

1. Место и сроки проведения практики:Наименование организации: МАИ, кафедра 304

Сроки проведения практики

дата начала практики: 10.02.2025дата окончания практики: 06.06.2025**2. Инструктаж по технике безопасности:**

_____/ Секретарев В.Е. / _____ 20__ г.
подпись проводившего *расшифровка подписи* *дата проведения*

3. Индивидуальное задание обучающегося:Написать программы на языке СИ по вариантам.**4. План выполнения индивидуального задания обучающегося:**

№ п/п	Место проведения	ТемаФ	Период выполнения
1		Задание 1, вариант 5	12.03.2023
2		Оформление отчета. Подведение итогов.	03.06.2023

Утверждаю

_____/ Секретарев В.Е. / _____ 20__ г.
подпись руководителя от МАИ *расшифровка подписи* *дата утверждения**

_____/ _____ / _____ 20__ г.
подпись руководителя от организации/предприятия *расшифровка подписи* *дата утверждения**

Ознакомлен

_____/ _____ / _____ 20__ г.

_____/ _____ / _____ 20__ г.

*подпись обучающегося**расшифровка подписи**дата ознакомления****Дата утверждения и ознакомления – дата начала практики*

[illegible]

data

Отчет по практике

Оглавление

Постановка задачи.....	5
Теория.....	6
Блок-схема.....	7
Описание функций.....	8
Код программы.....	9
Тесты.....	10
Вывод.....	11

Постановка задачи

ВАРИАНТ № 5

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:

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

- 1) подготовить программу, осуществляющую печать таблицы о самолетах совершающих посадку на каждом аэродроме в порядке возрастания времени посадки (использовать индексную сортировку методом «пузырька»);
- 2) обеспечить входной контроль бортового номера, времени посадки и аэродрома посадки, выполнить отладку и тестирование.

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

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

Теория

Сортировка пузырьком — это один из самых простых алгоритмов сортировки. Он получил своё название потому, что большие элементы, как пузыри, «всплывают» вверх (то есть к концу массива) после каждой итерации. Хотя алгоритм очень прост, он крайне неэффективен для больших массивов, поэтому используется в основном в учебных целях.

Принцип работы:

Алгоритм выполняет много проходов по массиву, каждый раз сравнивая пары соседних элементов. Если они стоят в неправильном порядке — меняет их местами.

Рассмотрим шаг за шагом:

1. Последовательное сравнение:

В начале алгоритм начинает с первого элемента и движется вправо:

- Сравнивается первый и второй элемент.
- Если первый больше второго — меняем их местами.
- Затем сравнивается второй и третий элемент, и так далее до конца массива.

Этот процесс называется одним проходом.

2. «Всплытие» максимального элемента:

После первого прохода самый большой элемент уже окажется в конце массива — его "вытолкнули" вправо за счёт последовательных обменов. Он уже на своём месте и не будет участвовать в следующих сравнениях.

Пример (первый проход):

У нас есть набор из 5 чисел расположенных в произвольном порядке. Преминяя алгоритм сортировки «пузырьком», на первом проходе мы можем наблюдать всплытие элемента 5

5, 3, 2, 4, 1 \rightarrow 3, 2, 4, 1, 5

Пример работы алгоритма

Отсортируем массив по возрастанию:

Исходный массив: 4, 2, 5, 1, 3

Первый проход:

- $4 > 2 \rightarrow$ меняем \rightarrow 2, 4, 5, 1, 3
- $4 < 5 \rightarrow$ ничего
- $5 > 1 \rightarrow$ меняем \rightarrow 2, 4, 1, 5, 3
- $5 > 3 \rightarrow$ меняем \rightarrow 2, 4, 1, 3, 5

Второй проход:

- $2 < 4 \rightarrow$ ничего
- $4 > 1 \rightarrow$ меняем \rightarrow 2, 1, 4, 3, 5
- $4 > 3 \rightarrow$ меняем \rightarrow 2, 1, 3, 4, 5

Третий проход:

- $2 > 1 \rightarrow$ меняем \rightarrow 1, 2, 3, 4, 5

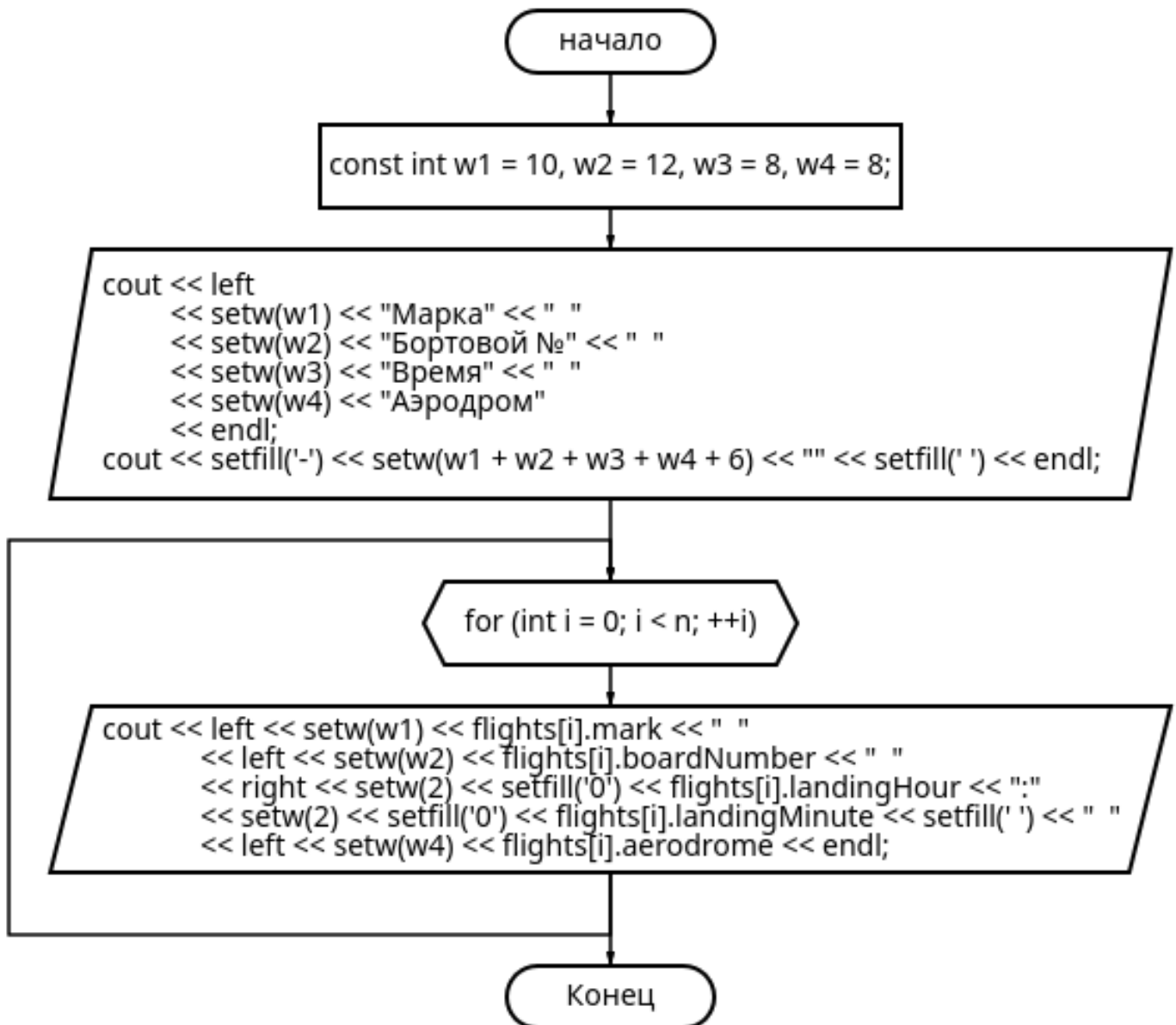
Четвёртый проход:

Ничего не меняем т. к. всё отсортировано.

