Министерство образования и науки Российской Федерации

1. Санкт-Петербургский Политехнический Университет Петра Великого
2. —
3. Институт компьютерных наук и кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «**База данных**»
2. по дисциплине «Структуры данных»
3. Выполнил
4. студент гр. 5151001/40001 Кириллов Д.А.

<*подпись*>

1. Преподаватель
2. асс. преподавателя Семьянов П.В.

<*подпись*>

1. Санкт-Петербург
2. 2025

# Цель работы

Написать базу данных и научиться работать со всеми типами данных в языке С.

# Постановка задачи

1. Придумать, какие структуры данных за что будут отвечать.
2. Реализовать хранение в памяти.
3. Написать функции вставки, удаления, поиска, чтения из файла и добавление в файл.

# идея базы данных

В программе реализована база данных студентов университета. Она хранит id, ФИО, Специальность, степень образования, курс/научного руководителя (зависит от степени обучения) и средний балл студентов. Хранение в памяти будет осуществляться с помощью AVL-дерева, т.к основные функции работы с деревом осуществляются за логарифмическое время.

# Используемые структуры

1. Стандартные структуры используются в поле студента.

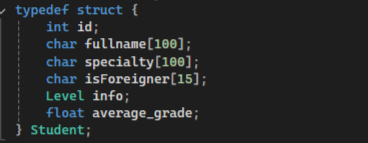


Рисунок 1-поле студента

1. Перечисления используются для более понятной записи выбора действий пользователя.

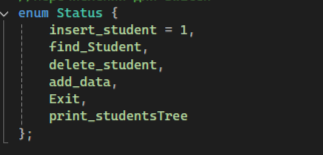


Рисунок 2 – действия пользователя

1. Объединения используются для записи курса, если студент получает ступень бакалавра/магистра, либо научного руководителя, если ступень аспиранта.

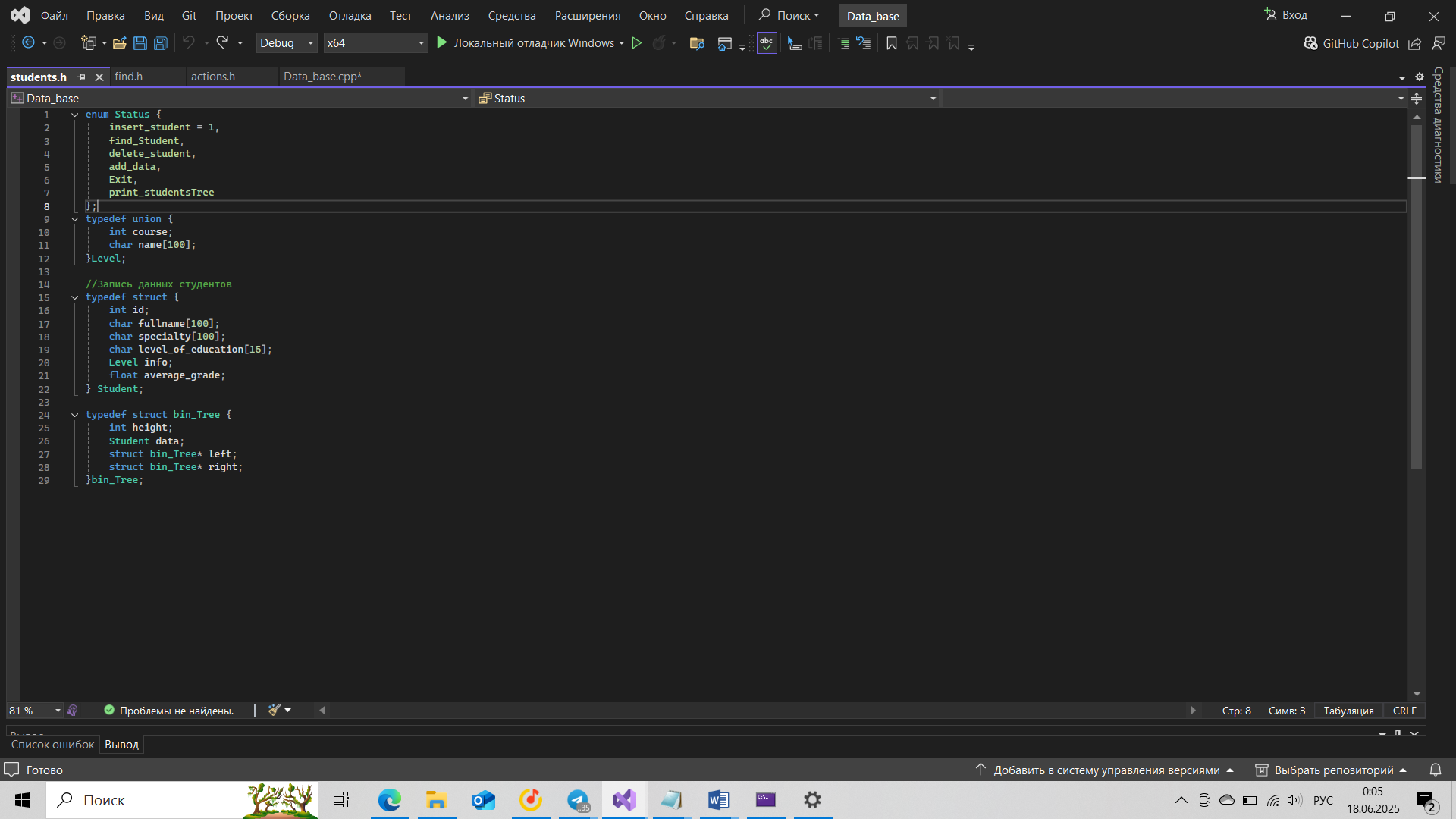


Рисунок 3 – объединения

1. Записи используются для хранения поля данных о пользователе.
2. Массивы используются для хранения символов char.
3. Динамические массивы используются при создании новых узлов в дереве.

