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 М30-125БВ-24

ФИО Егоров Александ Владиславович

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 _____ Дмитриева Е.А.

Москва

2025

1. **Место и сроки проведения практики:**

Наименование организации: _____МАИ, кафедра 304_____

Сроки проведения практики

дата начала практики: ___10.02.2025_____

дата окончания практики: ___06.06.2025_____

2. **Инструктаж по технике безопасности:**

_____ /__ Дмитриева Е.А.__/ _____ _____ 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

_____ /_Дмитриева Е.А._____/ _____ _____ 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*

5. Отзыв руководителя практики от организации/предприятия:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____ /_ Дмитриева Е.А._/ _____ _____ 20__г.

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

Отчет по практике

Оглавление

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 двигается 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 → 3, 2, 4, 1, 5

Пример работы алгоритма

Отсортируем массив по возрастанию:

Исходный массив: 4, 2, 5, 1, 3

Первый проход:

- 4 > 2 → меняем → 2, 4, 5, 1, 3
- 4 < 5 → ничего
- 5 > 1 → меняем → 2, 4, 1, 5, 3
- 5 > 3 → меняем → 2, 4, 1, 3, 5

Второй проход:

- 2 < 4 → ничего
- 4 > 1 → меняем → 2, 1, 4, 3, 5
- 4 > 3 → меняем → 2, 1, 3, 4, 5

Третий проход:

- 2 > 1 → меняем → 1, 2, 3, 4, 5

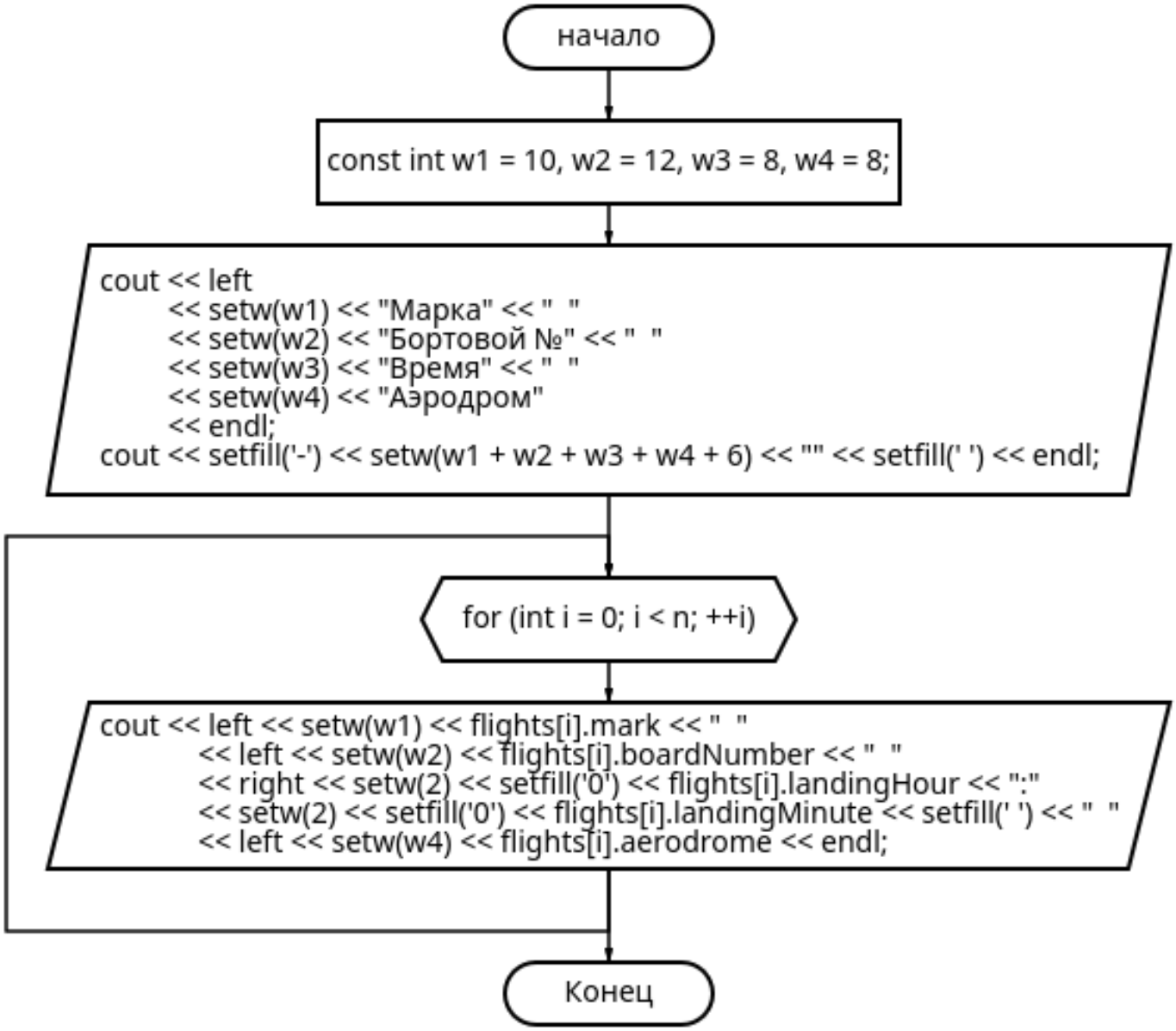
Четвёртый проход:

Ничего не меняем т. к. всё отсортировано.

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

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

Блок-схема


начало
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;
Конец

Описание функций

Функция copyStr:

1. Назначение: Копирует строку src в dest, не превышая maxLen (включая завершающий нуль);

