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 М3О-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

2. **Инструктаж по технике безопасности:**

_____ / Е.А.Давыдкина / 10 февраля 2025г.

подпись проводившего *расшифровка подписи* *дата проведения*

3. **Индивидуальное задание обучающегося:**

_____

_____

_____

_____

4. 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*

5. Отзыв руководителя практики от организации/предприятия:

_____

_____

_____

_____

_____

_____

_____

_____

_____ /__Е.А.Давыдкина___/ __07__ __июня___ 2025г.

подпись руководителя от организации/предприятии *расшифровка подписи* *дата*

6. Отчет обучающего по практике:

_____ /__А.Т.Нарзиев____/ _07_ ___июня___ 2025 г.

подпись обучающегося *расшифровка подписи* *дата*

Содержание

Задание

Кафедра 304 Курс: ПРОГРАММИРОВАНИЕ II семестр
Практика

ВАРИАНТ № 12

В зоне действия АСУ ВД имеется 3 аэродрома с номерами 1, 2, 3. В процессе функционирования данные о самолетах, совершающих посадку, фиксируются в файле, каждая запись которого имеет структуру типа:

11:15	**ТУ-154М**	**Б-3726**	**АП2**
время посадки	марка ЛА	бортовой номер	аэродром посадки

1) подготовить программу, осуществляющую печать таблицы о самолетах совершающих посадку на каждом аэродроме в порядке убывания времени посадки (использовать индексную сортировку методом «пузырька»);

2) обеспечить входной контроль времени посадки, бортового номера и аэродрома посадки, выполнить отладку и тестирование.

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

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

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.

1. Основные теоретические аспекты

1.1. 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
```

1.2.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
            }
        }
    }
}
```

1.3. 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)**

1.4.Валидация данных

Программа проверяет:
1. **Время посадки** (`HH:MM`):
- Должно быть в формате `ЧЧ:MM` (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);
}
```

1.5.Пример сортировки

Рассмотрим массив времен посадки в минутах:

Исходные данные: [125, 80, 300, 45]

Индексы: [0, 1, 2, 3]

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

- Сравниваем 1 и 2: 300 > 80 - порядок верный
- Массив: [300, 125, 80, 45]

3. Третья итерация:

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

4. Результат:

Отсортированные индексы: [2, 0, 1, 3]

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

Решение

2. Код программы:

2.1.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)
    {
```

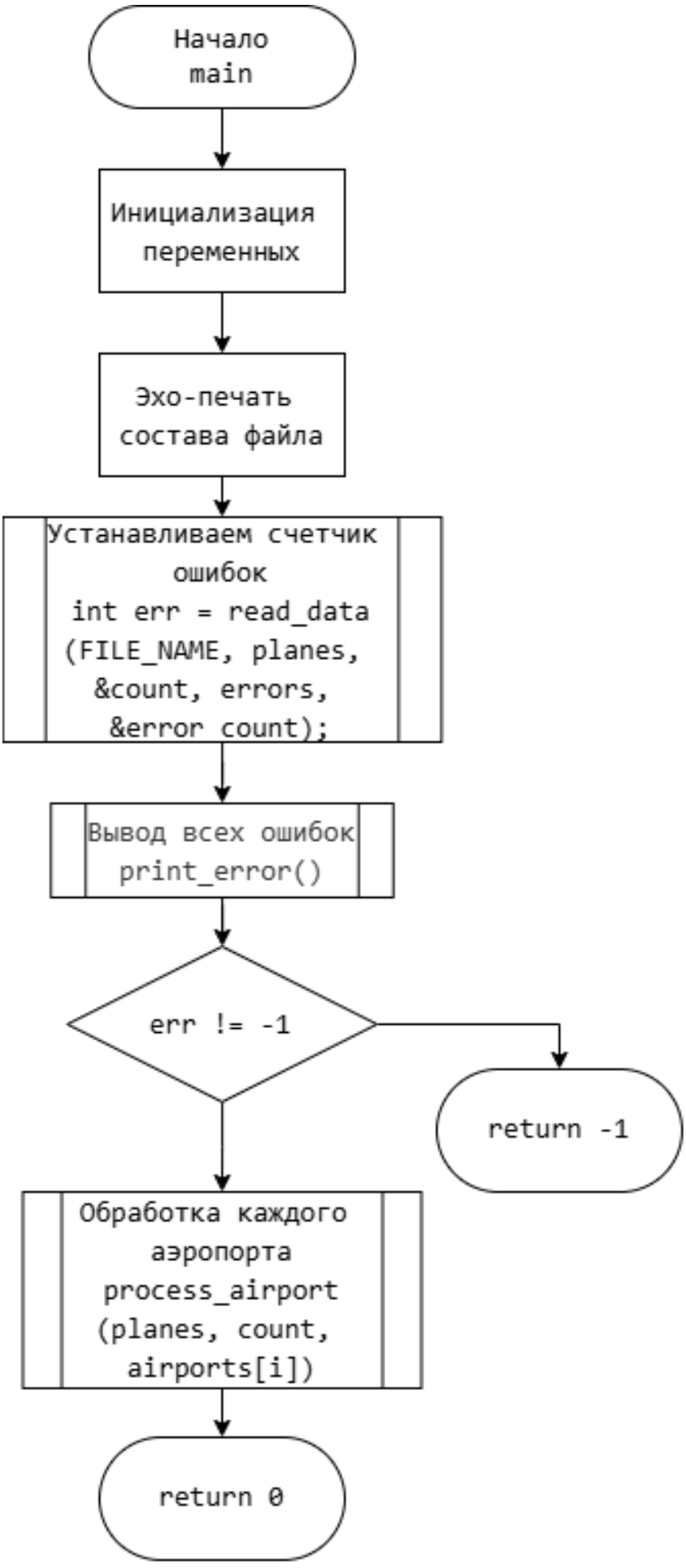
```
        char msg[250];
        snprintf(msg, sizeof(msg), "Минуты должны быть в диапазоне 00-59 (получено %02d)", minutes);
        strncpy(errors[*error_count].field_name, "Время", sizeof(errors[*error_count].field_name) - 1);
        strncpy(errors[*error_count].message, msg, sizeof(errors[*error_count].message) - 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 ******************************/
```

