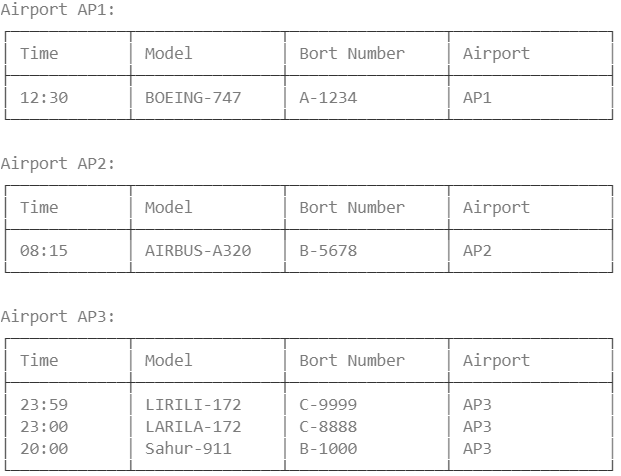
##### Тест №2

**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:**

Airport AP1:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 12:30 │ BOEING-747 │ A-1234 │ AP1 │  
└────────────┴───────────────┴────────────────┴────────────────┘  
  
Airport AP2:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 08:15 │ AIRBUS-A320 │ B-5678 │ AP2 │  
└────────────┴───────────────┴────────────────┴────────────────┘  
  
Airport 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 пробелы. Тест пройден.

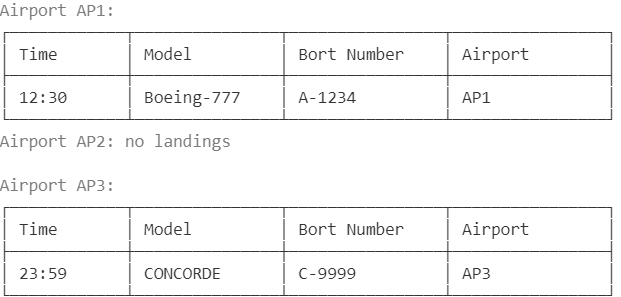
##### Тест №3

**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:**

Airport AP1:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 12:30 │ BOEING-777 │ A-1234 │ AP1 │  
└────────────┴───────────────┴────────────────┴────────────────┘  
  
Airport AP3:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 23:59 │ CONCORDE │ C-9999 │ 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

0:00,MINI-JET,A-0000,AP1

12:00,Boeing-777,A-1234,AP1

**Ожидаемый результат:**

Airport AP1:  
┌────────────┬───────────────┬────────────────┬────────────────┐

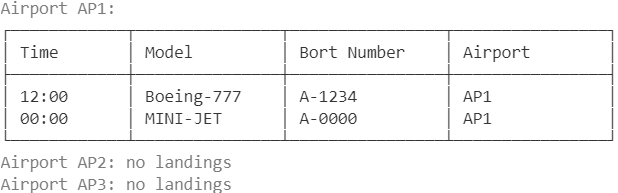
│ Time │ Model │ Bort Number │ Airport │

├────────────┼───────────────┼────────────────┼────────────────┤

│ 12:00 │ Boeing-777 │ A-1234 │ AP1 │

│ 00:00 │ MINI-JET │ A-0000 │ AP1 │

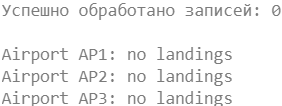
└────────────┴───────────────┴────────────────┴────────────────┘

**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.

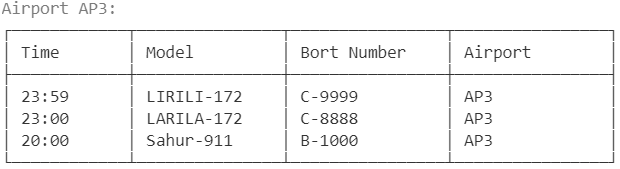
##### Тест №6

**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:**

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

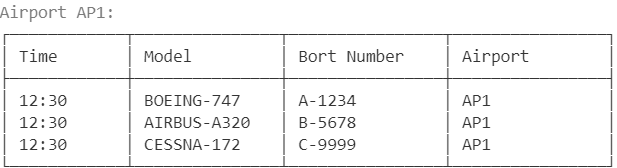
##### Тест №7

**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:**

Airport AP1:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ 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

