

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Пояснительная записка к курсовой работе

по дисциплине: «Программирование»

на тему: «Разработка базы данных с использованием связных списков, вариант 3»

Выполнил:

студент группы М3О-207Б-23 Ильин А. А.

Проверил:

Ивашенцев И. В.

Москва 2024 г.

## Задание

Разработать базу данных на языке С++.

Тематики базы данных выбираются согласно варианту.

Для хранения данных использовать текстовый файл.

Разработать структуры данных для хранения информации и функции:

* для ввода данных,
* выборки значения по заданному условию,
* вывода данных,
* удаления,
* записи новых данных,
* сортировки по определенному пользователем значению.

Обеспечить выдачу запросов по любым полям данных.

Всю обработку данных вести с помощью динамических структур данных.

Реализовать систему меню для взаимодействия с пользователем по всем пунктам задания.

**Схемы организации данных:**

1. Линейные двусвязные списки

Тематика базы данных: учет в деканате студентов: фамилия; имя; отчество; номер группы; дата рождения (год, месяц, число); домашний адрес, год поступления в вуз, результаты обучения по сессиям.

Оглавление

[Задание 2](#_Toc186097194)

[Блок схемы 4](#_Toc186097195)

[Описание интерфейса программы 17](#_Toc186097196)

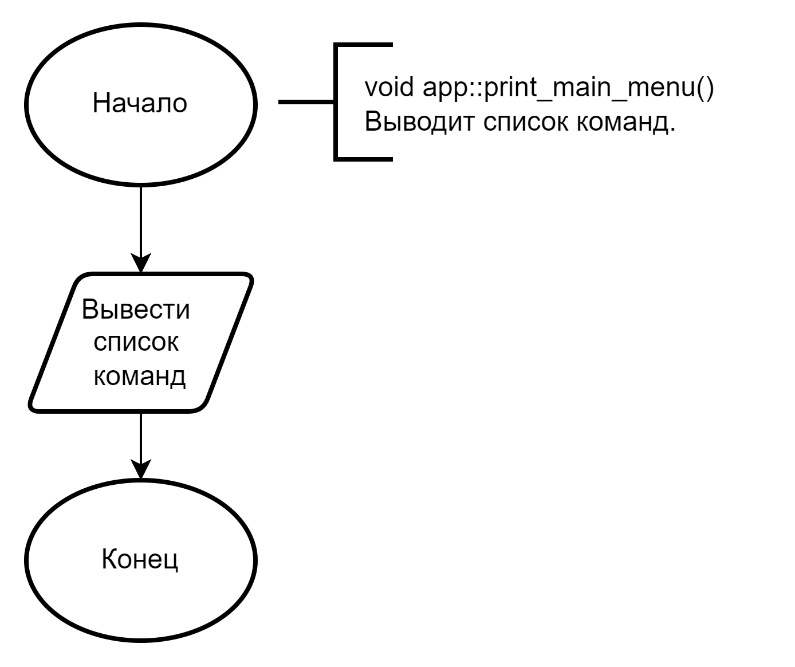
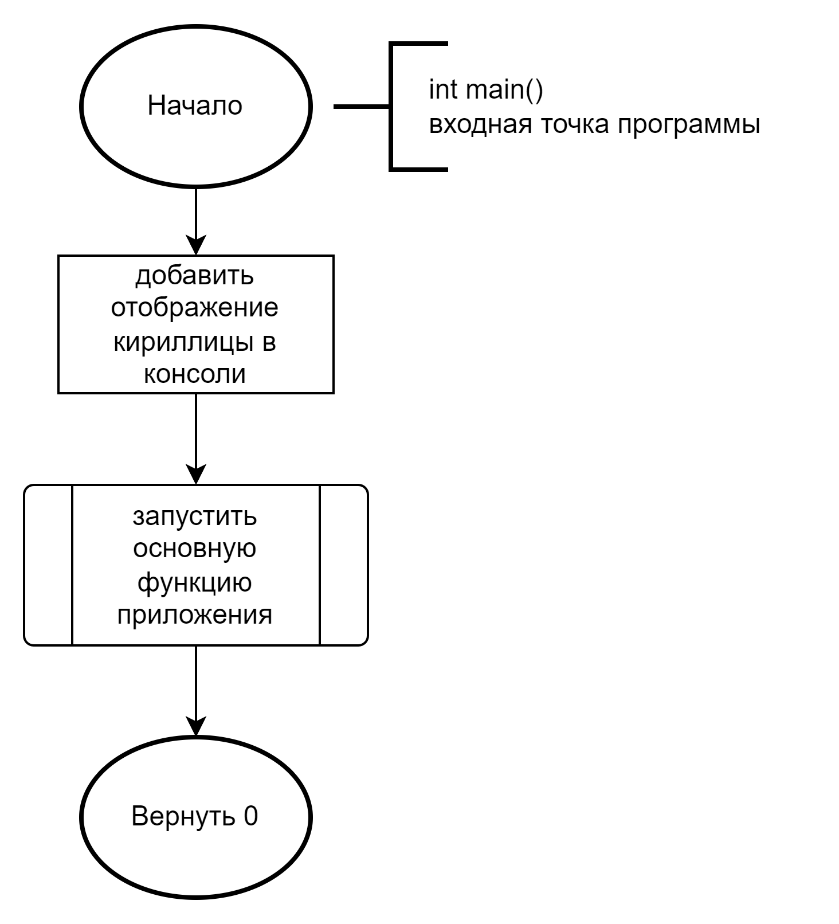
[Код программы 18](#_Toc186097197)

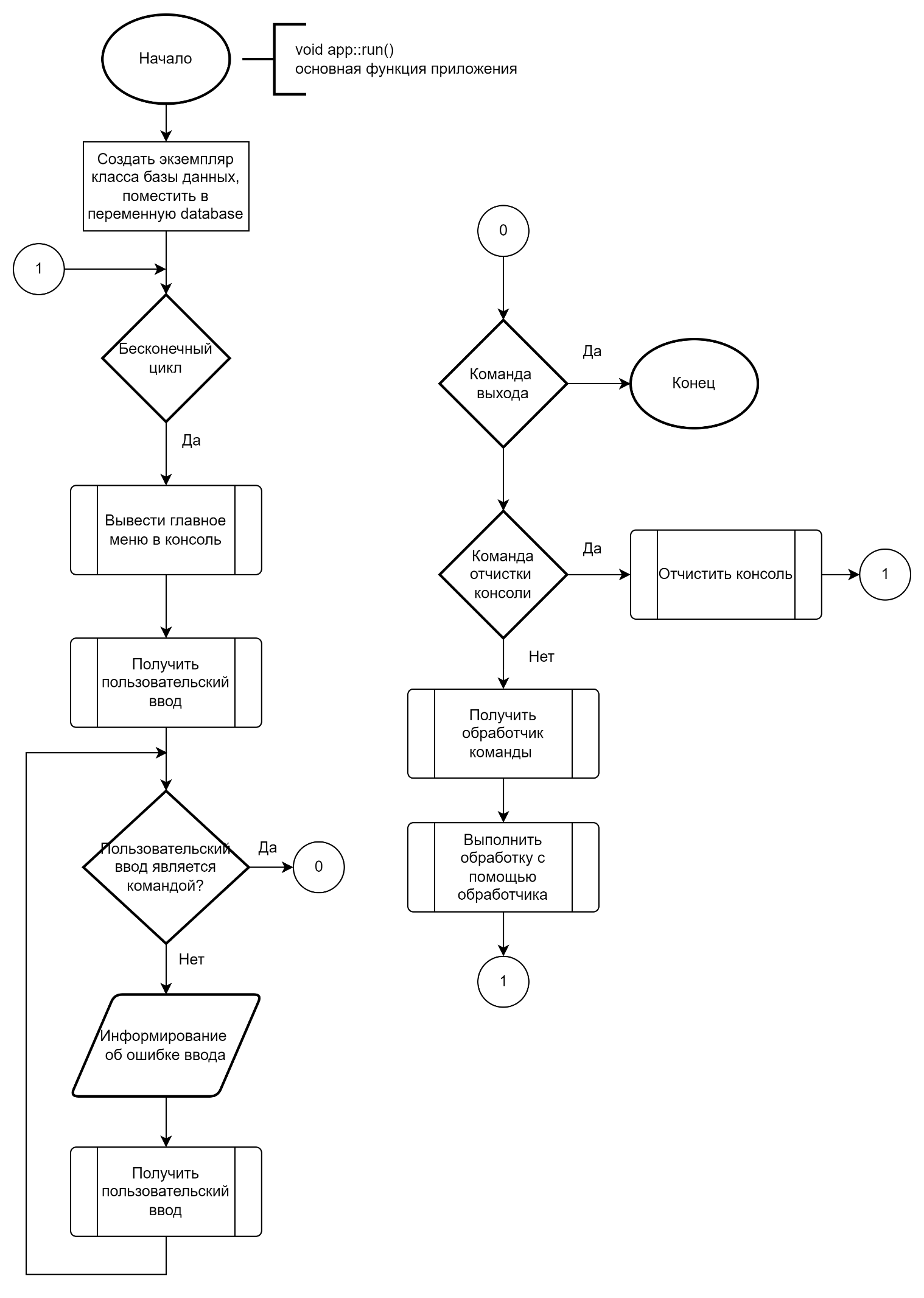
[Результаты работы 44](#_Toc186097198)

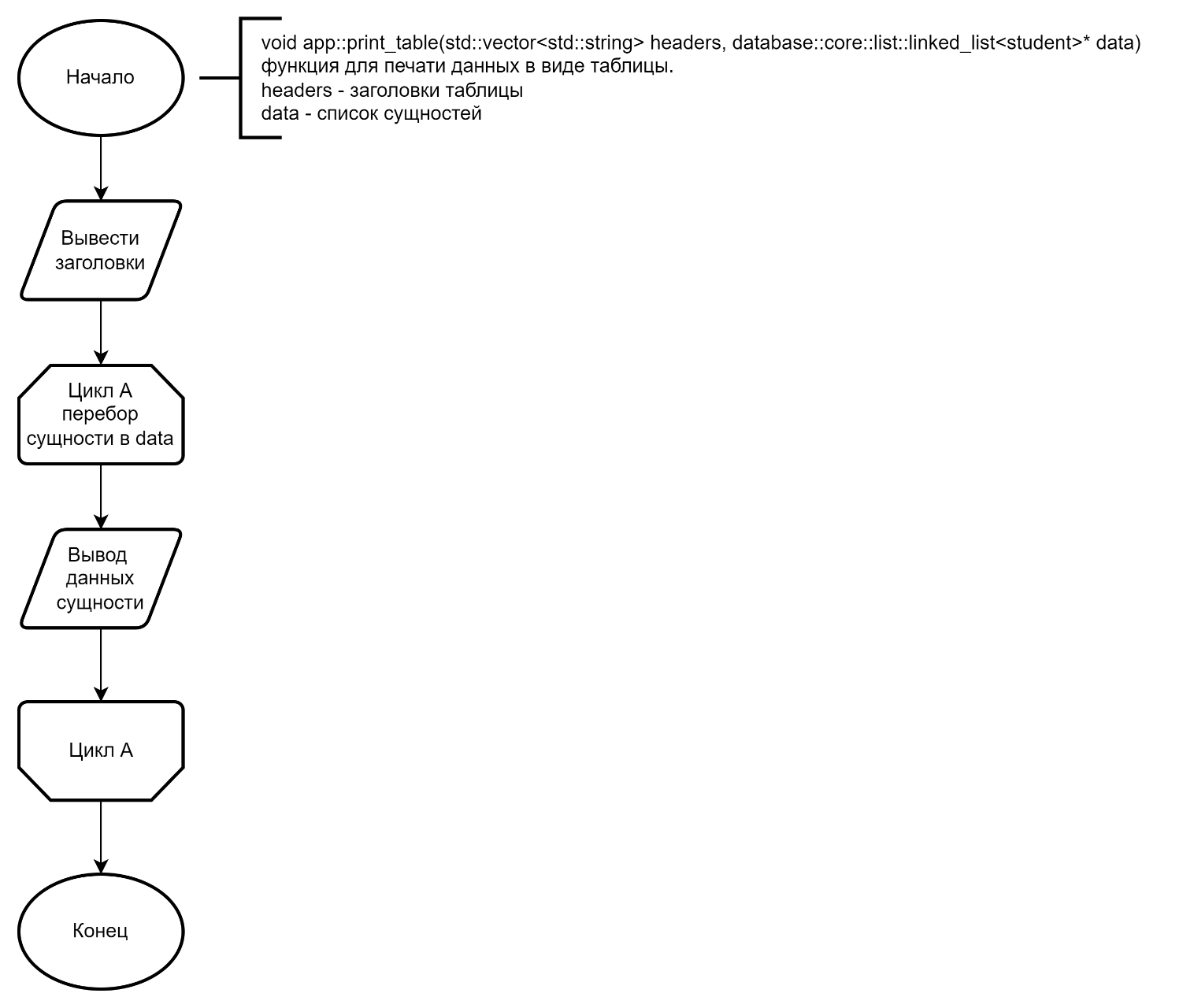
[Источники 47](#_Toc186097199)

