# Татарников Максим А-07-22 (ЛР№3, программирование)

**1. Постановка задачи**

Разработать и реализовать программу, используя принципы ООП. Решить проблему неоднозначности множественного наследования ("алмаза смерти") в данной программе.

Построить иерархию классов, состоящую из Планеты, физических характеристик, биологических характеристик, космоса. Класс Планета содержит в себе: название планеты, массу планеты, радиус планеты. Класс физических характеристик: уровень технологий, ускорение свободного падения. Класс биологических характеристик: население, уровень жизни.

Функции программы:

1. Бросить астероид
2. Заразить Вирусом.
3. Объявить войну
4. Разгневать Господа

**2. Разработка программы**

**2.1Описание классов**

Для работы программы создается суперкласс Planet.

Physics и Biology – дочерние классы Planet. Space – дочерний класс для классов Physics и Biology (множественное наследование). Данная иерархия демонстрирует проблему множественного наследования, заключающуюся в наследовании одних и тех же методов и свойств от нескольких материнских классов, что может привести к неоднозначности.

Planet

Biology

Physics

Space

*Рисунок 1.1 Диаграмма классов*

**Planet** – класс описания планеты.

**Функции класса Planet:**

*Конструктор:* Planet(string n, int m, int r)

*Сеттеры и геттеры:*

string GetName()

int GetWeight()

int GetRadius()

void SetName(string Name)

void SetWeight(int Weight)

void SetRadius(int Radius)

*Сеттеры* void PlanetPrint()

*Деструктор: ~*Planet()

**Physics** – класс описания физических характеристик.

**Функции класса Physics:**

*Конструктор:* Physics()

*Сеттеры и геттеры:*

double GetGAcceler()

int GetTechnic()

void SetTechnic(int any)

*Изменение данных:*

string IncTechnic()

*Вывод данных:*

void PhysPrint()

*Деструктор*: ~Physics()

**Biology** – класс описания биологических характеристик.

**Функции класса Biology:**

*Конструктор:* Biology()

*Сеттеры и геттеры:*

int GetQOLife()

int GetPopulation()

void SetQOLife(int any)

void SetPopulation(int Population)

*Деструктор: ~*Biology()

Изменение данных:

string IncPop()

string IncQOLife()

Вывод данных

string IncQOLife()

**Space** – класс описания космоса.

**Функции класса Space:**

*Конструктор:* Space()

*Геттер*

Int GetAst()

*Изменение данных*

string Asteroid(int m, int v, int S)

string War()

string Virus()

*Завершение программы*

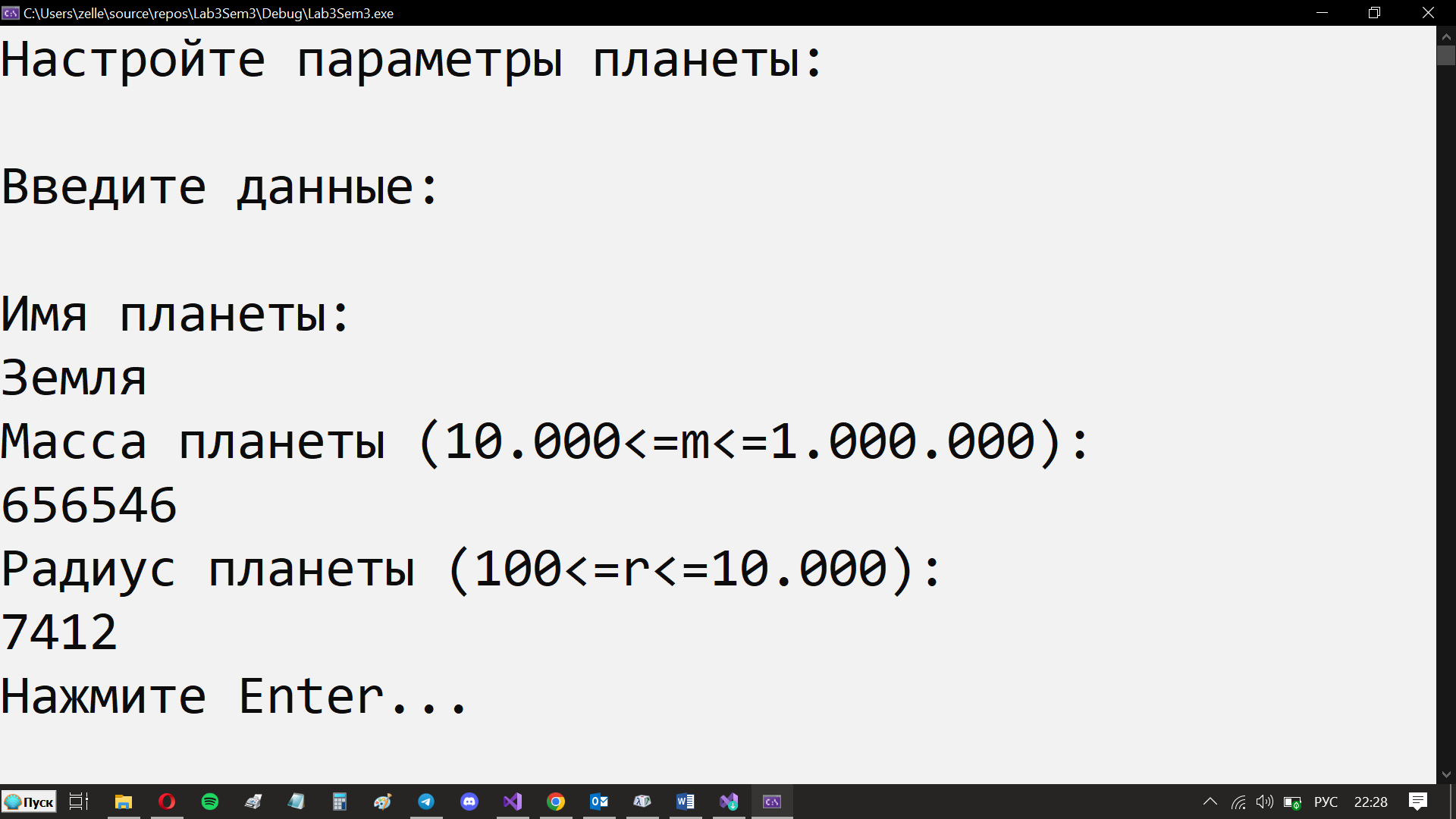
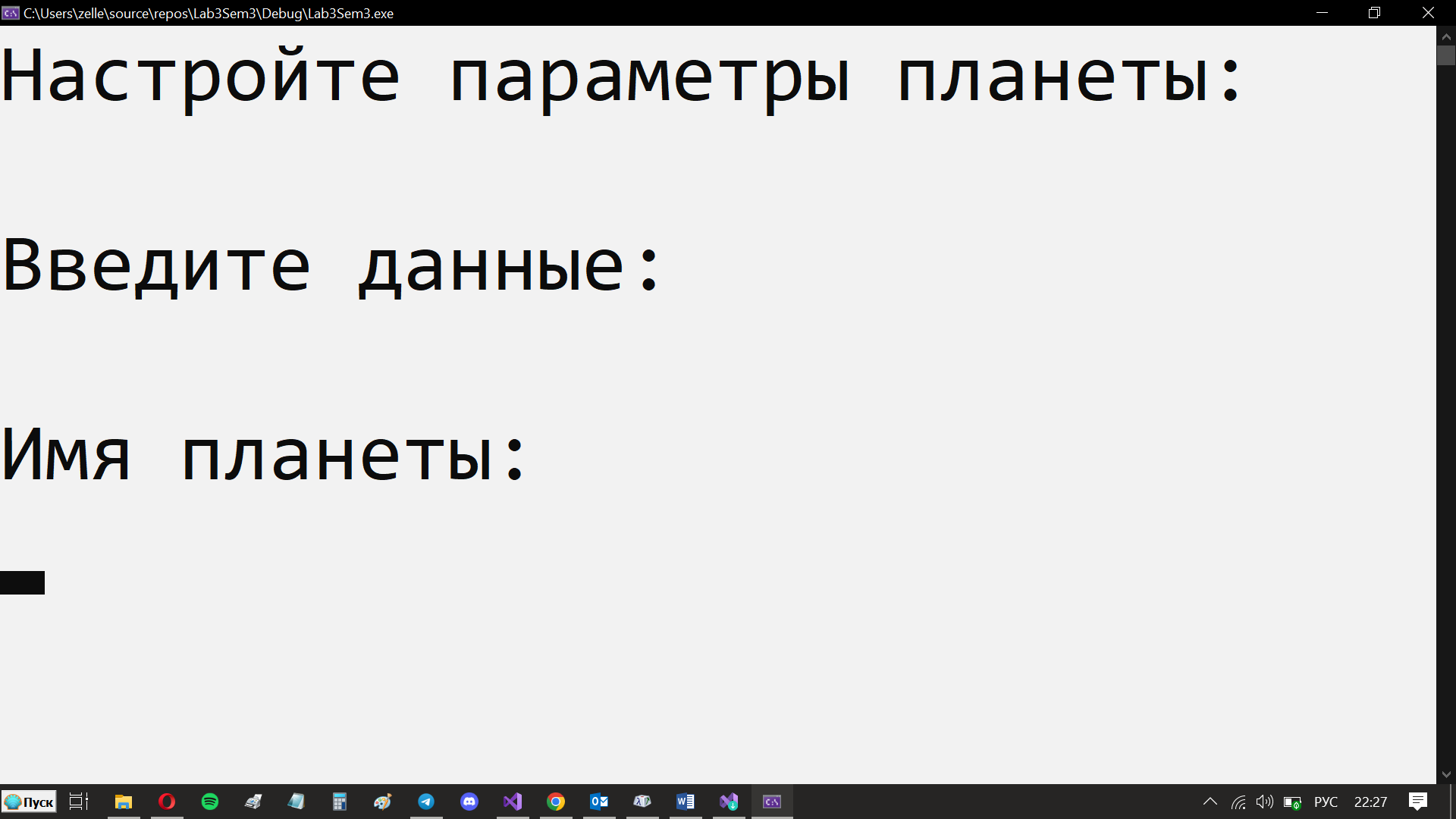
void KARA\_GOSPODNYA()