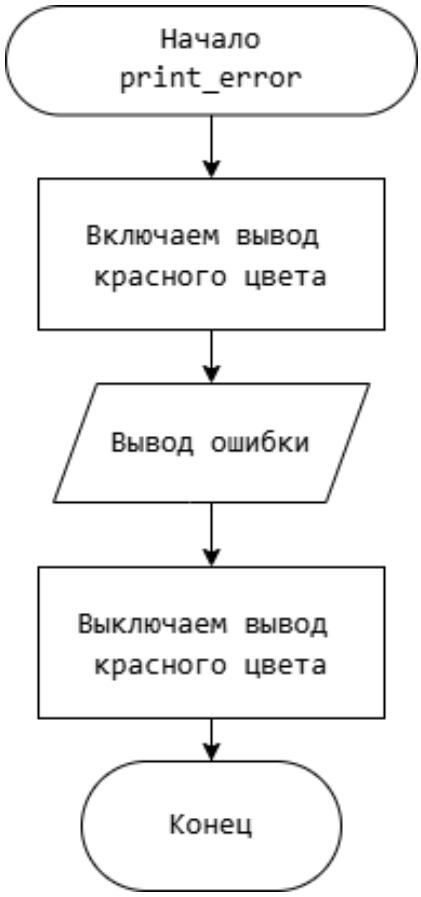

Начало
main
Инициализация
переменных
Эхо-печать
состава файла
Устанавливаем счетчик
ошибок
int err = read_data
(FILE_NAME, planes,
&count, errors,
&error_count);
Вывод всех ошибок
print_error()
err != -1
return -1
Обработка каждого
аэропорта
process_airport
(planes, count,
airports[i])
return 0