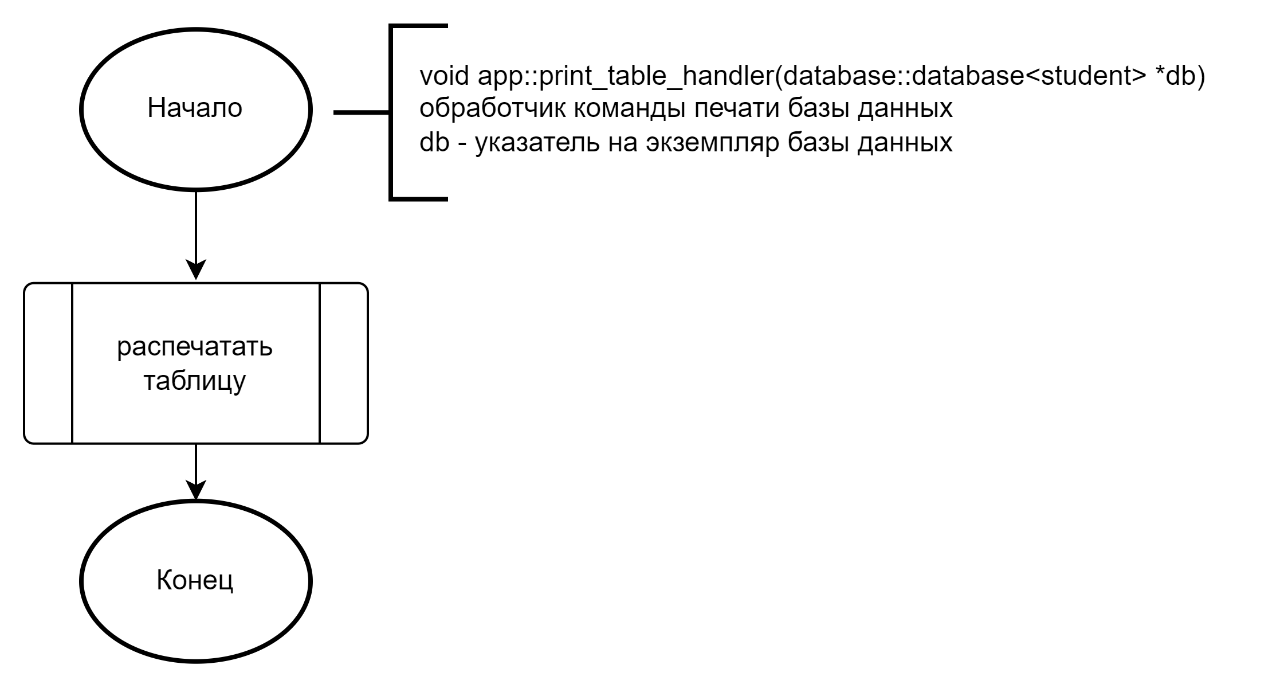
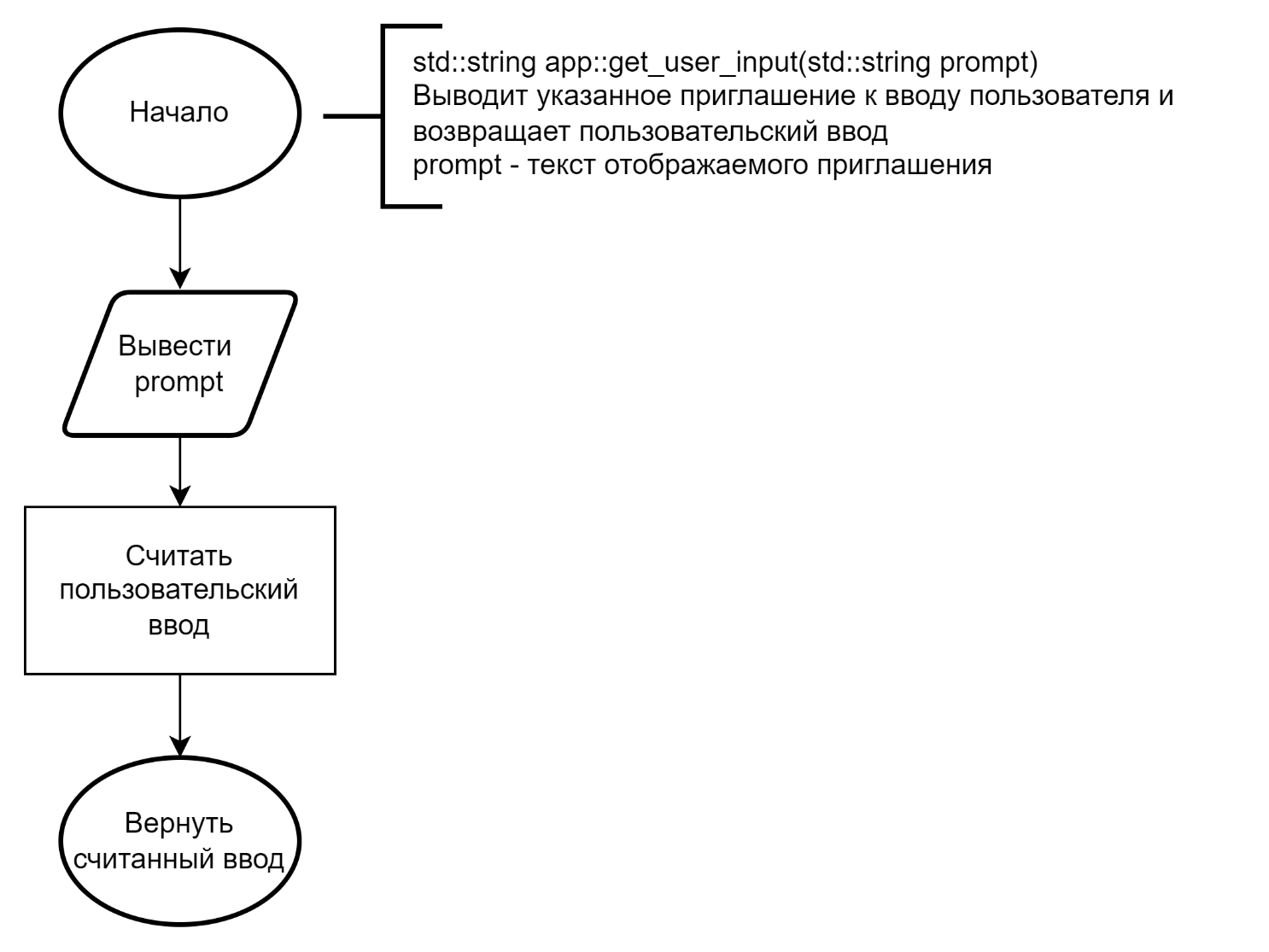
[Заключение 48](#_Toc186097200)