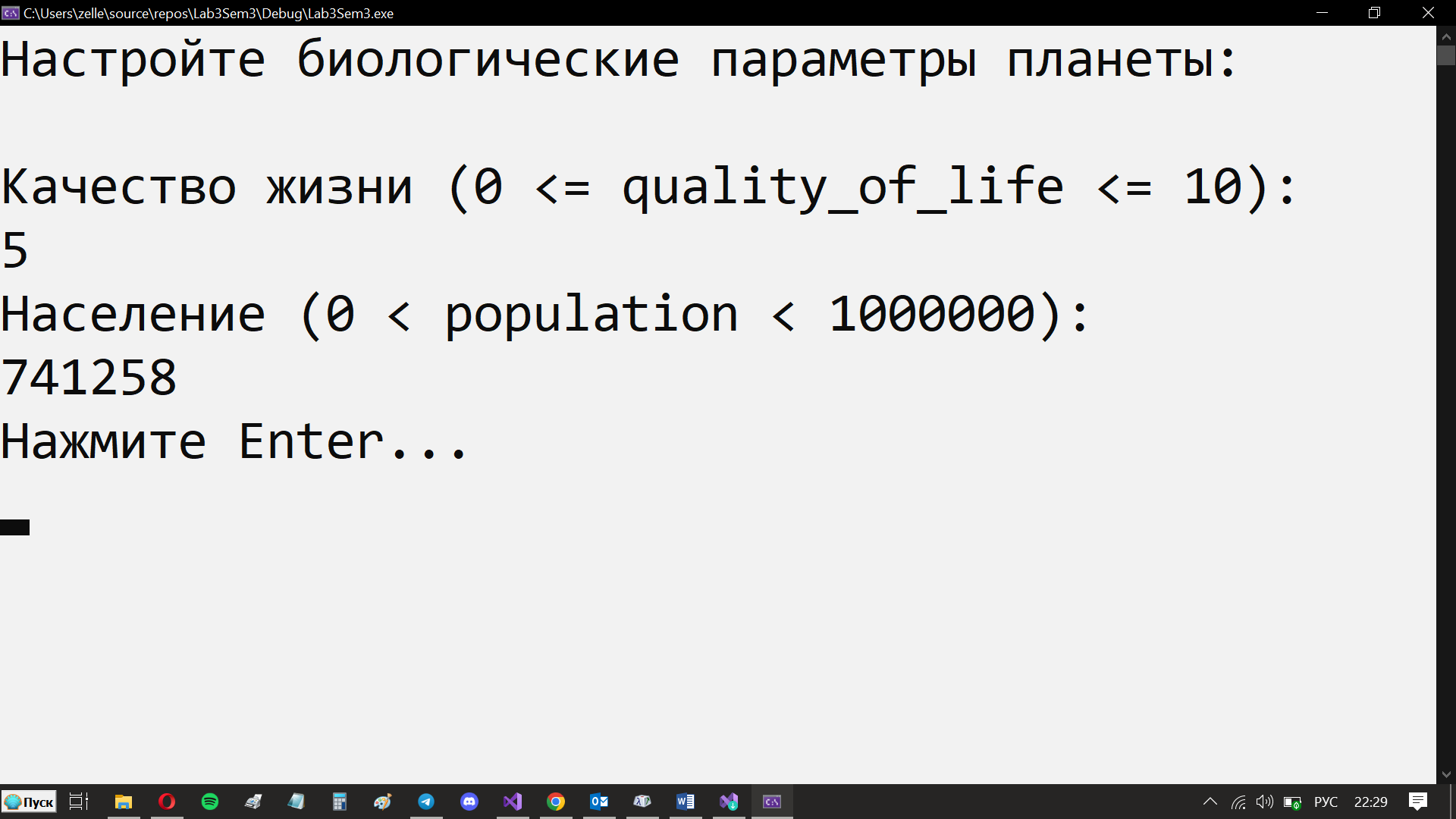
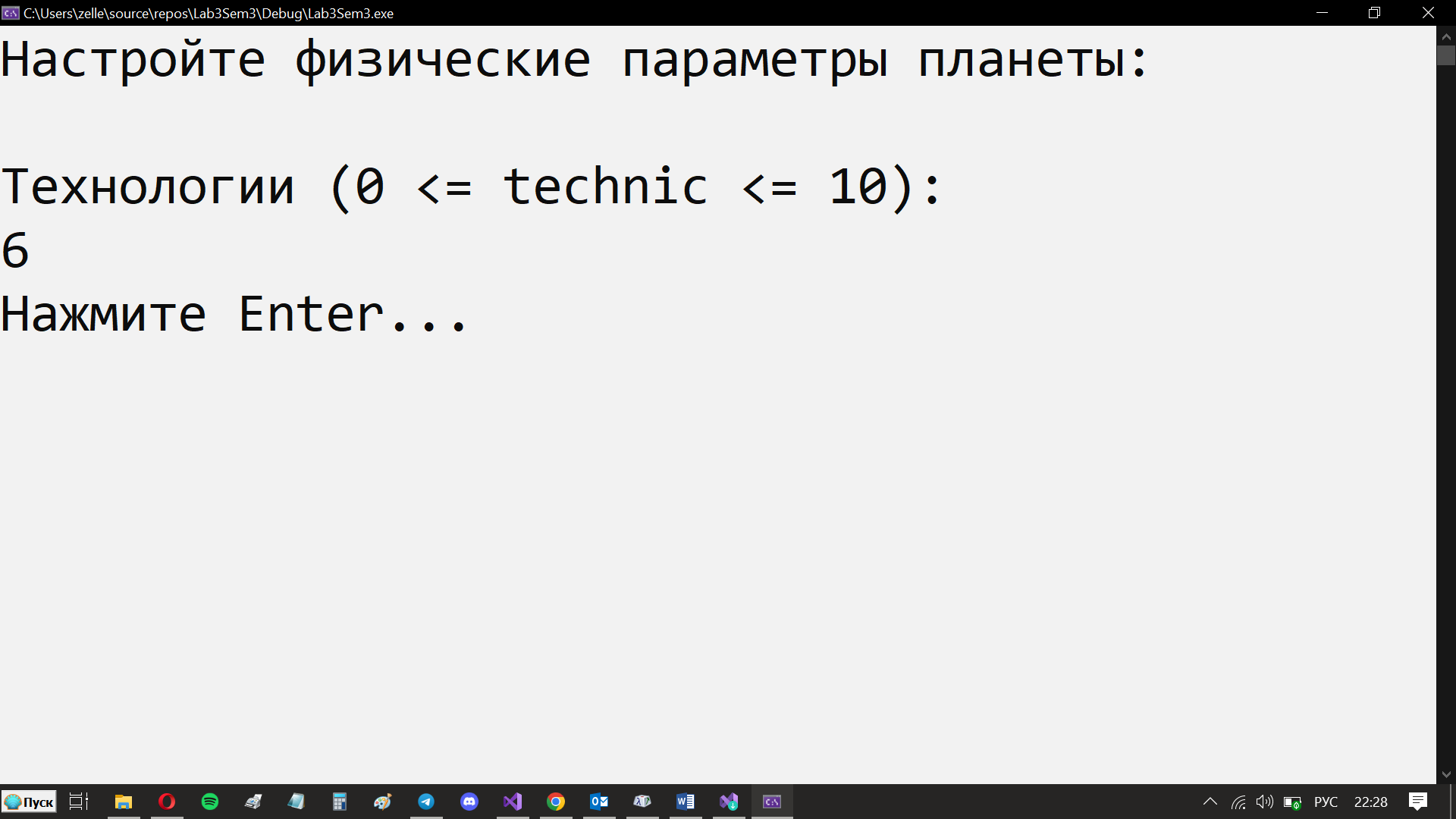
*Вывод данных*