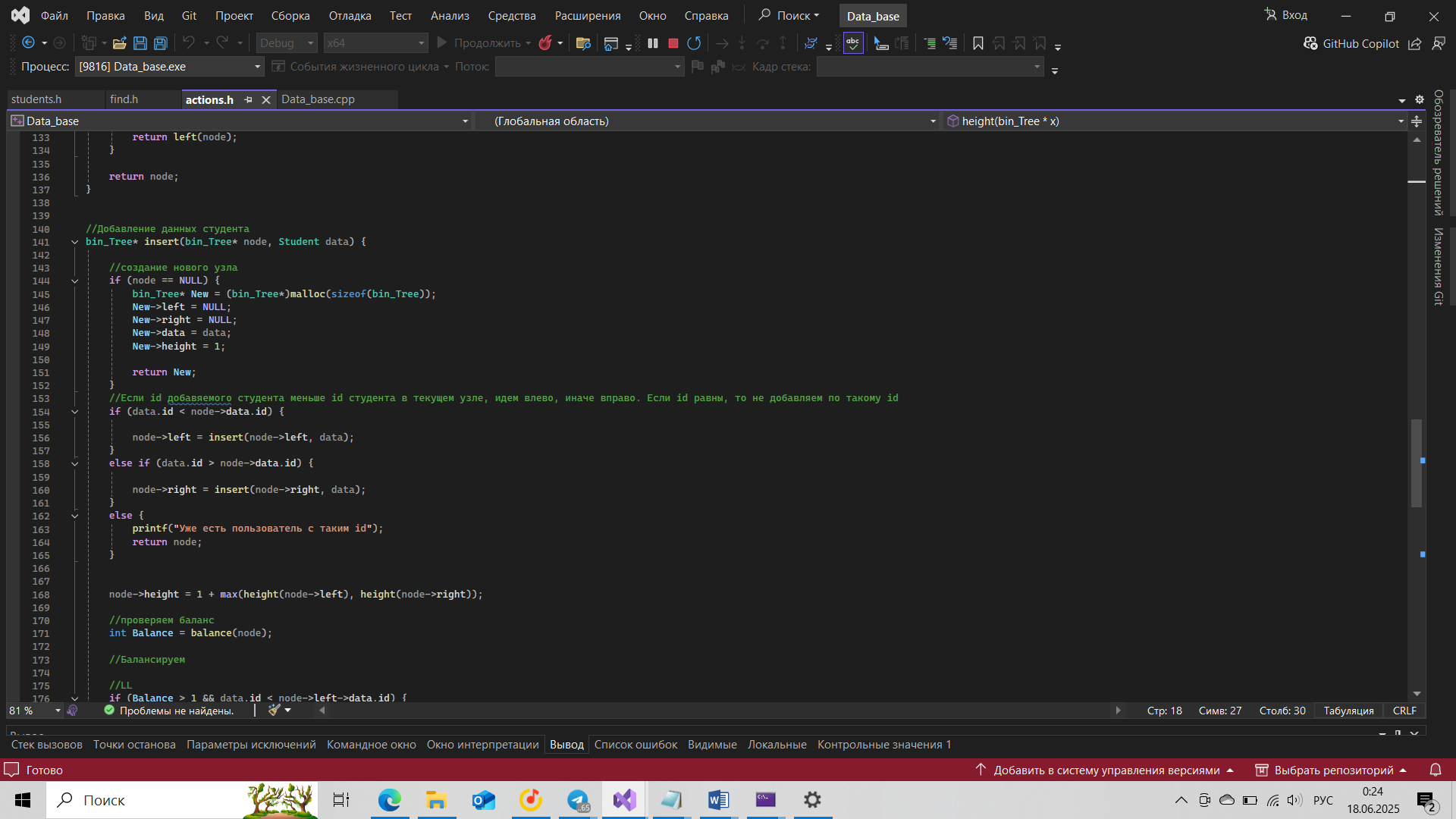
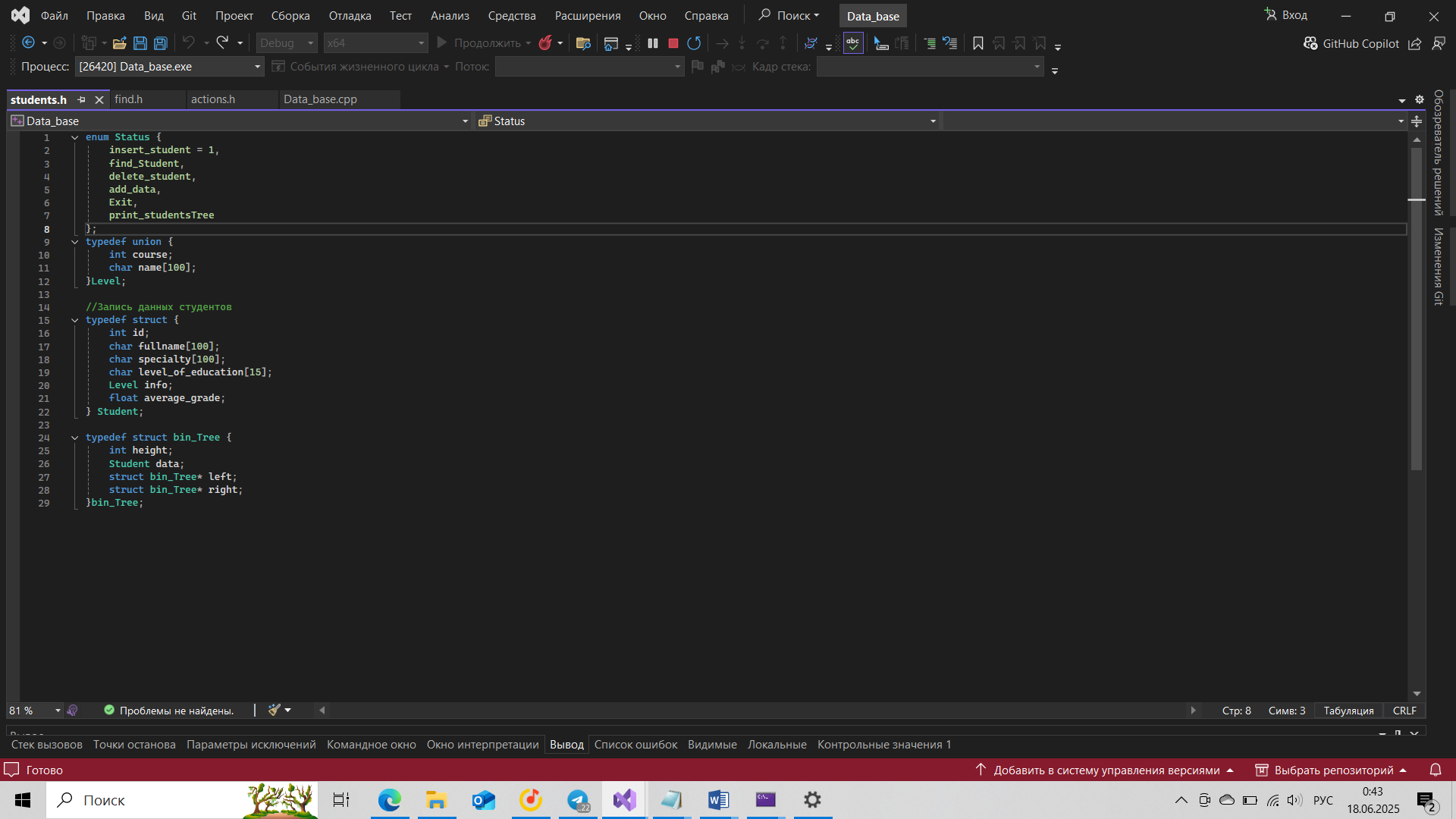


