**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №2

# «Разработка класса «Планета» и создание класса по варианту»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-21Б |  | преподаватель каф. ИУ5 |
| Анцифров Никита |  |  |
|  |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

**Постановка задачи**

1. Ознакомиться с материалом, изложенным в описании лабораторной работы
2. Создать класс «Планета» для планет солнечной системы, используя 4 характеристики планет разного типа
3. Создать много-файловый проект и отладить программу, которая создает один объект класса «Планета» и выводит значения его полей на экран.
4. Организовать интерфейс пользователя с программой в виде меню, позволяющего выполнять чтение БД «Солнечная система» из файла, запись БД «Солнечная система» в файл, сортировку БД, редактирование БД и вывод БД на экран
5. Создать текстовый файл с данными о планетах солнечной системы и сохранить его в папке проекта
6. Перегрузить конструктор копирования, деструктор и оператор присваивания.
7. Вставьте в конструкторы и деструктор печать типа «Создание (Удаление) ID n » , где n - номер объекта, для которого они вызываются.
8. Перегрузить операцию “ >> “ для класса «Планета» и ifstream и прочитать данные о планетах из файла в массив «Солнечная система» из объектов класса «Планета».
9. Перегрузить операцию “ << “ для классов «Планета» и ofstream и вывести на экран данные из массива.
10. Перегрузить операции сравнения “ < и ==” для класса «Планета», использовав для этого значение одного из полей.
11. Отсортировать массив планет солнечной системы, хранящийся в файле, с использованием перегруженных операций.
12. На основе разработанного класса «Планета» выполнить задание по варианту – разработать класс для анкеты для опроса населения (Вариант 4)

База данных, созданная из данных из анкеты для опроса населения о качестве услуг связи, состоит из ответов каждого человека на следующие вопросы:

1) Имя человека

2) Его возраст

3) Его пол

4) Оператор связи, которым он пользуется

5) Его оценка качества услуг оператора (положительная – 1, отрицательная – 0)

6) Его желание изменить оператора связи (желает – 1, не желает – 0)

**Описание входных, выходных и вспомогательных данных**

Для проекта с классом «Планета»:

**Planet** solar\_system\_planets [15] – массив из объектов класса для хранения данных о планетах солнечной системы

**char\*** filename – переменная для хранения имени файла, с которым осуществляется работа

**int** how\_many\_lines – переменная для общего количества сток в файле

**int** longest\_name – переменная для хранения количества символов самого длинного имени

**int** i, j, k – счетчики циклов

**Planet** test\_planet – объект класса для демонстрации отображения сведений в конструкторе

**int** num – вспомогательная переменная для числа символов test\_planet

**int** answ – переменная для хранения цифры ответа пользователя

**ifstream** fin – переменная для открытия файла для чтения

**char\*** current\_name – переменная для хранения названия планеты

**long** int current\_size – переменная для хранения размера планеты

**bool** current\_life – переменная для хранения показателя наличия жизни для планеты

**int** current\_sats – переменная для хранения числа спутников планеты

**ofstream** fout – переменная для открытия файла для записи в него информации

**int** column – переменная для хранения номера столбца для сортировки

**int** method – переменная для хранения способа сортировки столбца

**int** num\_line – переменная для хранения номера строки для редактирования

**int** num\_what – переменная для хранения номера ячейки для редактирования

**Planet\*** p1, p2 – временные указатели для функции swap ()

Для проекта с классом «Анкета»:

**unit** cell [15] – массив из объектов класса для хранения данных о человеке

**char\*** filename – переменная для хранения имени файла, с которым осуществляется работа

**int** how\_many\_lines – переменная для общего количества сток в файле

**int** longest\_name – переменная для хранения количества символов самого длинного имени

**int** longest\_provider – переменная для хранения количества символов самого длинного названия оператора связи

**int** temp1, temp2 – вспомогательные переменные

**int** i, j, k – счетчики циклов

**int** answ – переменная для хранения цифры ответа пользователя

**ifstream** fin – переменная для открытия файла для чтения

**ofstream** fout – переменная для открытия файла для записи в него информации

**int** column – переменная для хранения номера столбца для сортировки

**int** method – переменная для хранения способа сортировки столбца

**char\*** another\_name – переменная для хранения имени человека

**int** another\_age – переменная для хранения возраста человека

**char** another\_gend – переменная для хранения пола человека

**char\*** another\_prov – переменная для хранения названия оператора

**bool** another\_q – переменная для хранения показателя оценки качества

**bool** another\_wc – переменная для хранения показателя изменения оператора

**int** num\_line – переменная для хранения номера строки для редактирования

**int** num\_what – переменная для хранения номера ячейки для редактирования

**Описание прототипов функций**

Для проекта с классом «Планета»:

**int** menu(); - функция для работы меню

**void** read\_from\_file(char\*, Planet\*, int\*); - функция для чтения из файла без перегрузки операторов

**void** read\_from\_file\_2(char\*, Planet\*, int\*); - функция для чтения из файла с использованием перегрузки операторов

**void** write\_to\_file(char\*, Planet\*, int\*, int\*); - функция для записи в файл без перегрузки операторов

**void** write\_to\_file\_2(char\*, Planet\*, int\*, int\*); - функция для записи в файл с использованием перегрузки операторов

**void** sort\_menu(Planet\*, int\*); - функция для работы меню сортировки

**void** sort\_method(Planet\*, int\*, int); - функция для работы меню сортировки

**void** sort(Planet\*, int\*, int, int); - функция для сортировки с использованием методов класса

**void** sort\_2(Planet\*, int\*, int, int); - функция для сортировки с использованием перегрузки операторов

**void** edit\_menu(Planet\*, int\*); - функция для работы меню редактирования

**void** edit(Planet\*, int\*, int); - функция для осуществления редактирования

**void** longest\_name\_finder(Planet\*, int\*, int\*); - функция для поиска самого длинного имени

**void** swap(Planet\*, Planet\*); - функция для обмена ячейками (для сортировки)

Для проекта с классом «Анкета»:

**int** menu(); - функция для работы меню

**void** read\_from\_file(char\*, Planet\*, int\*); - функция для чтения из файла с использованием перегрузки операторов

**void** write\_to\_file(char\*, Planet\*, int\*, int\*); - функция для записи в файл с использованием перегрузки операторов

**void** sort\_menu(Planet\*, int\*); - функция для работы меню сортировки

**void** sort\_method(Planet\*, int\*, int); - функция для работы меню сортировки

**void** sort(Planet\*, int\*, int, int); - функция для сортировки с использованием перегрузки операторов

**void** edit\_menu(Planet\*, int\*); - функция для работы меню редактирования

**void** edit(Planet\*, int\*, int); - функция для осуществления редактирования

**void** swap(Planet\*, Planet\*); - функция для обмена ячейками (для сортировки)

**Разработка интерфейса класса**

Для проекта с классом «Планета»:

Класс:

**class** Planet – для хранения информации о планете

Поля класса:

**char\*** name – для хранения названия планеты

**long** **int** diameter – для хранения диаметра планеты

**bool** life – для хранения показателя наличия жизни на планете

**int** amount\_of\_sats – для хранения числа спутников планеты

**int** column – для хранения информации о столбце сортировки

Методы класса:

**char\*** get\_name(); – для получения имени планеты

**long** **int** get\_size(); – для получения размера планеты

**int** get\_amount\_of\_sats(); – для получения количества спутников

**bool** get\_life(); – для получения показателя жизни

**void** set\_name(char\*); – для установки нового имени (получает новое имя)

**void** set\_size(long int); – для установки нового размера (получает новый размер)

**void** set\_amount\_of\_sats(int); – для установки нового количества спутников (получает новое количество спутников)

**void** set\_life(bool); – для установки нового показателя жизни (получает новый показатель жизни)

**void** show(int\*); – для отображения на экране (получает значение самого длинного имени)

**Planet&** operator=(Planet&); – перегрузка оператора = (получает ссылку на объект класса)

**bool** operator==(Planet); – перегрузка оператора == (получает объект класса)

**bool** operator<(Planet); – перегрузка оператора < (получает объект класса)

**bool** operator>(Planet); – перегрузка оператора > (получает объект класса)

Дружественные функции:

friend **ifstream&** operator>> (ifstream&, Planet&); – перегрузка оператора >> (для чтения из файла)

friend **ofstream&** operator<< (ofstream&, Planet&); – перегрузка оператора << (для записи в файл)

friend **void** swap(Planet\*, Planet\*); – для обмена объектами класса

Для проекта с классом «Анкета»:

Класс:

**class** unit – для хранения информации об ответе человека на вопросы анкеты

Поля класса:

**char\*** name – для хранения имени человека

**int** age – для хранения возраста человека

**char** gend – для хранения пола человека

**char\*** provider – для хранения названия оператора связи

**bool** quality – для хранения оценки качества услуг

**bool** willchange – для хранения ответа на вопрос об изменении оператора

**int** column – для хранения информации о столбце сортировки

Методы класса:

**char\*** get\_name(); – для получения имени человека

**int** get\_age(); – для получения возраста человека

**char** get\_gend(); – для получения пола человека

**char\*** get\_provider(); – для получения названия оператора

**bool** get\_quality(); – для получения показателя оценки качества услуг

**bool** get\_willchange(); – для получения показателя желания изменить оператора

**void** set\_name(char\*); – для установки нового имени человека (получает новое имя)

**void** set\_age(int); – для установки нового возраста (получает новый возраст)

**void** set\_gend(char); – для установки нового пола (получает новый пол)

**void** set\_provider(char\*); – для установки нового названия оператора (получает название этого оператора)

**void** set\_quality(bool); – для установки нового показателя оценки качества услуг (получает новый показатель оценки качества услуг)

**void** set\_willchange(bool); – для установки нового показателя желания изменить оператора (получает новый показатель желания изменить оператора)

**void** show(int\*, int\*); – для отображения на экране (получает количество символов в самом длинном имени человека и названии оператора

**bool** operator==(unit); – перегрузка оператора == (получает объект класса)

**bool** operator<(unit); – перегрузка оператора < (получает объект класса)

**bool** operator>(unit); – перегрузка оператора > (получает объект класса)

Дружественные функции:

friend **ifstream&** operator>> (ifstream&, Planet&); – перегрузка >> (работа с файлом)