3. Описание функций

3.1. 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

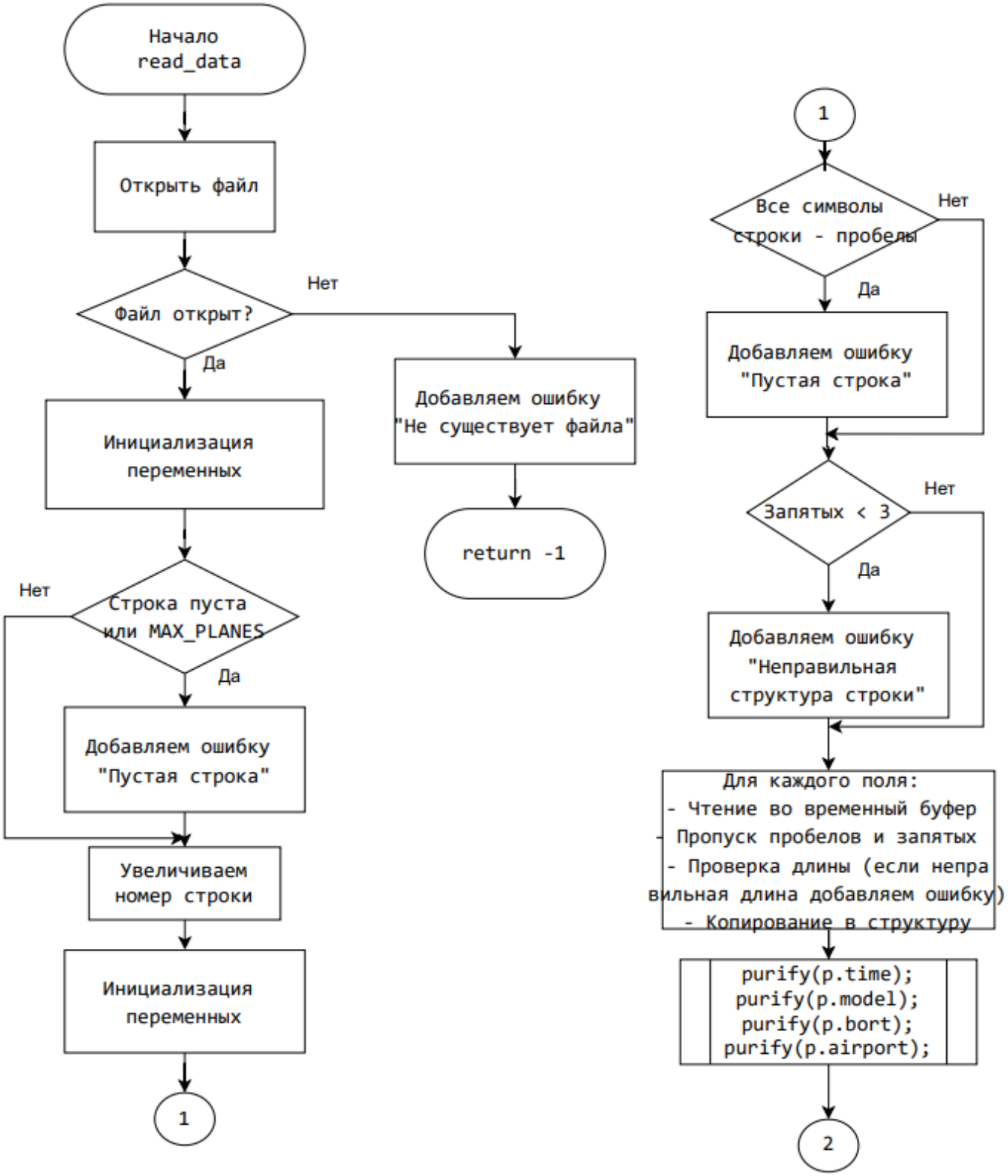


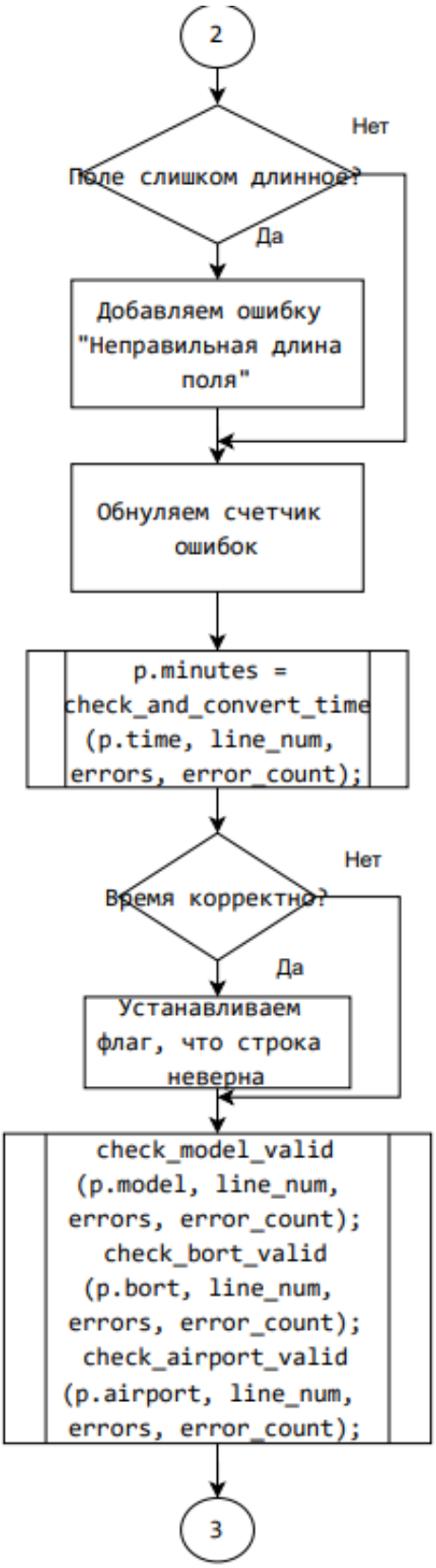
3.2. Функция read_data

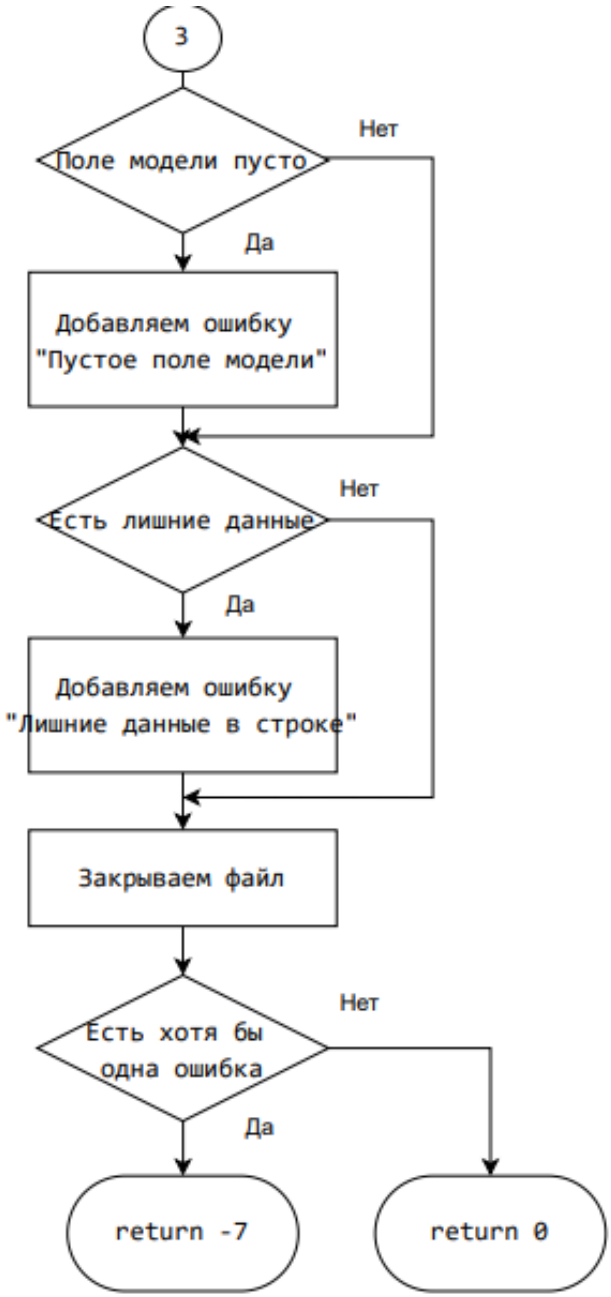
Назначение
Читает данные из файла, парсит строки, валидирует поля и заполняет массив структур Plane.
Прототип
```
int read_data(const char *filename, Plane *planes, int *count, ErrorInfo *errors, int
*error_count);
```
Обращение к функции:

```
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


Начало
read_data
Открыть файл
Файл открыт?
Нет
Да
Добавляем ошибку
"Не существует файла"
return -1
Инициализация
переменных
Строка пуста
или MAX_PLANES
Нет
Да
Добавляем ошибку
"Пустая строка"
Увеличиваем
номер строки
Инициализация
переменных
1
1
Все символы
строки - пробелы
Нет
Да
Добавляем ошибку
"Пустая строка"
Запятых < 3
Нет
Да
Добавляем ошибку
"Неправильная
структура строки"
Для каждого поля:
- Чтение во временный буфер
- Пропуск пробелов и запятых
- Проверка длины (если непра
вильная длина добавляем ошибку)
- Копирование в структуру
purify(p.time);