Для лучшей визуализации алгоритма сортировки, можно посмотреть видеоролик на YouTube:

<https://www.youtube.com/watch?v=kPRA0W1kECg> (4:01 сортировка пузырьком)

Блок-схема



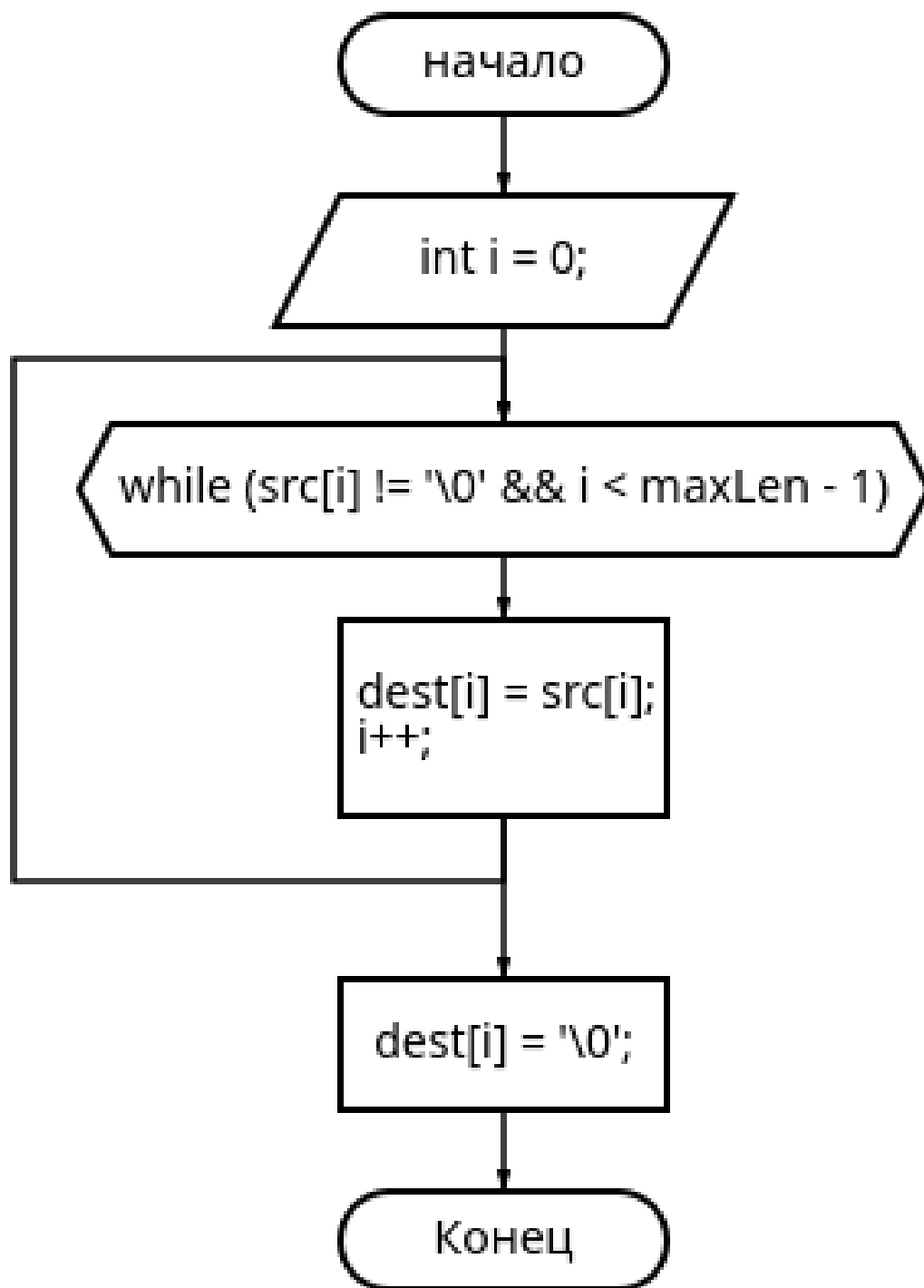
Описание функций

Функция copyStr:

1. Назначение: Копирует строку src в dest, не превышая maxLen (включая завершающий нуль);
2. Прототип функции: void copyStr(char* dest, const char* src, int maxLen);
3. Обращение: copyStr(f.mark, mark, MARK_SIZE);
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/ Выходной
copyStr	void	Копирует строку src в dest, не превышая maxLen	выходной
dest	char*	Массив символов, куда будет скопирована строка	входной
src	const char*	Исходная строка для копирования	входной
maxLen	int	Максимальный размер буфера dest	входной

5. Блоксхема:

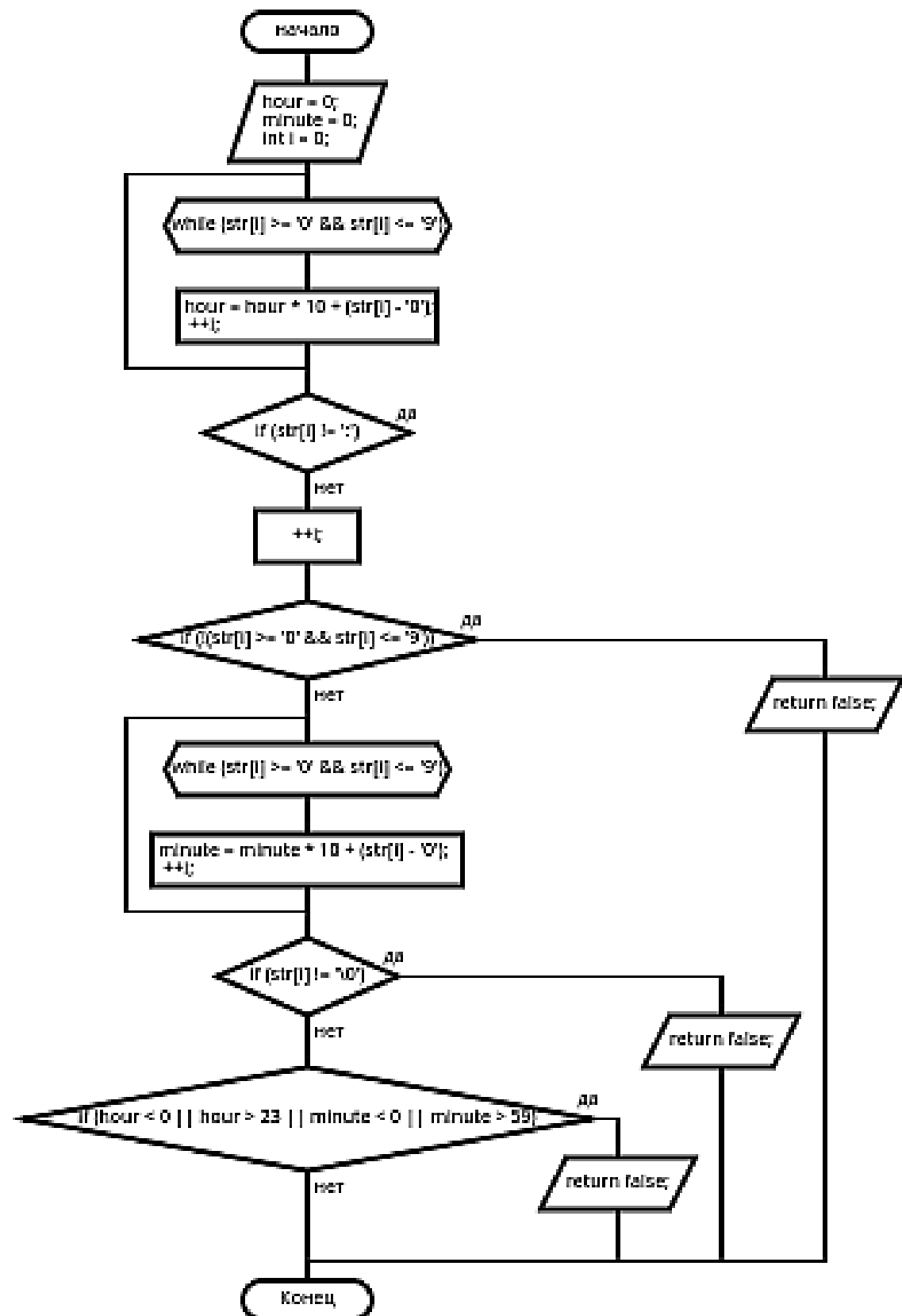


Функция `parseTime`:

1. Назначение: Разбирает строку времени ("ЧЧ:ММ" или "Ч:ММ") на часы и минуты;
2. Прототип функции: `bool parseTime(const char* str, int& hour, int& minute);`
3. Обращение: `parseTime(timeStr, f.landingHour, f.landingMinute);`
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/ Выходной
<code>parseTime</code>	<code>bool</code>	Разбирает строку времени на часы и минуты	выходной
<code>str</code>	<code>char*</code>	строка времени для разбора	входной
<code>hour</code>	<code>int&</code>	переменная для записи разобранных часов	входной
<code>minute</code>	<code>int&</code>	переменная для записи разобранных минут	входной

5. Блоксхема:



Функция `landingTimeToMinutes`:

1. Назначение: Возвращает время посадки в минутах от полуночи для данного рейса.
2. Прототип функции: `int landingTimeToMinutes(const Flight& f);`
3. Обращение: `landingTimeToMinutes(flights[idx[j+1]]);`
4. Описание параметров:

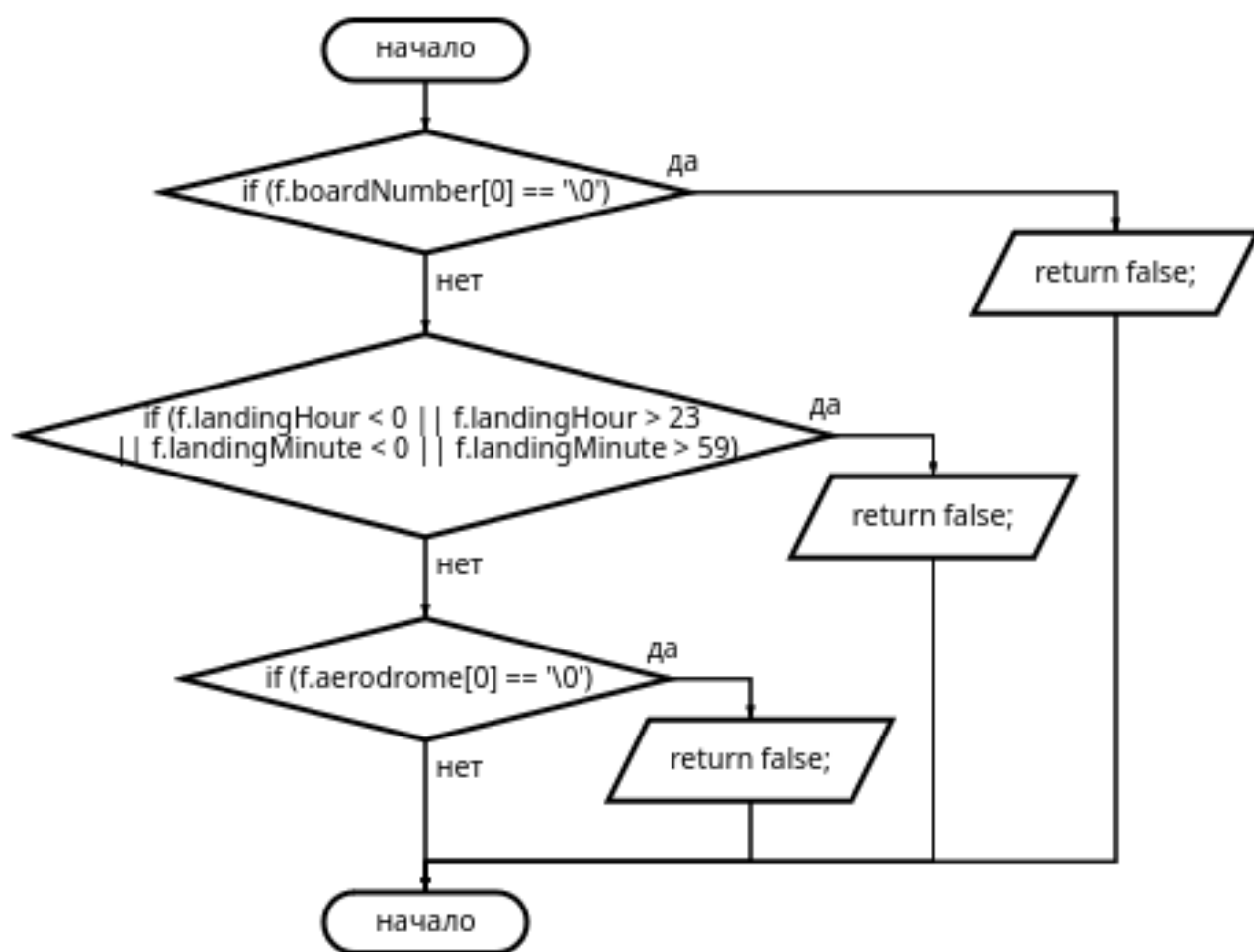
Идентификатор	Тип	Назначение	Входной/ Выходной
<code>landingTimeToMinutes</code>	<code>int</code>	Возвращает время посадки в минутах от полуночи для данного рейса	выходной
<code>f</code>	<code>const Flight&</code>	Структура <code>Flight</code> с данными рейса	входной

Функция `isValidFlight`:

1. Назначение: Проверяет корректность данных рейса;
2. Прототип функции: `bool isValidFlight(const Flight& f);`
3. Обращение: `isValidFlight(f);`
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/ Выходной
<code>isValidFlight</code>	<code>bool</code>	Проверяет корректность данных рейса	выходной
<code>f</code>	<code>const Flight&</code>	строка времени для разбора структура <code>Flight</code> для проверки	входной

5. Блоксхема:

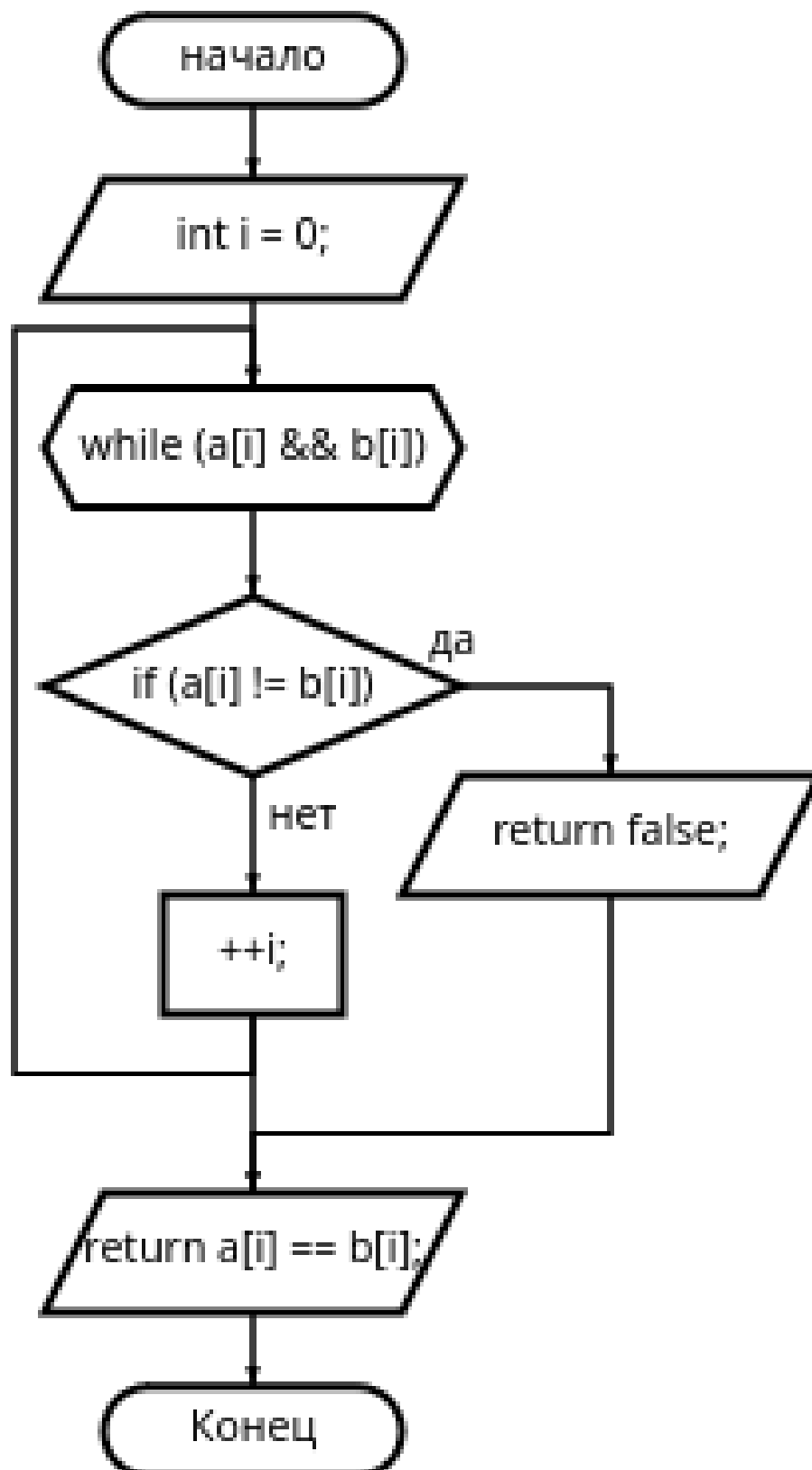


Функция areStringsEqual:

1. Назначение: Сравнивает две строки на равенство;
2. Прототип функции: `bool areStringsEqual(const char* a, const char* b);`
3. Обращение: `areStringsEqual(flights[k].aerodrome, targetAerodrome);`
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/ Выходной
<code>areStringsEqual</code>	<code>bool</code>	Сравнивает две строки на равенство	выходной
<code>a</code>	<code>const char*</code>	первая строка	входной
<code>b</code>	<code>const char*</code>	вторая строка	входной

5. Блоксхема:

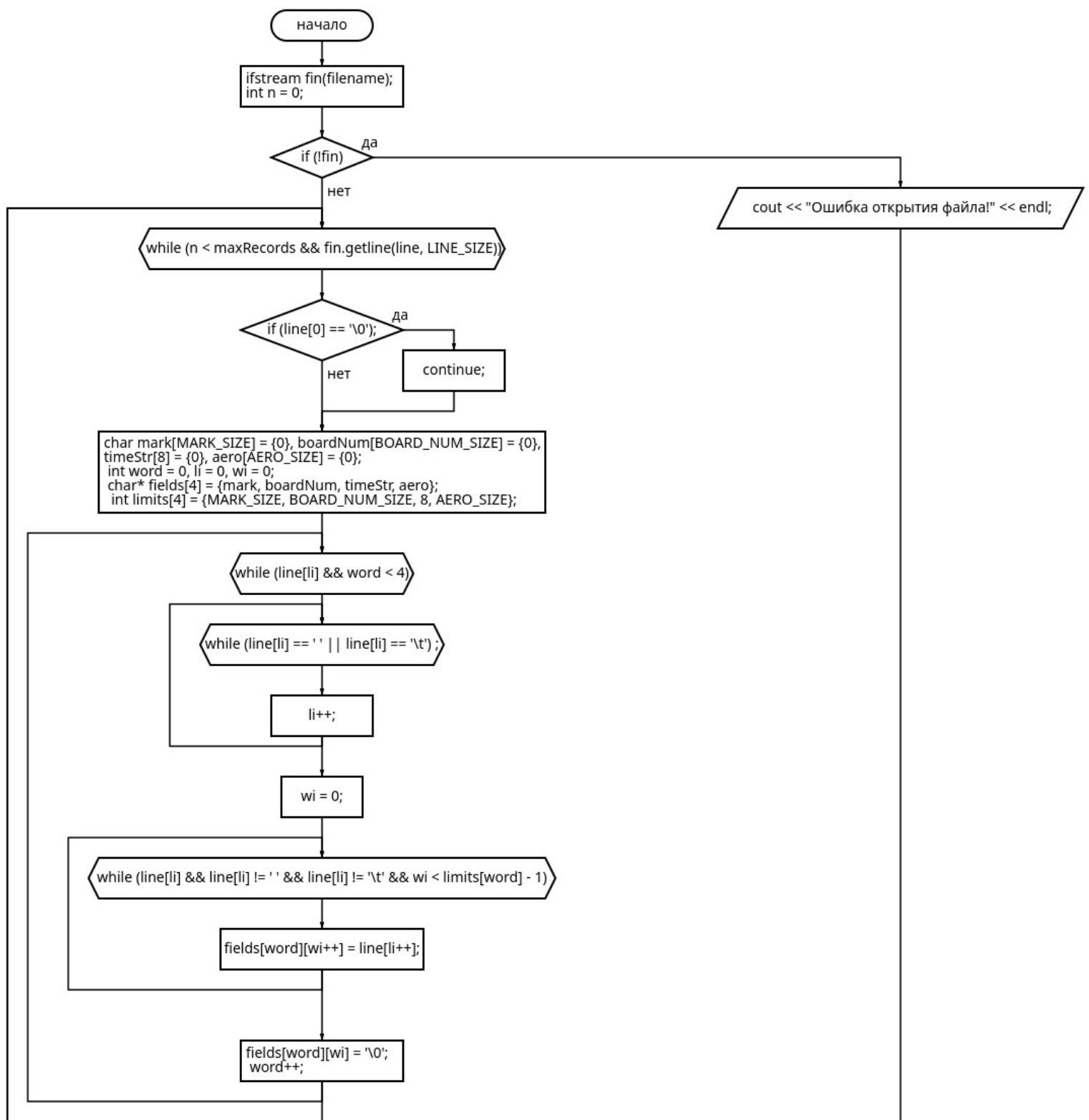


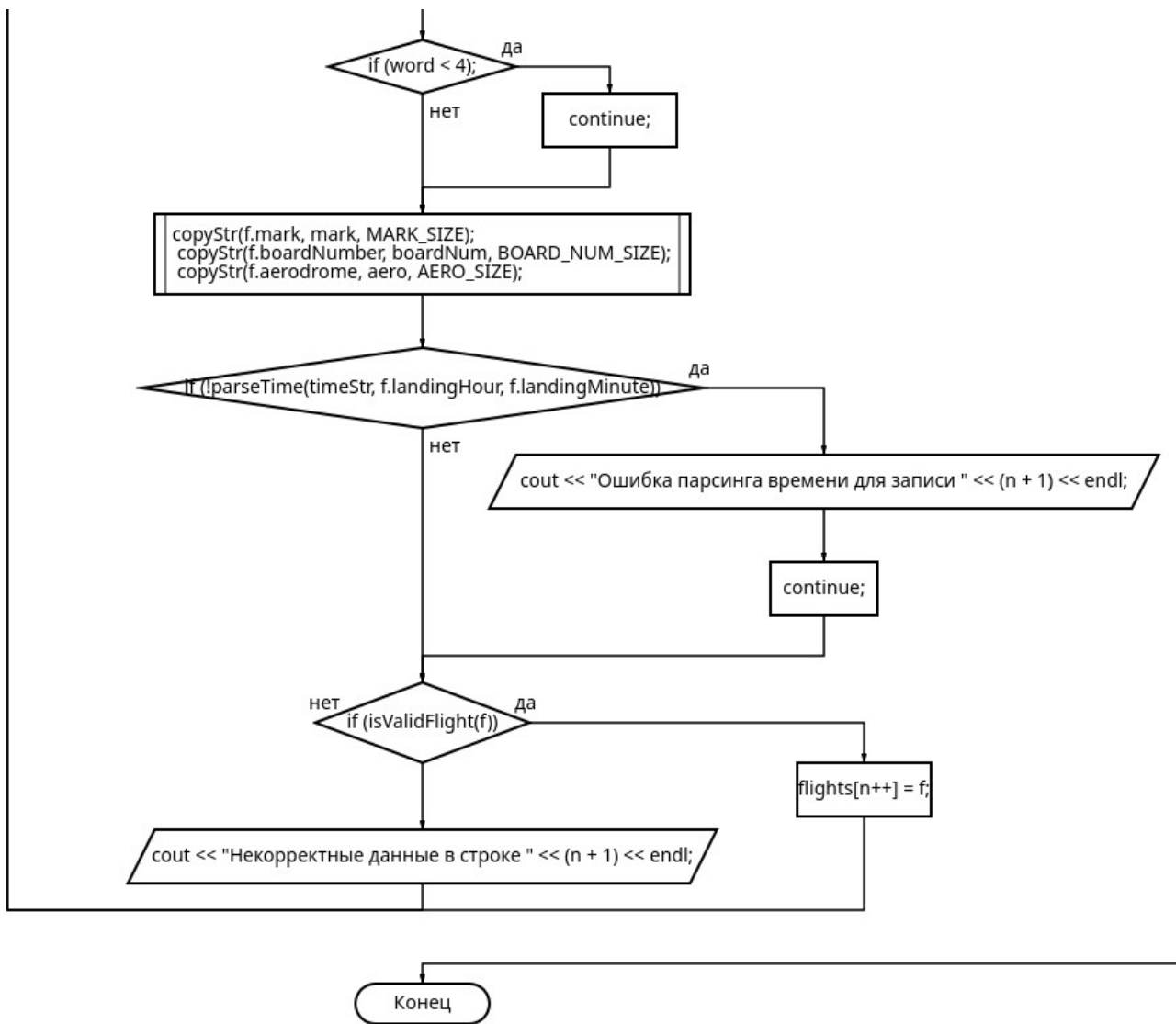
Функция readFlights:

1. Назначение: Считывает данные рейсов из файла filename в массив flights (максимум maxRecords штук);
2. Прототип функции: `int readFlights(const char* filename, Flight flights[], int maxRecords);`
3. Обращение: `readFlights(argv[1], flights, MAX_RECORDS);`
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/ Выходной
readFlights	int	Разбирает строку времени на часы и минуты	выходной
filename	const char*	имя файла для чтения данных	входной
flights	Flight	массив структур Flight, куда будут записаны считанные данные	входной
maxRecords	int	максимальное количество записей для чтения	входной

5. Блоксхема:



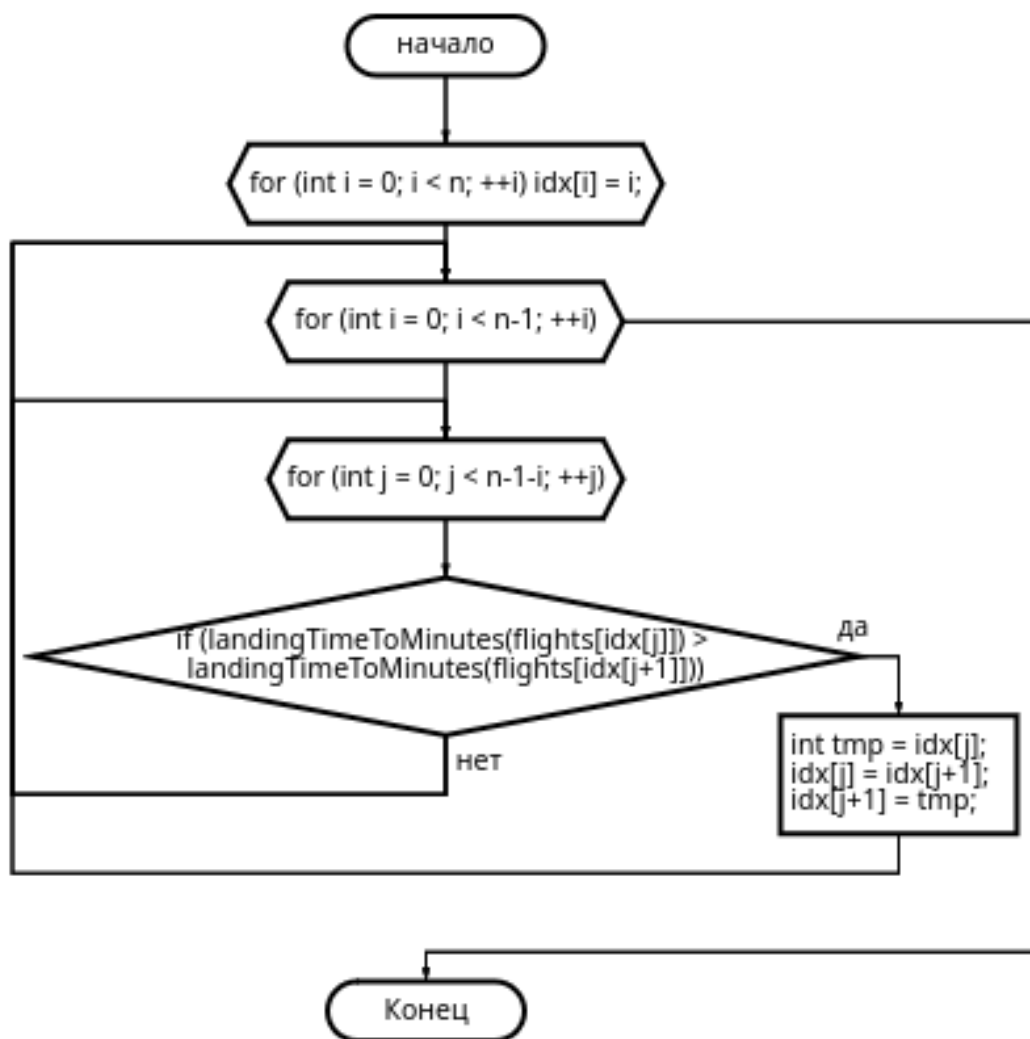


Функция `indexBubbleSort`:

1. Назначение: Сортирует индексы рейсов в массиве `idx` по времени посадки;
2. Прототип функции: `void indexBubbleSort(const Flight flights[], int idx[], int n);`
3. Обращение: `indexBubbleSort(flights, idx, n);`
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/ Выходной
<code>indexBubbleSort</code>	<code>void</code>	Сортирует индексы рейсов в массиве <code>idx</code> по времени посадки	выходной
<code>flights</code>	<code>const Flight</code>	массив структур <code>Flight</code> для сравнения времени посадки	входной
<code>idx</code>	<code>int</code>	массив индексов, который будет отсортирован по времени посадки	входной
<code>n</code>	<code>int</code>	количество рейсов/индексов для сортировки	входной