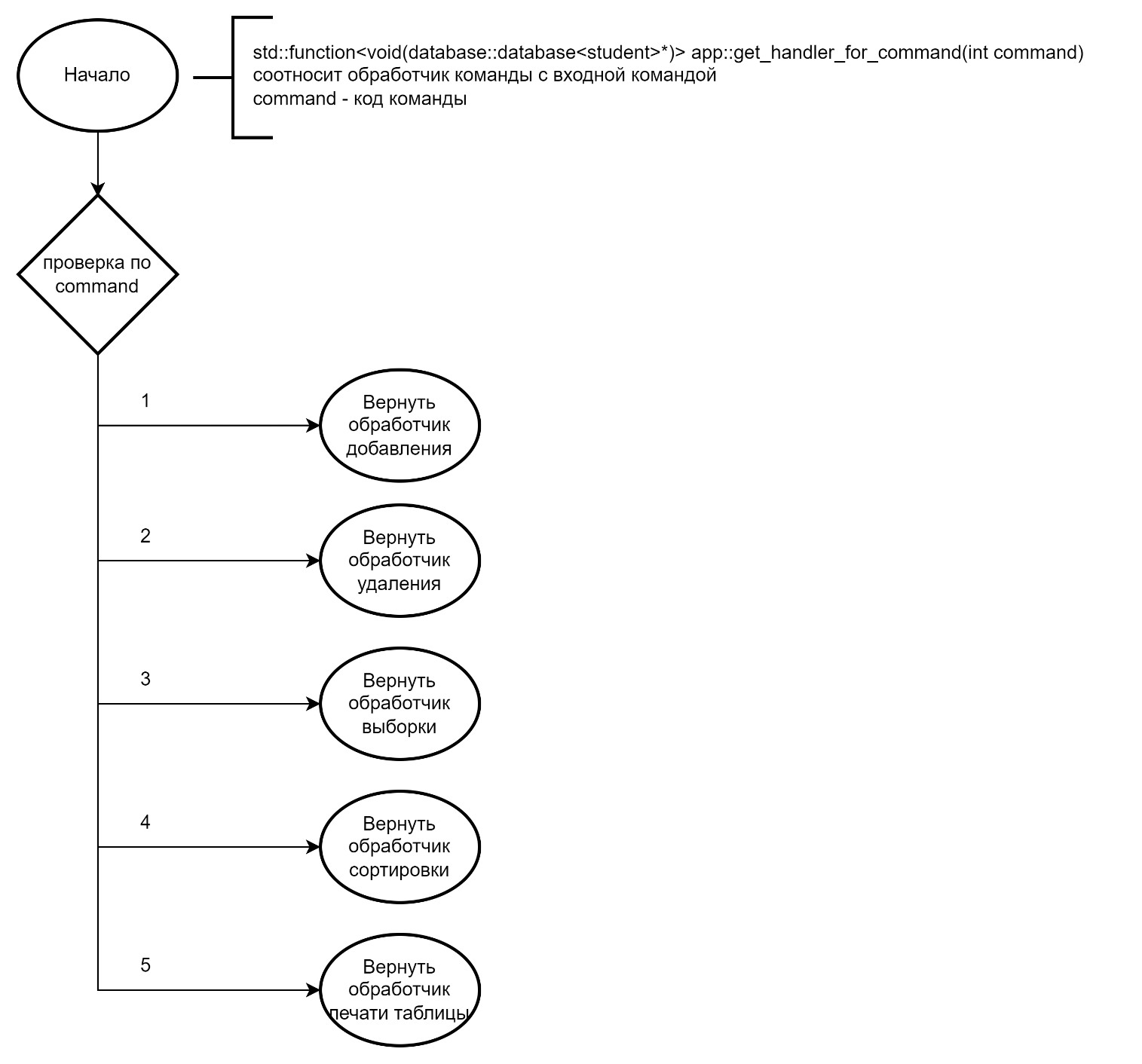
## Блок схемы

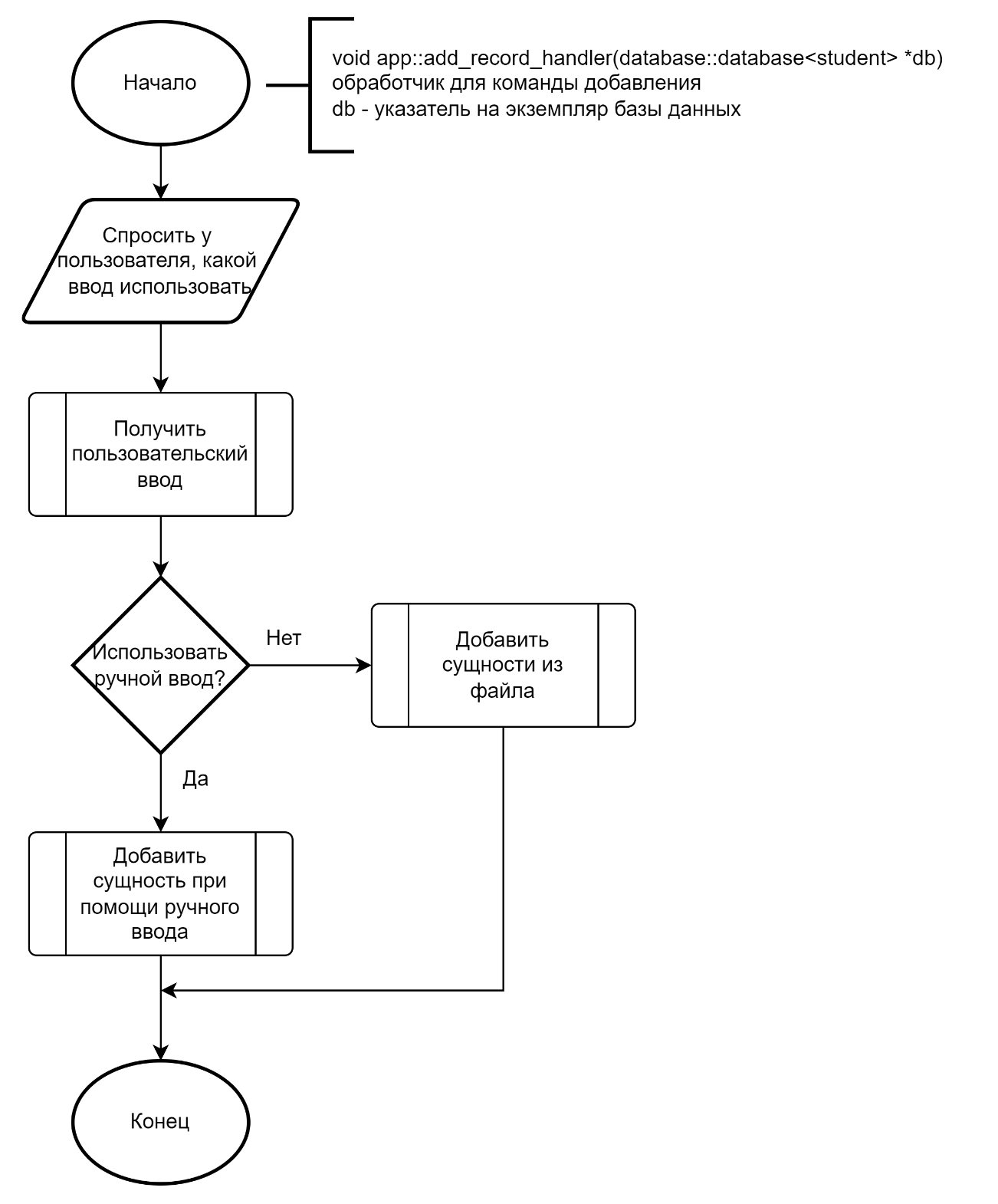


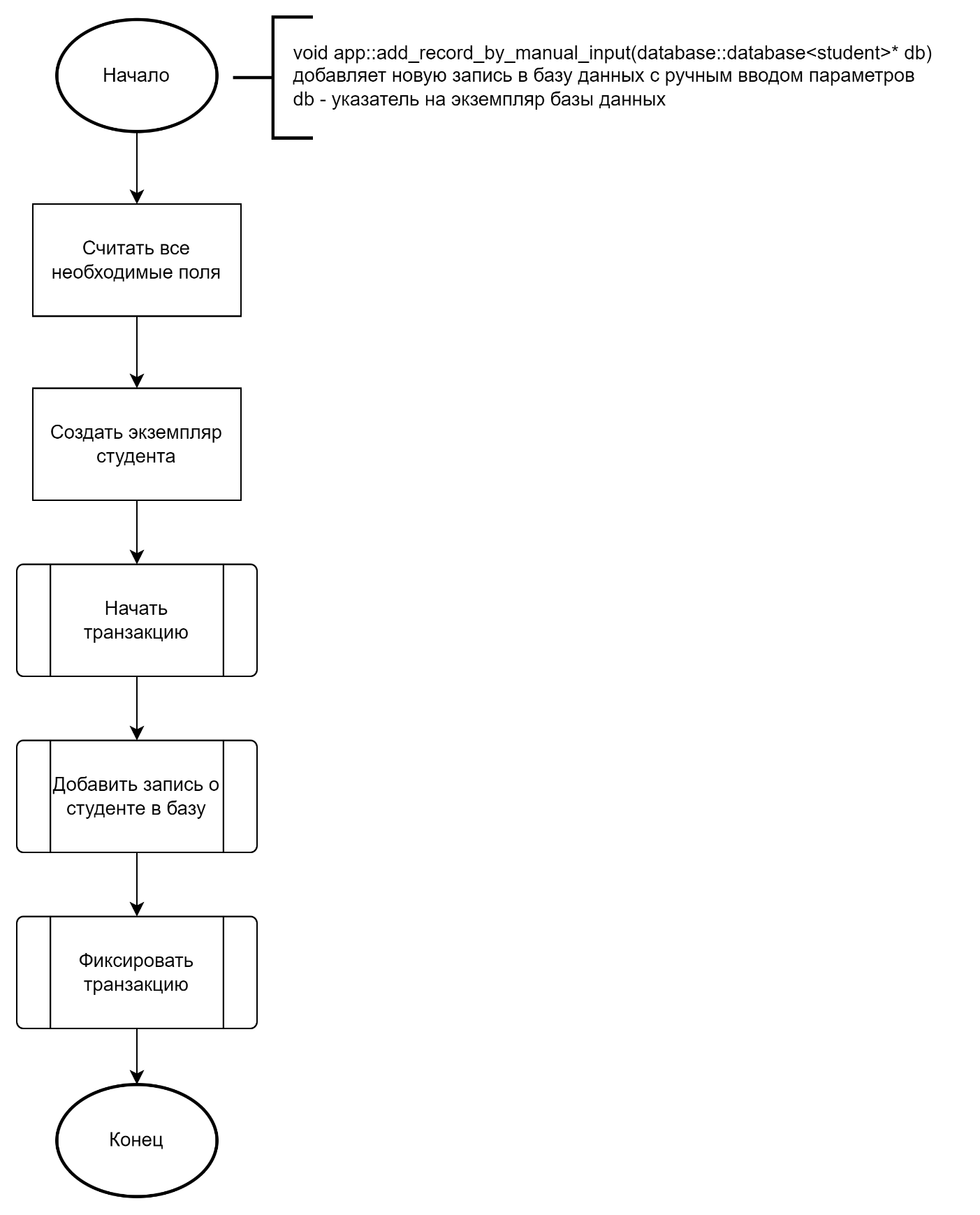


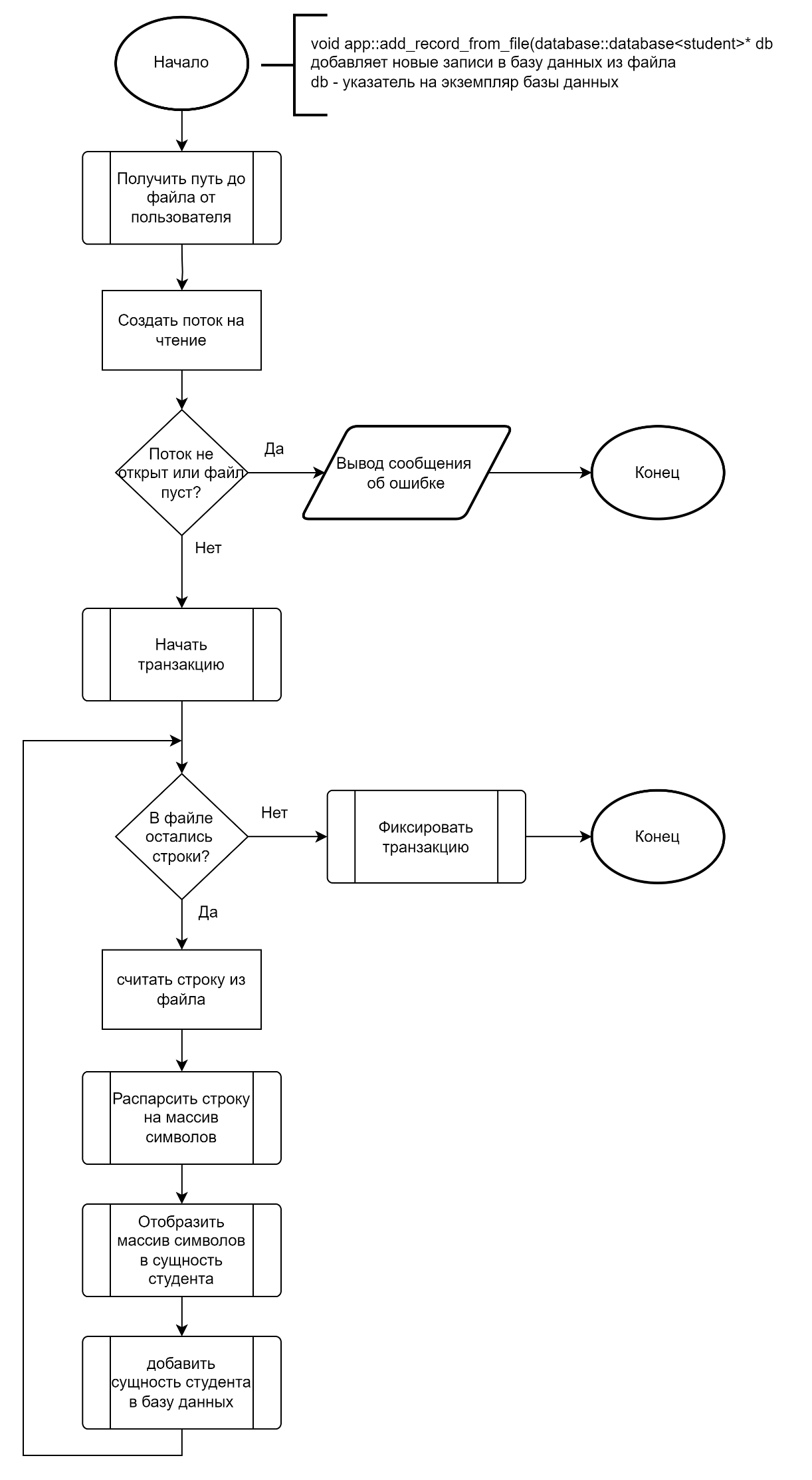


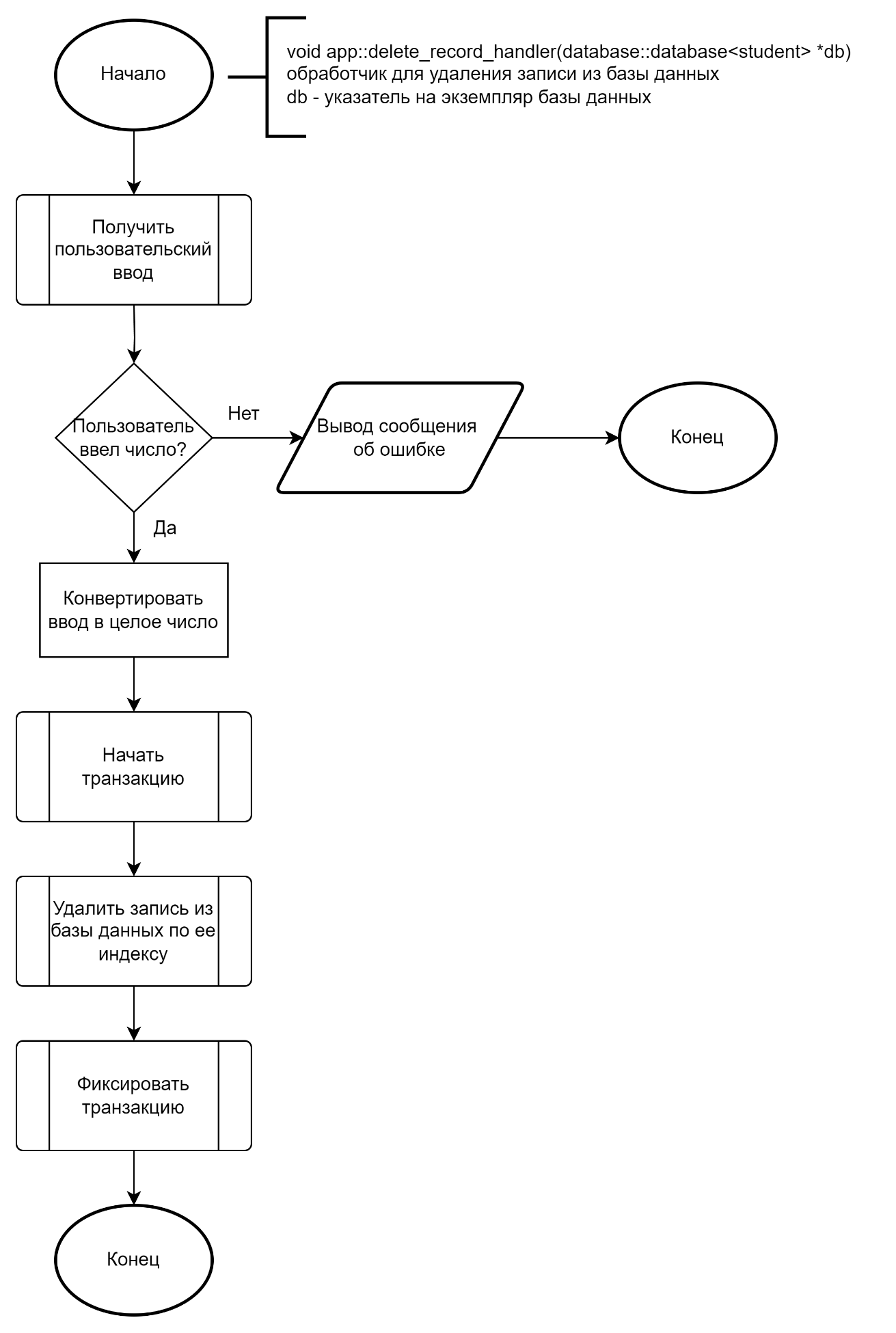


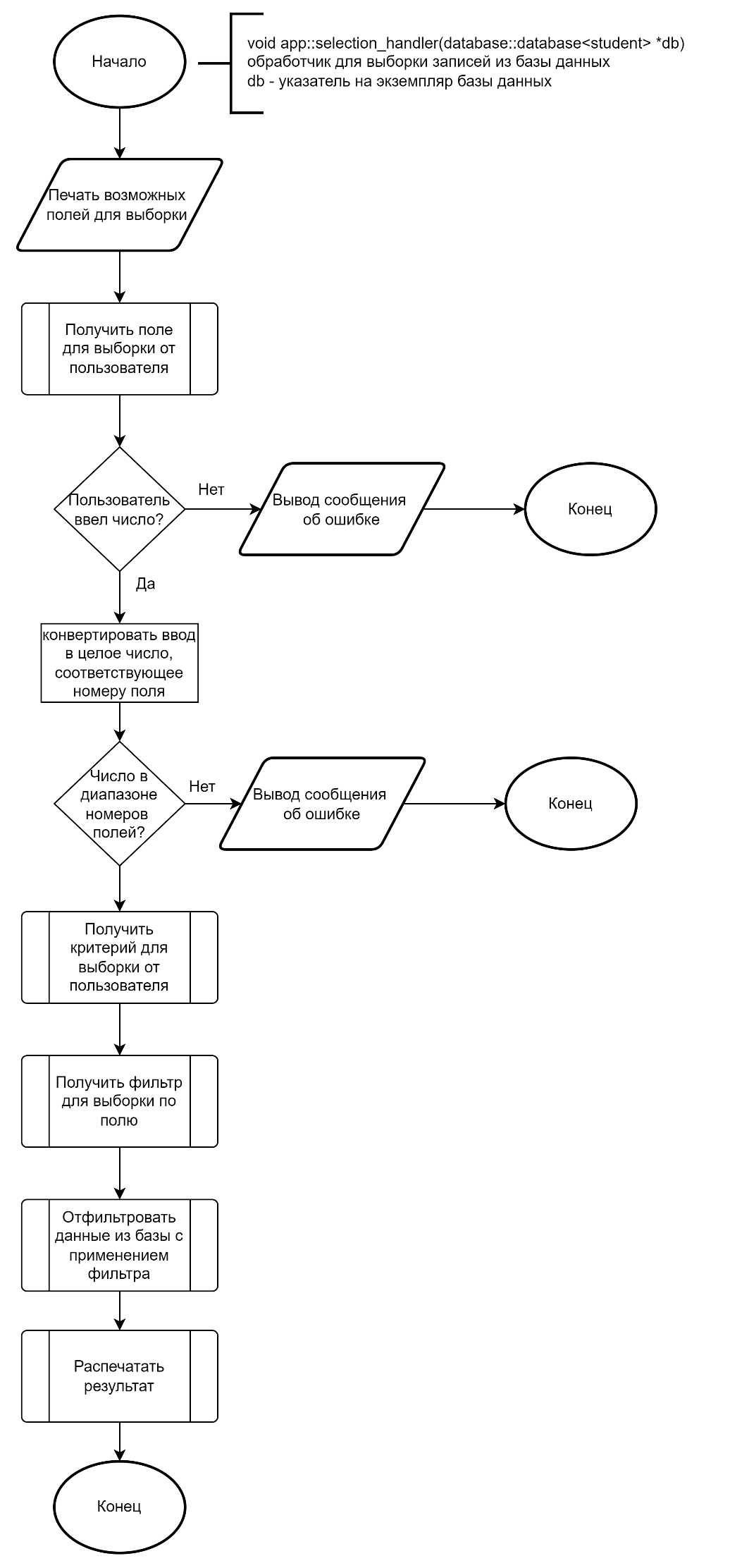


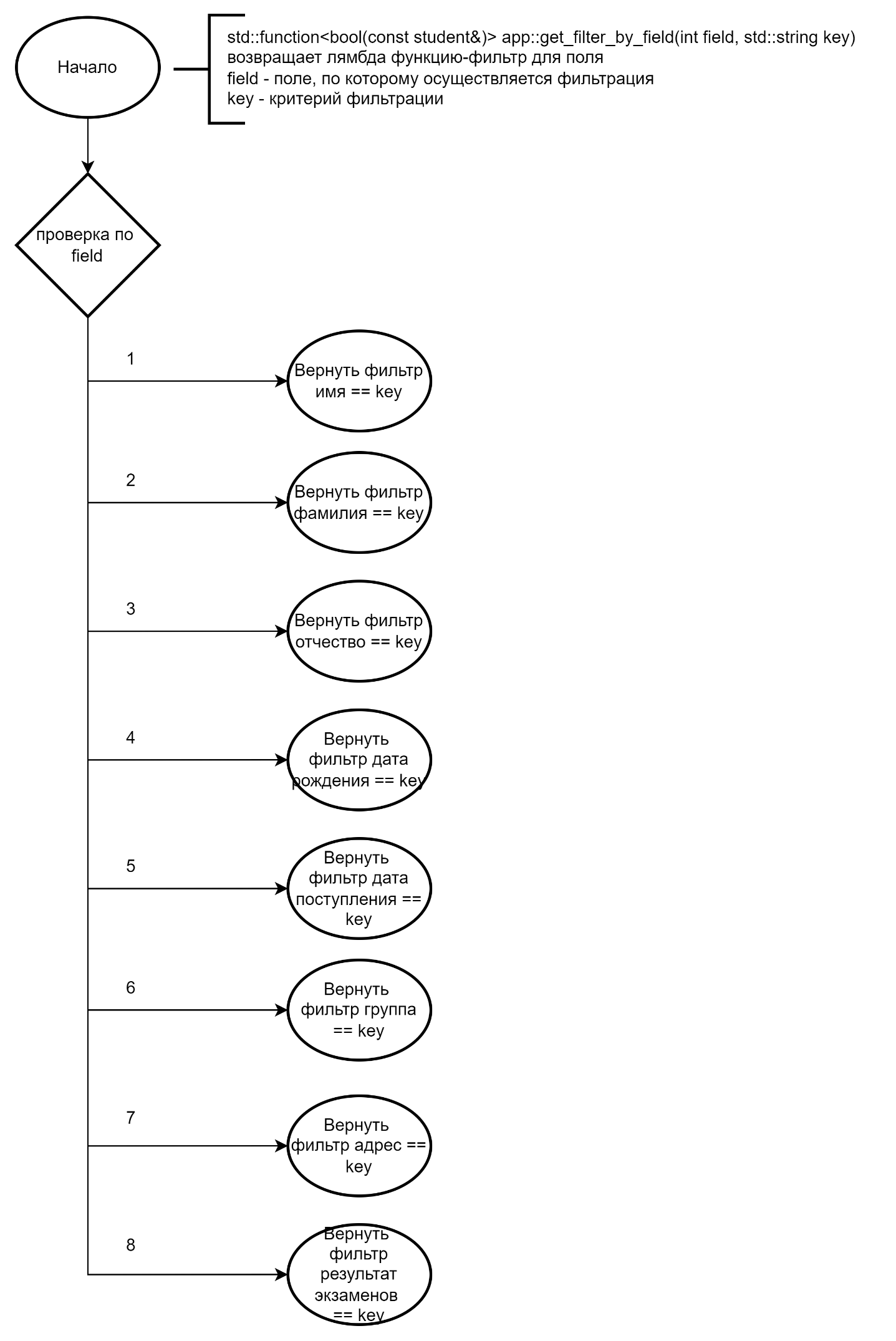


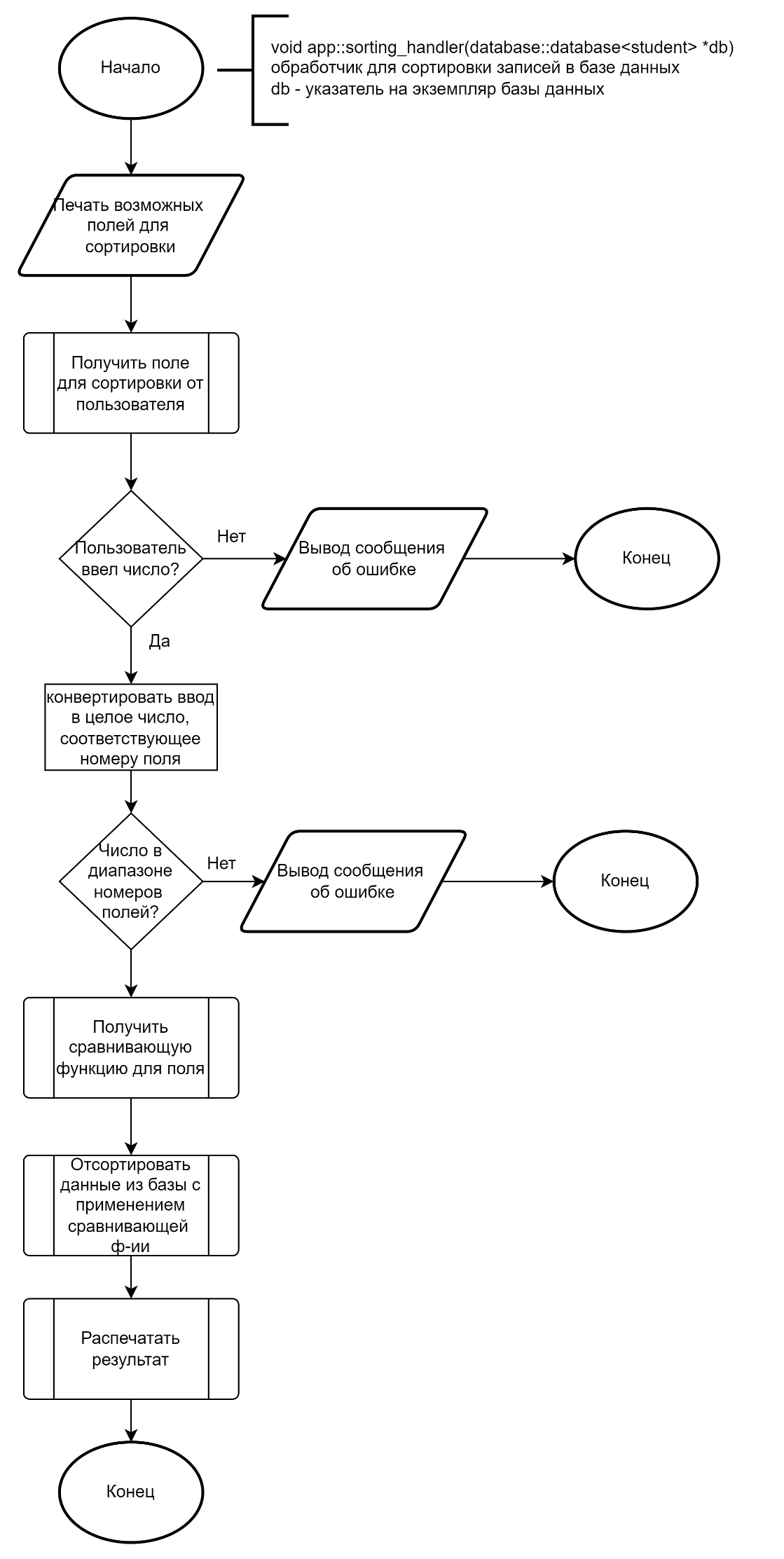


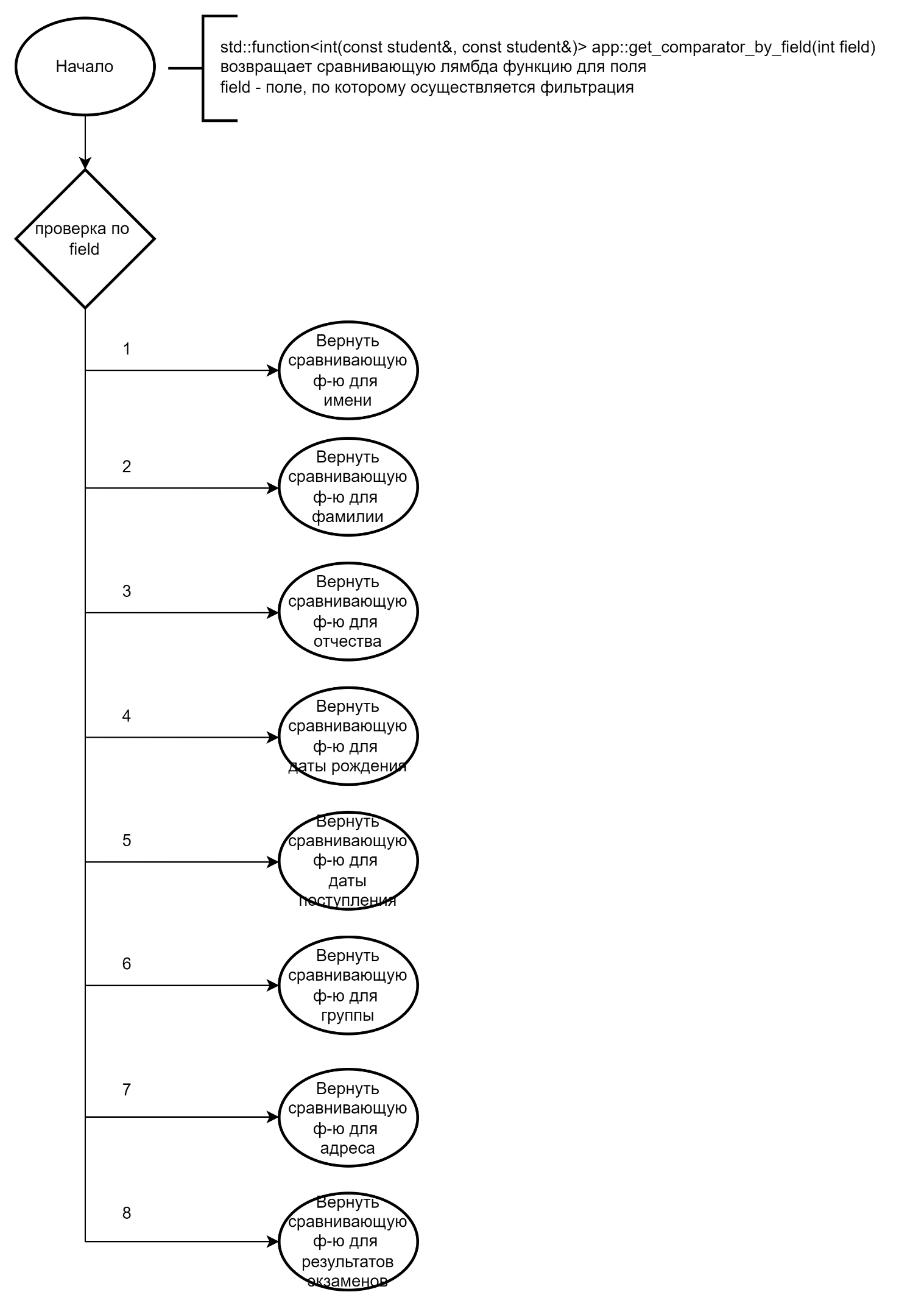




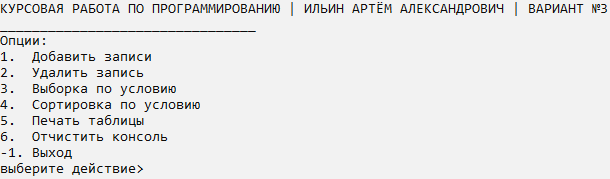








## Описание интерфейса программы

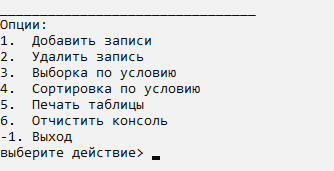


Взаимодействие с программой происходит через консольный интерфейс. При запуске программы выводится список из опций, а так же приглашение к вводу команды. Для выбора команды необходимо ввести цифру от 1 до 6 включительно или -1, для закрытия программы.

При вводе недействительной команды, программа выведет соответствующее сообщение с повторной просьбой ввода команды.



Чтобы отчистить консоль, необходимо выбрать команду 6:



Консоль после отчистки

## Код программы

kursach.h

#pragma clang diagnostic push

#pragma ide diagnostic ignored "modernize-use-nodiscard"

#ifndef KURSACH\_MAIN\_H

#define KURSACH\_MAIN\_H

#include <string>

#include <vector>

#include "linked\_list.h"

#include "student.h"

namespace app {

// start up

void run();

// command handling

bool is\_exist\_command(std::string command);

std::function<void(database::database<student>\*)> get\_handler\_for\_command(int command);

// database handlers

void add\_record\_handler(database::database<student> \*db);

void delete\_record\_handler(database::database<student> \*db);

void selection\_handler(database::database<student> \*db);

void sorting\_handler(database::database<student> \*db);

void print\_table\_handler(database::database<student> \*db);

// helpers

void print\_table(std::vector<std::string> headers, database::core::list::linked\_list<database::entities::student>\* data);

void print\_main\_menu();

std::string get\_user\_input(std::string prompt);

void add\_record\_by\_manual\_input(database::database<student>\* db);

void add\_record\_from\_file(database::database<student>\* db);

bool is\_number(const std::string& s);

void clear\_console();

std::function<bool(const student&)> get\_filter\_by\_field(int field, std::string key);

std::function<int(const student&, const student&)> get\_comparator\_by\_field(int field);

}

#endif

#pragma clang diagnostic pop

kursach.cpp

// kursach.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include <iostream>

#include <thread>

#include <chrono>

#include <set>

#include <exception>

#include "database.h"