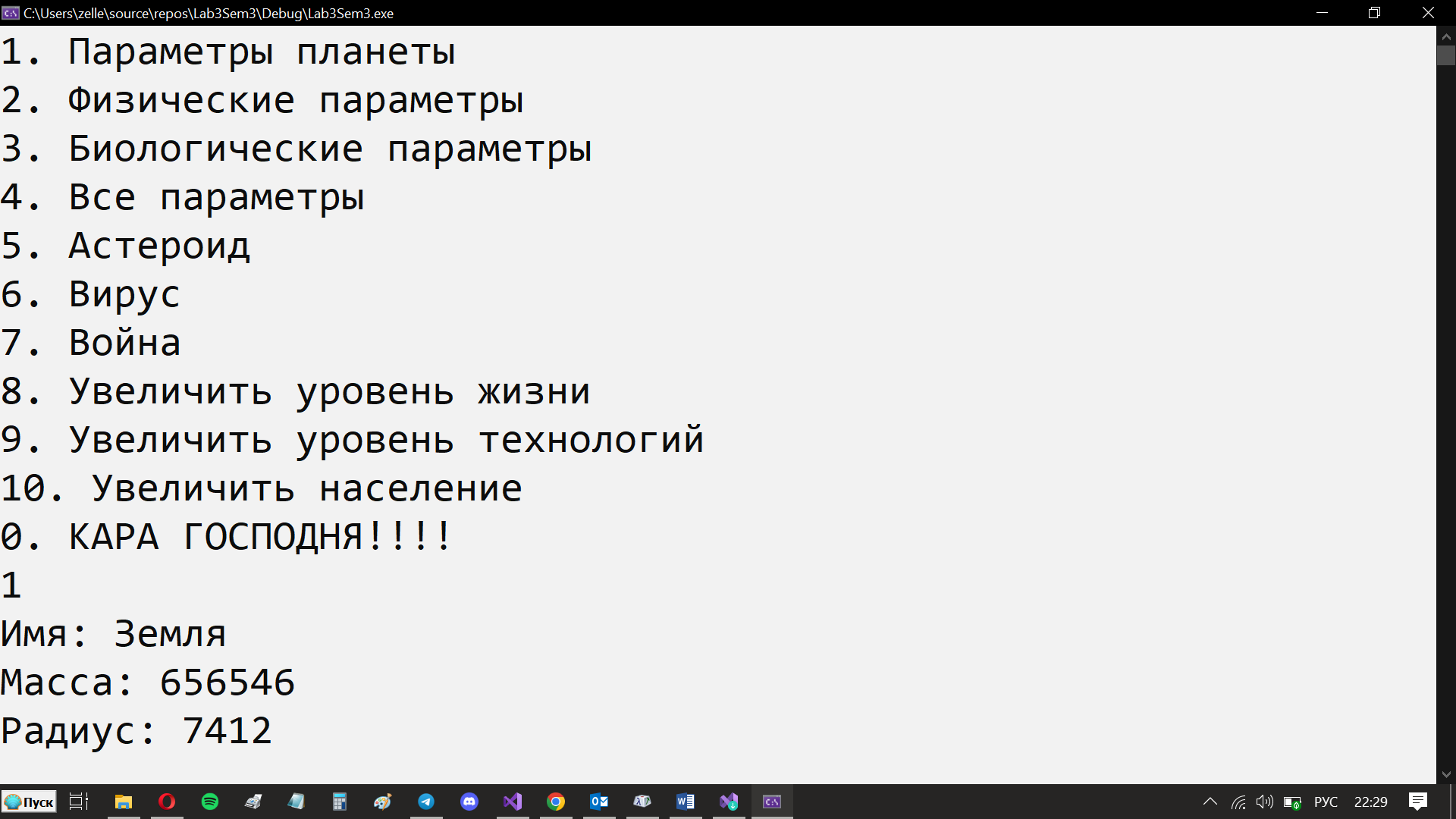
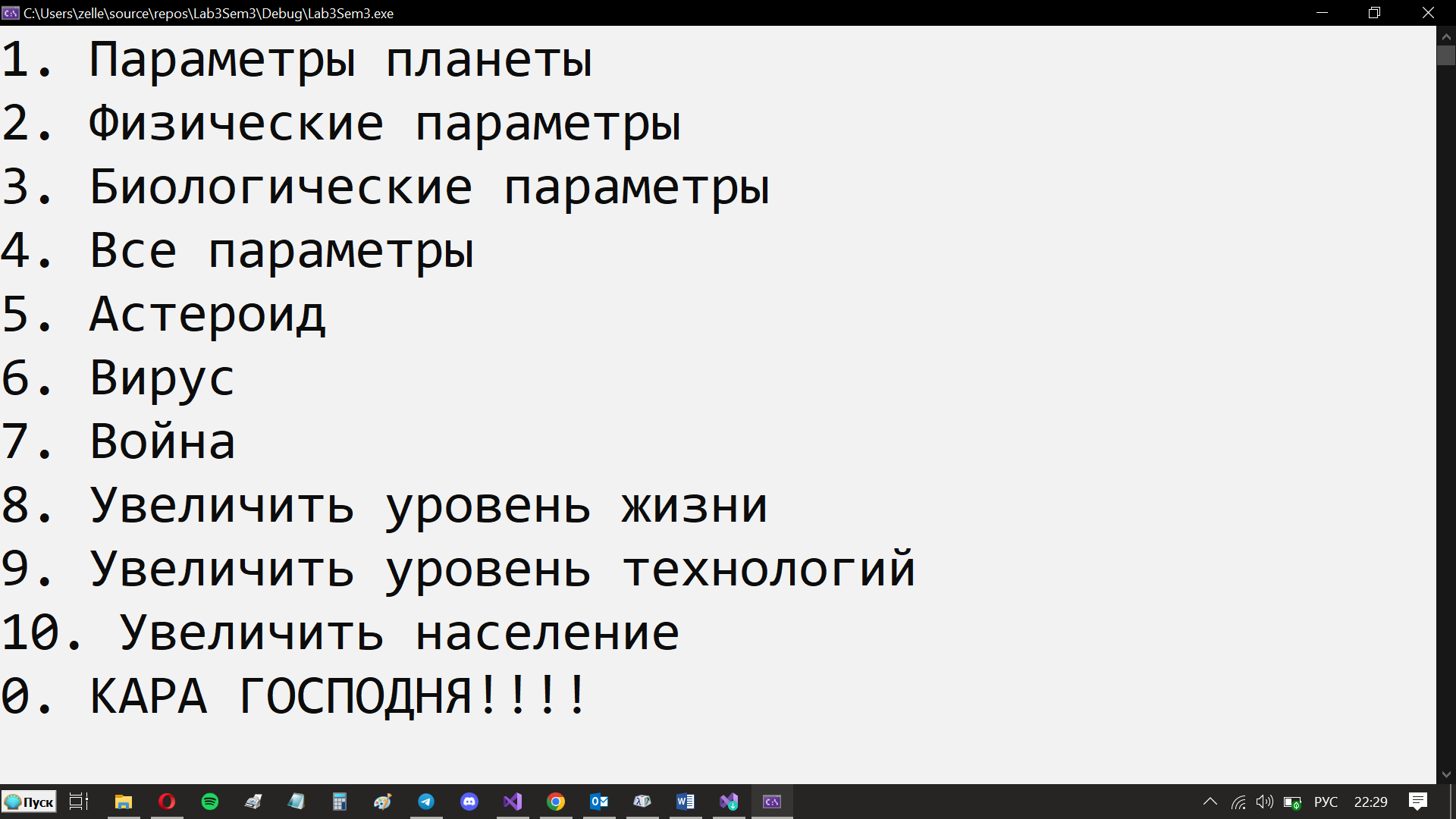
void Print()