Рисунок 4 – создание узла с использованием динамического массива.

1. Рекурсивный тип данных реализован в виде AVL-дерева.



# Описание реализации

Сначала программа считывает представленный файл со студентами. Потом для пользователя открывается окно действий.

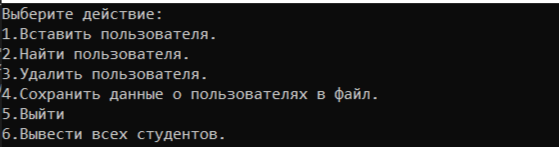


Рисунок 5 – окно действий.

1. При вставке пользователя нужно ввести все его данные (id, ФИО, Специальность, статус (местный/иностранец), родную страну/город и средний балл).

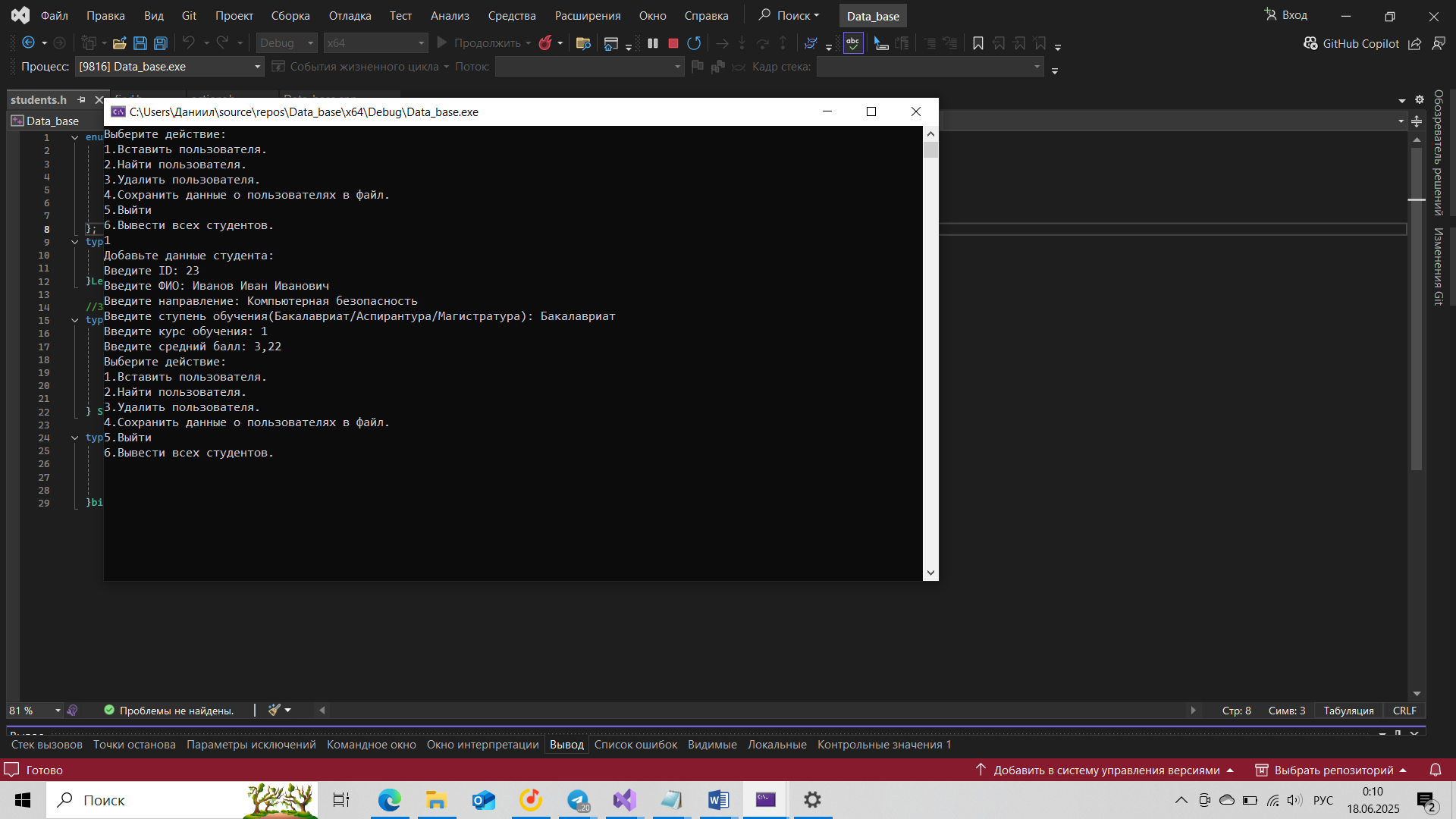


Рисунок 6 – добавление пользователя.

1. При поиске студента нужно ввести его id и после программа рекурсивно будет идти по дереву, сравнивая введенное id с id в правой и левой ветви и после выводить данные о нем.