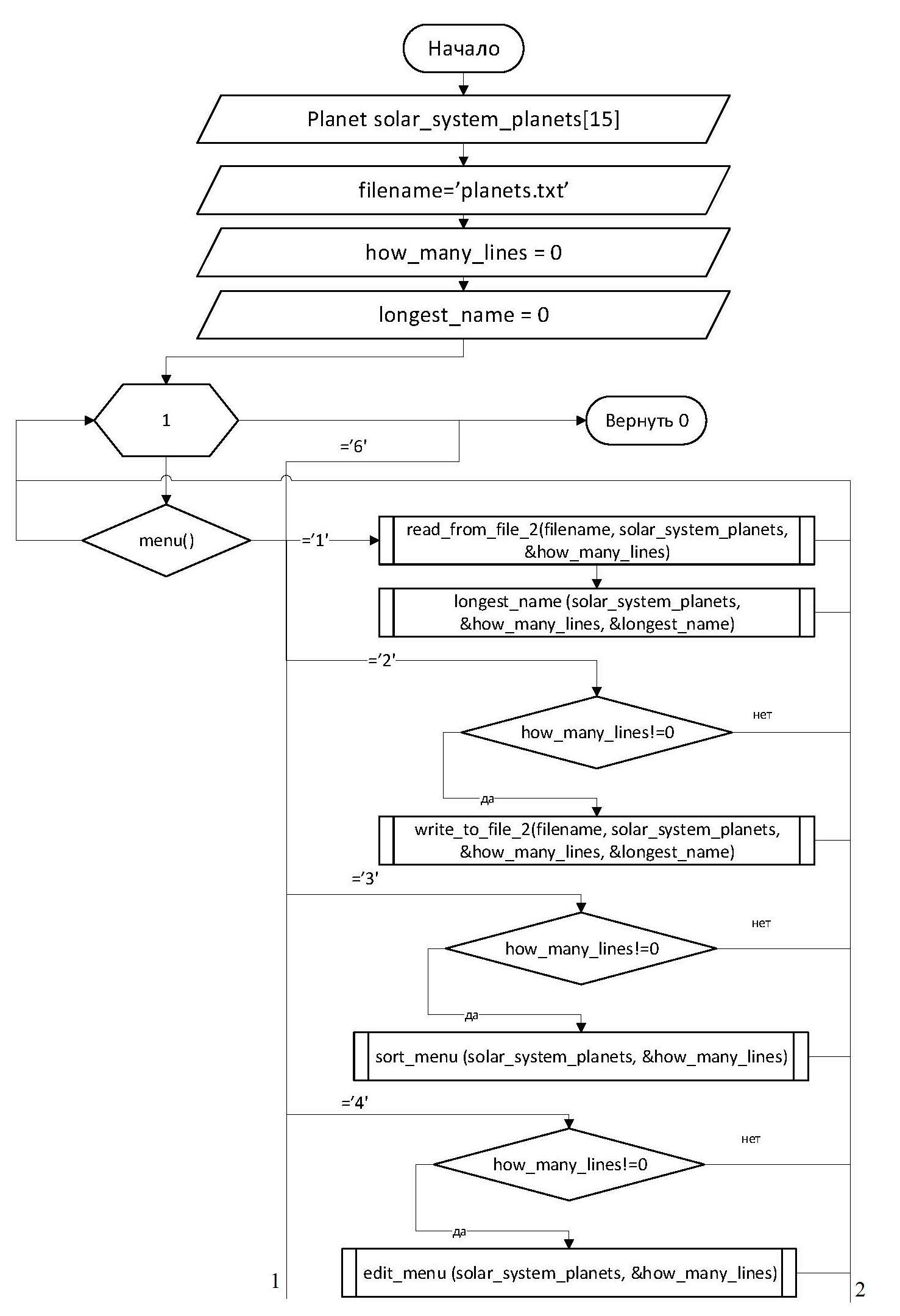
friend **ofstream&** operator<< (ofstream&, Planet&); – перегрузка << (работа с файлом)

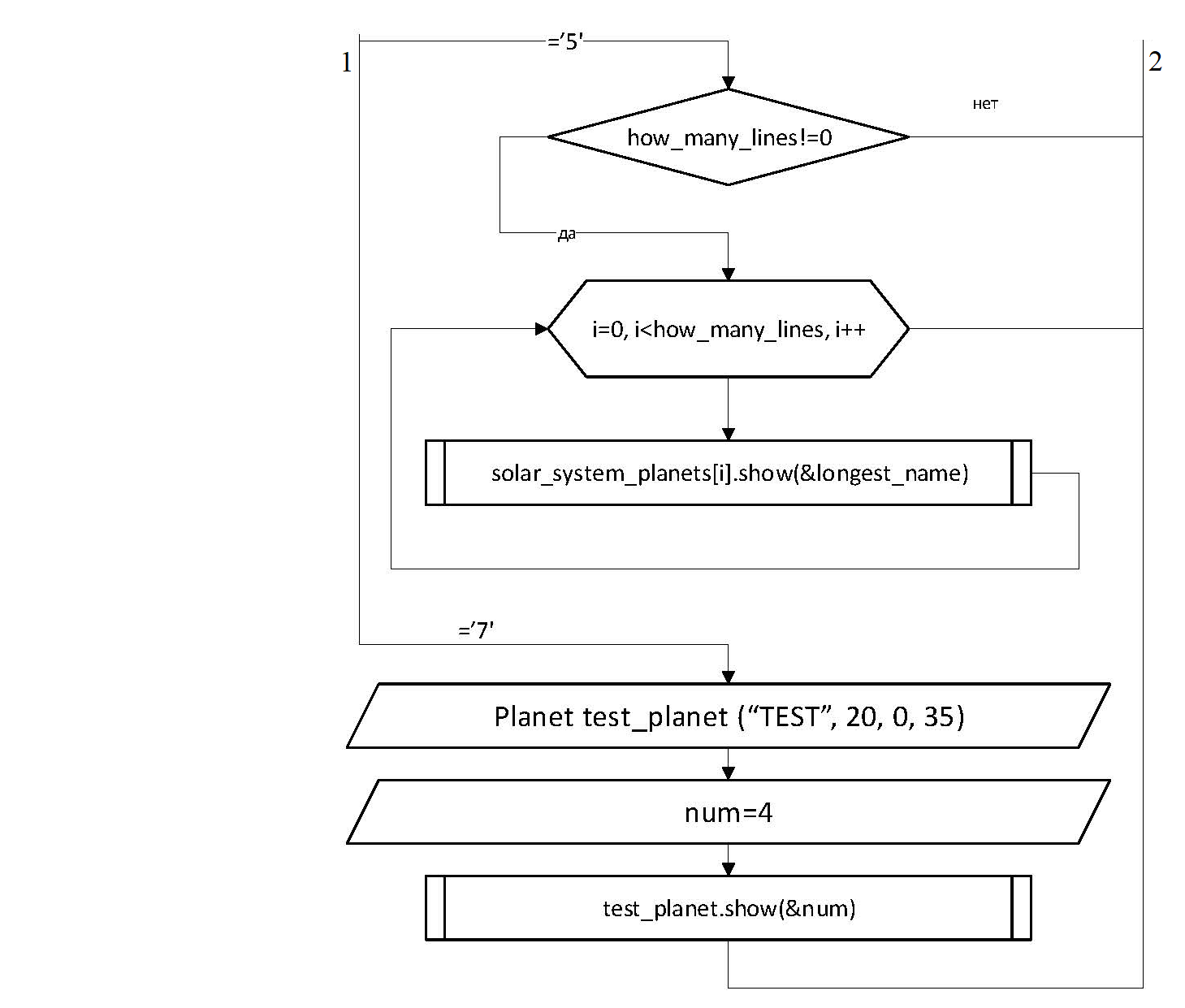
friend **void** swap(Planet\*, Planet\*); – для обмена объектами класса

**Алгоритм**

Для проекта с классом «Планета»