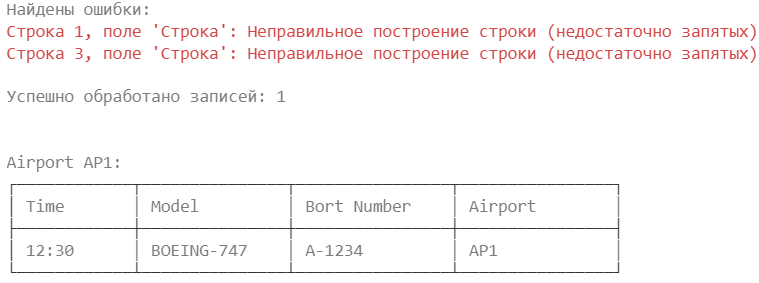
**Цель:** Проверить обработку файла с одной корректной записью среди ошибок.  
**Исходные данные:**  
Файл: test8.txt

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

**Ожидаемый результат:**

Строка 1, поле 'Строка': Неправильное построение строки (недостаточно запятых)

Строка 3, поле 'Строка': Неправильное построение строки (недостаточно запятых)  
  
Airport AP1:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ 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.

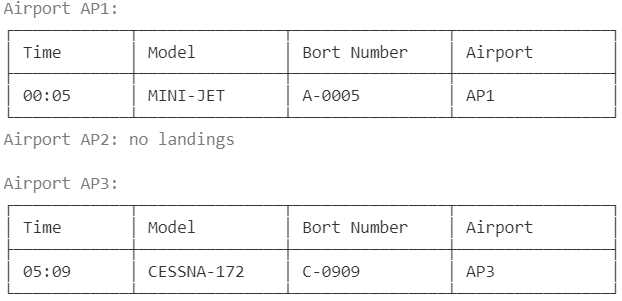
##### Тест №9

**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:**

Airport AP1:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 00:05 │ MINI-JET │ A-0005 │ AP1 │  
└────────────┴───────────────┴────────────────┴────────────────┘  
  
Airport AP3:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 05:09 │ CESSNA-172 │ C-0909 │ AP3 │  
└────────────┴───────────────┴────────────────┴────────────────┘

**Полученный результат:**  
  
**Вывод:** Время с ведущими нулями обработано корректно. Тест пройден.

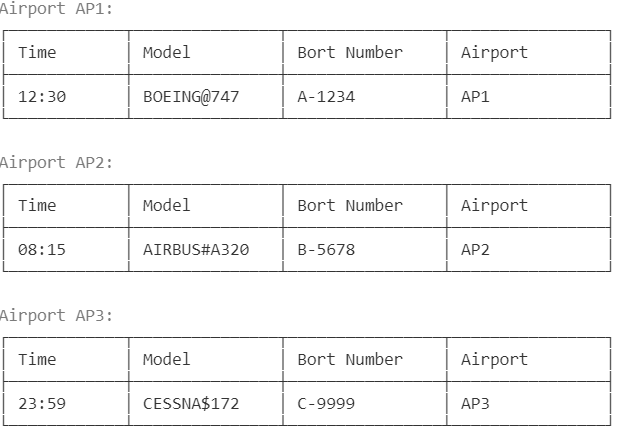
##### Тест 10

**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:**

Airport AP1:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 12:30 │ BOEING@747 │ A-1234 │ AP1 │  
└────────────┴───────────────┴────────────────┴────────────────┘  
  
Airport AP2:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 08:15 │ AIRBUS#A320 │ B-5678 │ AP2 │  
└────────────┴───────────────┴────────────────┴────────────────┘  
  
Airport AP3:  
┌────────────┬───────────────┬────────────────┬────────────────┐  
│ Time │ Model │ Bort Number │ Airport │  
├────────────┼───────────────┼────────────────┼────────────────┤  
│ 23:59 │ CESSNA$172 │ C-9999 │ AP3 │  
└────────────┴───────────────┴────────────────┴────────────────┘

**Полученный результат:**  
  
**Вывод:** Специальные символы корректно обработаны. 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.