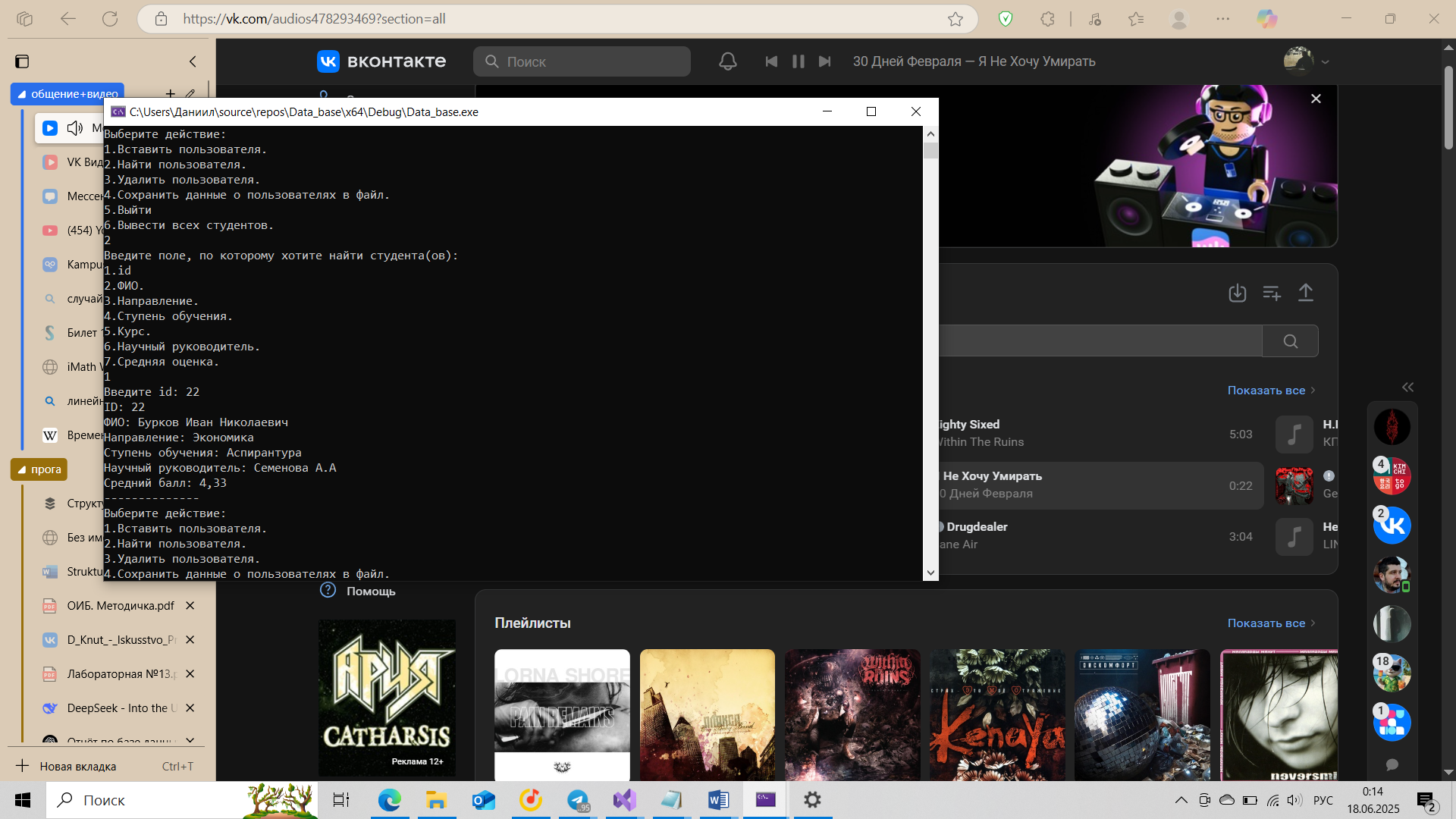


Рисунок 7 – поиск студента.

1. При удалении также надо ввести id пользователя и после в функции сначала будет искаться пользователь, потом будет происходить его удаление.