purify(p.model);
purify(p.bort);
purify(p.airport);
2

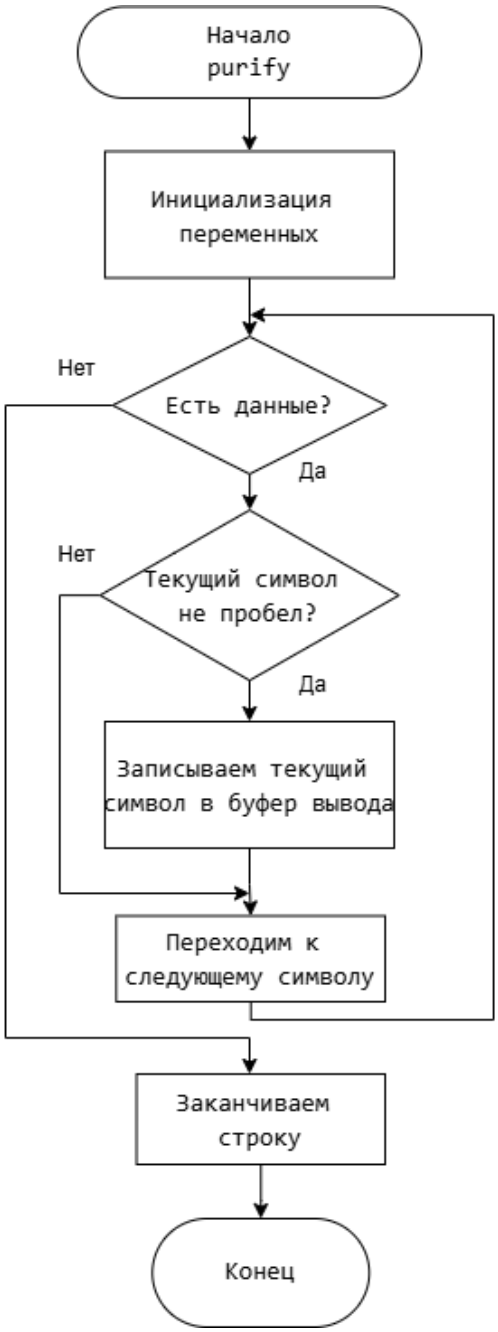

2
Поле слишком длинное?
Нет
Да
Добавляем ошибку "Неправильная длина поля"
Обнуляем счетчик ошибок
p.minutes = check_and_convert_time (p.time, line_num, errors, error_count);
Время корректно?
Нет
Да
Устанавливаем флаг, что строка неверна
check_model_valid (p.model, line_num, errors, error_count);
check_bort_valid (p.bort, line_num, errors, error_count);
check_airport_valid (p.airport, line_num, errors, error_count);
3


3
Поле модели пусто
Нет
Да
Добавляем ошибку "Пустое поле модели"
Есть лишние данные
Нет
Да
Добавляем ошибку "Лишние данные в строке"
Закрываем файл
Есть хотя бы одна ошибка
Нет
Да
return -7
return 0

3.3. 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



3.4. Функция check_and_convert_time

Назначение

Проверяет корректность формата времени (HH:MM) и конвертирует его в минуты.

Прототип

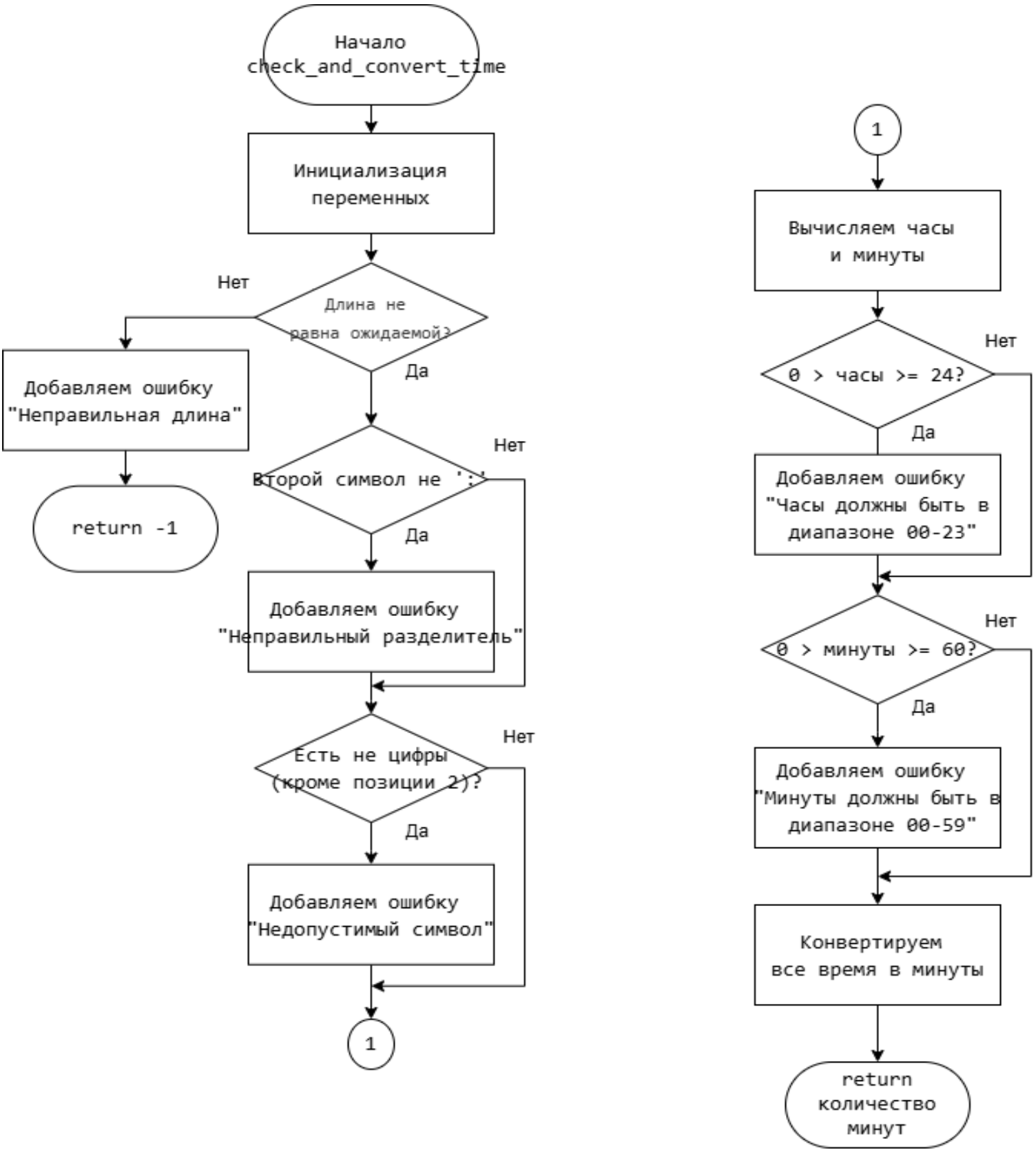
```
int check_and_convert_time(const char *time, int line_num, ErrorInfo *errors, int
*error_count);
```

Обращение к функции:

```
check_and_convert_time(p.time, line_num, errors, error_count);
```

Описание параметров:

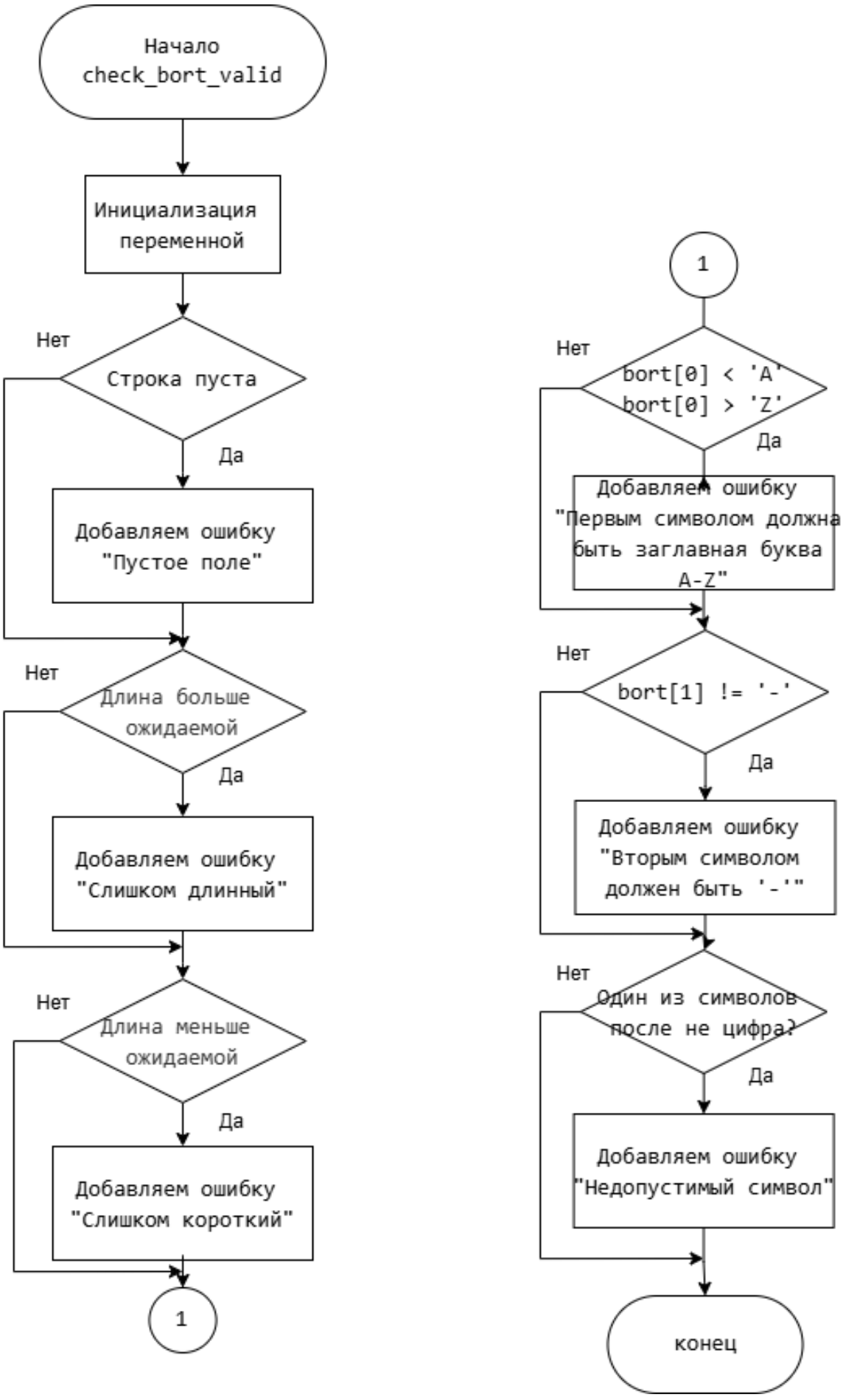
Идентификатор	Тип	Назначение	Входной/Выходной
time	const char*	Строка времени для проверки	Входной
line_num	int	Номер строки	Входной
errors	ErrorInfo*	Массив ошибок	Входной/Выходной
error_count	int*	Указатель на счетчик ошибок	Входной/Выходной


Начало
check_and_convert_time
Инициализация
переменных
Нет
Длина не
равна ожидаемой?
Да
Добавляем ошибку
"Неправильная длина"
return -1
Нет
Второй символ не ':'
Да
Добавляем ошибку
"Неправильный разделитель"
Нет
Есть не цифры
(кроме позиции 2)?
Да
Добавляем ошибку
"Недопустимый символ"
1
1
Вычисляем часы
и минуты
Нет
0 > часы >= 24?
Да
Добавляем ошибку
"Часы должны быть в
диапазоне 00-23"
Нет
0 > минуты >= 60?
Да
Добавляем ошибку
"Минуты должны быть в
диапазоне 00-59"
Конвертируем
все время в минуты
return
количество
минут

3.5. 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:

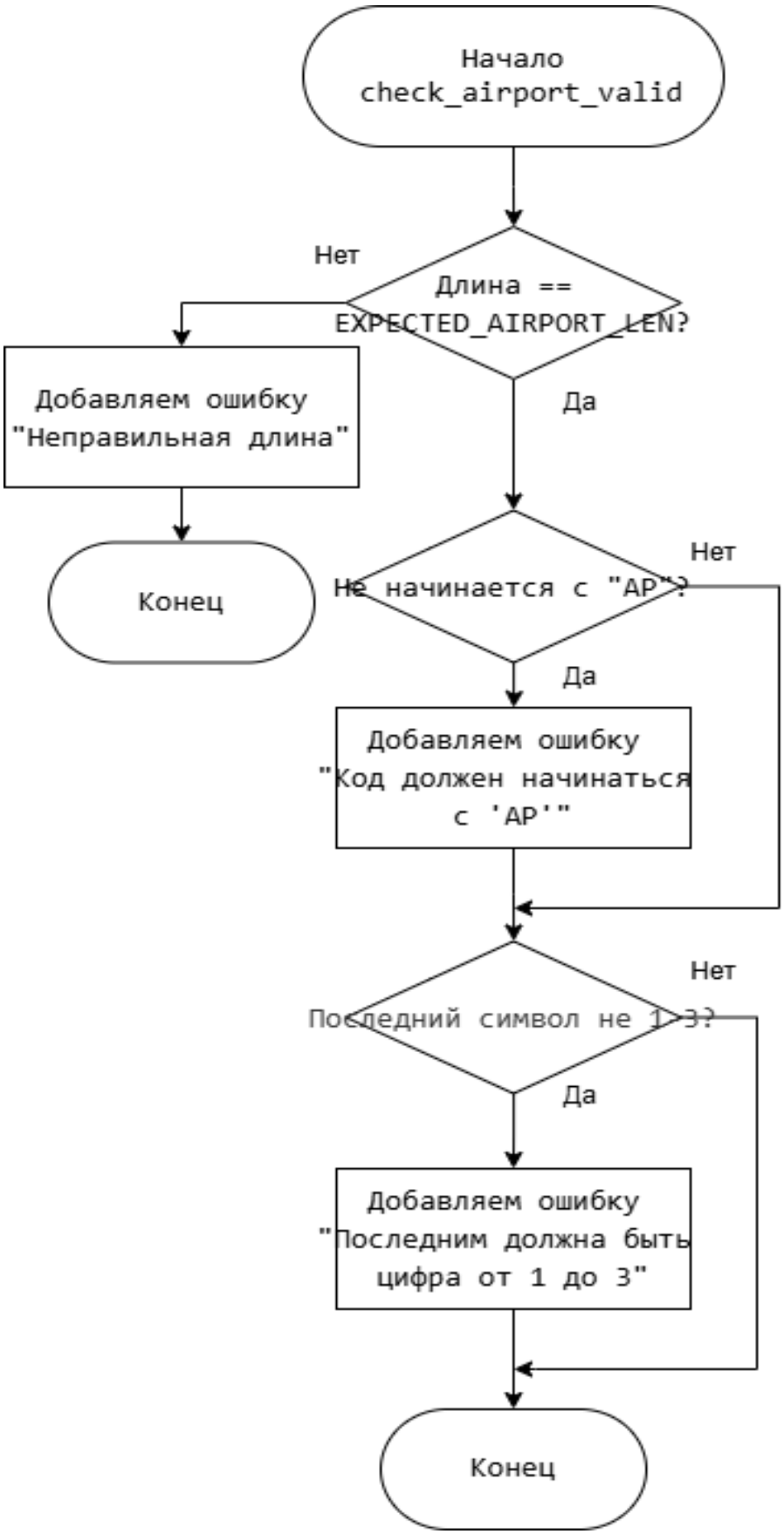
Идентификатор	Тип	Назначение	Входной/Выходной
bort	const char*	Бортовой номер для проверки	Входной
line_num	int	Номер строки в файле	Входной