5. Блоксхема:

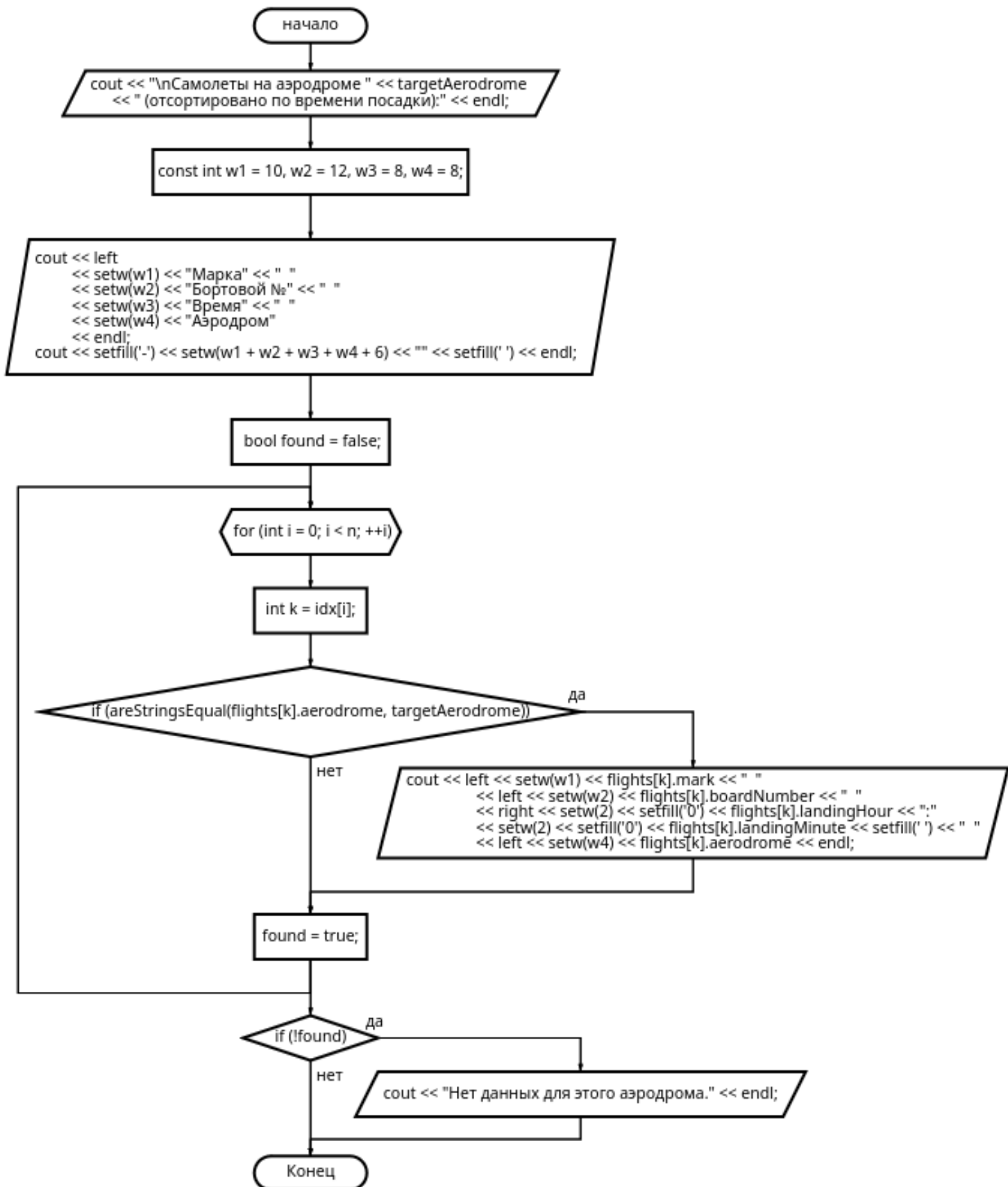


Функция printFlightsByAerodrome:

1. Назначение: Выводит рейсы, приземлившиеся на targetAerodrome, в порядке времени посадки;
2. Прототип функции: void printFlightsByAerodrome(const Flight flights[], const int idx[], int n, const char* targetAerodrome);
3. Обращение: printFlightsByAerodrome(flights, idx, n, "АПЗ");
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/ Выходной
printFlightsByAerodrome	void	Выводит рейсы, приземлившиеся на targetAerodrome, в порядке времени посадки	выходной
flights	const Flight	массив структур Flight	входной
idx	int	массив отсортированных индексов рейсов	входной
n	int	количество рейсов	входной
targetAerodrome	const char*	название аэродрома, для которого нужно вывести рейсы	входной

5. Блоксхема:



Функция `parseTime`:

1. Назначение: Разбирает строку времени ("ЧЧ:ММ" или "Ч:ММ") на часы и минуты;
2. Прототип функции: `bool parseTime(const char* str, int& hour, int& minute);`
3. Обращение: `parseTime(timeStr, f.landingHour, f.landingMinute);`
4. Описание параметров:

Идентификатор	Тип	Назначение	Входной/ Выходной
<code>parseTime</code>	<code>bool</code>	Разбирает строку времени на часы и минуты	выходной
<code>str</code>	<code>char*</code>	строка времени для разбора	входной
<code>hour</code>	<code>int&</code>	переменная для записи разобранных часов	входной
<code>minute</code>	<code>int&</code>	переменная для записи разобранных минут	входной

5. Блоксхема:

Код программы

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

// Максимальное количество рейсов, которые может обработать программа
const int MAX_RECORDS = 100;

// Размеры для строковых полей (включая завершающий нуль)
const int BOARD_NUM_SIZE = 20; // длина строки для бортового номера
const int AERO_SIZE = 10;      // длина строки для названия аэродрома
const int MARK_SIZE = 16;      // длина строки для марки самолета
const int LINE_SIZE = 64;      // максимальная длина строки при чтении из файла

// Структура для хранения информации о рейсе
struct Flight {
    char mark[MARK_SIZE];        // Марка самолёта, например "TU-154М"
    char boardNumber[BOARD_NUM_SIZE]; // Бортовой номер, например "В-3726"
    int landingHour;             // Часы посадки (0-23)
    int landingMinute;           // Минуты посадки (0-59)
    char aerodrome[AERO_SIZE];   // Аэродром посадки, например "AP2"
};

// Индексная структура для сортировки и доступа по индексу
struct FlightIndex {
    int idx;    // Индекс рейса в массиве Flight
    int time;   // Время посадки в минутах (для сортировки)
};

// Собственная реализация функции определения длины строки (аналог strlen)
int strLength(const char* s) {
    int len = 0;
    while (s[len] != '\0') {
        len++;
    }
    return len;
}

// Копирует строку src в dest, не превышая maxLen (включая завершающий нуль).
void copyStr(char* dest, const char* src, int maxLen) {
    int i = 0;
```

```

while (src[i] != '\0' && i < maxLen - 1) {
    dest[i] = src[i];
    i++;
}
dest[i] = '\0';
}