**main**

****

****

**menu**



**read\_from\_file**



**read\_from\_file\_2**



**write\_to\_file**



**write\_to\_file\_2**



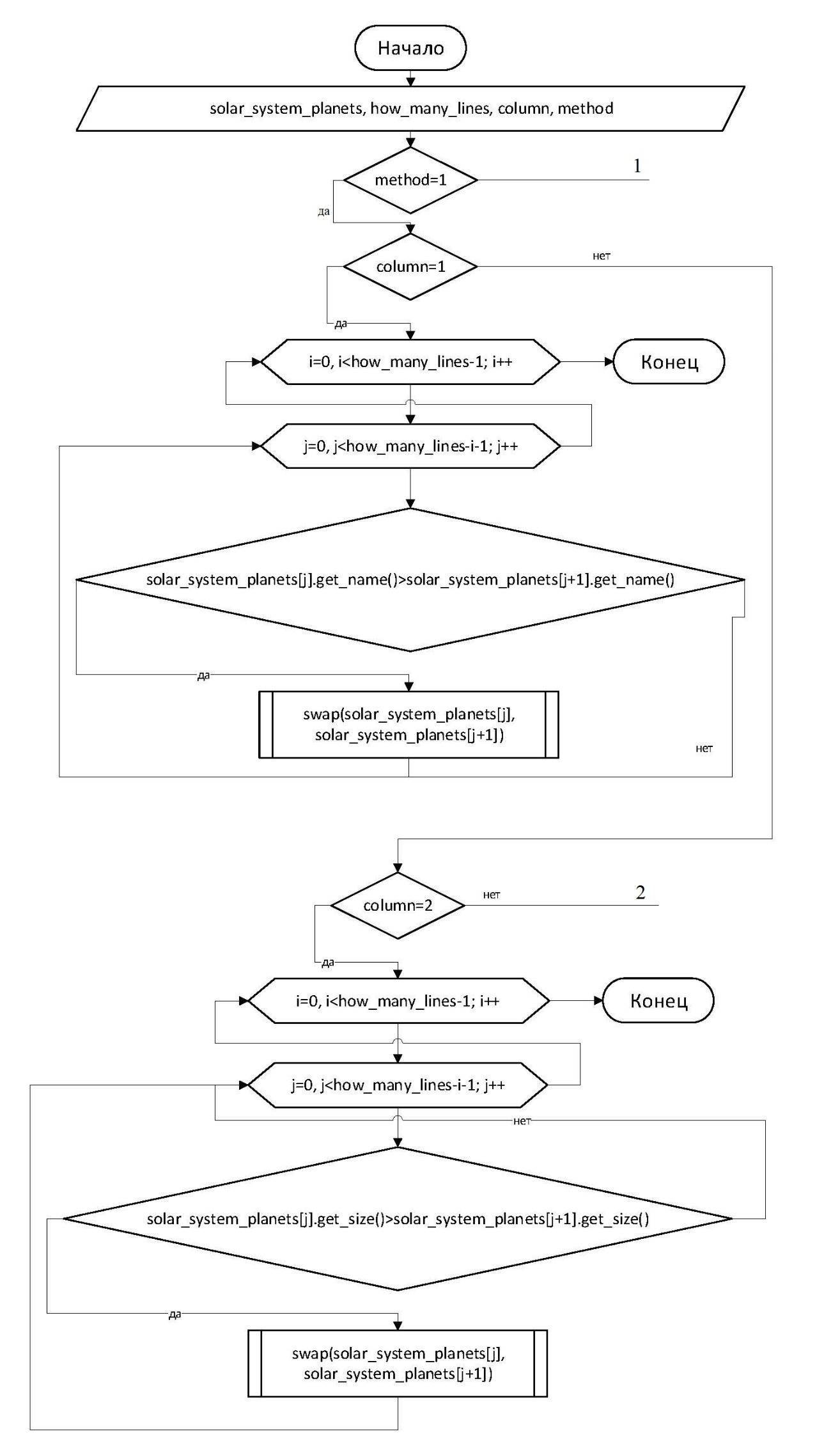
**sort\_menu**

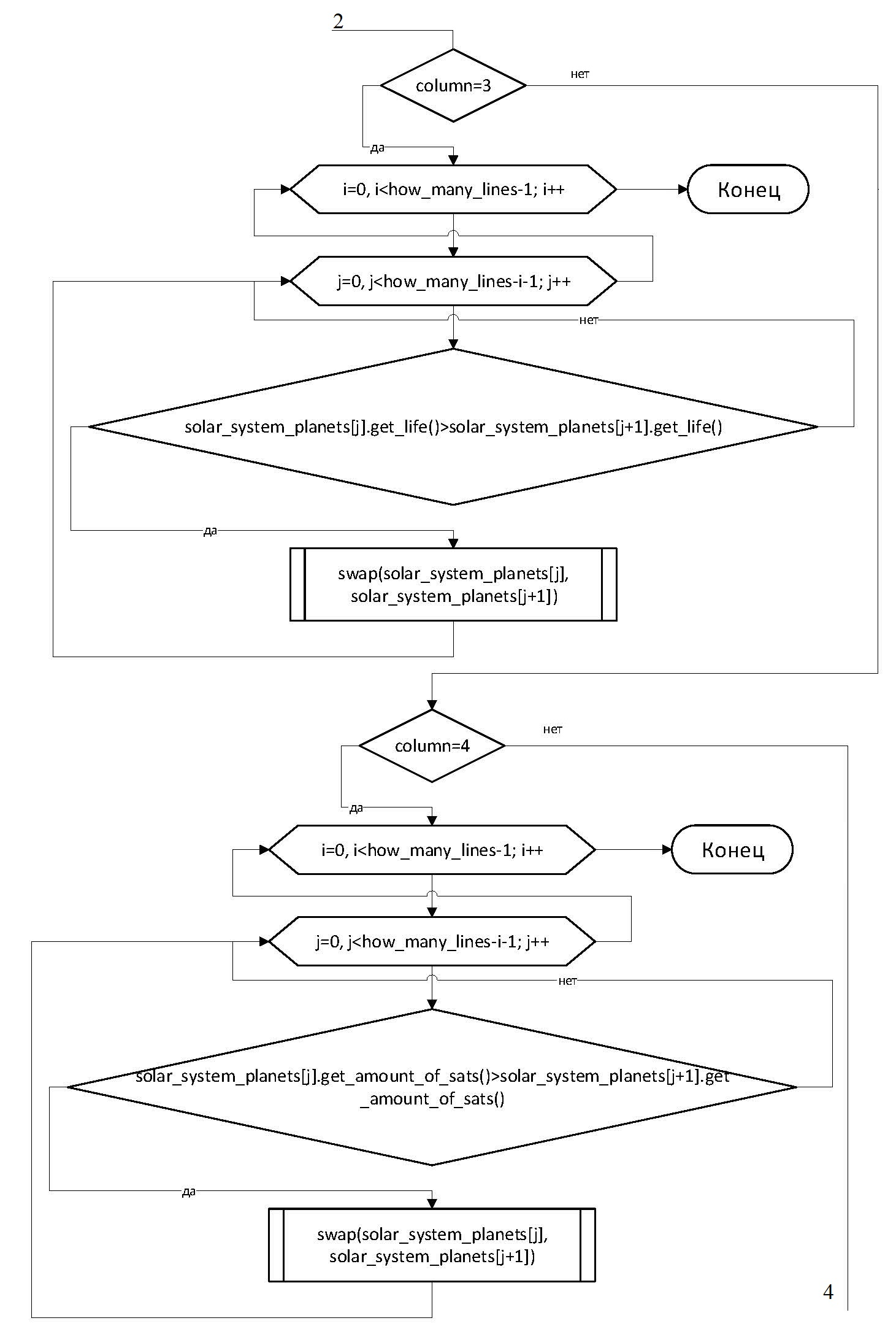


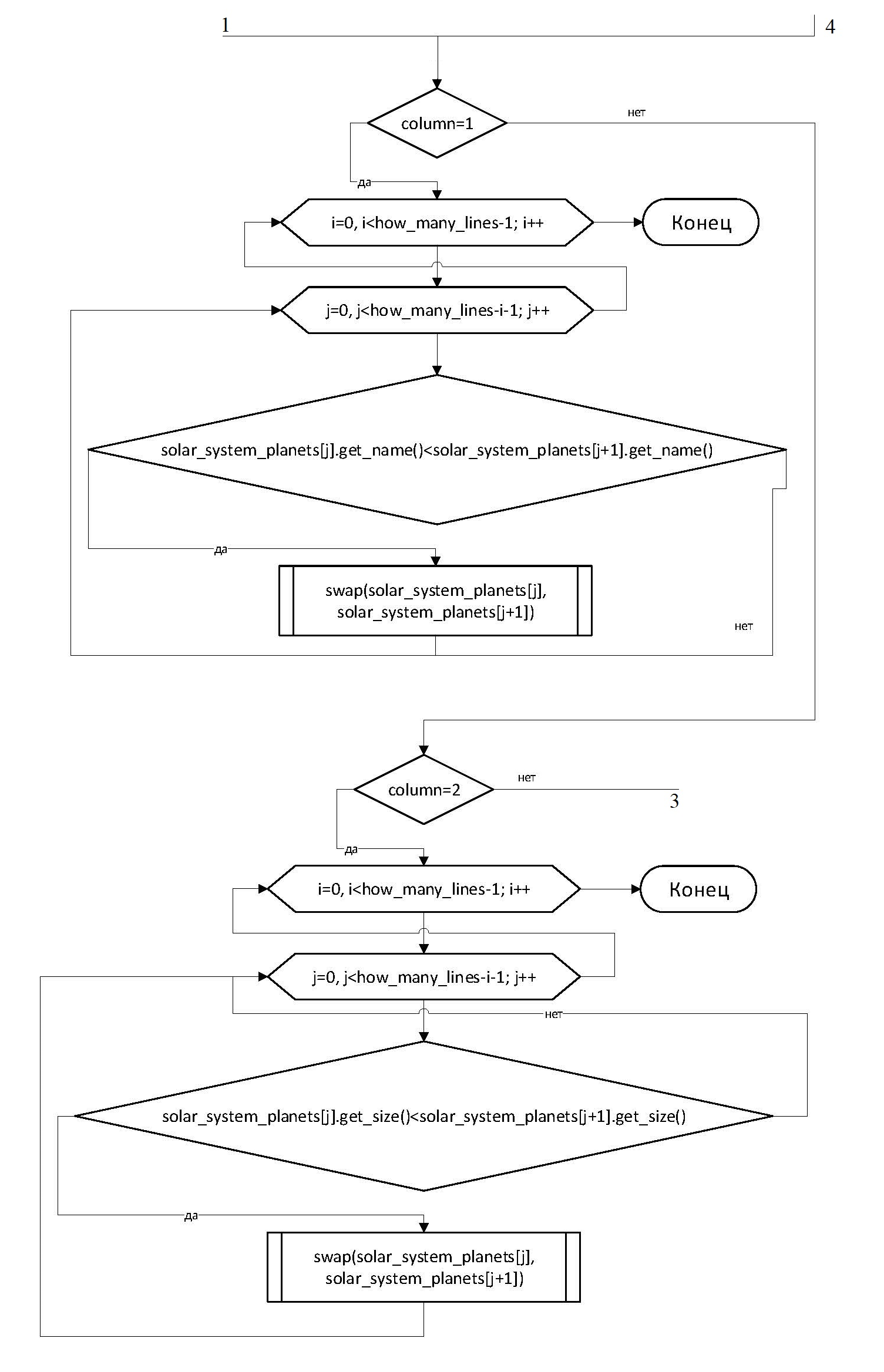
**sort\_method**

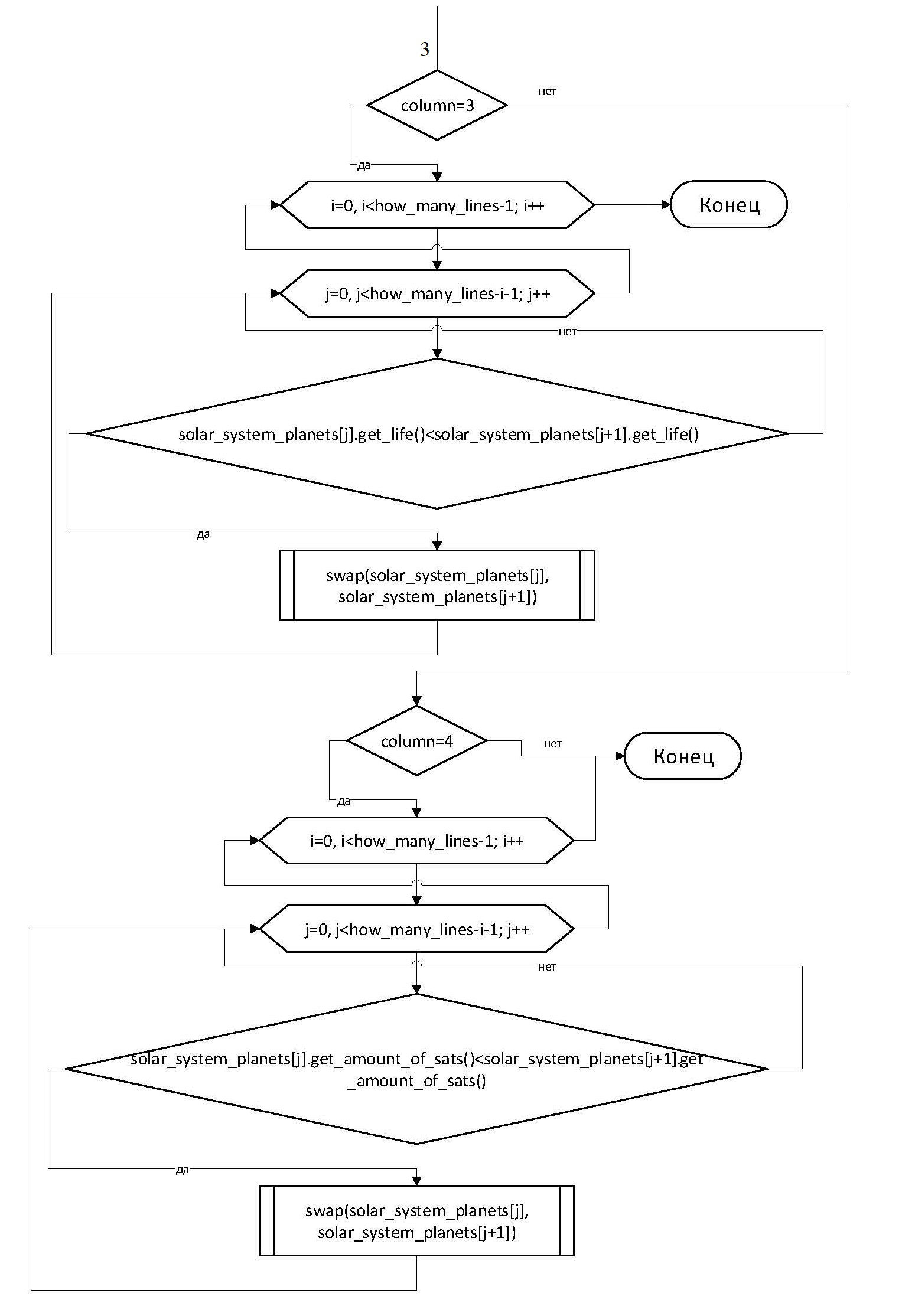


**sort**

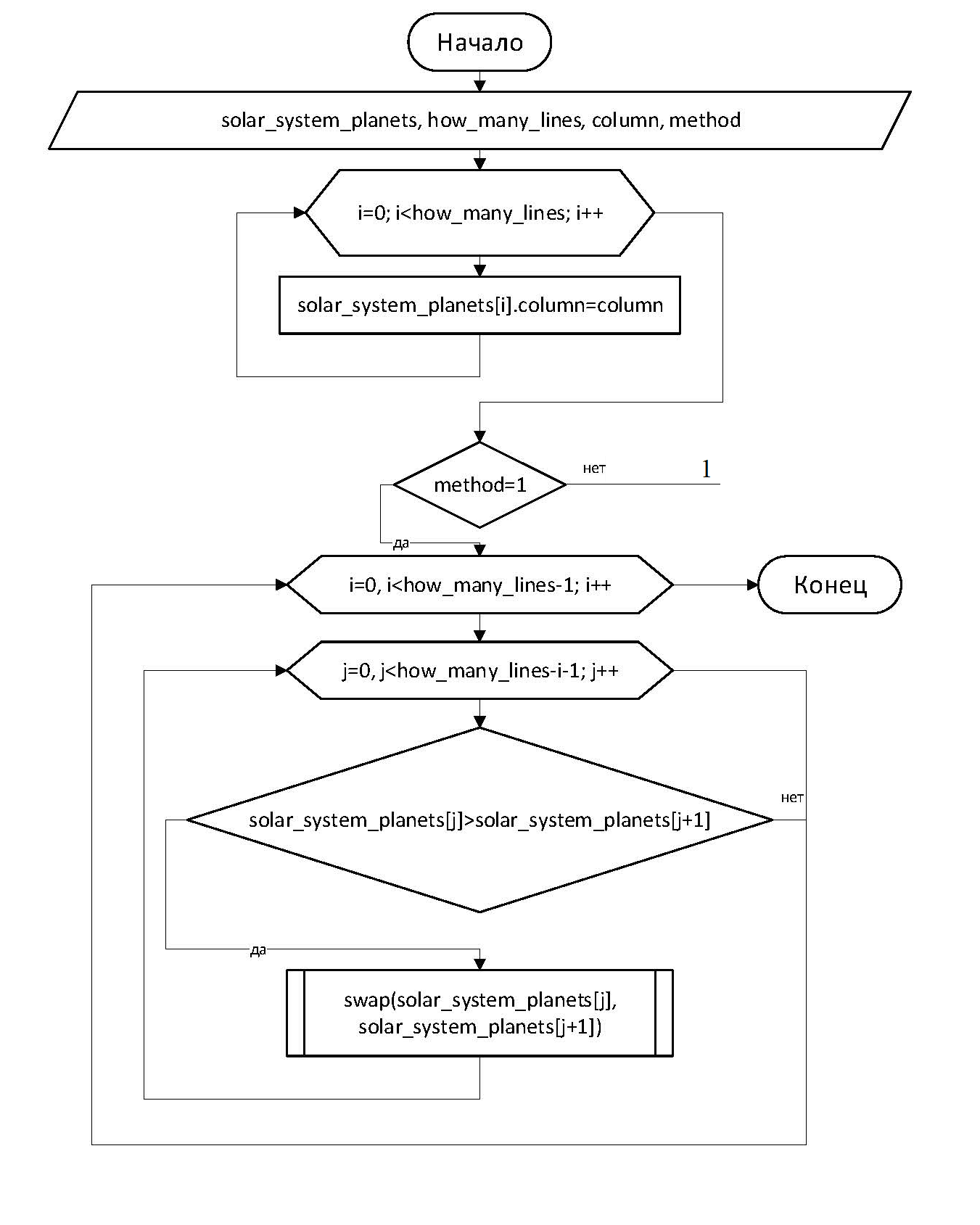
****

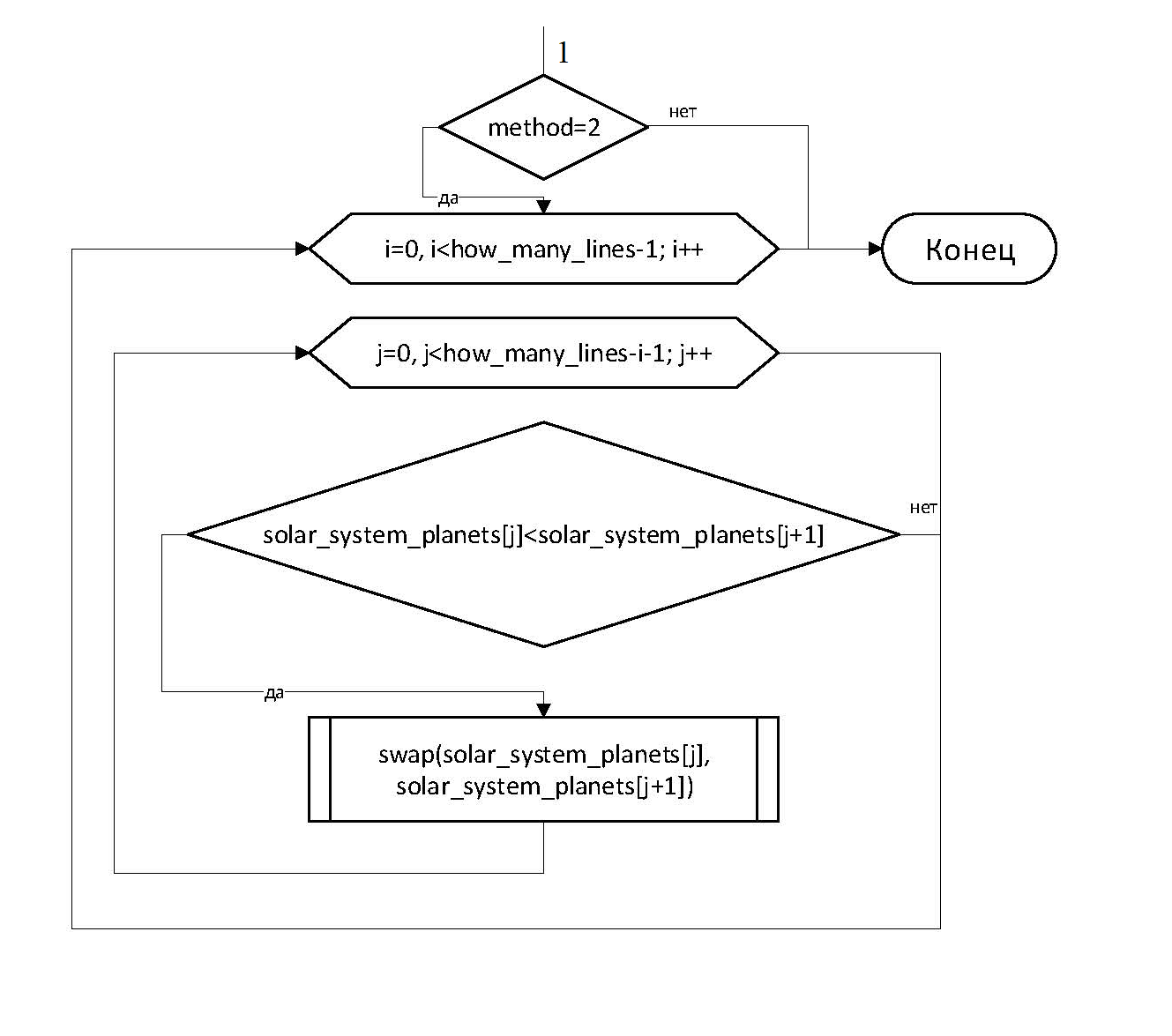
****

****

****

**sort\_2**

****

****

**edit\_menu**



**edit**





**longest\_name\_finder**

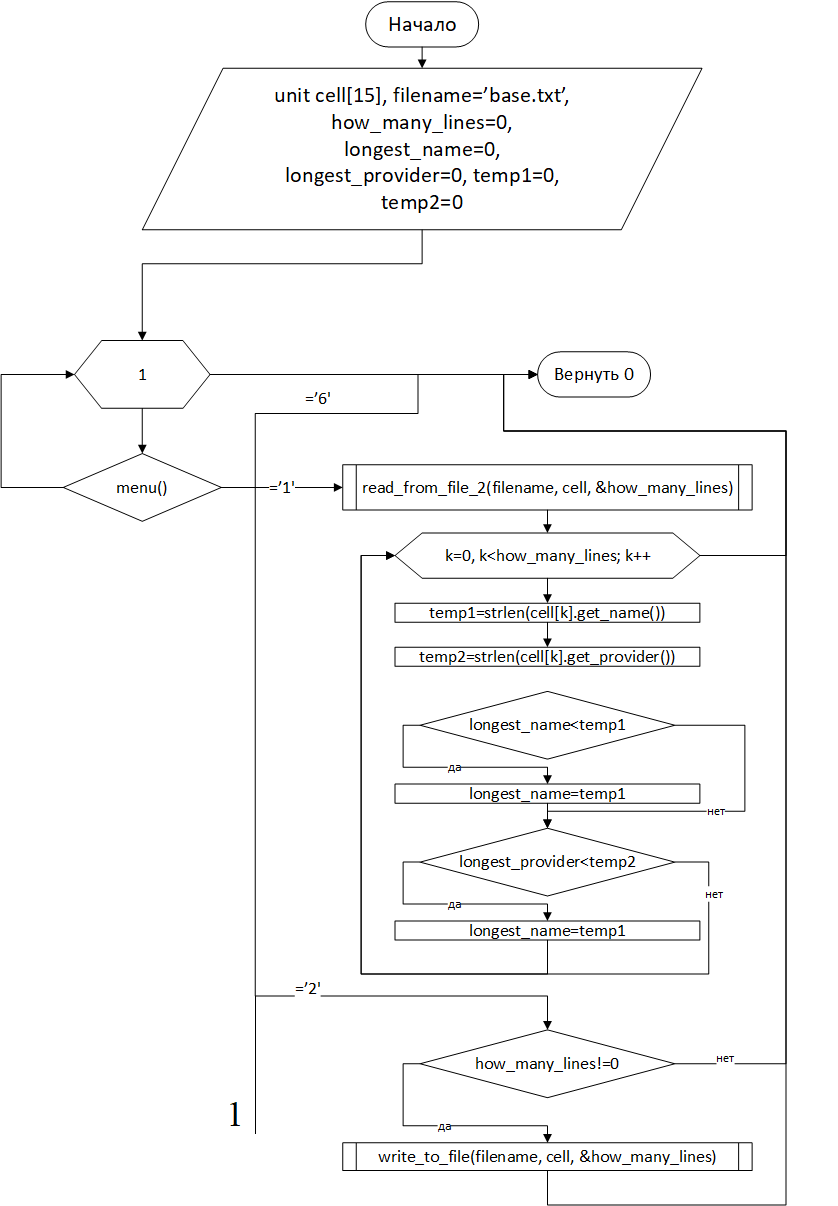


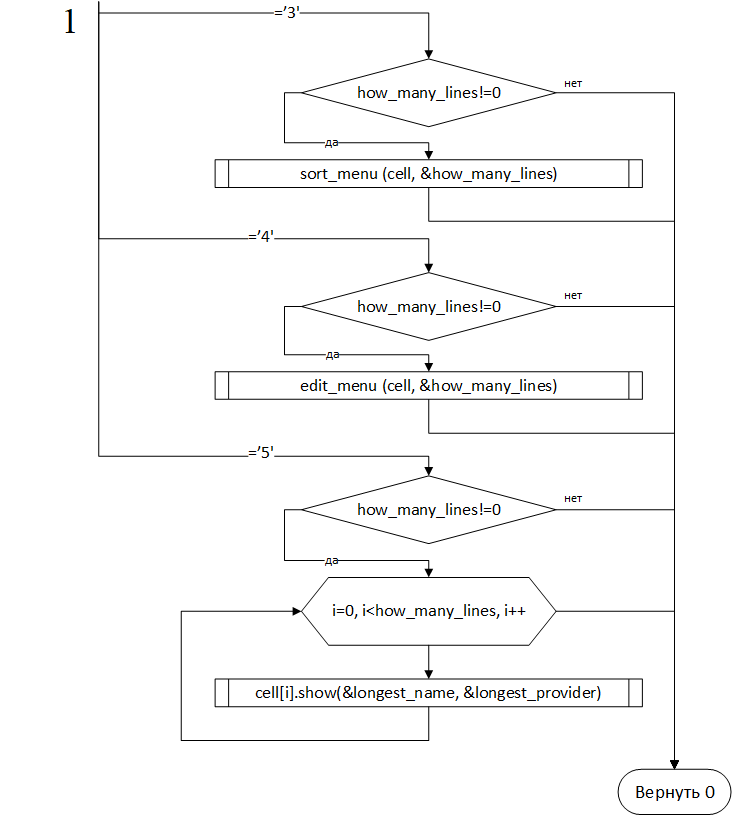
**swap**



Для проекта с классом «Анкета»

**main**

****

****

**menu**



**read\_from\_file**



**write\_to\_file**



**sort\_menu**



**sort\_method**



**edit\_menu**



**sort**



**edit**





**swap**



**Текст программы**

Для проекта с классом «Планета»

**planet.h**

#include <iostream>

using namespace std;

class Planet

{

friend ifstream& operator>>(ifstream&, Planet&);

friend ofstream& operator<<(ofstream&, Planet&);

friend void swap(Planet\*, Planet\*);

public:

Planet();

Planet(char\*, long int, bool, int);

~Planet();

char\* get\_name();

long int get\_size();

int get\_amount\_of\_sats();

bool get\_life();

void set\_name(char\*);

void set\_size(long int);

void set\_amount\_of\_sats(int);

void set\_life(bool);

void show(int\*);

Planet& operator=(Planet&);

bool operator==(Planet);

int column;

bool operator<(Planet);

bool operator>(Planet);

private:

char\* name;

long int diameter;

bool life;

int amount\_of\_sats;

};

**planet.cpp**

#include "planet.h"

#include <iostream>

#include <iomanip>

#include <fstream>

using namespace std;

Planet::Planet()

{

}

Planet::Planet(char\* another\_name, long int another\_diameter, bool another\_life, int another\_sats)

{

name=another\_name;

diameter=another\_diameter;

amount\_of\_sats=another\_sats;

life=another\_life;

cout << endl << "Создание объекта имени " << name << endl;

cout << "Создание объекта размера " << diameter << endl;

cout << "Создание объекта показателя наличия жизни " << life << endl;