#include "student.h"

#include "console\_table.h"

#include "kursach.h"

using database::entities::student;

using ConsoleTable = samilton::ConsoleTable;

const std::vector<std::string> headers = { "i", "FIRST NAME", "LAST NAME", "PATRONYMIC", "DATE OF BIRTH", "DATE OF ADMISSION", "GROUP", "ADDRESS", "EXAM RESULT" };

//const int sleepTime = 600;

int main()

{

setlocale(LC\_ALL, "");

std::cout << "КУРСОВАЯ РАБОТА ПО ПРОГРАММИРОВАНИЮ | ИЛЬИН АРТЁМ АЛЕКСАНДРОВИЧ | ВАРИАНТ №3" << '\n';

app::run();

return 0;

}

void app::run() {

auto database = database::database<student>("data.db");

while (true) {

app::print\_main\_menu();

auto user\_input = app::get\_user\_input("выберите действие");

while (!app::is\_exist\_command(user\_input)) {

std::cout << "Введенной команды не существует. Выберите команду из предложенных (ввод должен быть целым числом)." << '\n';

user\_input = app::get\_user\_input("выберите действие");

}

auto command = std::atoi(user\_input.c\_str());

if (command == -1) // exit from program

break;

if (command == 6) {

app::clear\_console();

continue;

}

auto handler = get\_handler\_for\_command(command);

handler(&database);

}

}

void app::print\_table(std::vector<std::string> headers, database::core::list::linked\_list<student>\* data) {

std::setlocale(LC\_ALL, "C");

ConsoleTable table(2, 2, samilton::Alignment::centre);

table.addRow(headers);

int i = 0;

auto current = data->begin();

do {

if (current == nullptr) {

std::cout << "В таблице нет ни одной записи." << '\n';

break;

}

auto student = current->data;

table.addRow({

std::to\_string(i),

student.firstName,

student.lastName,

student.patronymic,

student.dateOfBirth,

student.dateOfAdmission,

student.group,

student.address,

student.examResult

});

i++;

current = current->next;

} while (current != data->begin());

std::cout << table;

std::setlocale(LC\_ALL, "");

}

void app::clear\_console() {

system("CLS");

}

void app::print\_main\_menu() {

std::cout

<< "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << '\n'

<< "Опции:" << '\n'

<< "1. Добавить записи" << '\n'

<< "2. Удалить запись" << '\n'

<< "3. Выборка по условию" << '\n'

<< "4. Сортировка по условию" << '\n'

<< "5. Печать таблицы" << '\n'

<< "6. Отчистить консоль" << '\n'

<< "-1. Выход" << '\n';

}

std::string app::get\_user\_input(std::string prompt) {

std::cout

<< prompt << "> ";

std::string res;

std::getline(std::cin, res, '\n');

return res;

}

bool app::is\_exist\_command(string command) {

auto commands = std::set<string>{ "1", "2", "3", "4", "5", "6", "-1" };