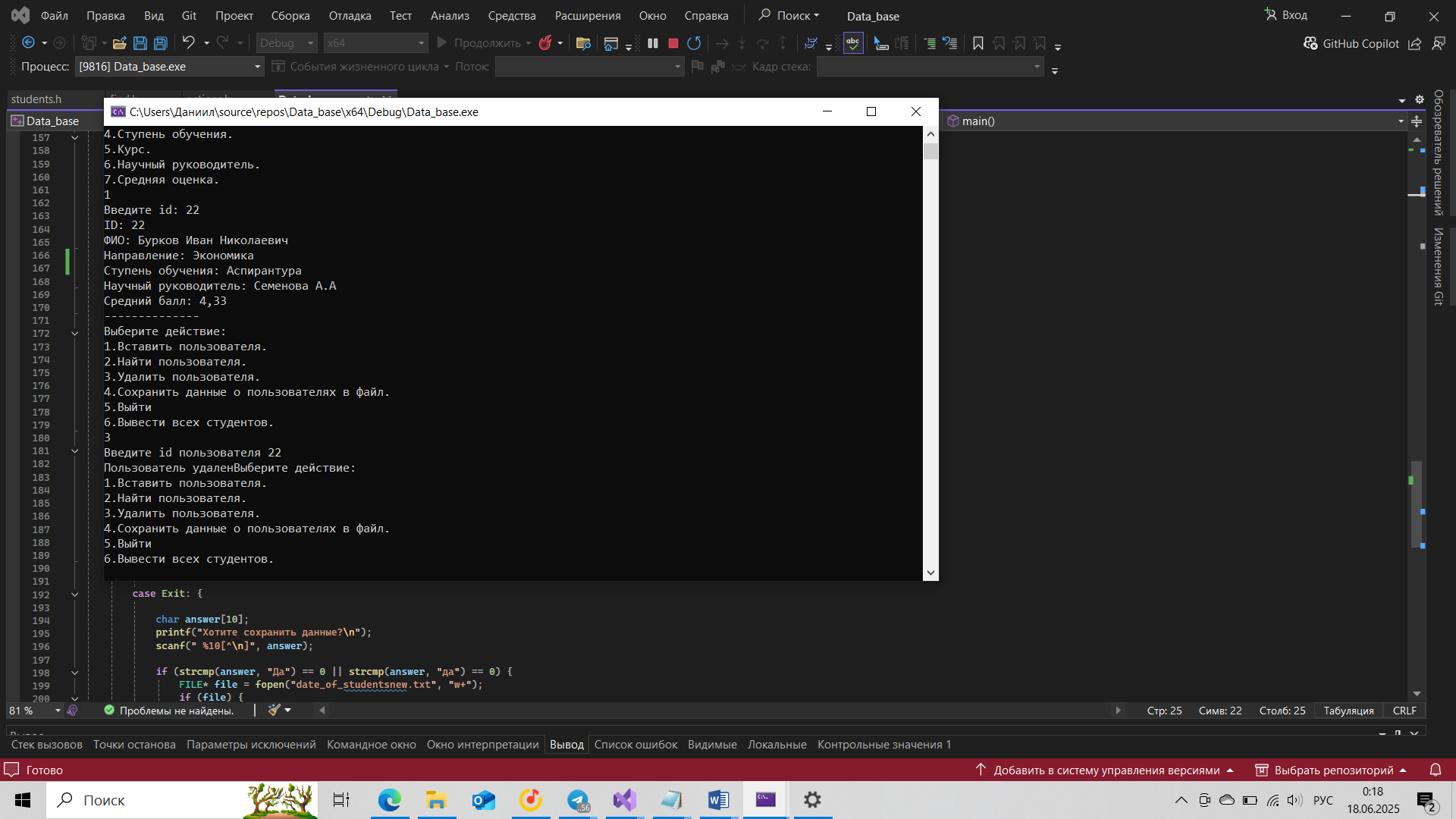


Рисунок 8 – удаление пользователя.

1. При сохранении мы будем рекурсивно сохранять данные о пользователях в файл.

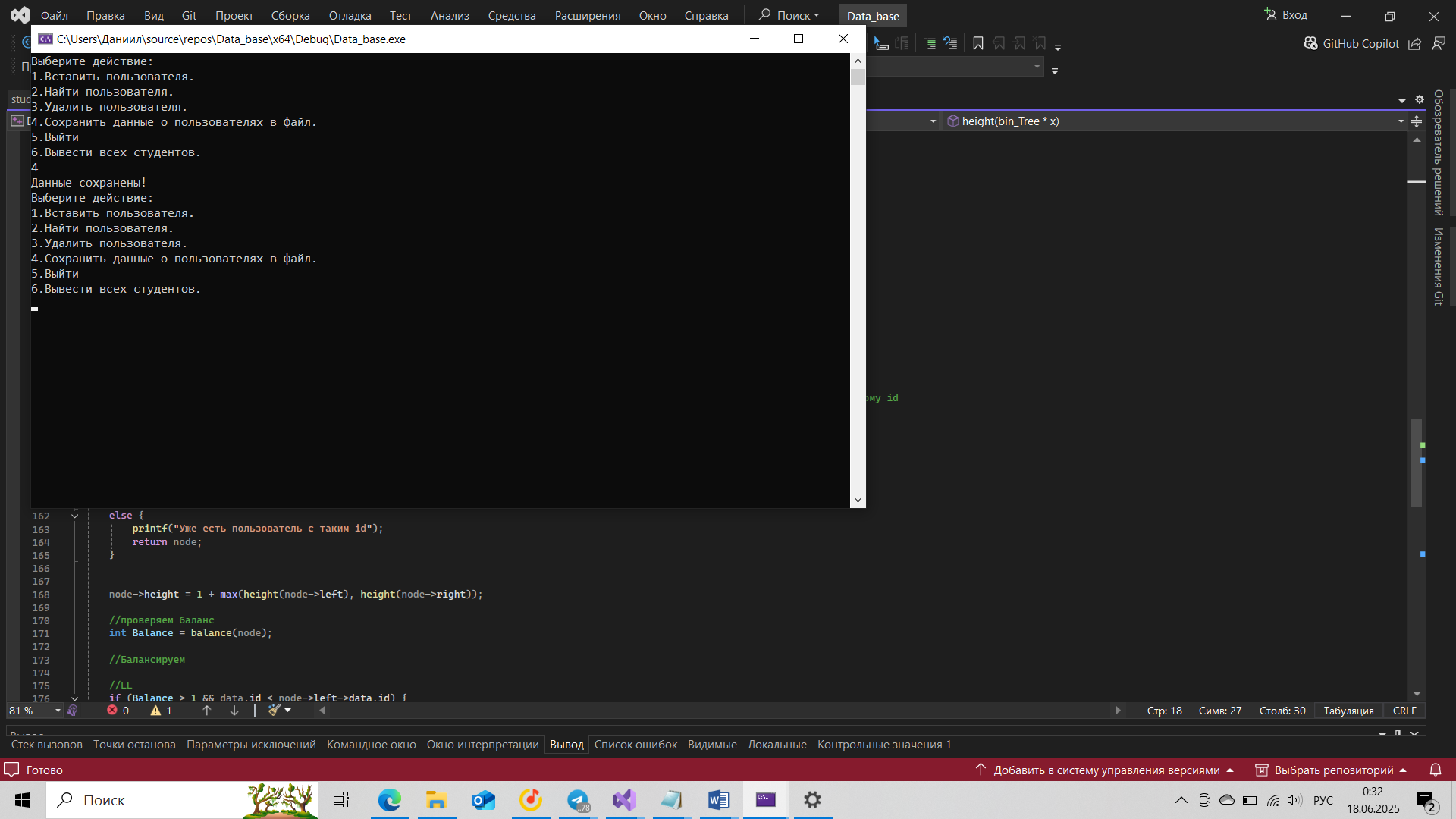


Рисунок 9 – сохранение изменений в файл.

1. При выходе будет выбор сохранять файл или нет.