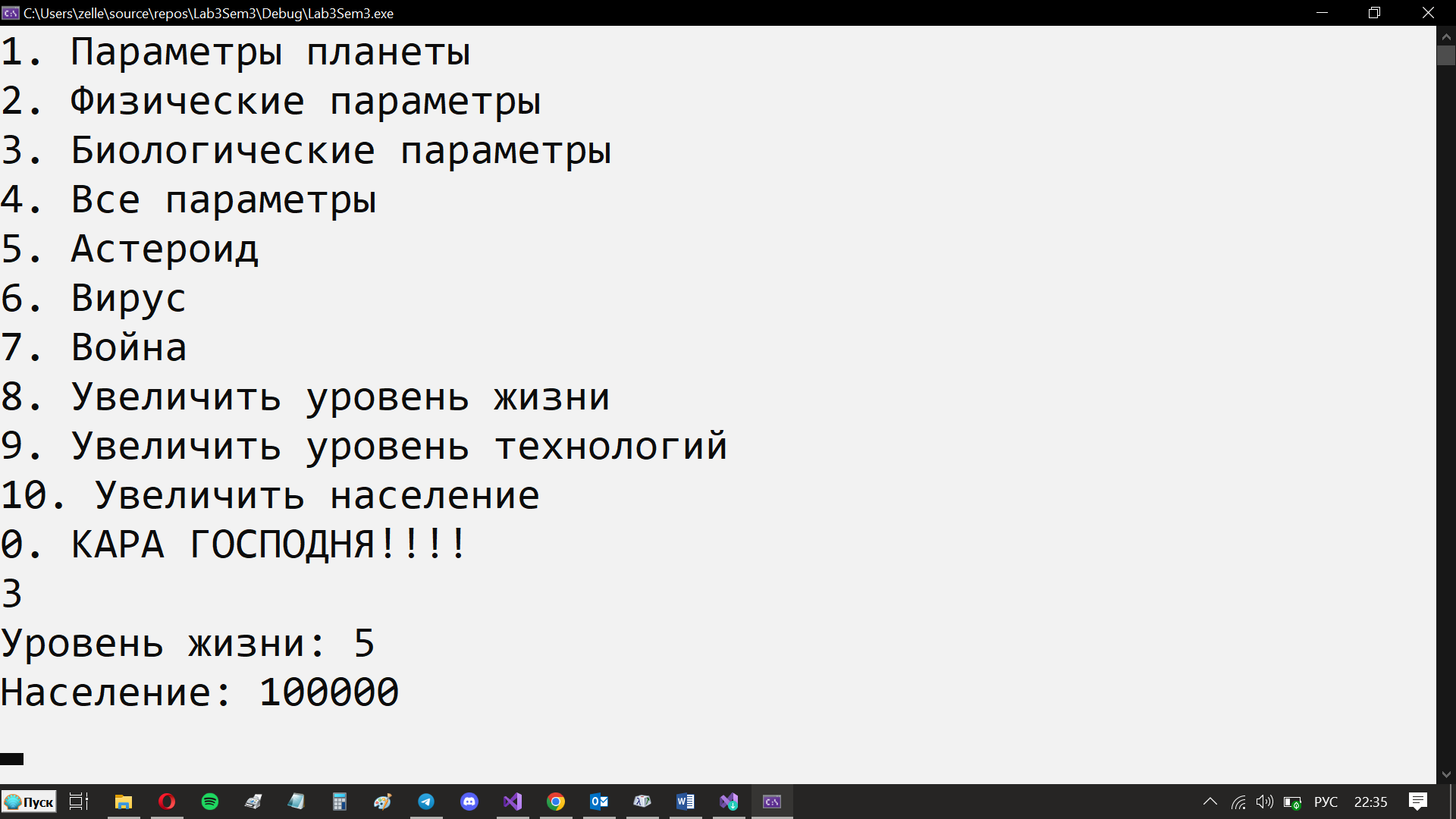
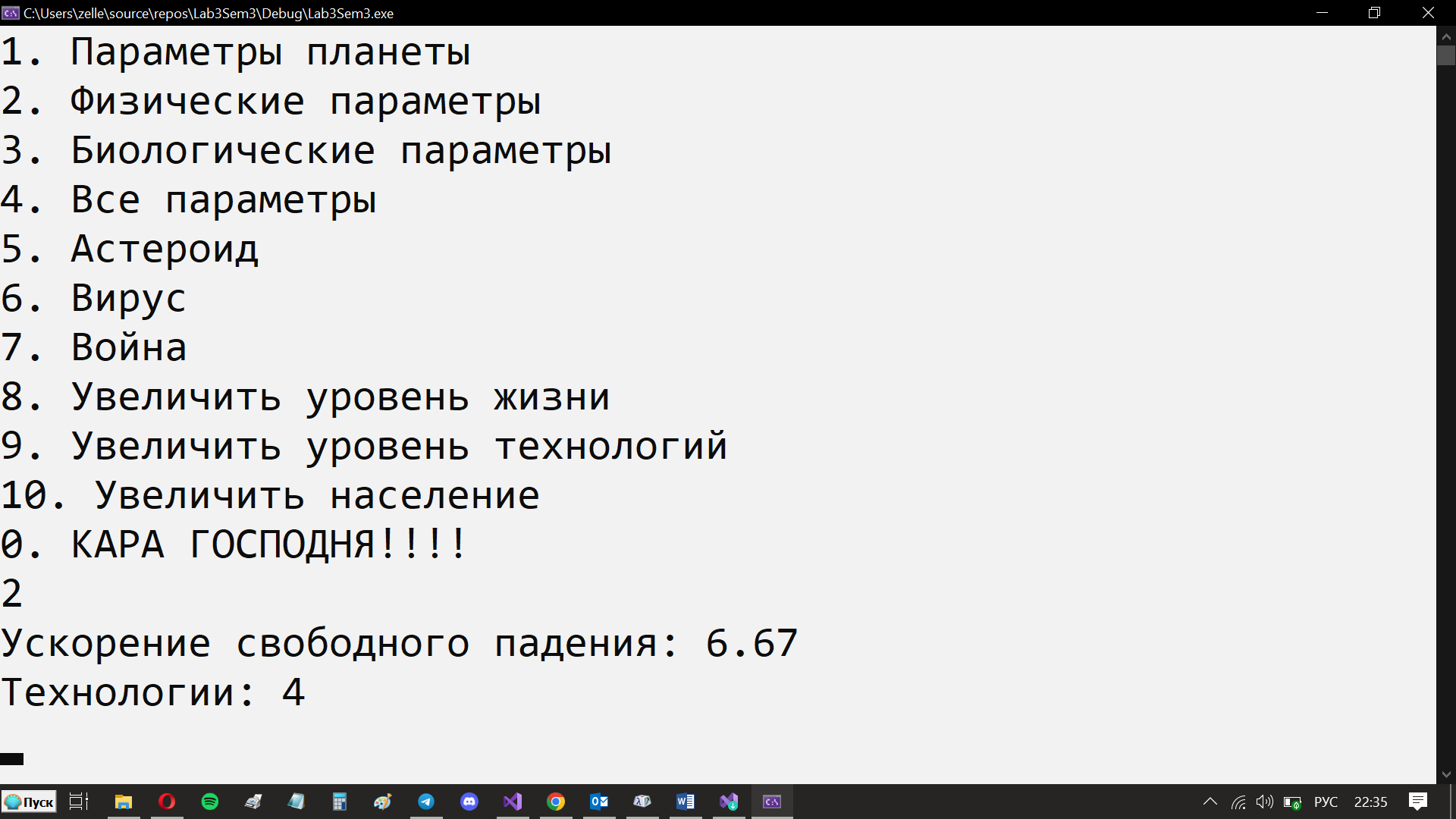
*Деструктор: ~*Space()