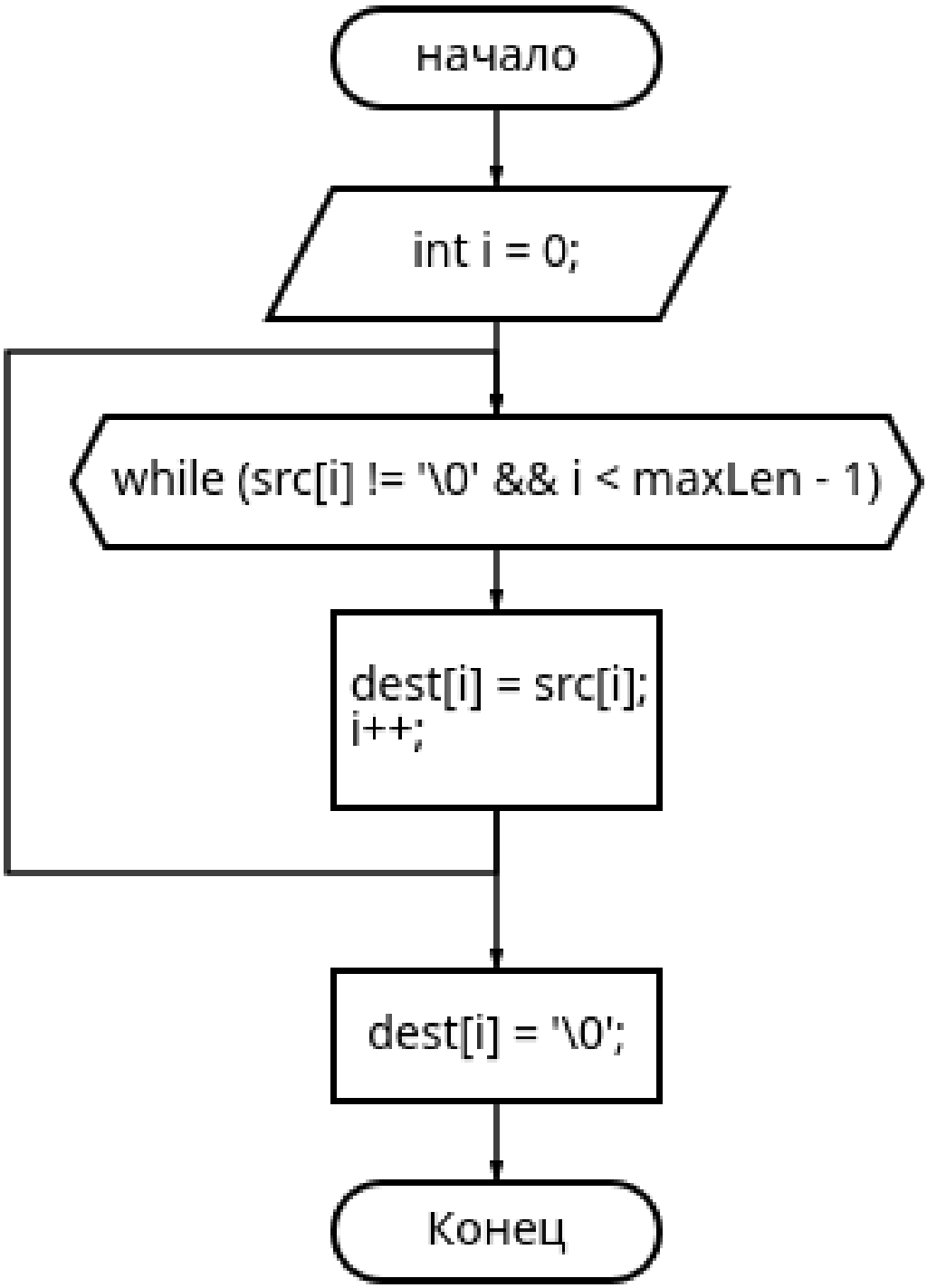
2. Прототип функции: void copyStr(char* dest, const char* src, int maxLen);

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 вимволов, куда будет скопирована строка	входной
src	const char*	Исходная строка для копирования	входной
maxLen	int	Максимальный размер буфера dest	входной

5. Блоксхема:


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

Функция parseTime:

1. Назначение: Разбирает строку времени ("ЧЧ:ММ" или "Ч:ММ") на часы и минуты;

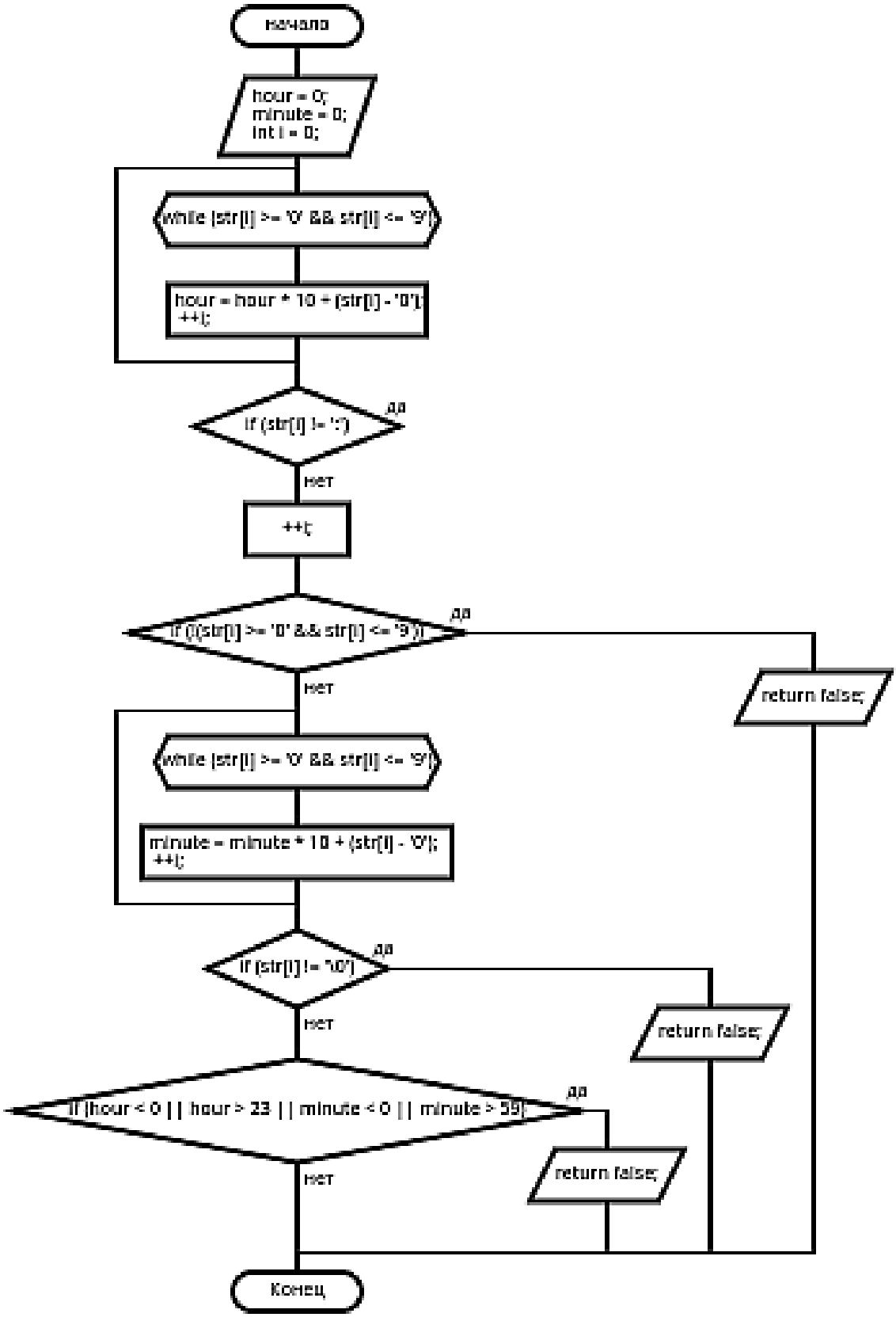
2. Прототип функции: bool parseTime(const char* str, int& hour, int& minute);