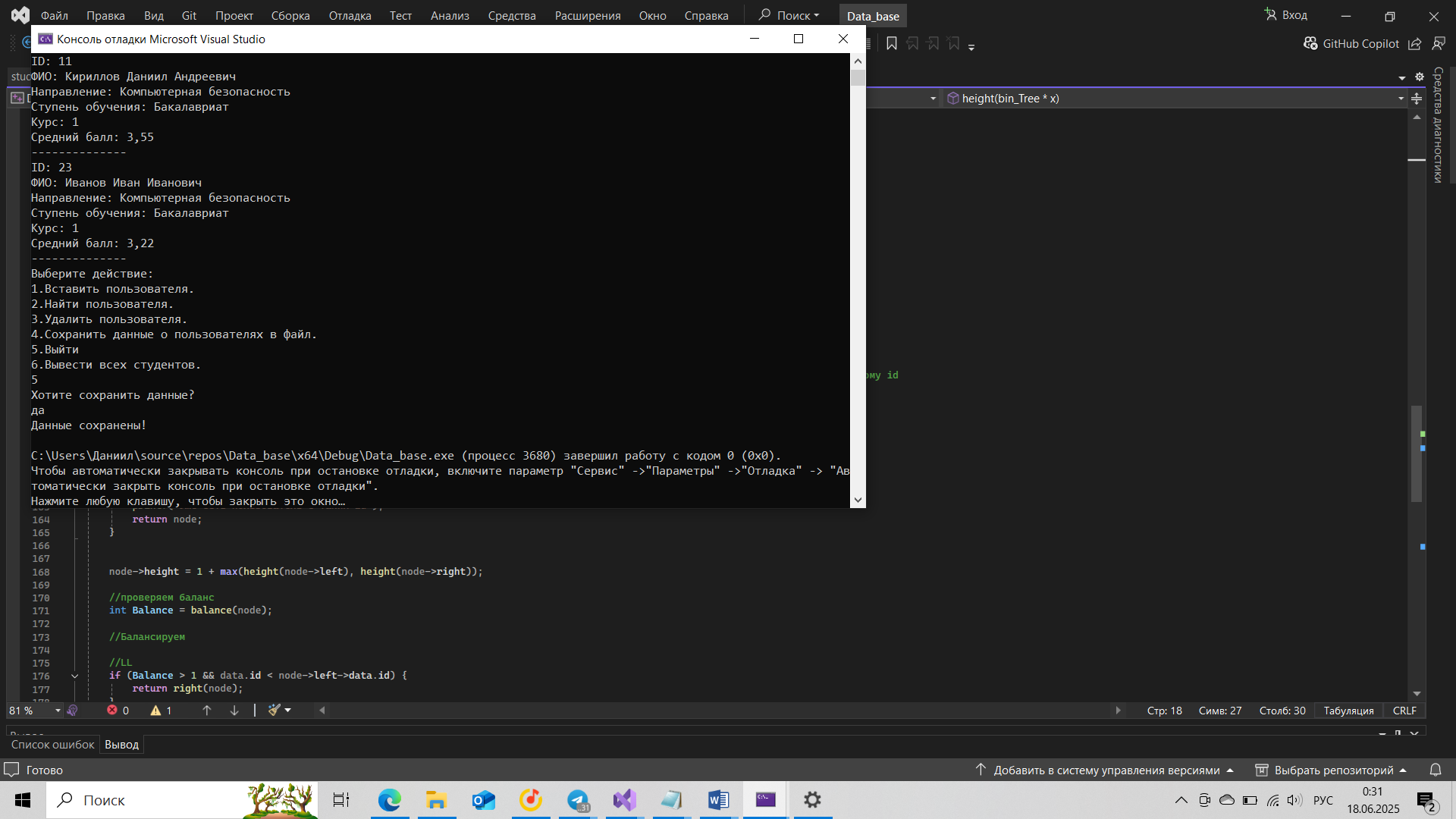


Рисунок 10 – выбор при выходе.

1. Также реализован вывод всех студентов.