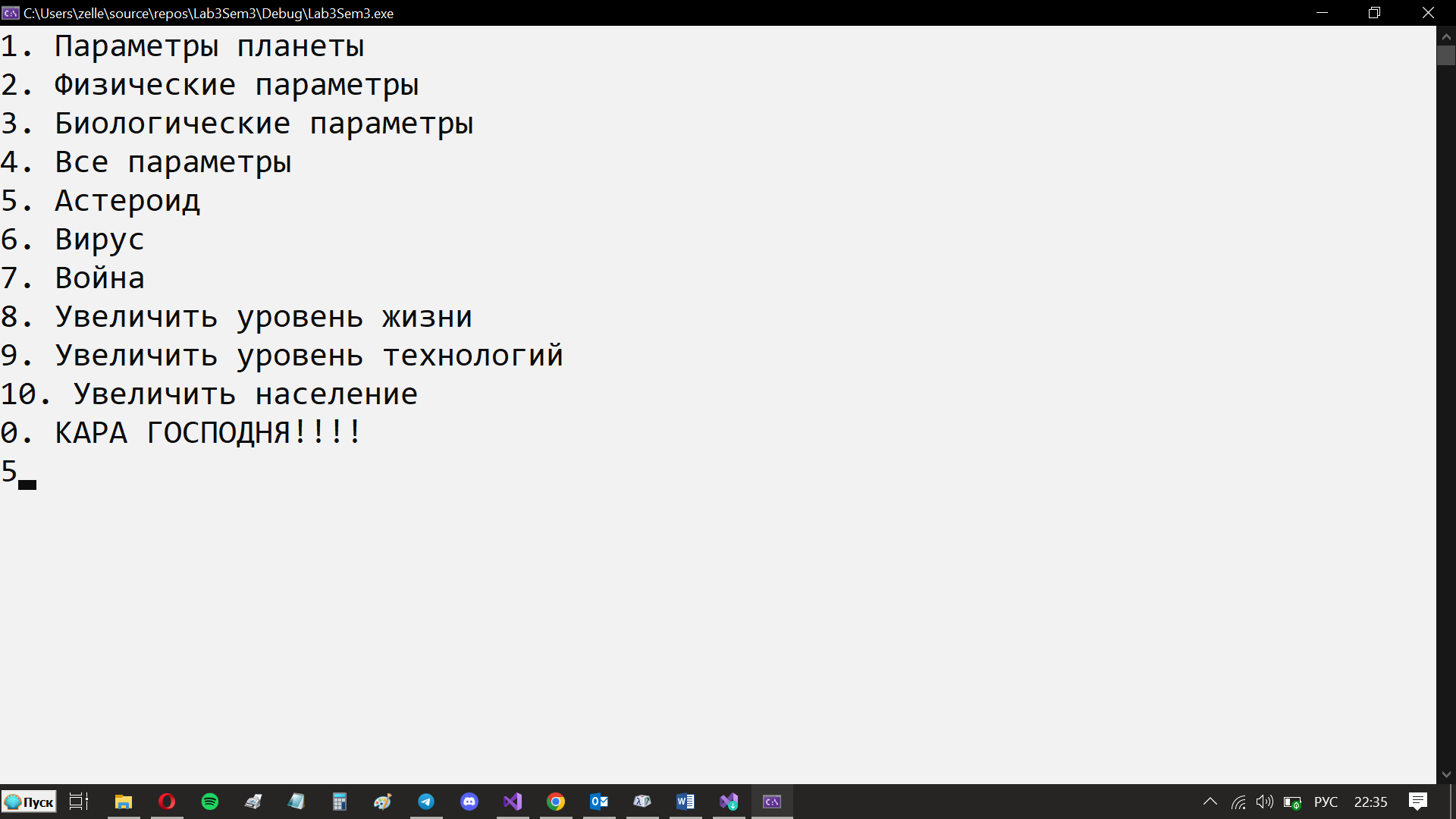
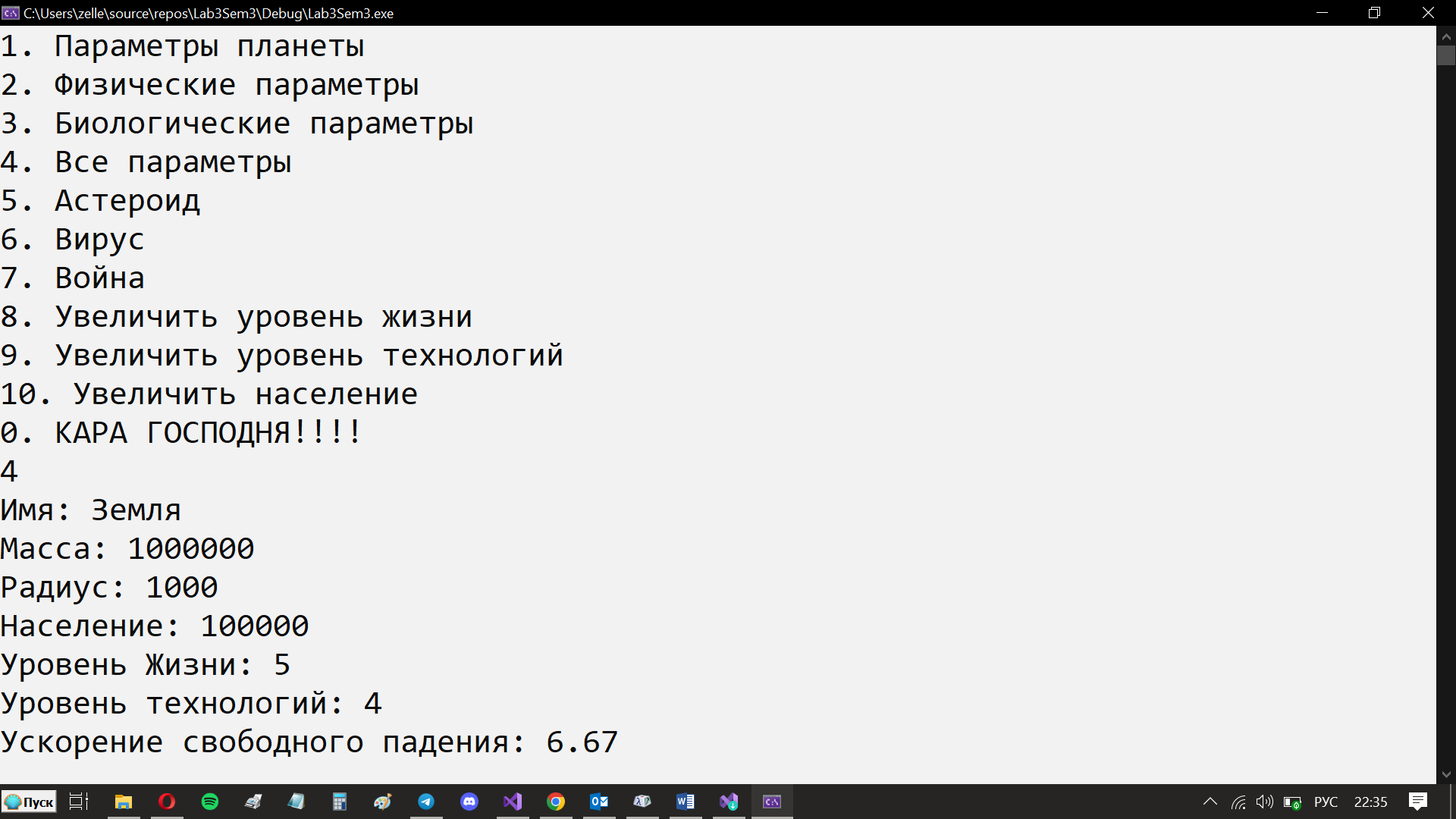
**2.2 Описание пользовательского интерфейса**