3. Обращение: parseTime(timeStr, f.landingHour, f.landingMinute);

4. Описание параметров:

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

5. Блоксхема:


начало
hour = 0;
minute = 0;
int i = 0;
while (str[i] >= '0' && str[i] <= '9')
hour = hour * 10 + (str[i] - '0');
++i;
if (str[i] != ':')
да
нет
++i;
if (!(str[i] >= '0' && str[i] <= '9'))
да
нет
return false;
while (str[i] >= '0' && str[i] <= '9')
minute = minute * 10 + (str[i] - '0');
++i;
if (str[i] != '\0')
да
нет
return false;
if (hour < 0 || hour > 23 || minute < 0 || minute > 59)
да
нет
return false;
Конец

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:

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

Функция isValidFlight:

1. Назначение: Проверяет корректность данных рейса;

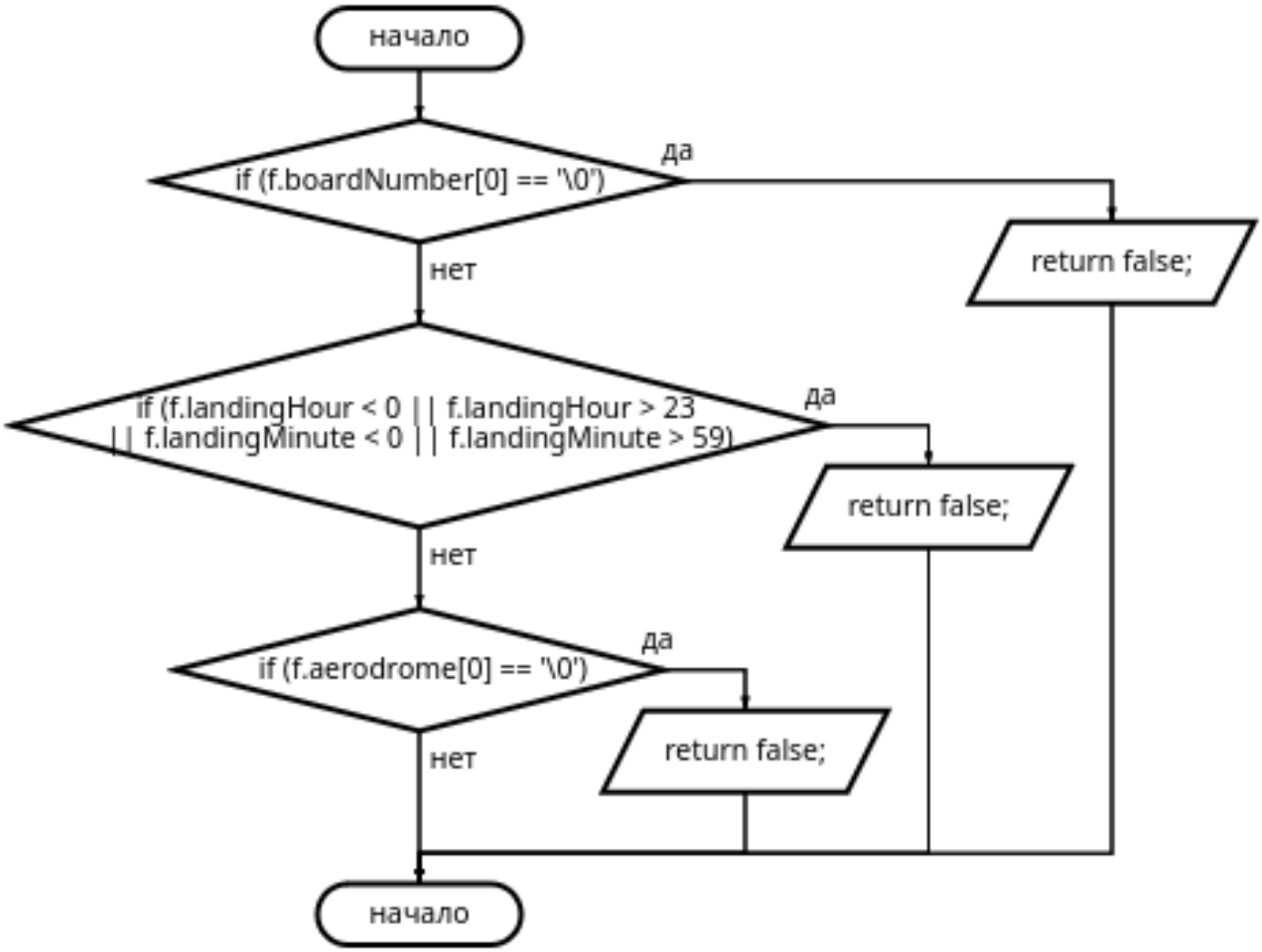
2. Прототип функции: bool isValidFlight(const Flight& f);

3. Обращение: isValidFlight(f);

4. Описание параметров:

Индентификатор	Тип	Назначение	Входной/ Выходной
isValidFlight	bool	Проверяет корректность данных рейса	выходной
f	const Flight&	строка времени для разбораструктура Flight для проверки	входной

5. Блоксхема:


начало
if (f.boardNumber[0] == '\0')
да
return false;
нет
if (f.landingHour < 0 || f.landingHour > 23
|| f.landingMinute < 0 || f.landingMinute > 59)
да
return false;
нет
if (f.aerodrome[0] == '\0')
да
return false;
нет
начало