return commands.contains(command);

}

std::function<void(database::database<student>\*)> app::get\_handler\_for\_command(int command) {

switch (command)

{

case 1:

return app::add\_record\_handler;

case 2:

return app::delete\_record\_handler;

case 3:

return app::selection\_handler;

case 4:

return app::sorting\_handler;

case 5:

return app::print\_table\_handler;

}

}

void app::add\_record\_handler(database::database<student> \*db) {

auto useManualInput = app::get\_user\_input("осуществлять ручной ввод? (Y/y - да)");

if (useManualInput == "Y" || useManualInput == "y" || useManualInput == "")

app::add\_record\_by\_manual\_input(db);

else

app::add\_record\_from\_file(db);

}

void app::add\_record\_by\_manual\_input(database::database<student>\* db) {

auto firstName = app::get\_user\_input("введите имя");

auto lastName = app::get\_user\_input("введите фамилию");

auto patronymic = app::get\_user\_input("введите отчество");

auto dateOfBirth = app::get\_user\_input("введите дату рождения");

auto dateOfAdmission = app::get\_user\_input("введите дату поступления");

auto group = app::get\_user\_input("введите номер группы");

auto address = app::get\_user\_input("введите адрес проживания");

auto examResult = app::get\_user\_input("введите результаты сессии (есть долги/нет долгов)");

auto newStudent = student(firstName, lastName, patronymic, dateOfBirth, dateOfAdmission, group, address, examResult);

db->startTransaction();

db->add(newStudent);

db->commitTransaction();

}

void app::add\_record\_from\_file(database::database<student>\* db) {

auto path = app::get\_user\_input("введите путь до файла с данными");

std::fstream file\_input(path);

if (!file\_input.good() || file\_input.peek() == EOF) {

std::cout << "Не удалось считать данные из указанного файла. Убедитесь, что файл по указанному пути существует и он не пуст." << '\n';

return;

}

std::string line;

int recordCounter = 0;

db->startTransaction();

while (std::getline(file\_input, line))

{

auto entityData = helpers::parse\_row(line, database::columnDelimiter);

auto entity = helpers::map(entityData);

db->add(entity);

recordCounter += 1;

}

db->commitTransaction();

std::cout << "Успешно добавлено " << recordCounter << " записей." << '\n';

}

void app::delete\_record\_handler(database::database<student> \*db) {

auto input = app::get\_user\_input("индекс для удаления");

if (!is\_number(input)) {

std::cout << "Индекс должен быть целым числом." << '\n';

return;

}

int index = atoi(input.c\_str());

db->startTransaction();

auto result = db->deleteBy(index);

auto message = (result ?