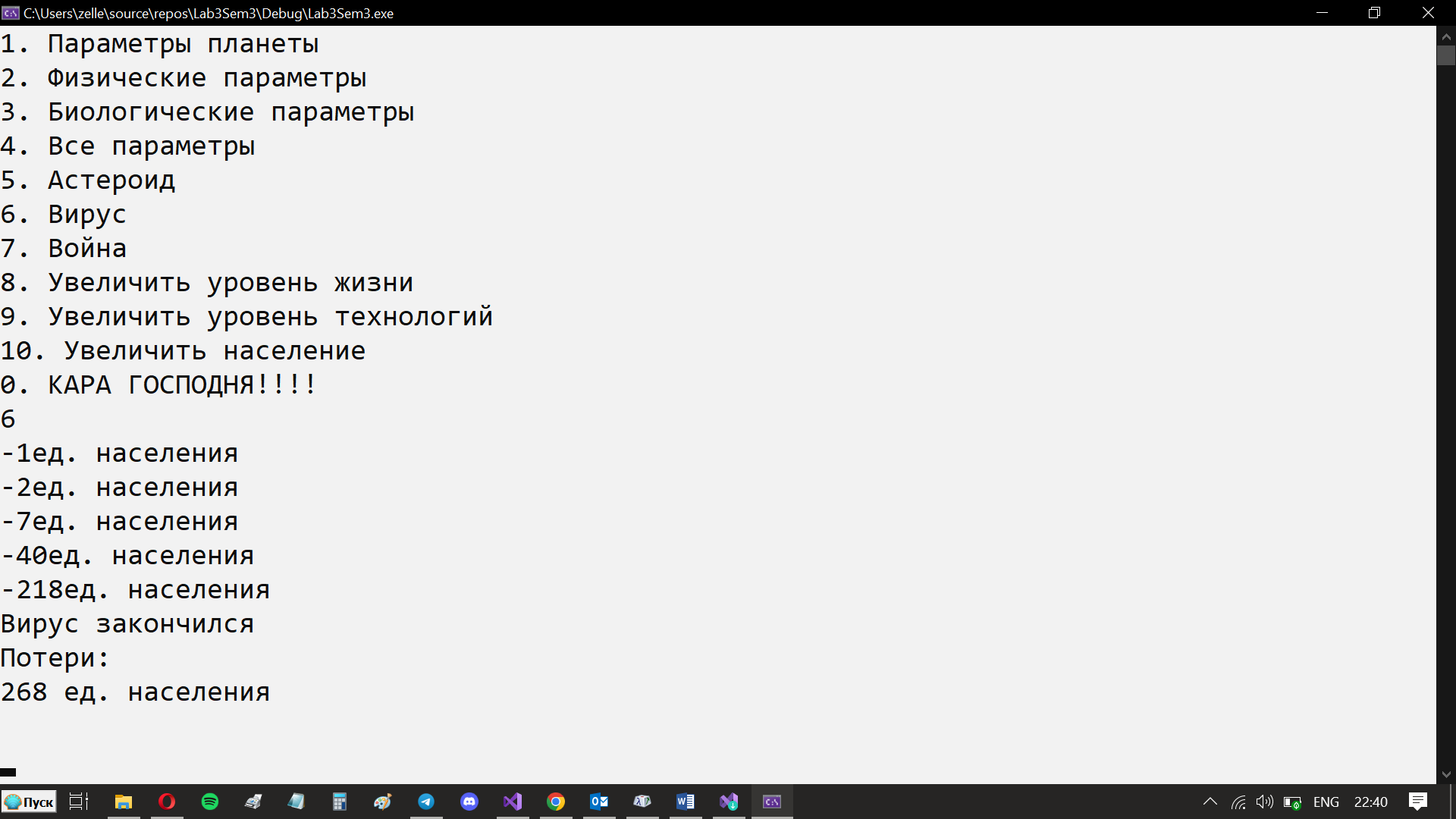
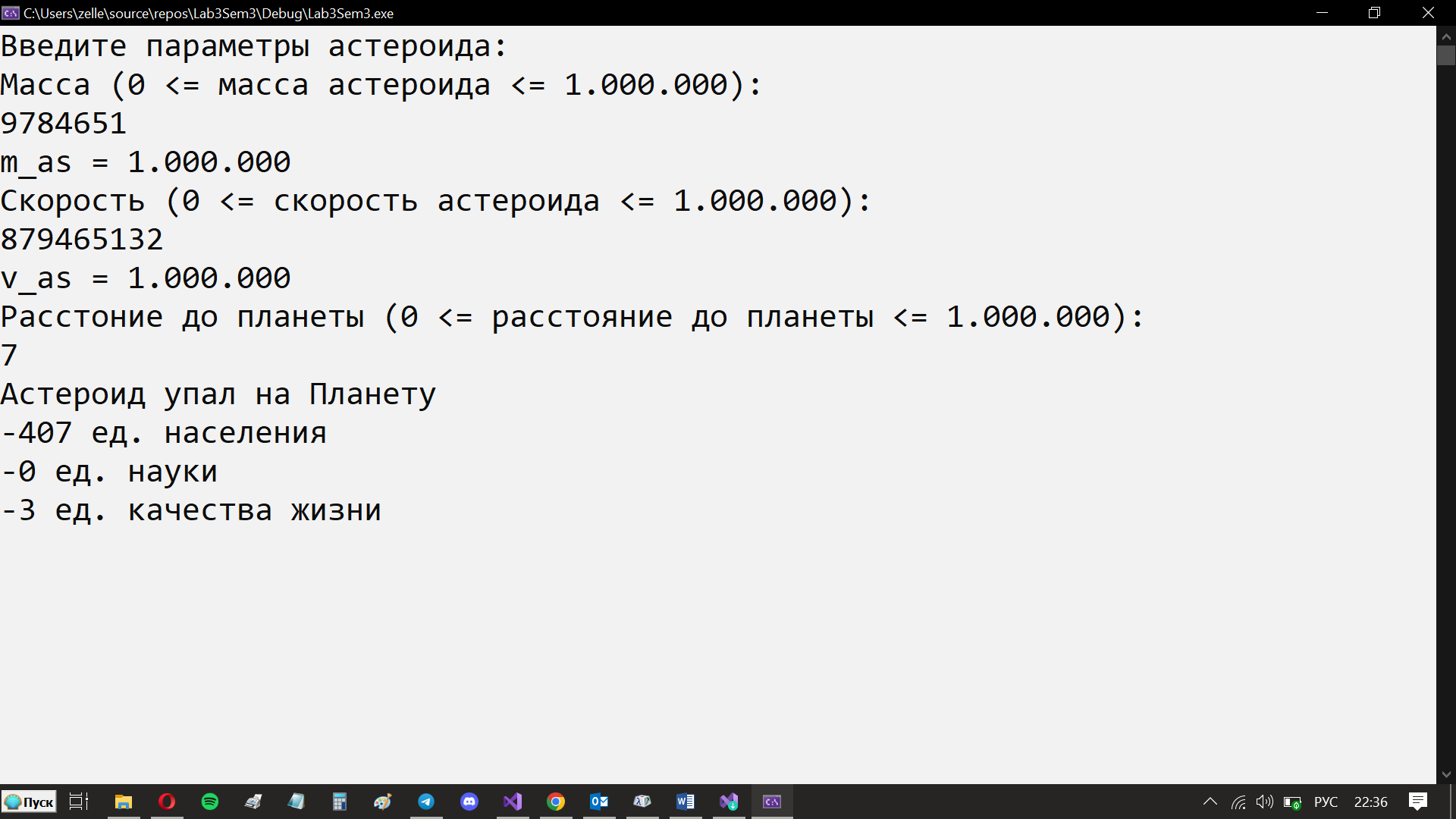