cout << "Создание объекта количества спутников " << amount\_of\_sats << endl << endl;

}

Planet::~Planet()

{

}

char\* Planet::get\_name()

{

return Planet::name;

}

long int Planet::get\_size()

{

return Planet::diameter;

}

int Planet::get\_amount\_of\_sats()

{

return Planet::amount\_of\_sats;

}

bool Planet::get\_life()

{

return Planet::life;

}

void Planet::set\_name(char\* new\_name)

{

Planet::name=new\_name;

}

void Planet::set\_size(long int new\_size)

{

Planet::diameter=new\_size;

}

void Planet::set\_amount\_of\_sats(int new\_amount\_of\_sats)

{

Planet::amount\_of\_sats=new\_amount\_of\_sats;

}

void Planet::set\_life(bool new\_life)

{

Planet::life=new\_life;

}

void Planet::show(int\* longest\_name)

{

cout << Planet::name << setw(12+(\*longest\_name)-strlen(Planet::name)) << right << Planet::diameter << setw(10) << Planet::life << setw(13) << Planet::amount\_of\_sats << endl;

}

Planet& Planet::operator=(Planet& solar\_system\_planets)

{

name=solar\_system\_planets.name;

diameter=solar\_system\_planets.diameter;

life=solar\_system\_planets.life;

amount\_of\_sats=solar\_system\_planets.amount\_of\_sats;

return (\*this);

}

ifstream& operator>>(ifstream& fin, Planet& solar\_system\_planets)

{

char\* another\_name=new char[20];

long int another\_diameter;

bool another\_life;

int another\_sats;

fin >> another\_name >> another\_diameter >> another\_life >> another\_sats;

solar\_system\_planets.set\_name(another\_name);

solar\_system\_planets.set\_size(another\_diameter);

solar\_system\_planets.set\_life(another\_life);

solar\_system\_planets.set\_amount\_of\_sats(another\_sats);

return fin;

}

ofstream& operator<<(ofstream& fout, Planet& solar\_system\_planets)

{

fout << solar\_system\_planets.get\_name() << " " << solar\_system\_planets.get\_size() << " " << solar\_system\_planets.get\_life() << " " << solar\_system\_planets.get\_amount\_of\_sats() << endl;

return fout;

}

bool Planet::operator==(Planet solar\_system\_planets)

{

if ((solar\_system\_planets.name == this->name) && (solar\_system\_planets.diameter == this->diameter) && (solar\_system\_planets.life == this->life) && (solar\_system\_planets.amount\_of\_sats == this->amount\_of\_sats))