Функция areStringsEqual:

1. Назначение: Сравнивает две строки на равенство;

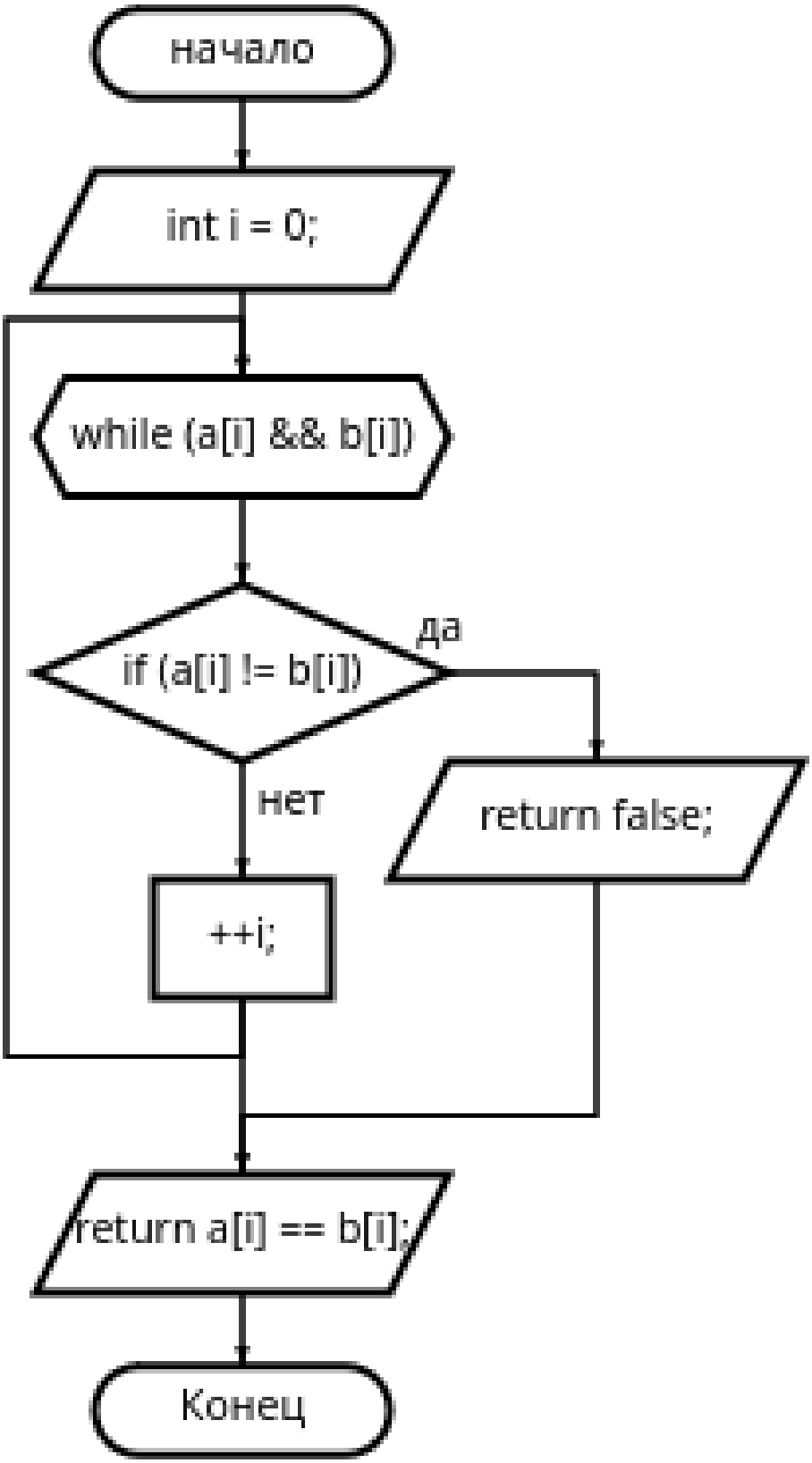
2. Прототип функции: bool areStringsEqual(const char* a, const char* b);

3. Обращение: areStringsEqual(flights[k].aerodrome, targetAerodrome);

4. Описание параметров:

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

5. Блоксхема:


начало
int i = 0;
while (a[i] && b[i])
if (a[i] != b[i])
да
нет
return false;
++i;
return a[i] == b[i];
Конец

Функция readFlights:

1. Назначение: Считывает данные рейсов из файла filename в массив flights (максимум maxRecords штук);