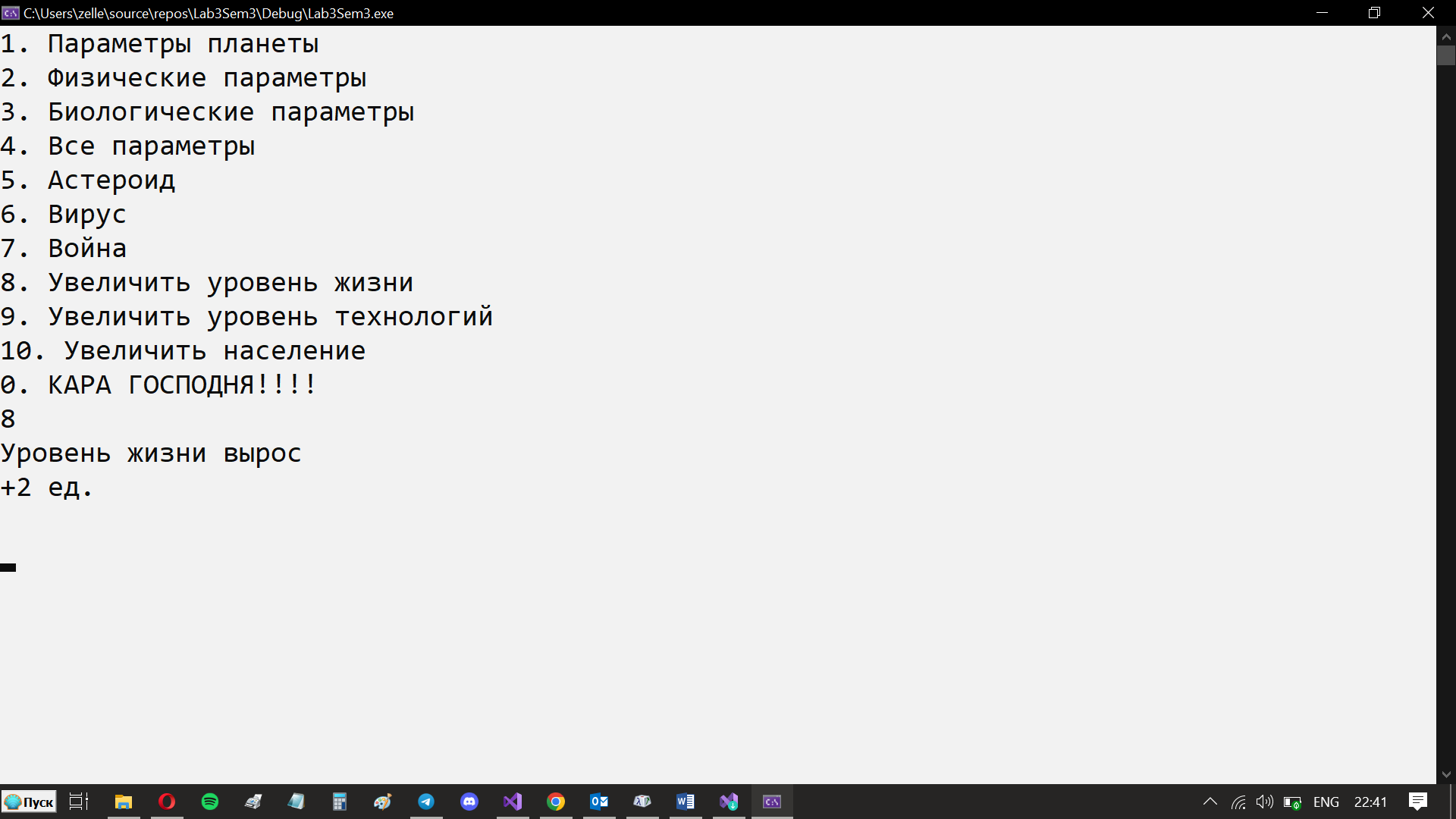
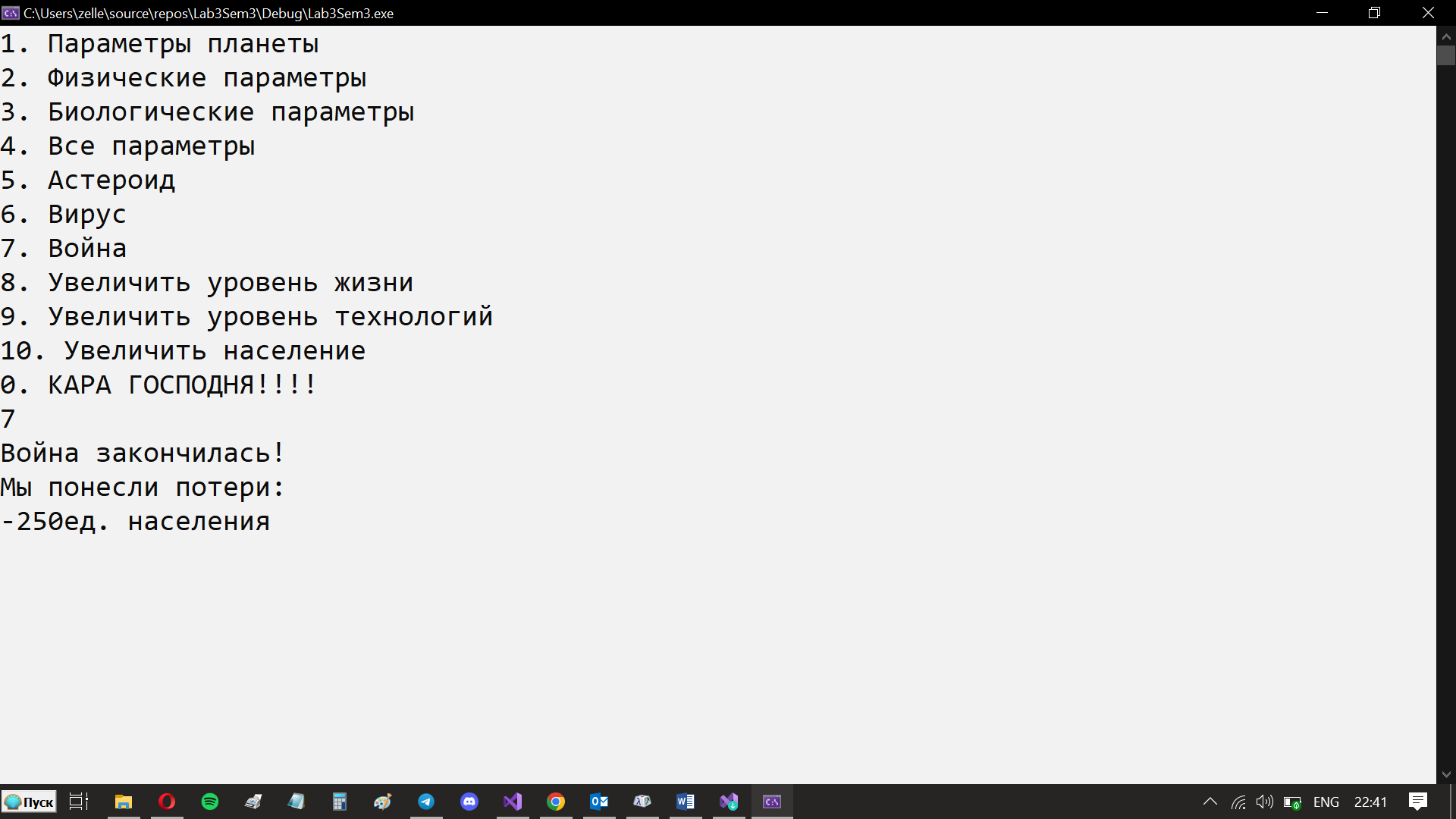