// Разбирает строку времени ("ЧЧ:ММ" или "Ч:ММ") на часы и минуты.
bool parseTime(const char* str, int& hour, int& minute) {
    hour = 0;
    minute = 0;
    int i = 0;
    if (!(str[i] >= '0' && str[i] <= '9')) return false;
    while (str[i] >= '0' && str[i] <= '9') {
        hour = hour * 10 + (str[i] - '0');
        ++i;
    }
    if (str[i] != ':') return false;
    ++i;
    if (!(str[i] >= '0' && str[i] <= '9')) return false;
    while (str[i] >= '0' && str[i] <= '9') {
        minute = minute * 10 + (str[i] - '0');
        ++i;
    }
    if (str[i] != '\0') return false;
    if (hour < 0 || hour > 23 || minute < 0 || minute > 59) return false;
    return true;
}

// Возвращает время посадки в минутах от полуночи
int landingTimeToMinutes(const Flight& f) {
    return f.landingHour * 60 + f.landingMinute;
}

// Проверяет, что символ - латинская буква или цифра
bool isLetterOrDigit(char c) {
    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) return true;
    if (c >= '0' && c <= '9') return true;
    return false;
}

// Проверяет, что строка состоит только из латинских букв и цифр
bool isAlphaNumStr(const char* s) {
    int i = 0;

```

```

    while (s[i] != '\0') {
        if (!isLetterOrDigit(s[i])) return false;
        i++;
    }
    return i > 0;
}

// Проверяет, что строка состоит из латинских букв, цифр и дефиса (должен быть хотя
бы один дефис)
bool isAlphaNumDashStr(const char* s) {
    int i = 0;
    bool hasDash = false;
    while (s[i] != '\0') {
        if (s[i] == '-') hasDash = true;
        else if (!isLetterOrDigit(s[i])) return false;
        i++;
    }
    return i > 0 && hasDash;
}

// Проверяет формат бортового номера: одна латинская буква, дефис, 4 цифры
bool isBoardNumberValid(const char* s) {
    if (!((s[0] >= 'A' && s[0] <= 'Z') || (s[0] >= 'a' && s[0] <= 'z')))) return
false;
    if (s[1] != '-') return false;
    int i = 2, cnt = 0;
    while (s[i] >= '0' && s[i] <= '9') { ++i; ++cnt; }
    if (cnt != 4) return false;
    if (s[i] != '\0') return false;
    return true;
}

// Проверяет, что аэродром – только AP1, AP2 или AP3
bool isAerodromeValid(const char* s) {
    return (
        (s[0] == 'A' && s[1] == 'P' && s[2] == '1' && s[3] == '\0') ||
        (s[0] == 'A' && s[1] == 'P' && s[2] == '2' && s[3] == '\0') ||
        (s[0] == 'A' && s[1] == 'P' && s[2] == '3' && s[3] == '\0')
    );
}

// Главная функция проверки полей структуры Flight (кроме специфических проверок
аэродрома)
bool isValidFlightData(const Flight& f) {

```

```

    if (!isBoardNumberValid(f.boardNumber)) return false;
    if (!isAlphaNumDashStr(f.mark)) return false;
    if (f.landingHour < 0 || f.landingHour > 23) return false;
    if (f.landingMinute < 0 || f.landingMinute > 59) return false;
    return true;
}

// Сравнивает две строки на полное совпадение
bool areStringsEqual(const char* a, const char* b) {
    int i = 0;
    while (a[i] && b[i]) {
        if (a[i] != b[i]) return false;
        ++i;
    }
    return a[i] == b[i];
}

// Чтение данных о рейсах из текстового файла
int readFlights(const char* filename, Flight flights[], int maxRecords) {
    ifstream fin(filename);
    int n = 0;
    if (!fin) {
        cout << "Ошибка открытия файла!" << endl;
        return 0;
    }

    char line[LINE_SIZE];
    int lineNumber = 0;
    while (n < maxRecords && fin.getline(line, LINE_SIZE)) {
        lineNumber++;
        if (line[0] == '\0') continue;
        char mark_buf[MARK_SIZE] = {0};
        char boardNum_buf[BOARD_NUM_SIZE] = {0};
        char timeStr_buf[8] = {0};
        char aero_buf[AERO_SIZE] = {0};
        int word = 0, li = 0, wi = 0;
        char* fields[4] = {mark_buf, boardNum_buf, timeStr_buf, aero_buf};
        int limits[4] = {MARK_SIZE, BOARD_NUM_SIZE, 8, AERO_SIZE};
        while (line[li] && word < 4) {
            while (line[li] == ' ' || line[li] == '\t') li++;
            if (line[li] == '\0' && word < 4) break;
            wi = 0;
            while (line[li] && line[li] != ' ' && line[li] != '\t' && wi <
limits[word] - 1) {

```

```

        fields[word][wi++] = line[li++];
    }
    fields[word][wi] = '\0';
    word++;
}
// ПРОВЕРКА 2: Поле аэродрома не пустое
if (strLength(aero_buf) == 0) {
    cout << "Ошибка: отсутствует значение для аэродрома (пустое поле) в
строке " << lineNumber << endl;
    continue;
}
// ПРОВЕРКА 3: Корректность значения аэродрома (AP1, AP2, AP3)
if (!isAerodromeValid(aero_buf)) {
    cout << "Ошибка: некорректный код аэродрома '" << aero_buf
        << "' в строке " << lineNumber << ". Ожидался AP1, AP2 или AP3."
    << endl;
    continue;
}
Flight f;
copyStr(f.mark, mark_buf, MARK_SIZE);
copyStr(f.boardNumber, boardNum_buf, BOARD_NUM_SIZE);
copyStr(f.aerodrome, aero_buf, AERO_SIZE);
if (!parseTime(timeStr_buf, f.landingHour, f.landingMinute)) {
    cout << "Ошибка парсинга времени для записи '" << timeStr_buf << "' в
строке " << lineNumber << endl;
    continue;
}
if (isValidFlightData(f)) {
    flights[n++] = f;
} else {
    cout << "Ошибка: некорректные данные (марка или бортовой номер) в
строке " << lineNumber << endl;
}
}
return n;
}

// Выводит таблицу рейсов (без сортировки)
void printFlights(const Flight flights[], int n) {
    const int w1 = 10, w2 = 12, w3 = 8, w4 = 8;
    cout << left
        << setw(w1) << "Марка" << " "
        << setw(w2) << "Бортовой №" << " "
        << setw(w3) << "Время" << " "

```

```

        << setw(w4) << "Аэродром"
        << endl;
    cout << setfill('-') << setw(w1 + w2 + w3 + w4 + 6) << "" << setfill(' ') <<
endl;
    for (int i = 0; i < n; ++i) {
        cout << left << setw(w1) << flights[i].mark << " "
            << left << setw(w2) << flights[i].boardNumber << " "
            << right << setw(2) << setfill('0') << flights[i].landingHour << ":"
            << setw(2) << setfill('0') << flights[i].landingMinute << setfill('
') << " "
            << left << setw(w4) << flights[i].aerodrome << endl;
    }
}

// Сортирует массив FlightIndex по времени посадки (от ранних к поздним)
void indexSort(FlightIndex indexes[], int n) {
    for (int i = 0; i < n-1; ++i) {
        for (int j = 0; j < n-1-i; ++j) {
            if (indexes[j].time > indexes[j+1].time) {
                FlightIndex tmp = indexes[j];
                indexes[j] = indexes[j+1];
                indexes[j+1] = tmp;
            }
        }
    }
}

// Выводит рейсы по индексной структуре (отсортированные по времени)
void printFlightsByIndex(const Flight flights[], const FlightIndex indexes[], int
n) {
    const int w1 = 10, w2 = 12, w3 = 8, w4 = 8;
    cout << left
        << setw(w1) << "Марка" << " "
        << setw(w2) << "Бортовой №" << " "
        << setw(w3) << "Время" << " "
        << setw(w4) << "Аэродром"
        << endl;
    cout << setfill('-') << setw(w1 + w2 + w3 + w4 + 6) << "" << setfill(' ') <<
endl;
    for (int i = 0; i < n; ++i) {
        const Flight& f = flights[indexes[i].idx];
        cout << left << setw(w1) << f.mark << " "
            << left << setw(w2) << f.boardNumber << " "
            << right << setw(2) << setfill('0') << f.landingHour << ":"

```