errors	ErrorInfo*	Массив ошибок	Входной/Выходной
error_count	int*	Указатель на счетчик ошибок	Входной/Выходной


Начало
check_bort_valid
Инициализация
переменной
Нет
Строка пуста
Да
Добавляем ошибку
"Пустое поле"
Нет
Длина больше
ожидаемой
Да
Добавляем ошибку
"Слишком длинный"
Нет
Длина меньше
ожидаемой
Да
Добавляем ошибку
"Слишком короткий"
1
1
Нет
bort[0] < 'A'
bort[0] > 'Z'
Да
Добавляем ошибку
"Первым символом должна
быть заглавная буква
A-Z"
Нет
bort[1] != '-'
Да
Добавляем ошибку
"Вторым символом
должен быть '-'"
Нет
Один из символов
после не цифра?
Да
Добавляем ошибку
"Недопустимый символ"
конец

3.6.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:

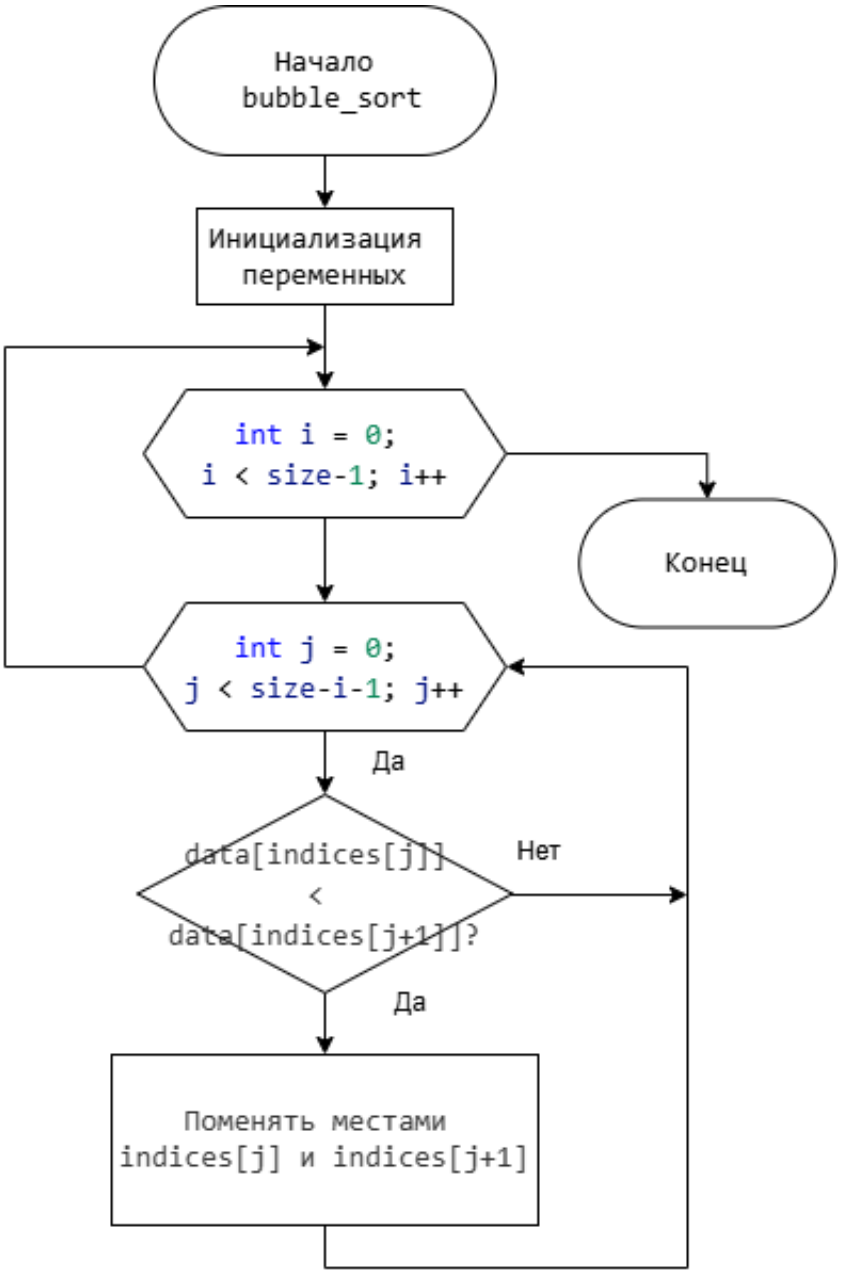
Идентификатор	Тип	Назначение	Входной/Выходной
airport	const char*	Код аэродрома для проверки	Входной
line_num	int	Номер строки в файле	Входной
errors	ErrorInfo*	Массив ошибок	Входной/Выходной
error_count	int*	Указатель на счетчик ошибок	Входной/Выходной


Начало
check_airport_valid
Нет
Длина ==
EXPECTED_AIRPORT_LEN?
Да
Добавляем ошибку
"Неправильная длина"
Конец
Нет
Не начинается с "AP"?
Да
Добавляем ошибку