"Успешно удалена запись с индексом " + input :

"Не удалось удалить значение по индексу " + input + ". Проверьте, что индекс был введен правильно.");

std::cout << message << '\n';

db->commitTransaction();

}

bool app::is\_number(const std::string& s) {

std::string::const\_iterator it = s.begin();

while (it != s.end() && std::isdigit(\*it)) ++it;

return !s.empty() && it == s.end();

}

void app::selection\_handler(database::database<student> \*db) {

std::cout << "Возможные поля для выборки: " << '\n';

for (int i = 1; i < headers.size(); i++) {

std::cout << " " << i << ". " << headers[i] << '\n';

}

auto input = app::get\_user\_input("введите номер поля");

if (!is\_number(input)) {

std::cout << "Номер поля должен быть целым числом." << '\n';

return;

}

auto field = atoi(input.c\_str());

if (field < 1 || field > headers.size()) {

std::cout << "Некорректный номер поля." << '\n';

return;

}

auto key = app::get\_user\_input("введите критерий выборки (" + headers[field] + " == your\_key)");

auto selector = app::get\_filter\_by\_field(field, key);

std::cout << field << " " << key << '\n';

auto selectionResult = db->selectBy(selector);

app::print\_table(headers, selectionResult);

}

std::function<bool(const student&)> app::get\_filter\_by\_field(int field, std::string key) {

switch (field)

{

case 1:

return [key](const student& s) {

return s.firstName == key;

};

case 2:

return [key](const student& s) {

return s.lastName == key;

};

case 3:

return [key](const student& s) {

return s.patronymic == key;

};

case 4:

return [key](const student& s) {

return s.dateOfBirth == key;

};

case 5:

return [key](const student& s) {

return s.dateOfAdmission == key;

};

case 6:

return [key](const student& s) {

return s.group == key;

};

case 7:

return [key](const student& s) {

return s.address == key;

};

case 8:

return [key](const student& s) {

return s.examResult == key;

};

}

}

void app::sorting\_handler(database::database<student> \*db) {

std::cout << "Возможные поля для сортировки: " << '\n';

for (int i = 1; i < headers.size(); i++) {

std::cout << " " << i << ". " << headers[i] << '\n';

}

auto input = app::get\_user\_input("введите номер поля");

if (!is\_number(input)) {

std::cout << "Номер поля должен быть целым числом." << '\n';

return;

}

auto field = atoi(input.c\_str());

if (field < 1 || field > headers.size()) {

std::cout << "Некорректный номер поля." << '\n';

return;

}

auto comparater = app::get\_comparator\_by\_field(field);

auto sorted = db->orderBy(comparater);

app::print\_table(headers, sorted);

}

std::function<int(const student&, const student&)> app::get\_comparator\_by\_field(int field) {

switch (field) {

case 1:

return [](const student& s1, const student& s2) {

if (s1.firstName > s2.firstName)

return 1;

else if (s1.firstName == s2.firstName)

return 0;

return -1;

};

case 2:

return [](const student& s1, const student& s2) {

if (s1.lastName > s2.lastName)

return 1;

else if (s1.lastName == s2.lastName)

return 0;

return -1;

};

case 3:

return [](const student& s1, const student& s2) {

if (s1.patronymic > s2.patronymic)

return 1;

else if (s1.patronymic == s2.patronymic)

return 0;

return -1;

};

case 4:

return [](const student& s1, const student& s2) {

if (s1.dateOfBirth > s2.dateOfBirth)

return 1;

else if (s1.dateOfBirth == s2.dateOfBirth)

return 0;

return -1;

};

case 5:

return [](const student& s1, const student& s2) {

if (s1.dateOfAdmission > s2.dateOfAdmission)

return 1;

else if (s1.dateOfAdmission == s2.dateOfAdmission)

return 0;

return -1;

};

case 6:

return [](const student& s1, const student& s2) {

if (s1.group > s2.group)

return 1;

else if (s1.group == s2.group)

return 0;

return -1;

};

case 7:

return [](const student& s1, const student& s2) {

if (s1.address > s2.address)

return 1;

else if (s1.address == s2.address)

return 0;

return -1;

};

case 8:

return [](const student& s1, const student& s2) {

if (s1.examResult > s2.examResult)

return 1;

else if (s1.examResult == s2.examResult)

return 0;

return -1;

};

}

}

void app::print\_table\_handler(database::database<student> \*db) {

std::cout << "Текущие записи в бд:" << '\n';

app::print\_table(headers, db->getAll());

}

database.h

#pragma clang diagnostic push

#pragma ide diagnostic ignored "modernize-use-nodiscard"

#ifndef KURSACH\_DATABASE\_H

#define KURSACH\_DATABASE\_H

#include <string>

#include <functional>

#include "linked\_list.h"

using std::string;

using database::core::list::linked\_list;

namespace database {

template<class T>

class database {

private:

// properties

bool hasActiveTransaction;

string path;

linked\_list<T>\* data;

// methods

void rollBack();

linked\_list<T>\* readFrom(string path);

void writeTo(string path);

public:

// constructors

database(string path);

~database();

// methods

void add(T item);

bool deleteBy(int id);

linked\_list<T>\* orderBy(std::function<int(const T&, const T&)> comparator);

linked\_list<T>\* selectBy(std::function<bool(const T&)> selector);

linked\_list<T>\* getAll();

void startTransaction();

bool commitTransaction();

};

}

#include "database.cpp"

#endif

#pragma clang diagnostic pop

database.cpp

#include <iostream>

#include <fstream>

#include <ostream>

#include <vector>

#include <exception>

#include <iomanip>

#include "sorting.h"

#include "helpers.h"

using database::core::sorting::quick\_sort;

namespace database {

#pragma region database

const string columnDelimiter = " | ";

template<class T>

database<T>::database(string path) {

this->path = path;

hasActiveTransaction = false;

data = readFrom(path);

}

template<class T>

database<T>::~database() {

data->~linked\_list();

}

template<class T>

linked\_list<T>\* database<T>::readFrom(string path) {

std::fstream file\_input(path);

try {

if (!file\_input.good() || file\_input.peek() == EOF)

return new linked\_list<T>{};

string line;

auto result = new linked\_list<T>{};

while (std::getline(file\_input, line))

{

auto entityData = helpers::parse\_row(line, columnDelimiter);

T entity = helpers::map(entityData);

result->push\_back(entity);

}

file\_input.close();

return result;

}

catch (const std::exception& ex) {

std::cout << ex.what() << '\n';

file\_input.close();

}

}

template<class T>

void database<T>::writeTo(string path) {

std::fstream file\_output(path, std::ios::out | std::ios::trunc);

try {

if (!file\_output.good())

throw std::runtime\_error("Can't open the file on the path: " + path);

for (int i = 0; i < data->size(); i++) {

auto entityData = helpers::map(data->nodeAt(i)->data);

for (int j = 0; j < entityData.size(); j++) {

file\_output << entityData[j];

if (j != entityData.size() - 1)

file\_output << columnDelimiter;

}

file\_output << '\n';

}

}

catch (const std::exception& ex) {

file\_output.close();

throw ex;

}

file\_output.close();

}

template<class T>

void database<T>::rollBack() {

data = readFrom(path);

}

template<class T>

void database<T>::add(T item) {

if (!hasActiveTransaction)

throw std::runtime\_error("Start the transaction before adding.");

data->push\_back(item);

}

template<class T>

bool database<T>::deleteBy(int id) {

if (!hasActiveTransaction)

throw std::runtime\_error("Start the transaction before deleting.");

if (id < 0 || id >= data->size())

return false;

data->erase(id);

return true;

}

template<class T>

linked\_list<T>\* database<T>::orderBy(std::function<int(const T&, const T&)> comparator) {

auto resultList = new linked\_list<T>();

data->copyTo(resultList);

::database::core::sorting::quick\_sort(resultList, comparator);

return resultList;

}

template<class T>

linked\_list<T>\* database<T>::selectBy(std::function<bool(const T&)> selector) {

auto resultList = new linked\_list<T>();

auto currentNode = data->begin();

while (currentNode != nullptr) {

if (selector(currentNode->data))

resultList->push\_back(currentNode->data);

currentNode = currentNode->next;

}

/\* do {

if (selector(currentNode->data))

resultList->push\_back(currentNode->data);

currentNode = currentNode->next;

} while (currentNode != data->begin());\*/

return resultList;

}

template<class T>

linked\_list<T>\* database<T>::getAll() {

return data;

}

template<class T>

void database<T>::startTransaction() {

if (hasActiveTransaction)

throw std::runtime\_error("Active transaction already exists.");

hasActiveTransaction = true;

}

template<class T>

bool database<T>::commitTransaction() {

if (!hasActiveTransaction)

throw std::runtime\_error("Start the transaction before commiting.");

try

{

writeTo(path);

hasActiveTransaction = false;

return true;

}

catch (const std::exception& ex)

{

std::cerr << "commit failed with error: " << ex.what() << '\n';

std::cout << "rollbacking the transactioin...." << '\n';

rollBack();

std::cout << "rallback success." << '\n';

hasActiveTransaction = false;

return false;

}

}

#pragma endregion

}

helpers.h

#pragma clang diagnostic push

#pragma ide diagnostic ignored "modernize-use-nodiscard"

#ifndef KURSACH\_HELPERS\_H

#define KURSACH\_HELPERS\_H

#include "student.h"

#include <vector>

#include <string>

#include <sstream>

using database::entities::student;

namespace helpers {

student map(std::vector<std::string> data);

std::vector<std::string> map(student entity);

std::vector<std::string> parse\_row(std::string s, std::string delimiter);

}

#endif

#pragma clang diagnostic pop

helpers.cpp

#include "helpers.h"

namespace helpers {

student map(std::vector<std::string> data) {

return student(data[0], data[1], data[2], data[3], data[4], data[5], data[6], data[7]);

}

std::vector<std::string> map(student entity) {

return {

entity.firstName,

entity.lastName,

entity.patronymic,

entity.dateOfBirth,

entity.dateOfAdmission,

entity.group,

entity.address,

entity.examResult

};

}

std::vector<std::string> parse\_row(std::string s, std::string delimiter) {

size\_t pos\_start = 0, pos\_end, delim\_len = delimiter.length();

std::string token;

std::vector<std::string> res;

while ((pos\_end = s.find(delimiter, pos\_start)) != std::string::npos) {

token = s.substr(pos\_start, pos\_end - pos\_start);

pos\_start = pos\_end + delim\_len;

res.push\_back(token);

}

res.push\_back(s.substr(pos\_start));

return res;

}

}

linked\_list.h

#pragma clang diagnostic push

#pragma ide diagnostic ignored "modernize-use-nodiscard"

#ifndef KURSACH\_LINKED\_LIST\_H

#define KURSACH\_LINKED\_LIST\_H

#include <optional>

#include <iostream>

namespace database {

namespace core {

namespace list {

template<class T>

class node {

public:

T data;

node\* next;

node\* prev;

explicit node(T newData);

node(T newData, node\* newNext, node\* newPrev);

~node() = default;

};

template<class T>

class linked\_list {

private:

node<T>\* head;

node<T>\* tail;

int \_size;

void link\_head\_tail(node<T>\* head, node<T>\* tail);

public:

linked\_list();

~linked\_list();

// node accessors

node<T>\* begin() const; // O(1)

node<T>\* end() const; // O(1)

node<T>\* nodeAt(int index) const; // O(n)

//capacity

bool empty() const; // O(1)

int size() const; // O(1)

//modifiers

void clear(); // O(n)

void insert(node<T>\* node, T data); // before node O(n)

void insert(int pos, T data); // before pos O(n)

void erase(node<T>\* node); // O(1)

void erase(int index); // O(n)

void push\_back(T data); // O(1)

void pop\_back(); //O(1)

void push\_front(T data); // O(1)

void pop\_front(); // O(1)

void copyTo(linked\_list<T>\* anotherList);

};

}

}

}

#include "linked\_list.cpp"

#endif

#pragma clang diagnostic pop

linked\_list.cpp

namespace database {

namespace core {

namespace list {

#pragma region node

template<class T>

::database::core::list::node<T>::node(T newData) {

data = newData;

next = nullptr;

prev = nullptr;

}

template<class T>

::database::core::list::node<T>::node(T newData, node\* newNext, node\* newPrev) {

data = newData;

next = newNext;

prev = newPrev;

}

#pragma endregion

#pragma region linked\_list

template<class T>

::database::core::list::linked\_list<T>::linked\_list() : head{}, tail{}, \_size{} {}

template<class T>

::database::core::list::linked\_list<T>::~linked\_list() {

database::core::list::node<T>\* current = head;

while (current != nullptr) {

database::core::list::node<T>\* next = current->next;

delete current;

current = next;

}

}

template<class T>

::database::core::list::node<T>\* ::database::core::list::linked\_list<T>::begin() const { return head; }

template<class T>

::database::core::list::node<T>\* ::database::core::list::linked\_list<T>::end() const { return tail; }

template<class T>

::database::core::list::node<T>\* ::database::core::list::linked\_list<T>::nodeAt(int index) const {

database::core::list::node<T>\* current = head;

for (int i = 0; i < index; i++) {

if (current == nullptr)

return nullptr;

current = current->next;

}

return current;