{

return true;

}

else

{

return false;

}

}

bool Planet::operator<(Planet solar\_system\_planets)

{

if ((this->column)==1)

{

if (this->name < solar\_system\_planets.name)

{

true;

}

else

{

return false;

}

}

if ((this->column)==2)

{

if (this->diameter < solar\_system\_planets.diameter)

{

true;

}

else

{

return false;

}

}

if ((this->column)==3)

{

if (this->life < solar\_system\_planets.life)

{

true;

}

else

{

return false;

}

}

if ((this->column)==4)

{

if (this->amount\_of\_sats < solar\_system\_planets.amount\_of\_sats)

{

true;

}

else

{

return false;

}

}

}

bool Planet::operator>(Planet solar\_system\_planets)

{

if ((this->column)==1)

{

if (this->name > solar\_system\_planets.name)

{

true;

}

else

{

return false;

}

}

if ((this->column)==2)

{

if (this->diameter > solar\_system\_planets.diameter)

{

true;

}

else

{

return false;

}

}

if ((this->column)==3)

{

if (this->life > solar\_system\_planets.life)

{

true;

}

else

{

return false;

}

}

if ((this->column)==4)

{

if (this->amount\_of\_sats > solar\_system\_planets.amount\_of\_sats)

{

true;

}

else

{

return false;

}

}

}

**main.cpp**

#include "planet.h"

#include "proto.h"

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

Planet solar\_system\_planets[15];

char\* filename="planets.txt";

int how\_many\_lines=0;

int longest\_name=0;

while (true)

{

switch (menu())

{

case 1: read\_from\_file\_2(filename, solar\_system\_planets, &how\_many\_lines);

longest\_name\_finder(solar\_system\_planets, &how\_many\_lines, &longest\_name);

break;

case 2: if ((how\_many\_lines)!=0)

{

write\_to\_file\_2(filename, solar\_system\_planets, &how\_many\_lines, &longest\_name);

}

else

{

cout << endl << "Ошибка: нет базы данных!" << endl << endl;

}

break;

case 3: if ((how\_many\_lines)!=0)

{

sort\_menu(solar\_system\_planets, &how\_many\_lines);

cout << endl << "Сортировка была поизведена успешно." << endl << endl;

}

else

{

cout << endl << "Ошибка: нет базы данных!" << endl << endl;