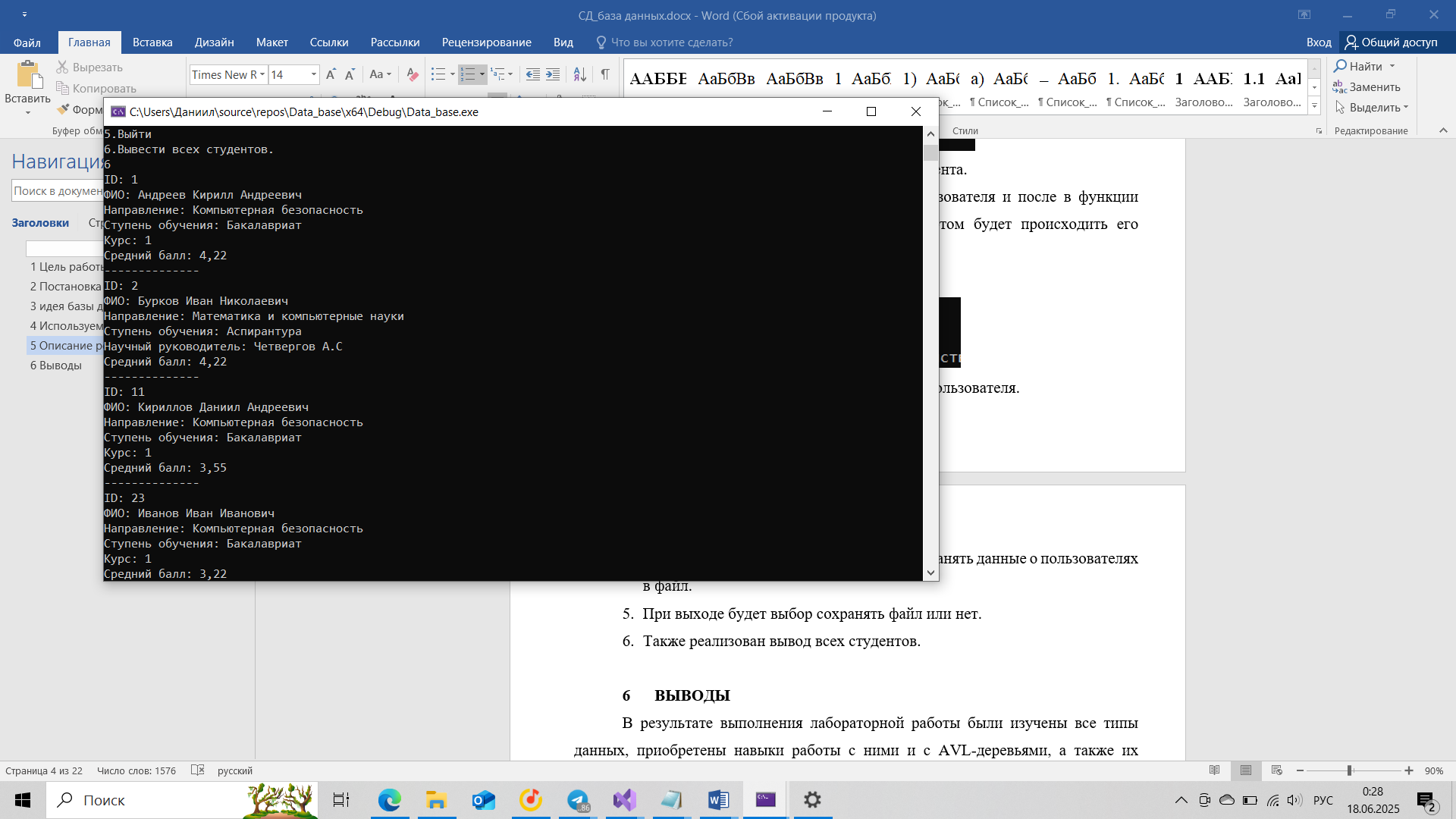


Рисунок 11 – вывод всех пользователей.

# Выводы

В результате выполнения лабораторной работы были изучены все типы данных, приобретены навыки работы с ними и с AVL-деревьями, а также их балансировкой.

**Приложение**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <string.h>

#include <Windows.h>

#include <stdio.h>

#include <locale.h>

//Перечисления для switch

enum Status {

insert\_student = 1,

find\_Student,

delete\_student,

add\_data,

Exit,

print\_studentsTree

};

typedef union {

char country[20];

char city[20];

}Level;

//Запись данных студентов

typedef struct {

int id;

char fullname[100];

char specialty[100];

char isForeigner[15];

Level info;

float average\_grade;

} Student;

//Инициализация дерева для хранения данных о студенте по id

typedef struct bin\_Tree {

int height;

Student data;

struct bin\_Tree\* left;

struct bin\_Tree\* right;

}bin\_Tree;

//выводит высоту дерева, либо 0, если узел не заполнен

int height(bin\_Tree\* x) {

return x ? x->height : 0;

}

//находит нарушен ли баланс (H\_left-H\_right)

int balance(bin\_Tree\* x) {

if (!x) { return 0; }

return height(x->left) - height(x->right);

}

//Левый поворот

bin\_Tree\* left(bin\_Tree\* B) {

if (B == NULL || B->right == NULL) {

return B; // ничего не поворачиваем

}

bin\_Tree\* A = B->right;

bin\_Tree\* temp1 = A->left;

A->left = B;

B->right = temp1;

A->height = max(height(A->left), height(A->right)) + 1;

B->height = max(height(B->right), height(B->left)) + 1;

return A;

}

//Правый поворот

bin\_Tree\* right(bin\_Tree\* B) {

bin\_Tree\* A = B->left;

bin\_Tree\* temp2 = A->right;

A->right = B;

B->left = temp2;

A->height = max(height(A->left), height(A->right)) + 1;

B->height = max(height(B->right), height(B->left)) + 1;

return A;

}

//Функция удаления студента по id

bin\_Tree\* del\_student(bin\_Tree\* root, int id) {

if (root == NULL) {

printf("Студент с ID %d не найден\n", id);

return NULL;

}

//Поиск узла(если id меньше корня - идем влево, иначе вправо)

if (id < root->data.id) {

root->left = del\_student(root->left, id);

}

else if (id > root->data.id) {

root->right = del\_student(root->right, id);

}

else {

//После нахождения узла рассматриваем 3 случая: нет потомков, 1 потомок, 2 потомка

if (root->left == NULL || root->right == NULL) {

bin\_Tree\* temp = root->left ? root->left : root->right;

// Нет потомков

if (temp == NULL) {

temp = root;

root = NULL;

}

// Один потомок

else {

\*root = \*temp; // Копируем содержимое потомка

}

free(temp);

}

else {

//два потомка

// Ищем минимальный узел

bin\_Tree\* temp = root->right;

while (temp && temp->left != NULL) {

temp = temp->left;

}

// Копируем данные из минимального узла в текущий узел

root->data = temp->data;

// Удаляем минимальный узел

root->right = del\_student(root->right, temp->data.id);

}

}

// Если дерево стало пустым

if (root == NULL) {

return NULL;

}

//Обновление высоты

root->height = 1 + max(height(root->left), height(root->right));

// Проверка баланса

int Balance = balance(root);

//Балансирование

// LL

if (Balance > 1 && balance(root->left) >= 0) {

return right(root);

}

// LR

if (Balance > 1 && balance(root->left) < 0) {

root->left = left(root->left);

return right(root);

}

// RR

if (Balance < -1 && balance(root->right) <= 0) {

return left(root);

}

// RL

if (Balance < -1 && balance(root->right) > 0) {

root->right = right(root->right);

return left(root);

}

return root;

}

//Добавление данных студента

bin\_Tree\* insert(bin\_Tree\* root, Student data) {

//создание нового узла

if (root == NULL) {

bin\_Tree\* New = (bin\_Tree\*)malloc(sizeof(bin\_Tree));

New->left = NULL;

New->right = NULL;

New->data = data;

New->height = 1;

return New;

}

//Если id добавяемого студента меньше id студента в текущем узле, идем влево, иначе вправо. Если id равны, то не добавляем по такому id

if (data.id < root->data.id) {

root->left = insert(root->left, data);

}

else if (data.id > root->data.id) {

root->right = insert(root->right, data);

}

else {

return root; // Если ID уже есть, то не добавляем

}

//обновляем высоту узла

root->height = 1 + max(height(root->left), height(root->right));

//проверяем баланс

int Balance = balance(root);