}

template<class T>

bool ::database::core::list::linked\_list<T>::empty() const { return head == nullptr; }

template<class T>

int ::database::core::list::linked\_list<T>::size() const { return \_size; }

template<class T>

void ::database::core::list::linked\_list<T>::clear() {

::database::core::list::node<T>\* current = head;

// delete nodes until there's none left

while (current != nullptr) {

::database::core::list::node<T>\* next = current->next;

delete current;

current = next;

}

head = nullptr;

tail = nullptr;

\_size = 0;

}

template<class T>

void ::database::core::list::linked\_list<T>::insert(::database::core::list::node<T>\* after\_node, T data) {

if (after\_node == nullptr) {

return;

}

// link the new node to its neighbors

::database::core::list::node<T>\* new\_node = new node(data, after\_node, after\_node->prev);

// link the prev node to the new node

if (after\_node->prev != nullptr) {

after\_node->prev->next = new\_node;

}

// link the after node to the new node

after\_node->prev = new\_node;

if (after\_node == head) {

head = new\_node;

}

\_size++;

}

template<class T>

void ::database::core::list::linked\_list<T>::insert(int pos, T data) {

if (pos == \_size) {

push\_back(data);

}

else if (pos == 0) {

push\_front(data);

}

else {

insert(nodeAt(pos), data);

}

}

template<class T>

void ::database::core::list::linked\_list<T>::erase(::database::core::list::node<T>\* node) {

if (node == nullptr) {

return;

}

// link the prev node to the next one

if (node->prev != nullptr) {

node->prev->next = node->next;

}

// link the next node to the prev one

if (node->next != nullptr) {

node->next->prev = node->prev;

}

// fix head

if (node == head) {

head = node->next;

}

// fix tail

if (node == tail) {

tail = node->prev;

}

delete node;

\_size--;

}

template<class T>

void ::database::core::list::linked\_list<T>::erase(int index) {

erase(nodeAt(index));

}

template<class T>

void ::database::core::list::linked\_list<T>::push\_back(T data) {

database::core::list::node<T>\* new\_node = new node(data);

if (tail == nullptr) {

// empty

head = new\_node;

tail = new\_node;

link\_head\_tail(head, tail);

}

else {

// link tail node to ours

new\_node->prev = tail;

// link our node to tail

tail->next = new\_node;

// fix tail

tail = new\_node;

}

\_size++;

}

template<class T>

void ::database::core::list::linked\_list<T>::pop\_back() {

if (tail == nullptr) { return; }

database::core::list::node<T>\* new\_last = tail->prev;

delete tail;

tail = new\_last;

if (tail == nullptr) {

// list is empty now, fix head

head = nullptr;

}

\_size--;

}

template<class T>

void ::database::core::list::linked\_list<T>::push\_front(T data) {

database::core::list::node<T>\* new\_node = new node(data);

if (head == nullptr) {

// empty

head = new\_node;

tail = new\_node;

link\_head\_tail(head, tail);

}

else {

// link head node to ours

new\_node->next = head;

// link our node to head

head->prev = new\_node;

// fix head

head = new\_node;

}

\_size++;

}

template<class T>

void ::database::core::list::linked\_list<T>::link\_head\_tail(node<T>\* head, node<T>\* tail){

head->next = tail;

tail->prev = head;

}

template<class T>

void ::database::core::list::linked\_list<T>::pop\_front() {

if (head == nullptr) { return; }

database::core::list::node<T>\* new\_first = head->next;

delete head;

head = new\_first;

if (head == nullptr) {

// list is empty now, fix tail

tail = nullptr;

}

}

template<class T>

void ::database::core::list::linked\_list<T>::copyTo(linked\_list<T>\* anotherList) {

auto currentNode = head;

if (currentNode == nullptr)

return;

while (currentNode != nullptr) {

anotherList->push\_back(currentNode->data);

currentNode = currentNode->next;

}

/\*do {

anotherList->push\_back(currentNode->data);

currentNode = currentNode->next;

} while (currentNode != begin());\*/

}

#pragma endregion

}

}

}

sorting.h

#pragma clang diagnostic push

#pragma ide diagnostic ignored "modernize-use-nodiscard"

#ifndef KURSACH\_SORTING\_H

#define KURSACH\_SORTING\_H

#include <functional>

#include "linked\_list.h"

using database::core::list::linked\_list;

namespace database {

namespace core {

namespace sorting {

template<class T>

void quick\_sort(linked\_list<T>\* list, std::function<int(const T&, const T&)> comparator);

}

}

}

#include "sorting.cpp"

#endif

#pragma clang diagnostic pop

sorting.cpp

#include "stack.h"

using database::core::list::node;

namespace database {

namespace core {

namespace sorting {

template<class T>

void swap(node<T>\* left, node<T>\* right) {

auto tmp = left->data;

left->data = right->data;

right->data = tmp;

}

template<class T>

void quick\_sort(linked\_list<T>\* list, std::function<int(const T&, const T&)> comparator) {

::database::core::stack::stack<std::pair<int, int>> sort\_stack = ::database::core::stack::stack<std::pair<int, int>>();

sort\_stack.push(std::pair(0, list->size() - 1));

while (!sort\_stack.empty()) {

auto [low\_i, high\_i] = sort\_stack.top();

sort\_stack.pop();

if (low\_i >= high\_i)

continue;

auto pivot = list->nodeAt(low\_i)->data;

int i = low\_i - 1, j = high\_i + 1;

while (true) {

do {

i++;

} while (comparator(list->nodeAt(i)->data, pivot) < 0);

do {

j--;

} while (comparator(list->nodeAt(j)->data, pivot) > 0);

if (i >= j)

break;

swap(list->nodeAt(i), list->nodeAt(j));

}

sort\_stack.push(std::pair(low\_i, j));

sort\_stack.push(std::pair(j + 1, high\_i));

}

}

}

}

}

stack.h

#pragma clang diagnostic push

#pragma ide diagnostic ignored "modernize-use-nodiscard"

#ifndef KURSACH\_STACK\_H

#define KURSACH\_STACK\_H

namespace database {

namespace core {

namespace stack {

template<class T>

class node {

public:

T data;

node\* next;

node(T data);

node(T data, node\* next);

~node() = default;

};

template<class T>

class stack {

private:

node<T>\* peek;

node<T>\* tail;

int \_size;

public:

stack();

~stack();

// node accessors

node<T>\* begin() const; // O(1)

node<T>\* end() const; // O(1)

//capacity

bool empty() const; // O(1)

int size() const; // O(1)

//modifiers

void clear(); // O(n)

void push(T data); // O(1)

T top(); // O(1)

T pop(); // O(1)

};

}

}

}

#include "stack.cpp"

#endif

#pragma clang diagnostic pop

stack.cpp

using database::core::stack::stack;

namespace database {

namespace core {

namespace stack {

#pragma region node

template<class T>

node<T>::node(T data) {

this->data = data;

next = nullptr;

}

template<class T>

node<T>::node(T data, node<T>\* next) {

this->data = data;

this->next = next;

}

#pragma endregion

#pragma region stack

template<class T>

stack<T>::stack() {

peek = nullptr;

tail = nullptr;

\_size = 0;

}

template<class T>

stack<T>::~stack() {

auto current = peek;

while (current != nullptr) {

auto next = current->next;

delete current;

current = next;

}

}

template<class T>

node<T>\* stack<T>::begin() const { return peek; }

template<class T>

node<T>\* stack<T>::end() const { return tail; }

template<class T>

bool stack<T>::empty() const { return peek == nullptr; }

template<class T>

int stack<T>::size() const { return \_size; }

template<class T>

void stack<T>::clear() {

auto current = peek;

while (current != nullptr) {

auto next = current->next;

delete current;

current = next;

}

peek = nullptr;

tail = nullptr;

\_size = 0;

}

template<class T>

void stack<T>::push(T data) {

auto newNode = new node(data, peek);

if (peek == nullptr) { // add first node

peek = newNode;

tail = newNode;

}

else {

peek = newNode;

}

}

template<class T>

T stack<T>::top() {

if (peek == nullptr)

throw std::runtime\_error("stack is empty.");

return peek->data;

}

template<class T>

T stack<T>::pop() {

auto newPeek = peek->next;

auto result = peek->data;

delete peek;

peek = newPeek;

if (peek == nullptr)

tail = nullptr;

return result;

}

#pragma endregion

}

}

}

student.h

#pragma clang diagnostic push

#pragma ide diagnostic ignored "modernize-use-nodiscard"

#ifndef KURSACH\_STUDENT\_H

#define KURSACH\_STUDENT\_H

#include <string>

#include <iostream>

#include <functional>

using std::string;

namespace database {

namespace entities {

class student {

public:

string firstName;

string lastName;

string patronymic;

string dateOfBirth;

string dateOfAdmission;

string group;

string address;

string examResult;

student();

student(

string firstName,

string lastName,

string patronymic,

string dateOfBirth,

string dateOfAdmission,

string group,

string address,

string examResult

);

friend std::ostream& operator<<(std::ostream& os, const student& s);

};

}

}

#endif

#pragma clang diagnostic pop

student.cpp

#include "student.h"

#include <iomanip>

namespace database {

namespace entities {

const int outputWidth = 7;

student::student() { }

student::student(

string firstName,

string lastName,

string patronymic,

string dateOfBirth,

string dateOfAdmission,

string group,

string address,

string examResult)

{

this->firstName = firstName;

this->lastName = lastName;

this->patronymic = patronymic;

this->dateOfBirth = dateOfBirth;

this->dateOfAdmission = dateOfAdmission;

this->group = group;

this->address = address;

this->examResult = examResult;

}

std::ostream& operator<<(std::ostream& os, const student& s) {

os << " | "

<< std::setw(outputWidth) << s.firstName << std::setw(outputWidth) << " | "

<< std::setw(outputWidth) << s.lastName << std::setw(outputWidth) << " | "

<< std::setw(outputWidth) << s.patronymic << std::setw(outputWidth) << " | "

<< std::setw(outputWidth) << s.dateOfBirth << std::setw(outputWidth) << " | "

<< std::setw(outputWidth) << s.group << std::setw(outputWidth) << " | "

<< std::setw(outputWidth) << s.address << std::setw(outputWidth) << " | "

<< std::setw(outputWidth) << s.examResult << std::setw(outputWidth) << " | ";

return os;