}

break;

case 4: if ((how\_many\_lines)!=0)

{

edit\_menu(solar\_system\_planets, &how\_many\_lines);

}

else

{

cout << endl << "Ошибка: нет базы данных!" << endl << endl;

}

break;

case 5: if ((how\_many\_lines)!=0)

{

cout << endl << "Название" << setw(6+longest\_name) << right << "Диаметр" << setw(10) << "Жизнь" << setw(13) << "Спутники" << endl;

for (int i=0; i<how\_many\_lines; i++)

{

solar\_system\_planets[i].show(&longest\_name);

}

cout << endl;

}

else

{

cout << endl << "Ошибка: нет базы данных!" << endl << endl;

}

break;

case 6: return 0;

case 7: Planet test\_planet("TEST",20,0,35);

int num;

num=4;

test\_planet.show(&num);

cout << endl;

break;

}

}

return 0;

}

int menu()

{

int answ;

cout << "МЕНЮ" << endl;

cout << "l) Считать из файла" << endl;

cout << "2) Запись в файл" << endl;

cout << "3) Сортировка" << endl;

cout << "4) Редактирование" << endl;

cout << "5) Вывод на экран" << endl;

cout << "6) Выход" << endl;

cout << "7) Тестирование отображения сведений о создании объекта в конструкторе" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 7): ";

cin >> answ;

}

while ((answ<1) && (answ>7));

return answ;

}

void read\_from\_file(char\* filename, Planet\* solar\_system\_planets, int\* how\_many\_lines)

{

ifstream fin;

fin.open(filename);

if (!fin.is\_open())

{

cout << "Ошибка: Файла не существует!" << endl << endl;

}

else

{

cout << endl << "Файл \'planets.txt\' успешно открыт." << endl;

char\* current\_name[15];

for (int j=0; j<15; j++)

{

current\_name[j]=new char[20];

}

long int current\_size;

bool current\_life;

int current\_sats;

int i=0;

while (!fin.eof())

{

if (i<15)

{

fin >> current\_name[i] >> current\_size >> current\_life >> current\_sats;

solar\_system\_planets[i].set\_name(current\_name[i]);

solar\_system\_planets[i].set\_size(current\_size);

solar\_system\_planets[i].set\_life(current\_life);

solar\_system\_planets[i].set\_amount\_of\_sats(current\_sats);

i++;

}

}

(\*how\_many\_lines)=i-1;

fin.close();

cout << "Чтение из файла \'planets.txt\' успешно завершено." << endl;

cout << endl;

}

}

void read\_from\_file\_2(char\* filename, Planet\* solar\_system\_planets, int\* how\_many\_lines)

{

ifstream fin;

fin.open(filename);

if (!fin.is\_open())

{

cout << "Ошибка: Файла не существует!" << endl << endl;

}

else

{

cout << endl << "Файл \'planets.txt\' успешно открыт." << endl;

int i=0;

while (!fin.eof())

{

if (i<15)

{

fin >> solar\_system\_planets[i];

i++;

}

}

(\*how\_many\_lines)=i-1;

fin.close();

cout << "Чтение из файла \'planets.txt\' успешно завершено." << endl;

cout << endl;

}

}

void write\_to\_file(char \*filename, Planet\* solar\_system\_planets, int\* how\_many\_lines, int\* longest\_name)

{

ofstream fout;

fout.open(filename);

if (!fout.is\_open())

{

cout << "Ошибка: Запись в файл невозможна!" << endl << endl;

}

else

{

for (int i=0; i<(\*how\_many\_lines); i++)

{

fout << solar\_system\_planets[i].get\_name() << setw(12+(\*longest\_name)-strlen(solar\_system\_planets[i].get\_name())) << right << solar\_system\_planets[i].get\_size() << setw(10) << solar\_system\_planets[i].get\_life() << setw(10) << solar\_system\_planets[i].get\_amount\_of\_sats() << endl;

}

fout.close();

cout << endl << "Запись в файл \'planets.txt\' успешно завершена." << endl;

cout << endl;

}

}

void write\_to\_file\_2(char \*filename, Planet\* solar\_system\_planets, int\* how\_many\_lines, int\* longest\_name)

{

ofstream fout;

fout.open(filename);

if (!fout.is\_open())

{

cout << "Ошибка: Запись в файл невозможна!" << endl << endl;

}

else

{

for (int i=0; i<(\*how\_many\_lines); i++)

{

fout << solar\_system\_planets[i];

}

fout.close();

cout << endl << "Запись в файл \'planets.txt\' успешно завершена." << endl;

cout << endl;

}

}

void sort\_menu(Planet\* solar\_system\_planets, int\* how\_many\_lines)

{

int answ;

cout << endl;

cout << "МЕНЮ СОРТИРОВКИ" << endl;

cout << "Столбец, по которому нужно отсортировать:" << endl;

cout << "1) Название" << endl;

cout << "2) Диаметр" << endl;

cout << "3) Показатель наличия жизни" << endl;

cout << "4) Количество спутников" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 4): ";

cin >> answ;

}

while ((answ<1) && (answ>4));

sort\_method(solar\_system\_planets, how\_many\_lines, answ);

}

void sort\_method(Planet\* solar\_system\_planets, int\* how\_many\_lines, int column)

{

int answ;

cout << "Способ, по которому нужно отсортировать:" << endl;

cout << "1) По возрастанию" << endl;

cout << "2) По убыванию" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 2): ";

cin >> answ;

}

while ((answ<1) && (answ>2));

sort\_2(solar\_system\_planets, how\_many\_lines, column, answ);

}

void sort(Planet\* solar\_system\_planets, int\* how\_many\_lines, int column, int method)

{

if (method==1)

{

switch(column)

{

case 1:

for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j].get\_name()>solar\_system\_planets[j+1].get\_name())

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

cout << endl << "Сортировка по названию по убыванию была успешно произведена." << endl << endl;

break;

case 2: for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j].get\_size()>solar\_system\_planets[j+1].get\_size())

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

cout << endl << "Сортировка по диаметру по возрастанию была успешно произведена." << endl << endl;

break;

case 3: for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j].get\_life()>solar\_system\_planets[j+1].get\_life())

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

cout << endl << "Сортировка по показателю наличия жизни по возрастанию была успешно произведена." << endl << endl;

break;

case 4: for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j].get\_amount\_of\_sats()>solar\_system\_planets[j+1].get\_amount\_of\_sats())

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

cout << endl << "Сортировка по количеству спутников по возрастанию была успешно произведена." << endl << endl;

break;

}

}

else

{

switch(column)

{

case 1:

for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j].get\_name()<solar\_system\_planets[j+1].get\_name())

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

cout << endl << "Сортировка по названию по убыванию была успешно произведена." << endl << endl;

break;

case 2: for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j].get\_size()<solar\_system\_planets[j+1].get\_size())

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

cout << endl << "Сортировка по диаметру по убыванию была успешно произведена." << endl << endl;

break;

case 3: for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j].get\_life()<solar\_system\_planets[j+1].get\_life())

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

cout << endl << "Сортировка по показателю наличия жизни по убыванию была успешно произведена." << endl << endl;

break;

case 4: for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j].get\_amount\_of\_sats()<solar\_system\_planets[j+1].get\_amount\_of\_sats())

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

cout << endl << "Сортировка по количеству спутников по убыванию была успешно произведена." << endl << endl;

break;

}

}

}

void sort\_2(Planet\* solar\_system\_planets, int\* how\_many\_lines, int column, int method)

{

for (int i=0; i<(\*how\_many\_lines); i++)

{

solar\_system\_planets[i].column=column;

}

if (method==1)

{

for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j]>solar\_system\_planets[j+1])

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

}

else

{

if (method==2)

{

for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (solar\_system\_planets[j]<solar\_system\_planets[j+1])

swap(solar\_system\_planets[j], solar\_system\_planets[j+1]);

}

}

}

}

}

void edit\_menu(Planet\* solar\_system\_planets, int\* how\_many\_lines)

{

int answ;

cout << endl;

cout << "МЕНЮ РЕДАКТИРОВАНИЯ" << endl;

cout << "Действие, по которое нужно совершить:" << endl;

cout << "1) Добавить строку" << endl;

cout << "2) Удалить строку" << endl;

cout << "3) Редактировать строку" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 3): ";

cin >> answ;

}

while ((answ<1) && (answ>3));

edit(solar\_system\_planets, how\_many\_lines, answ);

}

void edit(Planet\* solar\_system\_planets, int\* how\_many\_lines, int method)

{

if (method==1)

{

if ((\*how\_many\_lines)<15)

{

char\* current\_name;

current\_name=new char[20];

long int current\_size;

bool current\_life;

int current\_sats;

cout << "Введите новое название планеты: ";

cin >> current\_name;

solar\_system\_planets[(\*how\_many\_lines)].set\_name(current\_name);

cout << "Введите диаметр планеты: ";

cin >> current\_size;

solar\_system\_planets[(\*how\_many\_lines)].set\_size(current\_size);

cout << "Введите показатель наличия жизни (цифра 0, если нет, или 1, есть существует): ";

cin >> current\_life;

solar\_system\_planets[(\*how\_many\_lines)].set\_life(current\_life);

cout << "Введите количество спутников: ";

cin >> current\_sats;

solar\_system\_planets[(\*how\_many\_lines)].set\_amount\_of\_sats(current\_sats);

(\*how\_many\_lines)++;

cout << endl;

}

else

{

cout << endl << "Ошибка: невозможно добавить строку - переполнена допустимая база!";

}

}

if (method==2)

{

if ((\*how\_many\_lines)>0)

{

int answ;

cout << endl << "Введите номер строки для удаления: ";

cin >> answ;

for (int i=answ-1; i<=(\*how\_many\_lines); i++)

{

solar\_system\_planets[i]=solar\_system\_planets[i+1];

}

(\*how\_many\_lines)--;

cout << "Удаление " << answ << " строки успешно выполнено.";

cout << endl << endl;

}

else

{

cout << endl << "Ошибка: невозможно удалить строку!" << endl;

}

}

if (method==3)

{

int num\_line, num\_what;

cout << endl << "Введите номер строки для редактирования: ";

cin >> num\_line;

cout << endl << "Выберете, что Вы хотите отредактировать:" << endl;

cout << "1) Название" << endl;

cout << "2) Диаметр" << endl;

cout << "3) Показатель наличия жизни" << endl;

cout << "4) Количество спутников" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 4): ";

cin >> num\_what;

}

while ((num\_what<1) && (num\_what>4));

switch (num\_what)