```

        << setw(2) << setfill('0') << f.landingMinute << setfill(' ') << "
"
        << left << setw(w4) << f.aerodrome << endl;
    }
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        cout << "Использование: " << argv[0] << " <файл_данных>" << endl;
        return 1;
    }
    Flight flights[MAX_RECORDS];
    FlightIndex indexes[MAX_RECORDS];
    int n = readFlights(argv[1], flights, MAX_RECORDS);

    if (n == 0) {
        cout << "Нет данных для обработки (возможно, отсутствие файла)." << endl;
        return 1;
    }

    cout << "\nИсходные данные (только корректные записи):" << endl;
    printFlights(flights, n);

    // Заполнение индексной структуры
    for (int i = 0; i < n; ++i) {
        indexes[i].idx = i;
        indexes[i].time = landingTimeToMinutes(flights[i]);
    }
    // Сортировка индексной структуры по времени
    indexSort(indexes, n);

    cout << "\nДанные, отсортированные по времени посадки (через индексную
структуру):" << endl;
    printFlightsByIndex(flights, indexes, n);

    // По каждому аэродрому вывод через индексную структуру:
    const char* aerodromes[3] = {"AP1", "AP2", "AP3"};
    for (int ad = 0; ad < 3; ++ad) {
        cout << "\nСамолеты на аэродроме " << aerodromes[ad] << " (отсортировано
по времени посадки):" << endl;
        const int w1 = 10, w2 = 12, w3 = 8, w4 = 8;
        cout << left
            << setw(w1) << "Марка" << " "
            << setw(w2) << "Бортовой №" << " "

```



```

        << setw(w3) << "Время" << " "
        << setw(w4) << "Аэродром"
        << endl;
    cout << setfill('-') << setw(w1 + w2 + w3 + w4 + 6) << "" << setfill(' ')
<< endl;
    bool found = false;
    for (int i = 0; i < n; ++i) {
        const Flight& f = flights[indexes[i].idx];
        if (areStringsEqual(f.aerodrome, aerodromes[ad])) {
            cout << left << setw(w1) << f.mark << " "
                << left << setw(w2) << f.boardNumber << " "
                << right << setw(w3-6) << setfill('0') << f.landingHour <<
":."
                << setw(2) << setfill('0') << f.landingMinute << setfill('
') << " "
                << left << setw(w4) << f.aerodrome << endl;
            found = true;
        }
    }
    if (!found) {
        cout << "Нет данных для этого аэродрома." << endl;
    }
}

return 0;
}

```

Тесты

Корректные тесты:

```
> ./main data.txt
```

Исходные данные:

Марка	Бортовой №	Время	Аэродром
-------	------------	-------	----------

TU-154M	B-3726	11:15	AP2
AN-24	B-1234	10:00	AP1
YAK-42	B-5678	12:30	AP3
TU-134	B-9012	09:45	AP1

Самолеты на аэродроме AP1 (отсортировано по времени посадки):

Марка	Бортовой №	Время	Аэродром
-------	------------	-------	----------

TU-134	B-9012	09:45	AP1
AN-24	B-1234	10:00	AP1

Самолеты на аэродроме AP2 (отсортировано по времени посадки):

Марка	Бортовой №	Время	Аэродром
-------	------------	-------	----------

TU-154M	B-3726	11:15	AP2
---------	--------	-------	-----

Самолеты на аэродроме AP3 (отсортировано по времени посадки):

Марка	Бортовой №	Время	Аэродром
-------	------------	-------	----------

YAK-42	B-5678	12:30	AP3
--------	--------	-------	-----

Некорректные тесты:

1. Отсутствие аэропорта у рейса:

```
> ./main corrupted_data.txt
```

Ошибка: отсутствует значение для аэродрома (пустое поле) в строке 1

Исходные данные (только корректные записи):

Марка Бортовой № Время Аэродром

AN-24	B-1234	10:00	AP1
YAK-42	B-5678	12:30	AP3
TU-134	B-9012	09:45	AP1

Самолеты на аэродроме AP1 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

TU-134	B-9012	09:45	AP1
AN-24	B-1234	10:00	AP1

Самолеты на аэродроме AP2 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

Нет данных для этого аэродрома.

Самолеты на аэродроме AP3 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

YAK-42	B-5678	12:30	AP3
--------	--------	-------	-----

2. Неправильно указано время для рейса:

```
> ./main corrupted_data.txt
```

Ошибка парсинга времени для записи '1115' в строке 1

Исходные данные (только корректные записи):

Марка	Бортовой №	Время	Аэродром
-------	------------	-------	----------

AN-24	B-1234	10:00	AP1
YAK-42	B-5678	12:30	AP3
TU-134	B-9012	09:45	AP1

Самолеты на аэродроме AP1 (отсортировано по времени посадки):

Марка	Бортовой №	Время	Аэродром
-------	------------	-------	----------

TU-134	B-9012	09:45	AP1
AN-24	B-1234	10:00	AP1

Самолеты на аэродроме AP2 (отсортировано по времени посадки):

Марка	Бортовой №	Время	Аэродром
-------	------------	-------	----------

Нет данных для этого аэродрома.

Самолеты на аэродроме AP3 (отсортировано по времени посадки):

Марка	Бортовой №	Время	Аэродром
-------	------------	-------	----------

YAK-42	B-5678	12:30	AP3
--------	--------	-------	-----

3. Неправильно указан бортовой номер:

```
> ./main corrupted_data.txt
```

Ошибка: некорректные данные (марка или бортовой номер) в строке 1

Исходные данные (только корректные записи):

Марка Бортовой № Время Аэродром

AN-24	B-1234	10:00	AP1
YAK-42	B-5678	12:30	AP3
TU-134	B-9012	09:45	AP1

Самолеты на аэродроме AP1 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

TU-134	B-9012	09:45	AP1
AN-24	B-1234	10:00	AP1

Самолеты на аэродроме AP2 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

Нет данных для этого аэродрома.

Самолеты на аэродроме AP3 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

YAK-42	B-5678	12:30	AP3
--------	--------	-------	-----

4. Некорректная марка:

```
> ./main corrupted_data.txt
```

Ошибка: некорректные данные (марка или бортовой номер) в строке 1

Исходные данные (только корректные записи):

Марка Бортовой № Время Аэродром

AN-24	B-1234	10:00	AP1
YAK-42	B-5678	12:30	AP3
TU-134	B-9012	09:45	AP1

Самолеты на аэродроме AP1 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

TU-134	B-9012	09:45	AP1
AN-24	B-1234	10:00	AP1

Самолеты на аэродроме AP2 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

Нет данных для этого аэродрома.

Самолеты на аэродроме AP3 (отсортировано по времени посадки):

Марка Бортовой № Время Аэродром

YAK-42	B-5678	12:30	AP3
--------	--------	-------	-----

5. Указан несуществующий файл:

```
> ./main corrupted_data  
Ошибка открытия файла!  
Нет данных для обработки (возможно, отсутствие файла).
```

6. Не указан файл:

```
> ./main  
Использование: ./main <файл_данных>
```

Вывод

Разработку программы считаю завершенным в связи с достаточностью тестов.

_____	/ _____ /	_____ 20__ г.
_____	/ _____ /	_____ 20__ г.
<i>подпись обучающегося</i>	<i>расшифровка подписи</i>	<i>дата</i>

Рекомендации: В отчете по практике после подписи обучающегося ставится дата окончания практики. Подписи после текста отчета.