"Код должен начинаться
с 'AP'"
Нет
Последний символ не 1-3?
Да
Добавляем ошибку
"Последним должна быть
цифра от 1 до 3"
Конец

3.7. 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:

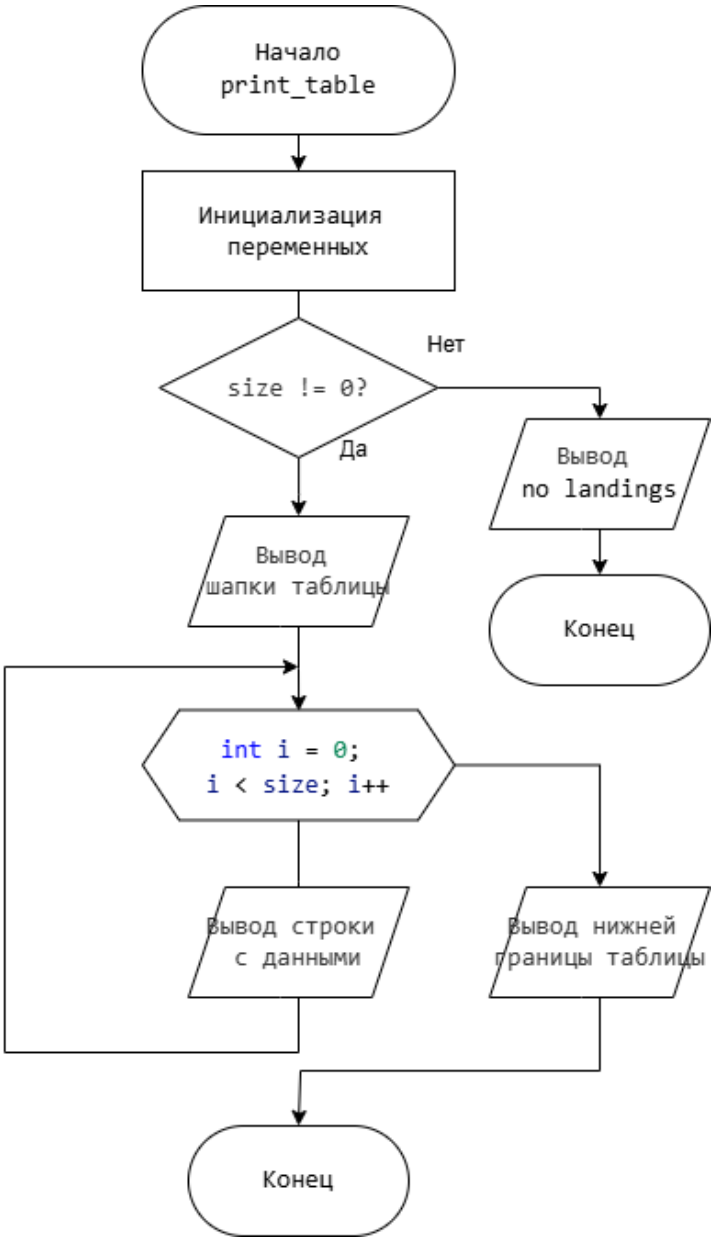
Идентификатор	Тип	Назначение	Входной/Выходной
indices	int*	Массив индексов	Входной/Выходной
size	int	Размер массива	Входной
data	Plane*	Массив структур Plane	Входной



3.8. 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:

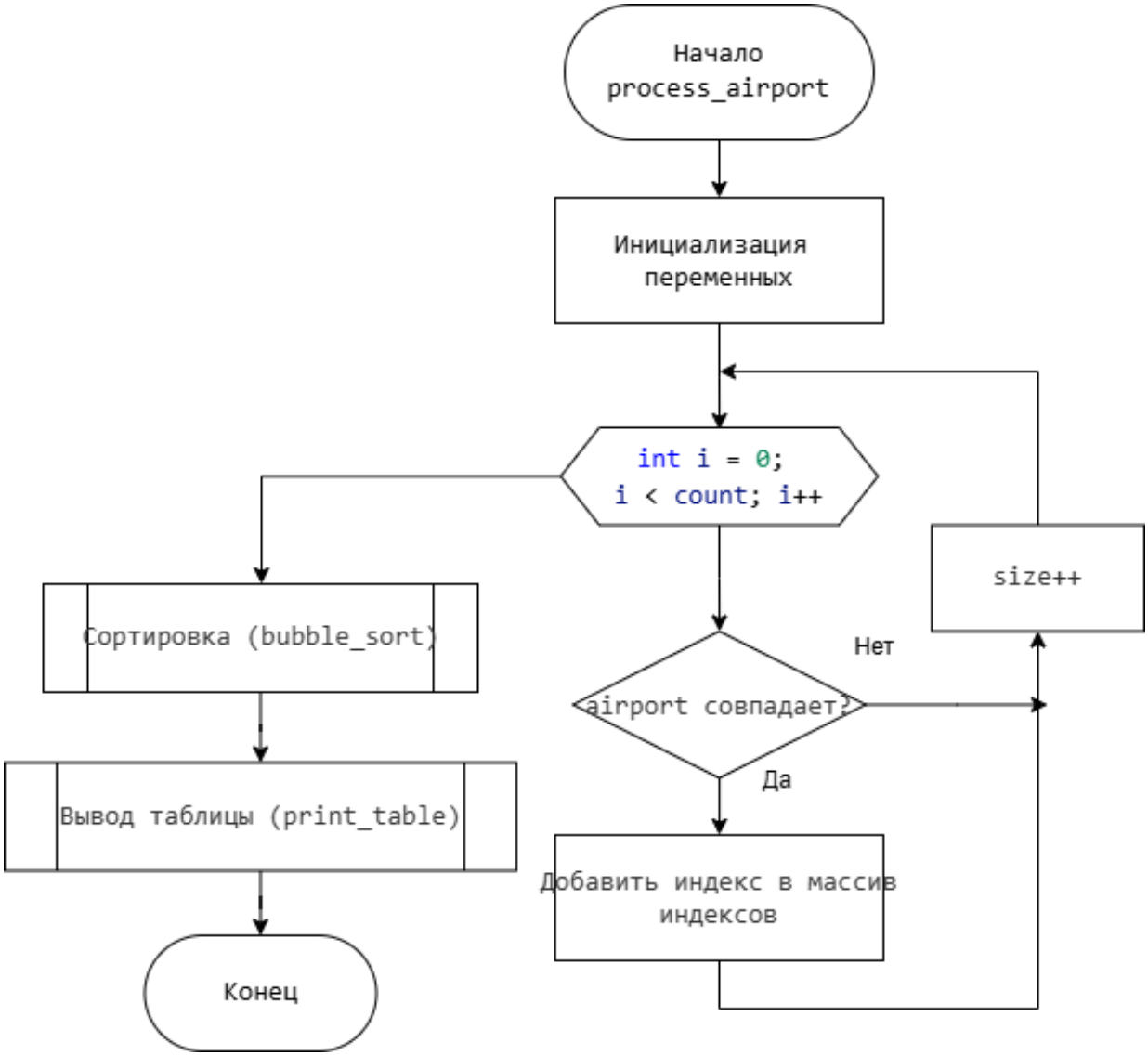
Идентификатор	Тип	Назначение	Входной/Выходной
indices	int*	Массив индексов	Входной
size	int	Количество записей	Входной
data	Plane*	Массив структур Plane	Входной
airport	const char*	Код аэродрома	Входной



3.9. 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



3.10. Функция check_model_valid

Назначение

Проверяет валидность поля “Модель самолета”.

Прототип

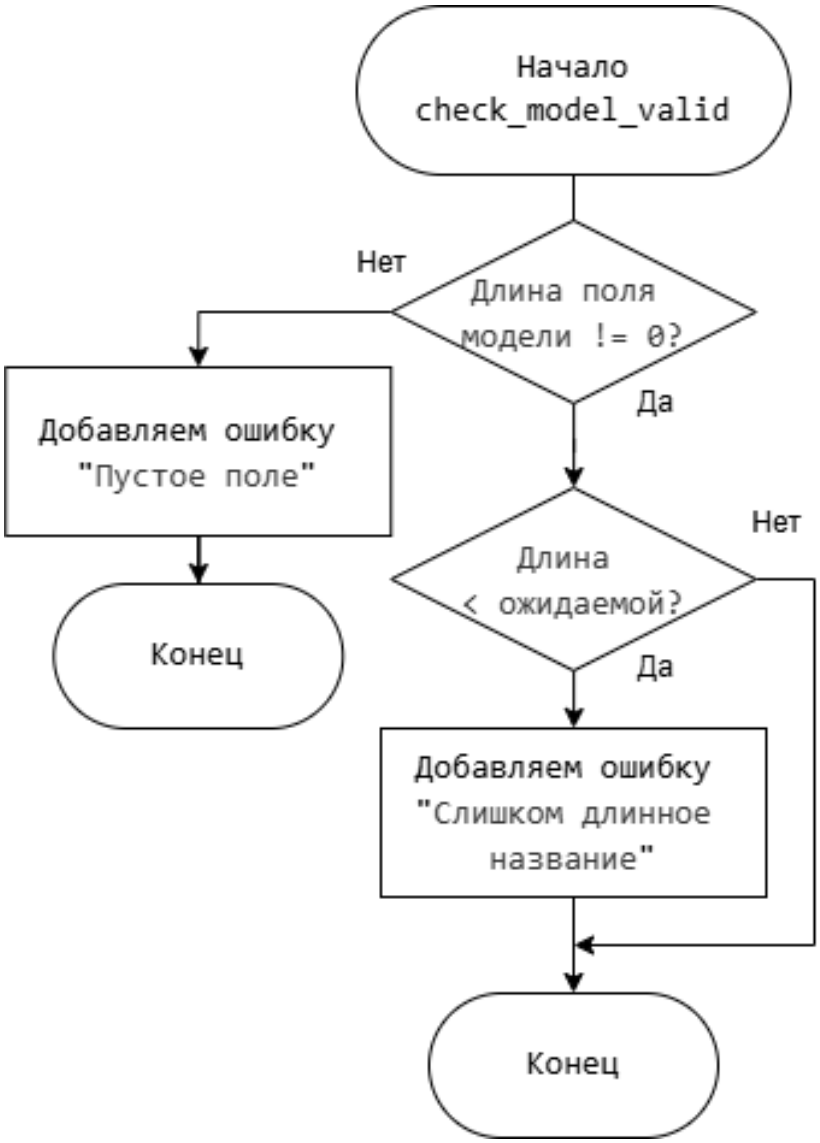
```
void check_model_valid(const char *model, int line_num, ErrorInfo *errors, int
*error_count);
```

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

1. *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:

```
Найдены ошибки:
Строка 1, поле 'Время': Часы должны быть в диапазоне 00-23 (получено 25)
Строка 2, поле 'Время': Минуты должны быть в диапазоне 00-59 (получено 60)
Строка 3, поле 'Время': Неправильный разделитель (должен быть ':', а получе
н '-')
```

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:

```
Найдены ошибки:
а 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)
```

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:

```
Найдены ошибки:
Строка 1, поле 'Аэродром': Последний символ должен быть цифрой 1-3 (получено 'X')
Строка 2, поле 'Аэродром': Неправильная длина (должно быть 3 символа, получено 2)
Строка 3, поле 'Аэродром': Поле слишком длинное (максимум 3 символов, получено 4)
```

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:

```
Найдены ошибки:
Строка 1, поле 'Модель': Пустое поле
Строка 2, поле 'Модель': Пустое поле
```

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:

```
Найдены ошибки:
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
```

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:

```
Найдены ошибки:
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
```

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:

```
Найдены ошибки:
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)
```

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:

```
Найдены ошибки:
Строка 1, поле 'Строка': Пустая строка
Строка 3, поле 'Строка': Пустая строка

Успешно обработано записей: 2
```

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

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:

```
Найдены ошибки:
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 'Бортовой номер': Первым символом должна быть заглавная буква A-Z (получено 'b')

Успешно обработано записей: 1


Airport AP1:
```

Time	Model	Bort Number	Airport
12:30	AIRBUS-A320	A-1233	AP1

```
Airport AP2: no landings
Airport AP3: no landings
```

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:

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

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:

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

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:

Airport AP1:

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

Airport AP2: no landings

Airport AP3:

Time	Model	Bort Number	Airport
23:59	CONCORDE	C-9999	AP3

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 00:00	Boeing-777 MINI-JET	A-1234 A-0000	AP1 AP1

Полученный результат:

Airport AP1:

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

Airport AP2: no landings
Airport AP3: no landings

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:

```
Успешно обработано записей: 0

Airport AP1: no landings
Airport AP2: no landings
Airport AP3: no landings
```

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:

```
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

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:

```
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

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:

```
Найдены ошибки:
Строка 1, поле 'Строка': Неправильное построение строки (недостаточно запятых)
Строка 3, поле 'Строка': Неправильное построение строки (недостаточно запятых)

Успешно обработано записей: 1


Airport AP1:
```

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

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:

```
Airport AP1:
```

Time	Model	Bort Number	Airport
00:05	MINI-JET	A-0005	AP1

```
Airport AP2: no landings

Airport AP3:
```

Time	Model	Bort Number	Airport
05:09	CESSNA-172	C-0909	AP3

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:

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

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.