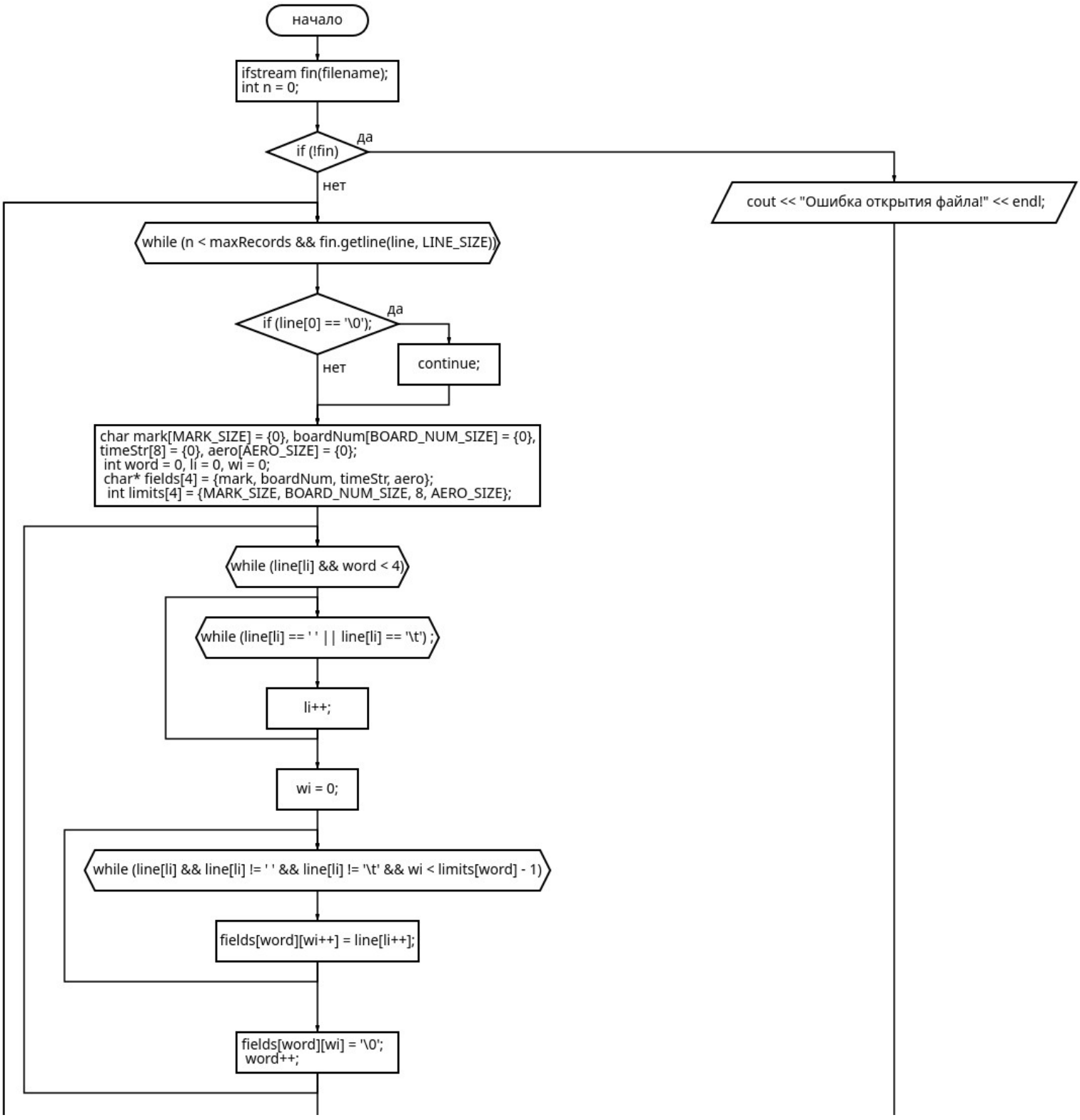
2. Прототип функции: int readFlights(const char* filename, Flight flights[], int maxRecords);

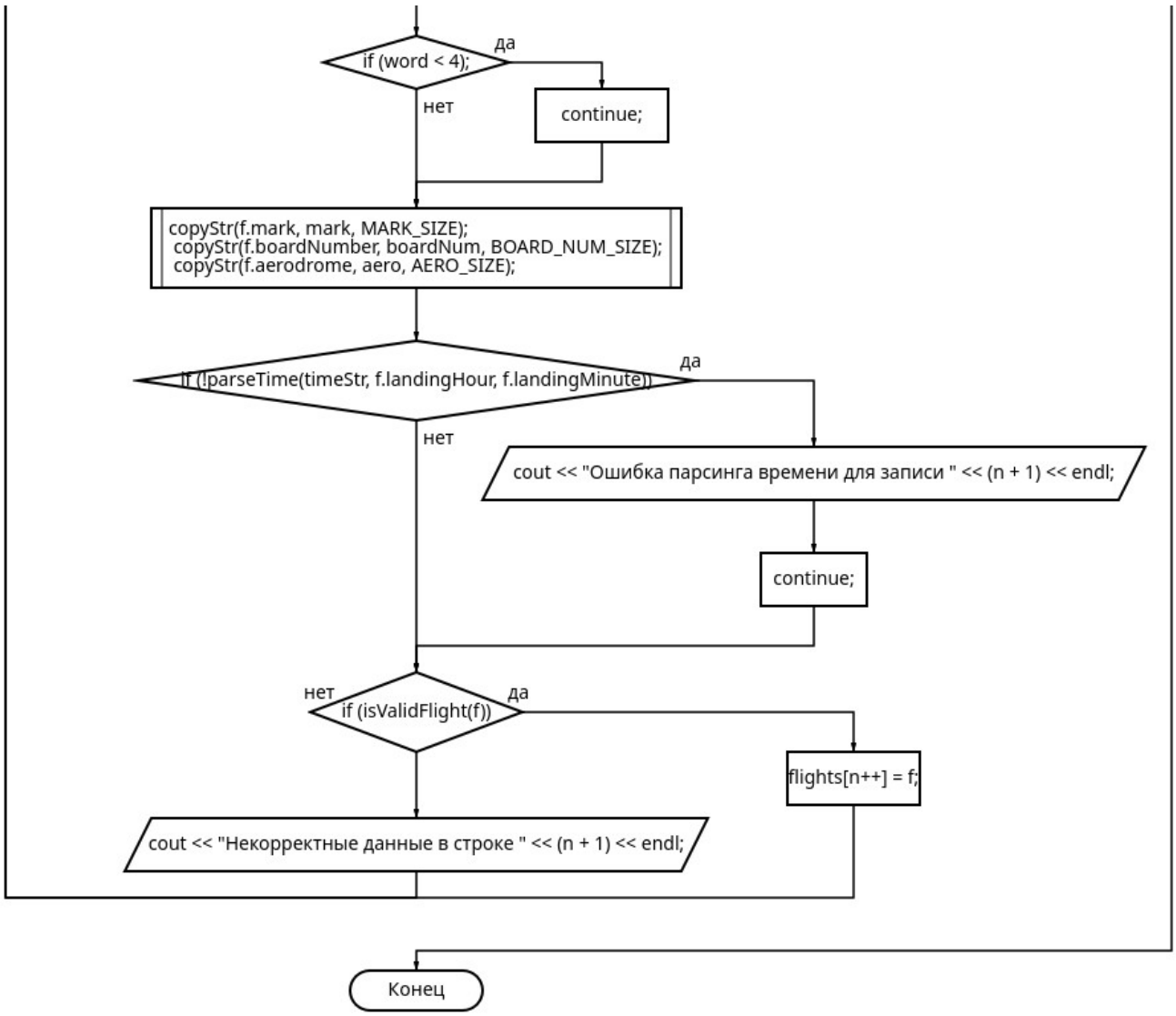
3. Обращение: readFlights(argv[1], flights, MAX_RECORDS);

4. Описание параметров:

Индентификатор	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:


начало
ifstream fin(filename);
int n = 0;
if (!fin)
да
нет
cout << "Ошибка открытия файла!" << endl;
while (n < maxRecords && fin.getline(line, LINE_SIZE))
if (line[0] == '\0');
да
continue;
нет
char mark[MARK_SIZE] = {0}, boardNum[BOARD_NUM_SIZE] = {0},
timeStr[8] = {0}, aero[AERO_SIZE] = {0};
int word = 0, li = 0, wi = 0;
char* fields[4] = {mark, boardNum, timeStr, aero};
int limits[4] = {MARK_SIZE, BOARD_NUM_SIZE, 8, AERO_SIZE};
while (line[li] && word < 4)
while (line[li] == ' ' || line[li] == '\t') ;
li++;
wi = 0;
while (line[li] && line[li] != ' ' && line[li] != '\t' && wi < limits[word] - 1)
fields[word][wi++] = line[li++];
fields[word][wi] = '\0';
word++;


if (word < 4);
да
нет
continue;
copyStr(f.mark, mark, MARK_SIZE);
copyStr(f.boardNumber, boardNum, BOARD_NUM_SIZE);
copyStr(f.aerodrome, aero, AERO_SIZE);
if (!parseTime(timeStr, f.landingHour, f.landingMinute))
да
нет
cout << "Ошибка парсинга времени для записи " << (n + 1) << endl;
continue;
нет
if (isValidFlight(f))
да
flights[n++] = f;
cout << "Некорректные данные в строке " << (n + 1) << endl;
Конец

Функция indexBubbleSort:

1. Назначение: Сортирует индексы рейсов в массиве idx по времени посадки;

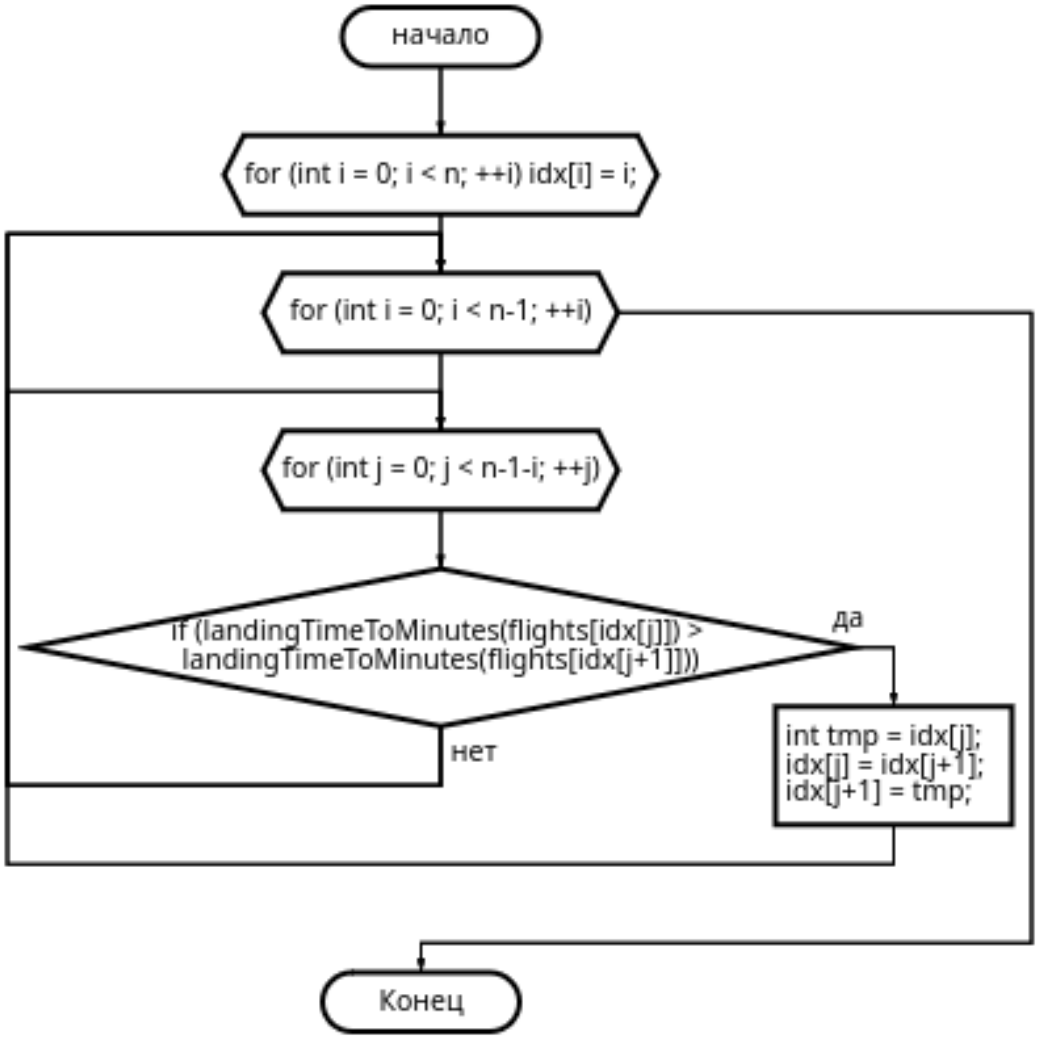
2. Прототип функции: void indexBubbleSort(const Flight flights[], int idx[], int n);

3. Обращение: indexBubbleSort(flights, idx, n);

4. Описание параметров:

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

5. Блоксхема:


начало
for (int i = 0; i < n; ++i) idx[i] = i;
for (int i = 0; i < n-1; ++i)
for (int j = 0; j < n-1-i; ++j)
if (landingTimeToMinutes(flights[idx[j]]) >
landingTimeToMinutes(flights[idx[j+1]]))
да
нет
int tmp = idx[j];
idx[j] = idx[j+1];
idx[j+1] = tmp;
Конец

Функция printFlightsByAerodrome:

1. Назначение: Выводит рейсы, приземлившиеся на targetAerodrome, в порядке времени посадки;