//Балансируем

//LL

if (Balance > 1 && data.id < root->left->data.id) {

return right(root);

}

//RR

if (Balance <-1 && data.id > root->right->data.id) {

return left(root);

}

//LR

if (Balance > 1 && data.id > root->left->data.id) {

root->left = left(root->left);

return right(root);

}

//RL

if (Balance < -1 && data.id < root->right->data.id) {

root->right = right(root->right);

return left(root);

}

return root;

}

//Рекурсивная функция сохранения студентов в файл

void savestudent(FILE\* file, bin\_Tree\* root) {

//Если дошли до незаполненного узла, возвращаемся

if (root == NULL) return;

//Рекурсивно идем влево и когда начинаем возвращаться, записываем студента

savestudent(file, root->left);

// Запись данных студента

fprintf(file, "%d,%s,%s,%s,", root->data.id, root->data.fullname, root->data.specialty,root->data.isForeigner);

// Запись данных в зависимости от ступени обучения

if (strcmp(root->data.isForeigner, "Иностранец") == 0) {

fprintf(file, "%s,%.2f\n", root->data.info.country, root->data.average\_grade);

}

else {

fprintf(file, "%s,%.2f\n", root->data.info.city, root->data.average\_grade);

}

//Рекурсивно идем в правую ветвь

savestudent(file, root->right);

}

//Функция вывода данных студента на экран

void print\_Student(Student s) {

printf("ID: %d\n", s.id);

printf("ФИО: %s\n", s.fullname);

printf("Направление: %s\n", s.specialty);

printf("Статус: %s\n", s.isForeigner);

if (strcmp(s.isForeigner, "Иностранец") == 0)

printf("Страна: %s\n", s.info.country);

else

printf("Родной город: %s\n", s.info.city);

printf("Средний балл: %.2f\n", s.average\_grade);

printf("--------------\n");

}

//Функция вывода дерева на экран

void printTree(bin\_Tree\* root) {

if (root == NULL)

return;

printTree(root->left);

print\_Student(root->data);

printTree(root->right);

}

//освобождаем память

void free\_tree(bin\_Tree\* root) {

if (root) {

free\_tree(root->left);

free\_tree(root->right);

free(root);

}

}

//Функция рекурсивного поиска студента в дереве по id

bin\_Tree\* find\_student(bin\_Tree\* x, int id) {

if (x == NULL) {

printf("\nПользователь не найден\n\n");

return NULL;

}

if (x->data.id == id) {

print\_Student(x->data);

}

else if (x->data.id > id) {

return find\_student(x->left, id);

}

else if (x->data.id < id) {

return find\_student(x->right, id);

}

}

int main() {

// Чтоб работали русские символы

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Инициализация дерева студентов

bin\_Tree\* root = NULL;

// Открываем файл для чтения данных

FILE\* file = fopen("date\_of\_students.txt", "r");

Student s;

char info\_time[100];

char line[500];

while (fgets(line, sizeof(line), file)) {

int result = sscanf(line, "%d,%99[^,],%99[^,],%15[^,],%99[^,],%f",

&s.id, s.fullname, s.specialty, s.isForeigner, info\_time, &s.average\_grade);

if (strcmp(s.isForeigner, "Иностранец") == 0) {

strncpy(s.info.country, info\_time, sizeof(s.info.country));

}

else {

strncpy(s.info.city, info\_time, sizeof(s.info.city));

}

root = insert(root, s);

}

fclose(file);

bool A = true;

while (A) {

printf("Выберите действие:\n1.Вставить пользователя.\n2.Найти пользователя.\n3.Удалить пользователя.\n4.Сохранить данные о пользователях в файл.\n5.Выйти\n6.Вывести всех студентов.\n");

int action;

scanf("%d", &action);

switch (action) {

case insert\_student:

{

Student s;

printf("Добавьте данные студента:\n");

printf("Введите ID: ");

scanf("%d", &s.id);

getchar();

printf("Введите ФИО: ");

fgets(s.fullname, sizeof(s.fullname), stdin);

s.fullname[strcspn(s.fullname, "\n")] = 0;

printf("Введите специализацию: ");

fgets(s.specialty, sizeof(s.specialty), stdin);

s.specialty[strcspn(s.specialty, "\n")] = 0;

printf("Введите статус (Иностранец или Местный): ");

fgets(s.isForeigner, sizeof(s.isForeigner), stdin);

s.isForeigner[strcspn(s.isForeigner, "\n")] = 0;

if (strcmp(s.isForeigner, "Иностранец") == 0) {

printf("Введите страну: ");

fgets(s.info.country, sizeof(s.info.country), stdin);

s.info.country[strcspn(s.info.country, "\n")] = 0;

}

else {

printf("Введите родной город: ");

fgets(s.info.city, sizeof(s.info.city), stdin);

s.info.city[strcspn(s.info.city, "\n")] = 0;

}

printf("Введите средний балл: ");

scanf("%f", &s.average\_grade);

getchar();

root = insert(root, s);

printf("Пользователь добавлен!\n");

}

break;

case find\_Student:

{

int id;

printf("Введите id пользователя ");

scanf("%d", &id);

find\_student(root, id);

}

break;

case delete\_student:

{

int ID;

printf("Введите id пользователя ");

scanf("%d", &ID);

root = del\_student(root, ID);

}

break;

case add\_data: {

FILE\* file = fopen("date\_of\_students.txt", "w+");

if (file) {

savestudent(file, root);

fclose(file);

printf("Данные сохранены!\n");

}

else {

printf("Ошибка сохранения данных!\n");

}

}

break;

case Exit: {

char answer[10];

printf("Хотите сохранить данные?\n");

scanf(" %10[^\n]", answer);

if (strcmp(answer, "Да") == 0 || strcmp(answer, "да") == 0) {

FILE\* file = fopen("date\_of\_students.txt", "w+");

if (file) {

savestudent(file, root);

fclose(file);

printf("Данные сохранены!\n");

}

else {

printf("Ошибка сохранения данных!\n");

}

}

A = false;

break;

}

case print\_studentsTree:

printTree(root);

break;

default:

printf("Неверный выбор!\n");

break;

}

}

free\_tree(root);

return 0;

}