{

case 1: char\* current\_name;

current\_name=new char[20];

cout << endl << "Введите название планеты: ";

cin >> current\_name;

solar\_system\_planets[num\_line-1].set\_name(current\_name);

cout << endl << "Название планеты изменено успешно." << endl << endl;

break;

case 2: long int current\_size;

cout << endl << "Введите диаметр планеты: ";

cin >> current\_size;

solar\_system\_planets[num\_line-1].set\_size(current\_size);

cout << endl << "Диаметр планеты изменен успешно." << endl << endl;

break;

case 3: bool current\_life;

cout << endl << "Введите показатель наличия жизни (цифра 0, если нет, или 1, есть существует): ";

cin >> current\_life;

solar\_system\_planets[num\_line-1].set\_life(current\_life);

cout << endl << "Показатель наличия жизни планеты изменен успешно." << endl << endl;

break;

case 4: int current\_sats;

cout << endl << "Введите количество спутников: ";

cin >> current\_sats;

solar\_system\_planets[num\_line-1].set\_amount\_of\_sats(current\_sats);

cout << endl << "Количество спутников планеты изменено успешно." << endl << endl;

break;

}

}

}

void longest\_name\_finder(Planet\* solar\_system\_planets, int\* how\_many\_lines, int\* longest\_name)

{

for (int k=0; k<(\*how\_many\_lines); k++)

{

if ((\*longest\_name)<strlen(solar\_system\_planets[k].get\_name()))

{

(\*longest\_name)=strlen(solar\_system\_planets[k].get\_name());

}

}

}

void swap(Planet\* p1, Planet\* p2)

{

swap((\*p1).name, (\*p2).name);

swap((\*p1).diameter, (\*p2).diameter);

swap((\*p1).life, (\*p2).life);

swap((\*p1).amount\_of\_sats, (\*p2).amount\_of\_sats);

}

**proto.h**

int menu();

void read\_from\_file(char\*, Planet\*, int\*);

void read\_from\_file\_2(char\*, Planet\*, int\*);

void write\_to\_file(char\*, Planet\*, int\*, int\*);

void write\_to\_file\_2(char\*, Planet\*, int\*, int\*);

void sort\_menu(Planet\*, int\*);

void sort\_method(Planet\*, int\*, int);

void sort(Planet\*, int\*, int, int);

void sort\_2(Planet\*, int\*, int, int);

void edit\_menu(Planet\*, int\*);

void edit(Planet\*, int\*, int);

void longest\_name\_finder(Planet\*, int\*, int\*);

void swap(Planet\*, Planet\*);

Для проекта с классом «Анкета»

**unit.h**

#include <iostream>

using namespace std;

class unit

{

friend ifstream& operator>>(ifstream&, unit&);

friend ofstream& operator<<(ofstream&, unit&);

friend void swap(unit\*, unit\*);

public:

unit();

~unit();

char\* get\_name();

int get\_age();

char get\_gend();

char\* get\_provider();

bool get\_quality();

bool get\_willchange();

void set\_name(char\*);

void set\_age(int);

void set\_gend(char);

void set\_provider(char\*);

void set\_quality(bool);

void set\_willchange(bool);

void show(int\*, int\*);

bool operator==(unit);

int column;

bool operator<(unit);

bool operator>(unit);

private:

char\* name;

int age;

char gend;

char\* provider;

bool quality;

bool willchange;

};

**unit.cpp**

#include "unit.h"

#include "proto.h"

#include <iostream>

#include <iomanip>

#include <fstream>

using namespace std;

unit::unit()

{

}

unit::~unit()

{

}

char\* unit::get\_name()

{

return unit::name;

}

int unit::get\_age()

{

return unit::age;

}

char unit::get\_gend()

{

return unit::gend;

}

char\* unit::get\_provider()

{

return unit::provider;

}

bool unit::get\_quality()

{

return unit::quality;

}

bool unit::get\_willchange()

{

return unit::willchange;

}

void unit::set\_name(char\* new\_name)

{

unit::name=new\_name;

}

void unit::set\_age(int new\_age)

{

unit::age=new\_age;

}

void unit::set\_gend(char new\_gend)

{

unit::gend=new\_gend;

}

void unit::set\_provider(char\* new\_prov)

{

unit::provider=new\_prov;

}

void unit::set\_quality(bool new\_q)

{

unit::quality=new\_q;

}

void unit::set\_willchange(bool new\_wc)

{

unit::willchange=new\_wc;

}

void unit::show(int\* longest\_name, int\* longest\_provider)

{

cout << unit::name << setw(\*longest\_name-strlen(unit::get\_name())+10) << right << unit::age << setw(10) << right << unit::gend << setw(\*longest\_provider+10) << right << unit::provider << setw(10) << right << unit::quality << setw(20) << right << unit::willchange << endl;

}

ifstream& operator>>(ifstream& fin, unit& cell)

{

char\* another\_name=new char[20];

int another\_age;

char another\_gend;

char\* another\_prov=new char[20];

bool another\_q;

bool another\_wc;

fin >> another\_name >> another\_age >> another\_gend >> another\_prov >> another\_q >> another\_wc;

cell.set\_name(another\_name);

cell.set\_age(another\_age);

cell.set\_gend(another\_gend);

cell.set\_provider(another\_prov);

cell.set\_quality(another\_q);

cell.set\_willchange(another\_wc);

return fin;

}

ofstream& operator<<(ofstream& fout, unit& cell)

{

fout << cell.get\_name() << " " << cell.get\_age() << " " << cell.get\_gend() << " " << cell.get\_provider() << " " << cell.get\_quality() << " " << cell.get\_willchange() << endl;

return fout;

}

bool unit::operator==(unit cell)

{

if ((cell.name == this->name) && (cell.age == this->age) && (cell.gend == this->gend) && (cell.provider == this->provider) && (cell.quality == this->quality) && (cell.willchange == this->willchange))

{

return true;

}

else

{

return false;

}

}

bool unit::operator<(unit cell)

{

if ((this->column)==1)

{

if (this->name < cell.name)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==2)

{

if (this->age < cell.age)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==3)

{

if (this->gend < cell.gend)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==4)

{

if (this->provider < cell.provider)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==5)

{

if (this->quality < cell.quality)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==6)

{

if (this->willchange < cell.willchange)

{

return true;

}

else

{

return false;

}

}

else

return false;

}

}

}

}

}

}

bool unit::operator>(unit cell)

{

if ((this->column)==1)

{

if (this->name > cell.name)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==2)

{

if (this->age > cell.age)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==3)

{

if (this->gend > cell.gend)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==4)

{

if (this->provider > cell.provider)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==5)

{

if (this->quality > cell.quality)

{

return true;

}

else

{

return false;

}

}

else

{

if ((this->column)==6)

{

if (this->willchange > cell.willchange)

{

return true;

}

else

{

return false;

}

}

else

return false;

}

}

}

}

}

}

**main.cpp**

#include "unit.h"

#include "proto.h"

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

unit cell[15];

char\* filename="base.txt";

int how\_many\_lines=0;

int longest\_name=0;

int longest\_provider=0;

int temp1=0;

int temp2=0;

cout << "База данных, созданная из данных из анкеты для опроса населения о качестве услуг связи, состоит из ответов каждого человека на следующие вопросы: " << endl;

cout << "1) Имя человека" << endl;

cout << "2) Его возраст" << endl;

cout << "3) Его пол" << endl;

cout << "4) Оператор связи, которым он пользуется" << endl;

cout << "5) Его оценка качества услуг оператора (положительная - 1, отрицательная - 0)" << endl;

cout << "6) Его желание изменить оператора связи (0 - не желает, 1 - желает)" << endl;

cout << "База данных содержит максимум 15 ячеек." << endl << endl;

while (true)

{

switch (menu())

{

case 1: read\_from\_file(filename, cell, &how\_many\_lines);

for (int k=0; k<how\_many\_lines; k++)

{

temp1=strlen(cell[k].get\_name());

temp2=strlen(cell[k].get\_provider());

if (longest\_name<temp1)

{

longest\_name=temp1;

}

if (longest\_provider<temp2)

{

longest\_provider=temp2;

}

}

break;

case 2: if ((how\_many\_lines)!=0)

{

write\_to\_file(filename, cell, &how\_many\_lines);

}

else

{

cout << endl << "Ошибка: нет базы данных!" << endl << endl;

}

break;

case 3: if ((how\_many\_lines)!=0)

{

sort\_menu(cell, &how\_many\_lines);

cout << endl << "Сортировка была поизведена успешно." << endl;

cout << endl;

}

else

{

cout << endl << "Ошибка: нет базы данных!" << endl << endl;

}

break;

case 4: if ((how\_many\_lines)!=0)

{

edit\_menu(cell, &how\_many\_lines);

}

else

{

cout << endl << "Ошибка: нет базы данных!" << endl << endl;

}

break;

case 5: if ((how\_many\_lines)!=0)

{

cout << endl;

cout << "Имя" << setw(longest\_name-3+10) << right << "Возраст" << setw(10) << right << "Пол" << setw(longest\_provider+10) << right << "Оператор" << setw(10) << right << "Оценка" << setw(20) << right << "Желание изменить" << endl;

for (int i=0; i<how\_many\_lines; i++)

{

cell[i].show(&longest\_name, &longest\_provider);

}

cout << "Всего строк: " << how\_many\_lines << endl;

cout << endl;

}

else

{

cout << endl << "Ошибка: нет базы данных!" << endl << endl;

}

break;

case 6: return 0;

}

}

return 0;

}

int menu()

{

int answ;

cout << "МЕНЮ" << endl;

cout << "l) Считать из файла" << endl;

cout << "2) Запись в файл" << endl;

cout << "3) Сортировка" << endl;

cout << "4) Редактирование" << endl;

cout << "5) Вывод на экран" << endl;

cout << "6) Выход" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 6): ";

cin >> answ;

}

while ((answ<1) && (answ>6));

return answ;

}

void read\_from\_file(char\* filename, unit\* cell, int\* how\_many\_lines)

{

ifstream fin;

fin.open(filename);

if (!fin.is\_open())

{

cout << "Ошибка: Файла не существует!" << endl << endl;

}

else

{

cout << endl << "Файл \'base.txt\' успешно открыт." << endl;

int i=0;

while (!fin.eof())

{

if (i<15)

{

fin >> cell[i];

i++;

}

}

(\*how\_many\_lines)=i-1;

fin.close();

cout << "Чтение из файла \'base.txt\' успешно завершено." << endl;

cout << endl;

}

}

void write\_to\_file(char \*filename, unit\* cell, int\* how\_many\_lines)

{

ofstream fout;

fout.open(filename);

if (!fout.is\_open())

{

cout << "Ошибка: Запись в файл невозможна!" << endl << endl;