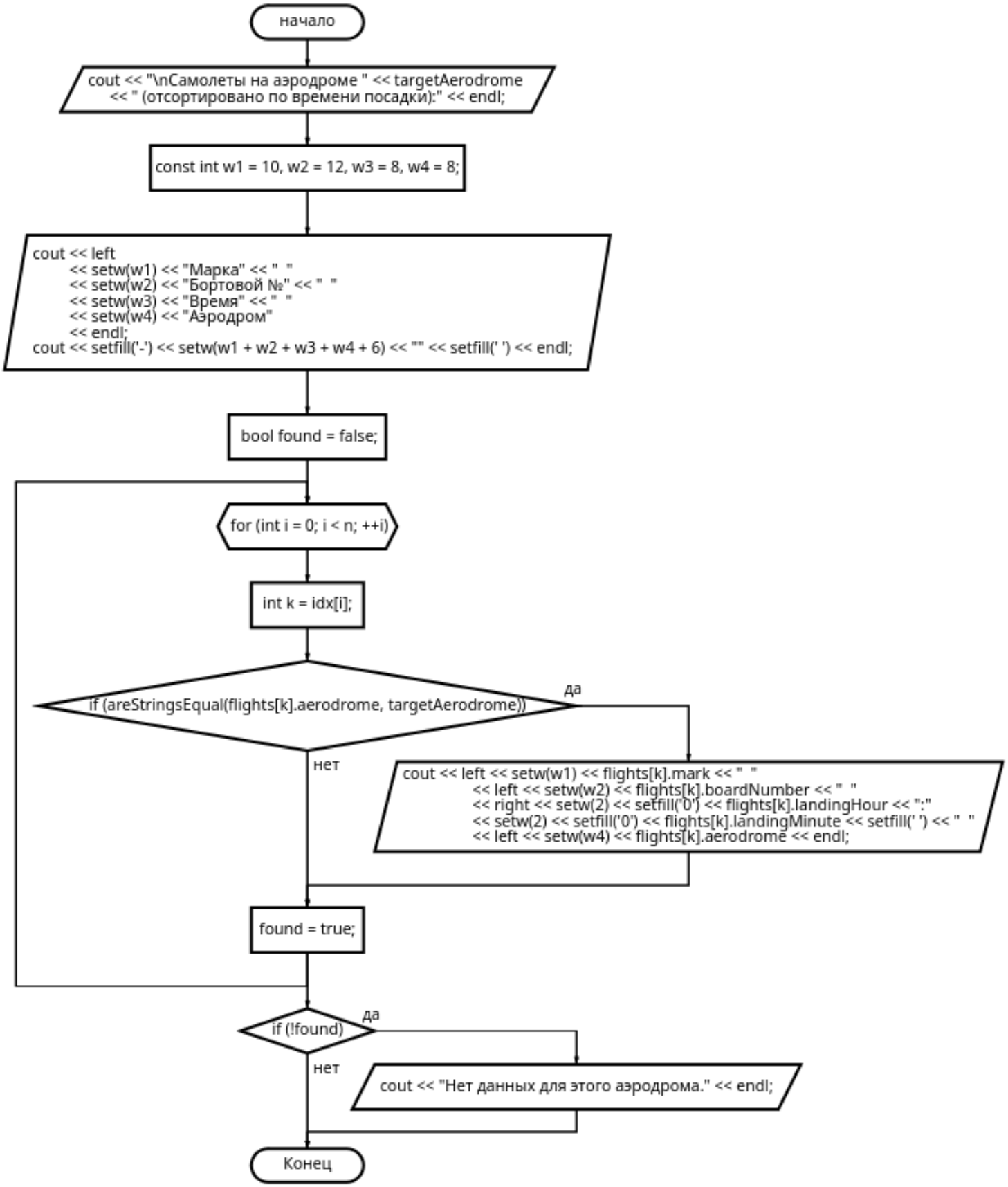
2. Прототип функции: void printFlightsByAerodrome(const Flight flights[], const int idx[], int n, const char* targetAerodrome);

3. Обращение: printFlightsByAerodrome(flights, idx, n, "АП3");

4. Описание параметров:

Индентификатор	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:


начало
cout << "\nСамолеты на аэродроме " << targetAerodrome
<< " (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)
int k = idx[i];
if (areStringsEqual(flights[k].aerodrome, targetAerodrome))
да
нет
cout << left << setw(w1) << flights[k].mark << " "
<< left << setw(w2) << flights[k].boardNumber << " "
<< right << setw(2) << setfill('0') << flights[k].landingHour << ":"
<< setw(2) << setfill('0') << flights[k].landingMinute << setfill(' ') << " "
<< left << setw(w4) << flights[k].aerodrome << endl;
found = true;
if (!found)
да
нет
cout << "Нет данных для этого аэродрома." << endl;
Конец

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:

Индентификатор	Тип	Назначение	Входной/ Выходной
parseTime	bool	Разбирает строку времени на часы и минуты	выходной
str	char*	строка времени для разбора	входной
hour	int&	переменная для записи разобранных часов	входной
minute	int&	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-154M"
    char boardNumber[BOARD_NUM_SIZE];// Бортовой номер, например "B-3726"
    int landingHour;                  // Часы посадки (0-23)
    int landingMinute;                // Минуты посадки (0-59)
    char aerodrome[AERO_SIZE];        // Аэродром посадки, например "AP2"
};

// Копирует строку src в dest, не превышая maxLen (включая завершающий нуль).
void copyStr(char* dest, const char* src, int maxLen) {
    int i = 0;
    // Копируем посимвольно, не превышая maxLen-1 (оставляем место под '\0')
    while (src[i] != '\0' && i < maxLen - 1) {
        dest[i] = src[i];
        i++;
    }
    dest[i] = '\0'; // Добавляем завершающий нуль-символ
}

// 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;
    // Читаем часы (одна или две цифры)
    if (!(str[i] >= '0' && str[i] <= '9')) return false;
    while (str[i] >= '0' && str[i] <= '9') {
        hour = hour * 10 + (str[i] - '0');
        ++i;
    }
    // Проверяем наличие разделителя ':'
    if (str[i] != ':') return false;
    ++i;
    // Читаем минуты (две цифры)
    if (!(str[i] >= '0' && str[i] <= '9')) return false;
    while (str[i] >= '0' && str[i] <= '9') {
        minute = minute * 10 + (str[i] - '0');
        ++i;
    }
    // Проверяем, что строка закончилась
    if (str[i] != '\0') return false;
    // Проверяем диапазоны значений
    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; //
латинские буквы
    if (c >= '0' && c <= '9') return true; // цифры
    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; // строка не пуста
}

// 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++;
    }
    // Требуем хотя бы один дефис
    return i > 0 && hasDash;
}

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

// Проверяет, что аэродром корректный: 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
bool isValidFlight(const Flight& f) {
    // Проверка бортового номера: латинская буква, дефис, 4 цифры
    if (!isBoardNumberValid(f.boardNumber)) return false;