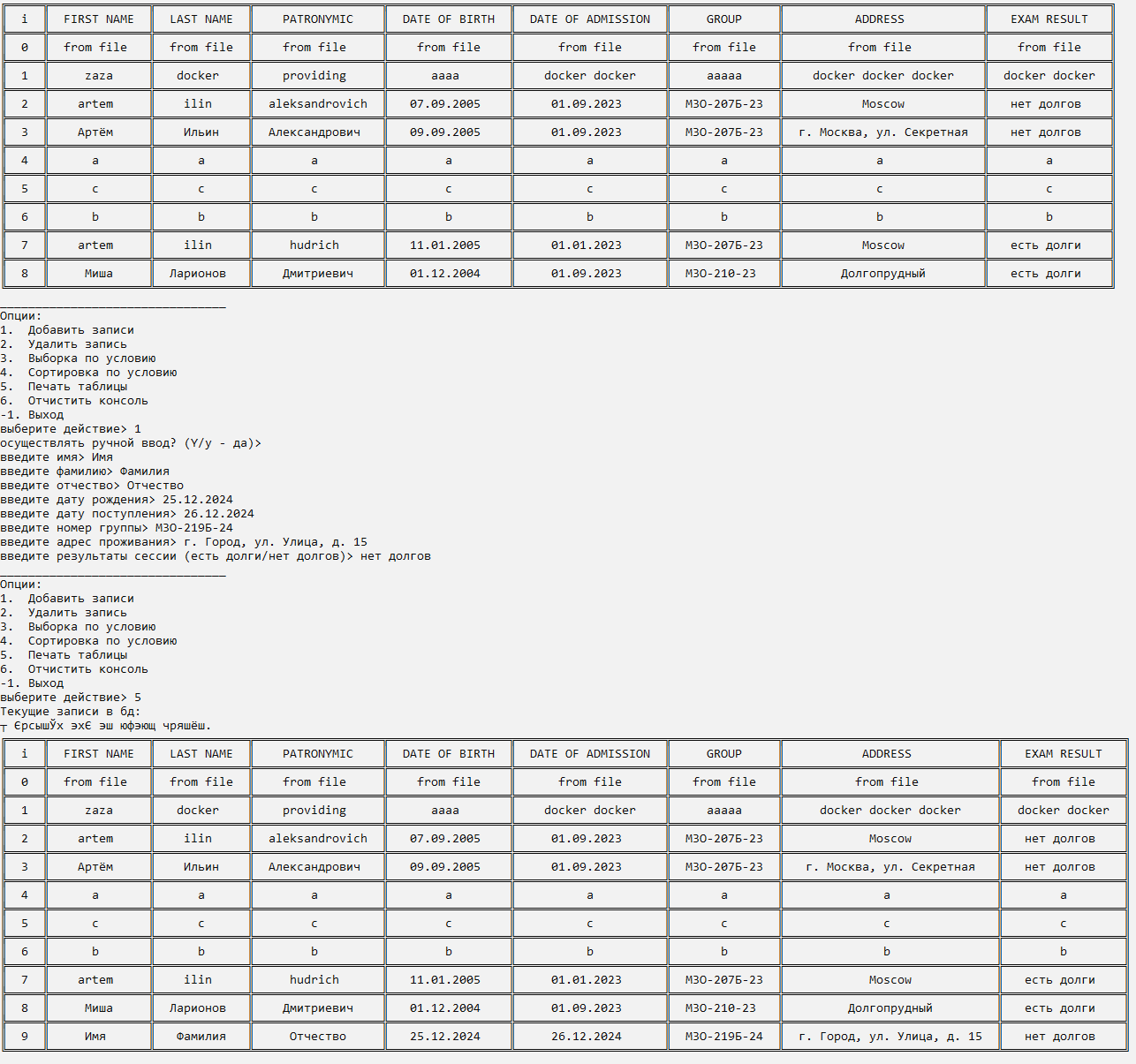
}

}

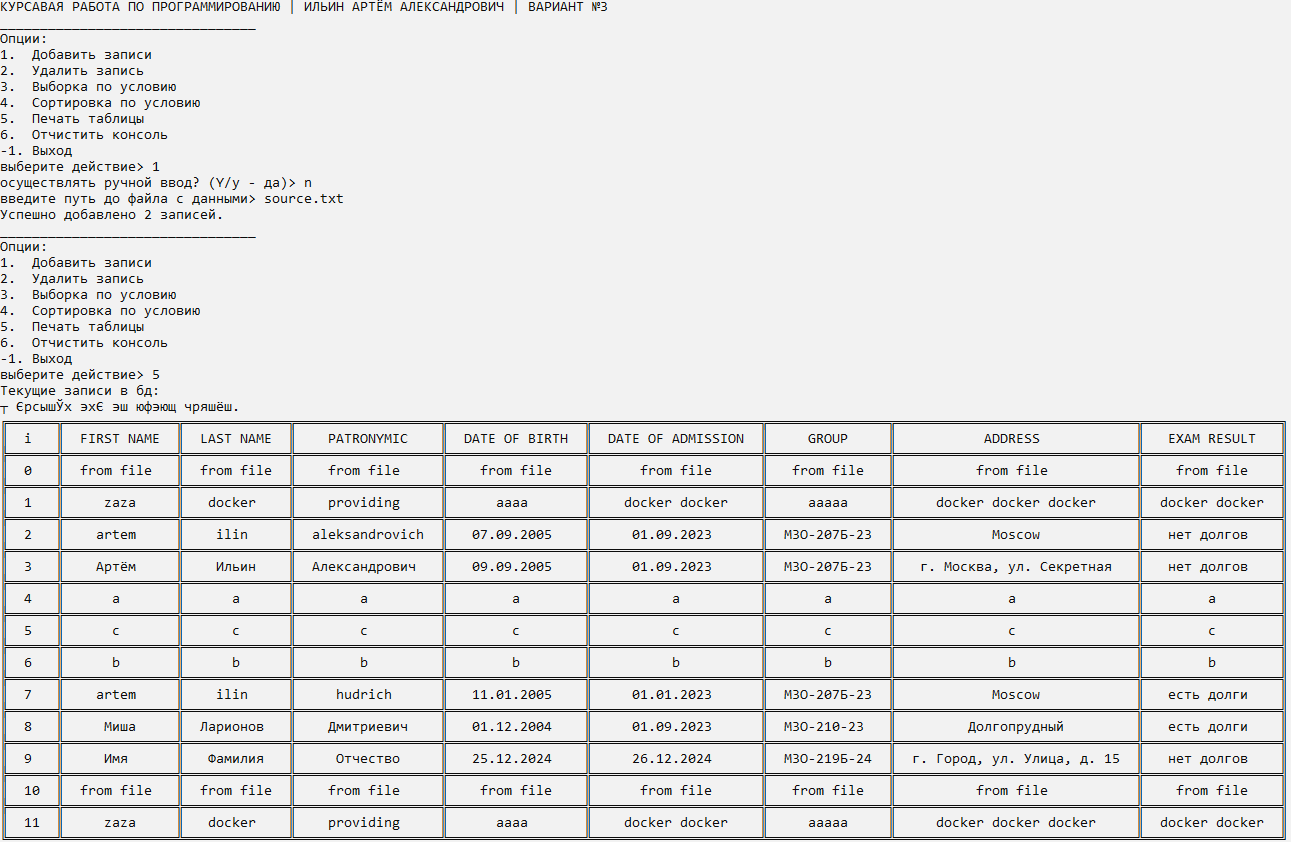
}

## Результаты работы

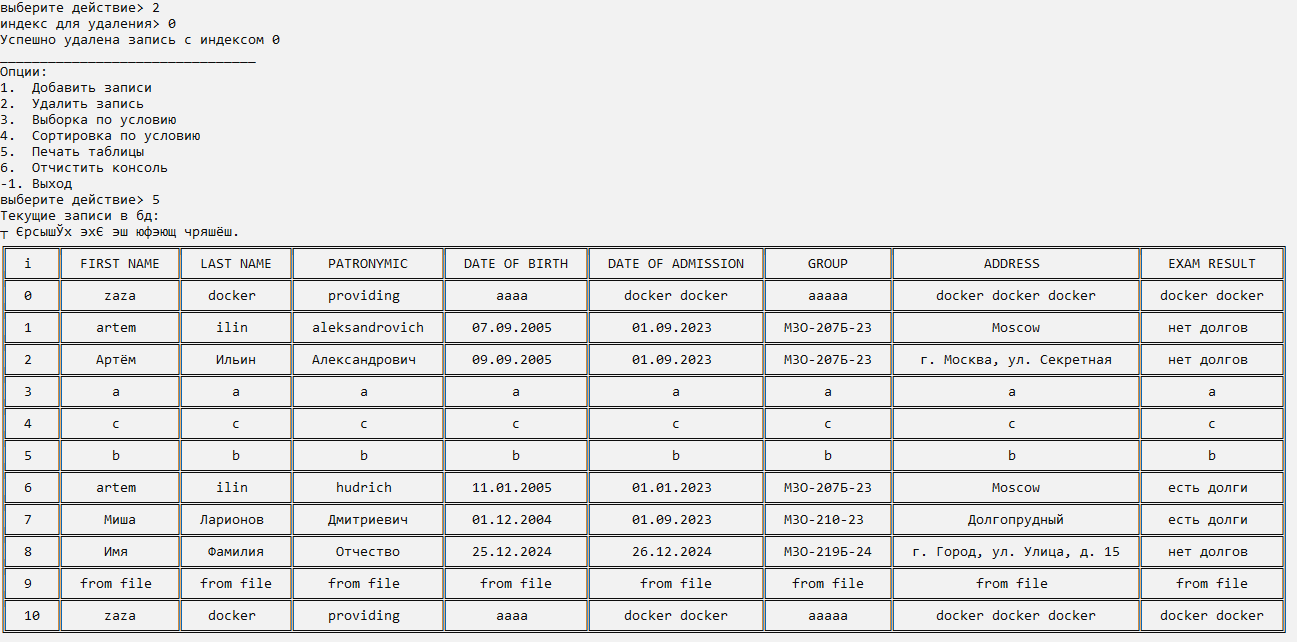
Все операции выполнены последовательно, для просмотра предыдущего состояния таблицы необходимо просмотреть предыдущий снимок экрана.



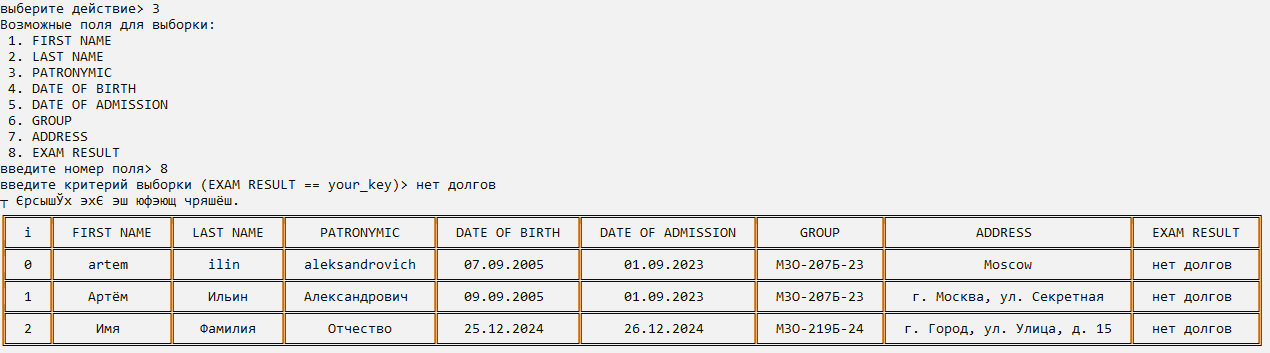
Добавление новой записи с помощью ручного ввода



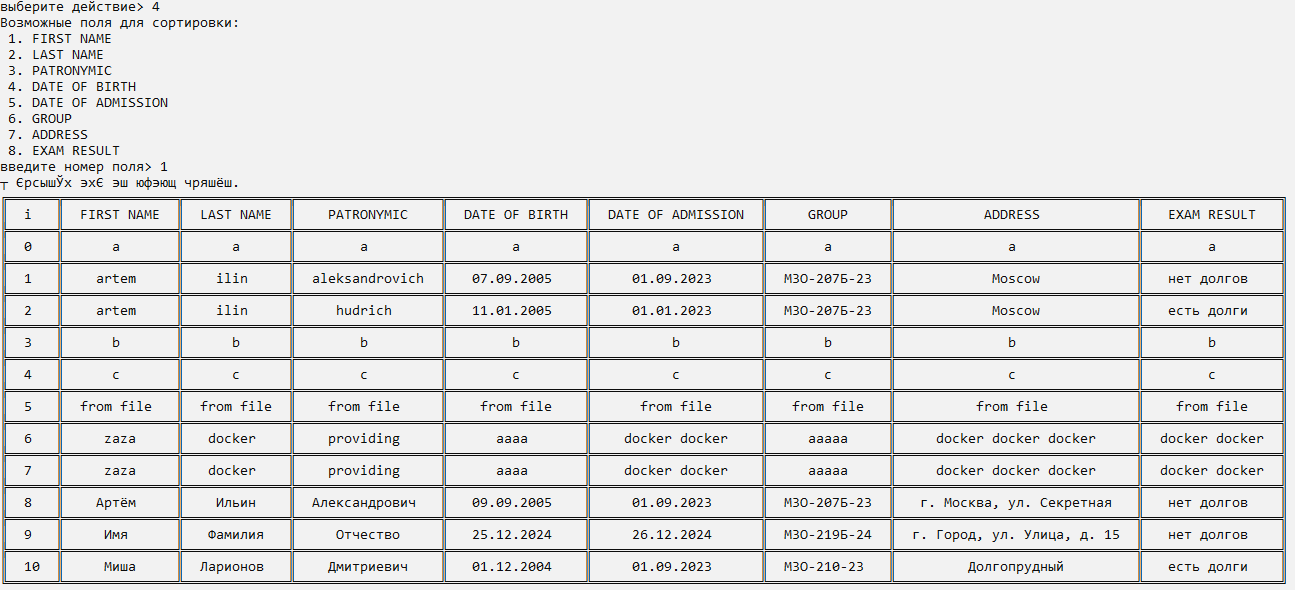
Добавление записей из файла



Результат удаления записи с индексом 0



Результат выборки записей по столбцу EXAM RESULT и ключу “нет долгов”



Результат сортировки по столбцу FIRST NAME

## Источники

1. Материалы лекций
2. <https://ru.wikipedia.org/wiki/Стек>
3. <https://en.wikipedia.org/wiki/Doubly_linked_list>

## Заключение

В данной курсовой работе было изучено применение алгоритмов Quick Sort для сортировки двусвязного списка. Была изучена структура данных «Стек» и способы взаимодействия с ней.