}

else

{

for (int i=0; i<(\*how\_many\_lines); i++)

{

fout << cell[i];

}

fout.close();

cout << endl << "Запись в файл \'base.txt\' успешно завершена." << endl;

cout << endl;

}

}

void sort\_menu(unit\* cell, int\* how\_many\_lines)

{

int answ;

cout << endl;

cout << "МЕНЮ СОРТИРОВКИ" << endl;

cout << "Столбец, по которому нужно отсортировать:" << endl;

cout << "1) Имя" << endl;

cout << "2) Возраст" << endl;

cout << "3) Пол" << endl;

cout << "4) Оператор связи" << endl;

cout << "5) Положительная/отрицательная оценка качества услуг" << endl;

cout << "6) Желание изменить оператора связи" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 6): ";

cin >> answ;

}

while ((answ<1) && (answ>6));

sort\_method(cell, how\_many\_lines, answ);

}

void sort\_method(unit\* cell, int\* how\_many\_lines, int column)

{

int answ;

cout << "Способ, по которому нужно отсортировать:" << endl;

cout << "1) По возрастанию" << endl;

cout << "2) По убыванию" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 2): ";

cin >> answ;

}

while ((answ<1) && (answ>2));

sort(cell, how\_many\_lines, column, answ);

}

void sort(unit\* cell, int\* how\_many\_lines, int column, int method)

{

for (int i=0; i<(\*how\_many\_lines); i++)

{

cell[i].column=column;

}

if (method==1)

{

for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (cell[j]>cell[j+1])

swap(cell[j], cell[j+1]);

}

}

}

else

{

if (method==2)

{

for (int i=0; i<(\*how\_many\_lines)-1; i++)

{

for (int j=0; j<(\*how\_many\_lines)-i-1; j++)

{

if (cell[j]<cell[j+1])

swap(cell[j], cell[j+1]);

}

}

}

}

}

void edit\_menu(unit\* cell, int\* how\_many\_lines)

{

int answ;

cout << endl;

cout << "МЕНЮ РЕДАКТИРОВАНИЯ" << endl;

cout << "Действие, по которое нужно совершить:" << endl;

cout << "1) Добавить строку" << endl;

cout << "2) Удалить строку" << endl;

cout << "3) Редактировать строку" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 3): ";

cin >> answ;

}

while ((answ<1) && (answ>3));

edit(cell, how\_many\_lines, answ);

}

void edit(unit\* cell, int\* how\_many\_lines, int method)

{

if (method==1)

{

if ((\*how\_many\_lines)<15)

{

char\* another\_name=new char[20];

int another\_age;

char another\_gend;

char\* another\_prov=new char[20];

bool another\_q;

bool another\_wc;

cout << endl << "Введите имя человека: ";

cin >> another\_name;

cell[(\*how\_many\_lines)].set\_name(another\_name);

cout << "Введите возраст человека: ";

cin >> another\_age;

cell[(\*how\_many\_lines)].set\_age(another\_age);

cout << "Введите пол человека: ";

cin >> another\_gend;

cell[(\*how\_many\_lines)].set\_gend(another\_gend);

cout << "Введите название оператора связи: ";

cin >> another\_prov;

cell[(\*how\_many\_lines)].set\_provider(another\_prov);

cout << "Введите показатель его оценки качества связи (0 - отрицательный, 1 - положительный): ";

cin >> another\_q;

cell[(\*how\_many\_lines)].set\_quality(another\_q);

cout << "Введите показатель его желания изменить оператора (0 - отрицательный, 1 - положительный): ";

cin >> another\_wc;

cell[(\*how\_many\_lines)].set\_willchange(another\_wc);

(\*how\_many\_lines)++;

cout << endl << "Добавление строки в базу было произведено успешно." << endl;

cout << endl;

}

else

{

cout << endl << "Ошибка: невозможно добавить строку - переполнена допустимая база!";

}

}

if (method==2)

{

if ((\*how\_many\_lines)>0)

{

int answ;

cout << endl << "Введите номер строки для удаления: ";

cin >> answ;

for (int i=answ-1; i<=(\*how\_many\_lines); i++)

{

cell[i]=cell[i+1];

}

(\*how\_many\_lines)--;

cout << endl << "Удаление " << answ << " строки успешно выполнено.";

cout << endl << endl;

}

else

{

cout << endl << "Ошибка: невозможно удалить строку!" << endl;

}

}

if (method==3)

{

int num\_line, num\_what;

cout << endl << "Введите номер строки для редактирования: ";

cin >> num\_line;

cout << endl << "Выберете, что Вы хотите отредактировать:" << endl;

cout << "1) Имя" << endl;

cout << "2) Возраст" << endl;

cout << "3) Пол" << endl;

cout << "4) Оператор связи" << endl;

cout << "5) Оценка качества услуг" << endl;

cout << "6) Желание изменить оператора" << endl;

do

{

cout << "Пожалуйста выберете нужный вариант (от 1 до 6): ";

cin >> num\_what;

}

while ((num\_what<1) && (num\_what>6));

switch (num\_what)

{

case 1: char\* another\_name;

another\_name=new char[20];

cout << endl << "Введите новое имя: ";

cin >> another\_name;

cell[num\_line-1].set\_name(another\_name);

cout << endl << "Имя изменено успешно." << endl << endl;

break;

case 2: int another\_age;

cout << endl << "Введите новый возраст: ";

cin >> another\_age;

cell[num\_line-1].set\_age(another\_age);

cout << endl << "Возраст изменен успешно." << endl << endl;

break;

case 3: char another\_gend;

cout << endl << "Введите новый пол: ";

cin >> another\_gend;

cell[num\_line-1].set\_gend(another\_gend);

cout << endl << "Пол изменен успешно." << endl << endl;

break;

case 4: char\* another\_provider;

another\_provider=new char[20];

cout << endl << "Введите новое название оператора: ";

cin >> another\_provider;

cell[num\_line-1].set\_provider(another\_provider);

cout << endl << "Название оператора изменено успешно." << endl << endl;

break;

case 5: bool another\_quality;

cout << endl << "Введите новую оценку качества услуг: ";

cin >> another\_quality;

cell[num\_line-1].set\_quality(another\_quality);

cout << endl << "Показатель качества услуг изменен успешно." << endl << endl;

break;

case 6: bool another\_willchange;

cout << endl << "Введите показатель желания изменить оператора: ";

cin >> another\_willchange;

cell[num\_line-1].set\_willchange(another\_willchange);

cout << endl << "Показатель желания изменить оператора изменен успешно." << endl << endl;

break;

}

}

}

void swap(unit\* u1, unit\* u2)

{

swap((\*u1).name, (\*u2).name);

swap((\*u1).age, (\*u2).age);

swap((\*u1).gend, (\*u2).gend);

swap((\*u1).provider, (\*u2).provider);

swap((\*u1).quality, (\*u2).quality);

swap((\*u1).willchange, (\*u2).willchange);

}

**proto.h**

int menu();

void read\_from\_file(char\*, unit\*, int\*);

void write\_to\_file(char\*, unit\*, int\*);

void sort\_menu(unit\*, int\*);

void sort\_method(unit\*, int\*, int);

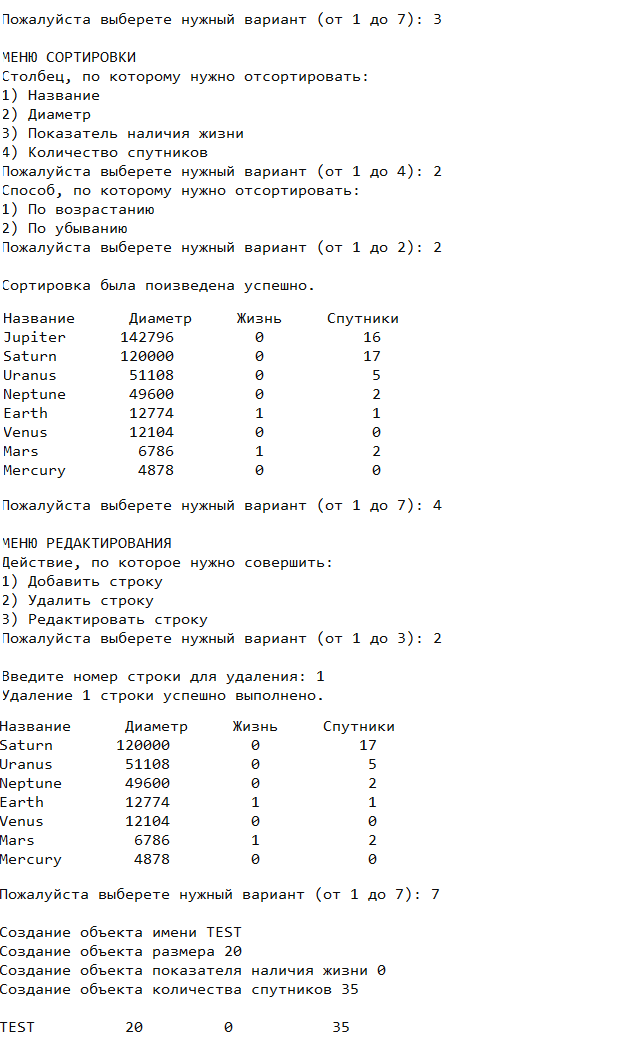
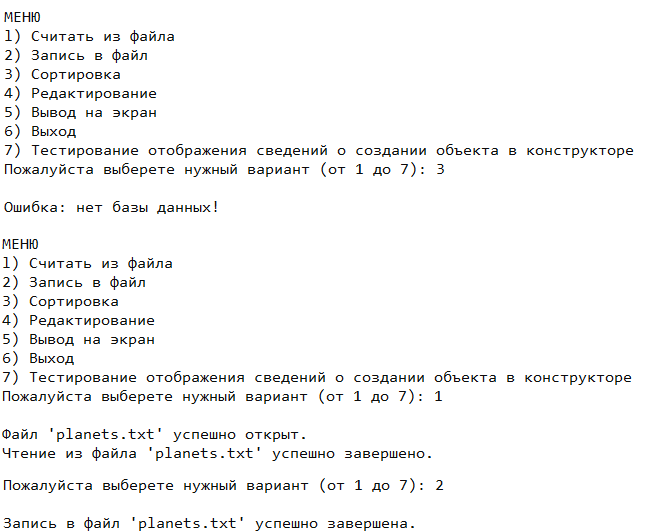
void sort(unit\*, int\*, int, int);

void edit\_menu(unit\*, int\*);

void edit(unit\*, int\*, int);

void swap(unit\*, unit\*);

**Анализ результатов**

Для проекта с классом «Планета»

Для проекта с классом «Анкета»