    // Проверка марки самолета: латинские буквы, цифры, дефис (минимум один
дефис)
    if (!isAlphaNumDashStr(f.mark)) return false;

    // Проверка аэродрома: только AP1, AP2 или AP3
    if (!isAerodromeValid(f.aerodrome)) return false;

    // Проверка времени
    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); // Открываем файл для чтения
    int n = 0; // Количество успешно считанных записей
    if (!fin) {
        cout << "Ошибка открытия файла!" << endl;
        return 0;
    }
    char line[LINE_SIZE];
    while (n < maxRecords && fin.getline(line, LINE_SIZE)) {
        if (line[0] == '\0') continue; // Пропускаем пустые строки
```

```
        char mark[MARK_SIZE] = {0}, boardNum[BOARD_NUM_SIZE] = {0},
timeStr[8] = {0}, aero[AERO_SIZE] = {0};
        int word = 0, li = 0, wi = 0;
        // Разбиваем строку на четыре слова по пробелу/табуляции
        char* fields[4] = {mark, boardNum, timeStr, aero};
        int limits[4] = {MARK_SIZE, BOARD_NUM_SIZE, 8, AERO_SIZE};
        while (line[li] && word < 4) {
            // Пропускаем пробелы и табуляции
            while (line[li] == ' ' || line[li] == '\t') li++;
            wi = 0;
            // Копируем текущее слово
            while (line[li] && line[li] != ' ' && line[li] != '\t' && wi <
limits[word] - 1) {
                fields[word][wi++] = line[li++];
            }
            fields[word][wi] = '\0';
            word++;
        }
        // Если не хватает слов — строка пропускается
        if (word < 4) continue;

        Flight f;
        // Копируем разобранные данные в структуру
        copyStr(f.mark, mark, MARK_SIZE);
        copyStr(f.boardNumber, boardNum, BOARD_NUM_SIZE);
        copyStr(f.aerodrome, aero, AERO_SIZE);

        // Разбираем время
        if (!parseTime(timeStr, f.landingHour, f.landingMinute)) {
            cout << "Ошибка парсинга времени для записи " << (n + 1) << endl;
            continue;
        }
        // Проверяем корректность данных, сохраняем запись, если всё корректно
        if (isValidFlight(f)) {
            flights[n++] = f;
        } else {
            cout << "Некорректные данные в строке " << (n + 1) << endl;
        }
    }
    return n;
}
```

```
// Выводит таблицу рейсов
void printFlights(const Flight flights[], int n) {
    // Ширина столбцов для красивого вывода
    const int w1 = 10, w2 = 12, w3 = 8, w4 = 8;
    // Заголовки столбцов
    cout << left
         << setw(w1) << "Марка" << "  "
         << setw(w2) << "Бортовой №" << "  "
         << setw(w3) << "Время" << "  "
         << setw(w4) << "Аэродром"
         << endl;
    // Разделительная линия
    cout << setfill('-') << setw(w1 + w2 + w3 + w4 + 6) << "" << setfill(' ')
<< endl;
    // Выводим данные по каждому рейсу
    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 индексы рейсов по времени посадки (от ранних к поздним)
void indexBubbleSort(const Flight flights[], int idx[], int n) {
    for (int i = 0; i < n; ++i) idx[i] = i; // Инициализация массива индексов
    // Пузырьковая сортировка по времени посадки
    for (int i = 0; i < n-1; ++i) {
        for (int j = 0; j < n-1-i; ++j) {
            if (landingTimeToMinutes(flights[idx[j]]) >
landingTimeToMinutes(flights[idx[j+1]])) {
                int tmp = idx[j];
                idx[j] = idx[j+1];
                idx[j+1] = tmp;
            }
        }
    }
}
```

```
// Выводит рейсы, приземлившиеся на указанный аэродром, отсортированные по
времени посадки
void printFlightsByAerodrome(const Flight flights[], const int idx[], int n,
const char* targetAerodrome) {
    cout << "\nСамолеты на аэродроме " << targetAerodrome << " (отсортировано
по времени посадки):" << endl;
    // Ширина столбцов для вывода
    const int w1 = 10, w2 = 12, w3 = 8, w4 = 8;
    // Заголовки
    cout << left
         << setw(w1) << "Марка" << "  "
         << setw(w2) << "Бортовой №" << "  "
         << setw(w3) << "Время" << "  "
         << setw(w4) << "Аэродром"
         << endl;
    // Разделительная линия
    cout << setfill('-') << setw(w1 + w2 + w3 + w4 + 6) << "" << setfill(' ')
<< endl;
    bool found = false;
    // Проходим по отсортированным индексам и выводим только нужный аэродром
    for (int i = 0; i < n; ++i) {
        int k = idx[i];
        if (areStringsEqual(flights[k].aerodrome, targetAerodrome)) {
            cout << left << setw(w1) << flights[k].mark << "  "
                 << left << setw(w2) << flights[k].boardNumber << "  "
                 << right << setw(2) << setfill('0') <<
flights[k].landingHour << ":"
                 << setw(2) << setfill('0') << flights[k].landingMinute <<
setfill(' ') << "  "
                 << left << setw(w4) << flights[k].aerodrome << endl;
            found = true;
        }
    }
    if (!found) {
        cout << "Нет данных для этого аэродрома." << endl;
    }
}

// Главная функция программы
int main(int argc, char* argv[]) {
    // Проверяем наличие аргумента командной строки (имя файла с данными)
    if (argc < 2) {
```

```
        cout << "Использование: " << argv[0] << " <файл_данных>" << endl;
        return 1;
    }
    // Основной массив для хранения рейсов
    Flight flights[MAX_RECORDS];
    // Массив индексов для сортировки
    int idx[MAX_RECORDS];
    // Считываем рейсы из файла
    int n = readFlights(argv[1], flights, MAX_RECORDS);

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

    // Выводим исходные данные
    cout << "Исходные данные:" << endl;
    printFlights(flights, n);
    // Сортируем индексы по времени посадки
    indexBubbleSort(flights, idx, n);

    // Выводим данные по каждому аэродрому
    printFlightsByAerodrome(flights, idx, n, "AP1");
    printFlightsByAerodrome(flights, idx, n, "AP2");
    printFlightsByAerodrome(flights, idx, n, "AP3");

    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 завершонным в связи с достаточностью тестов.