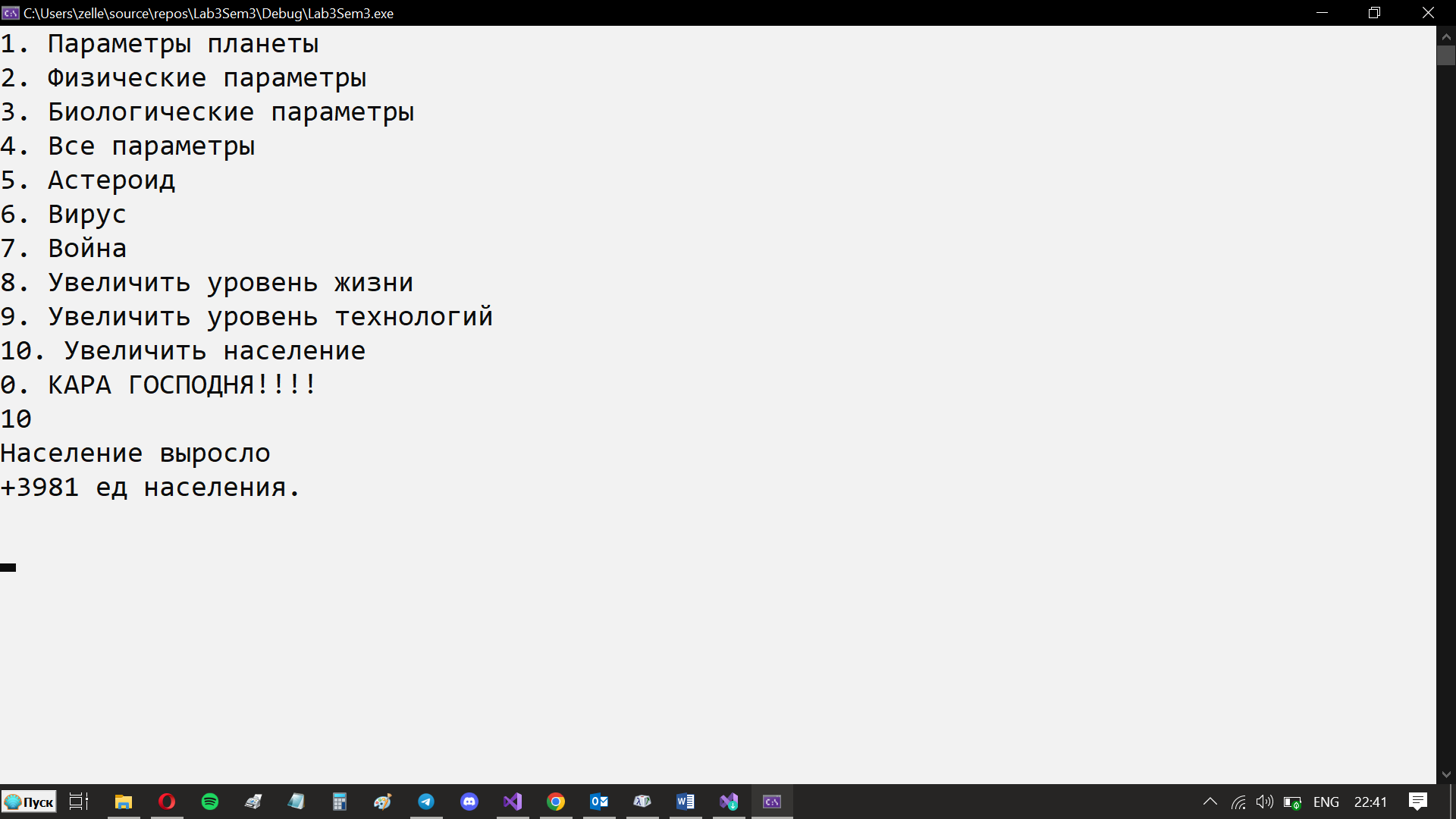
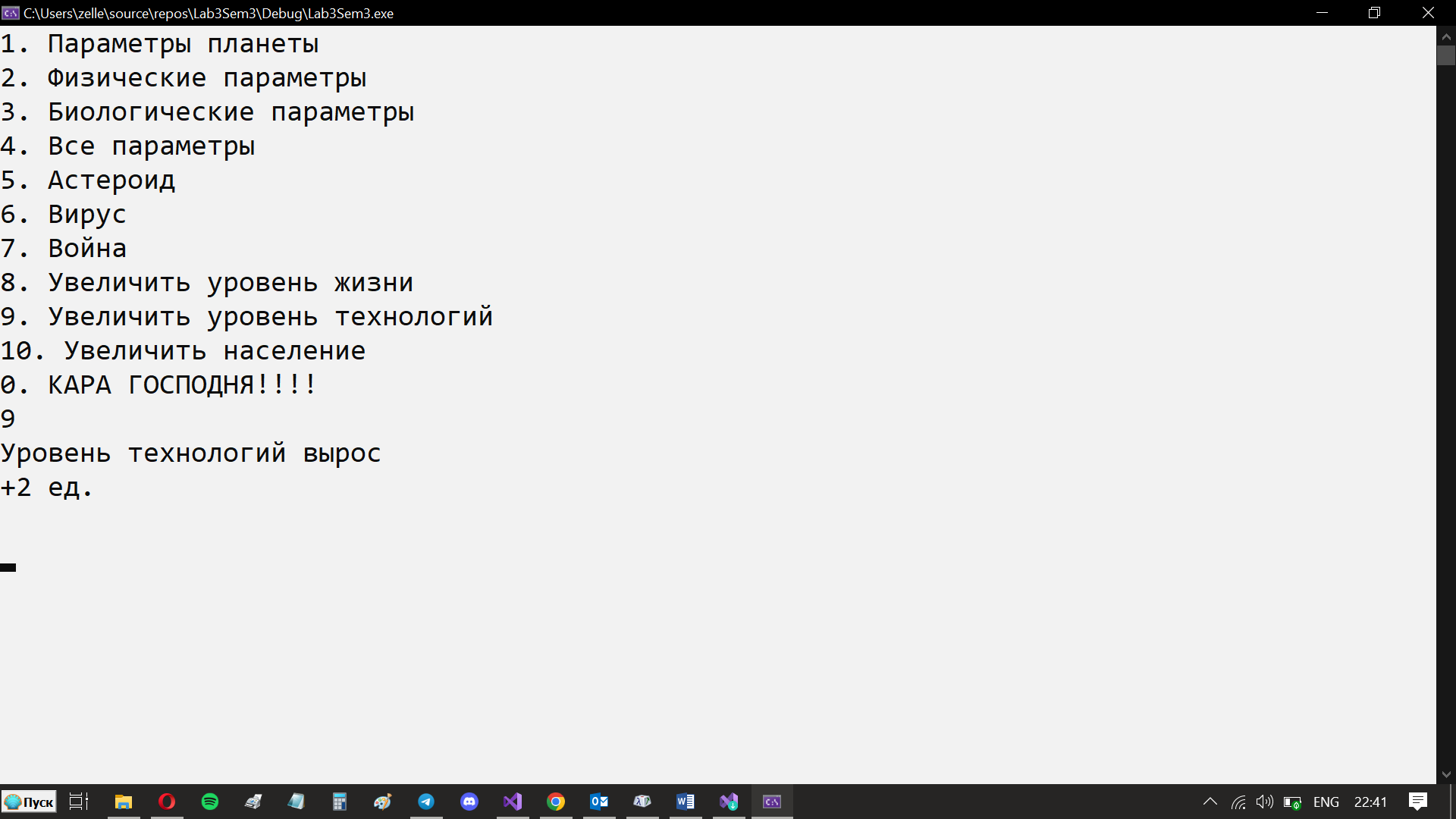


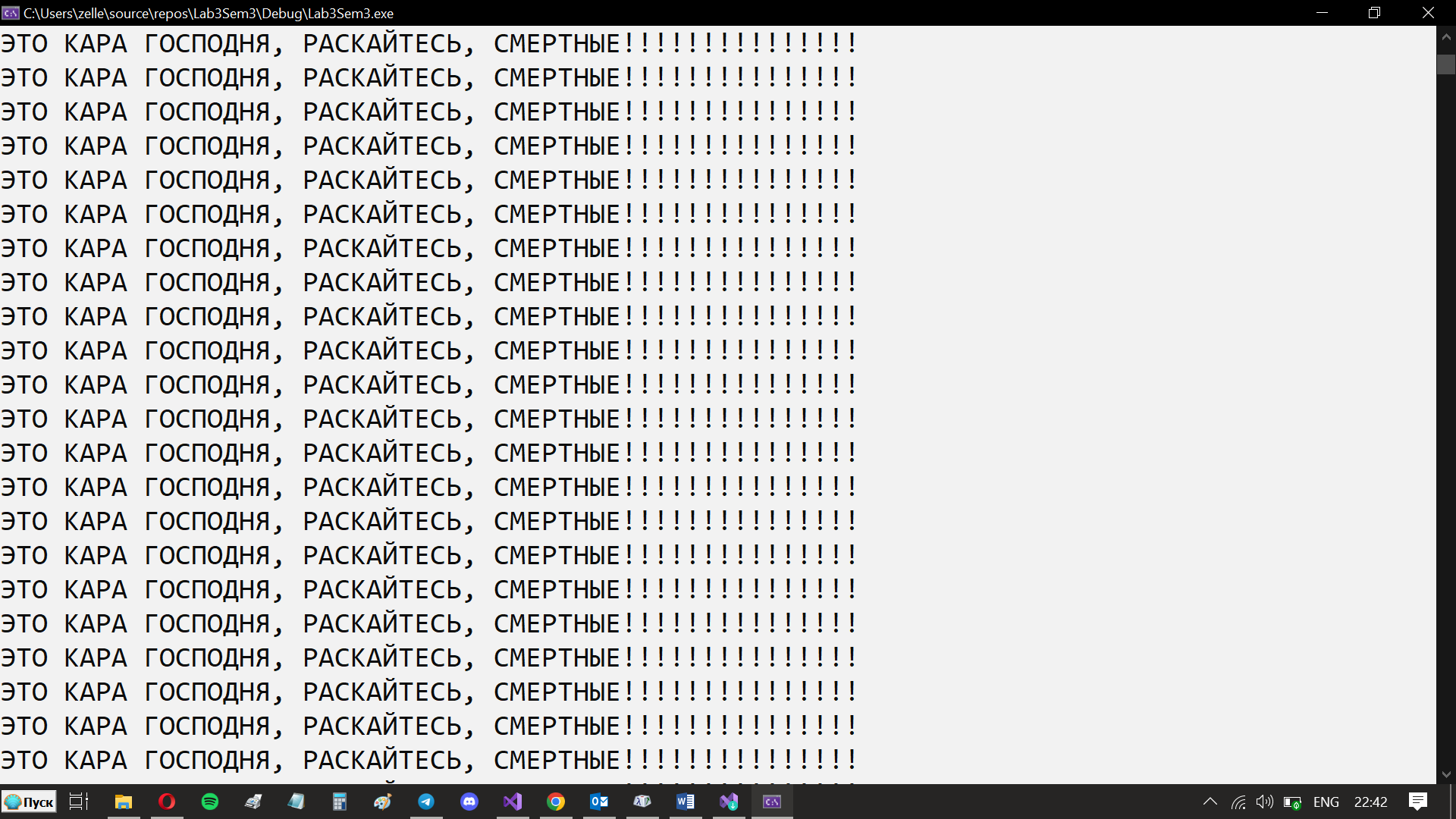
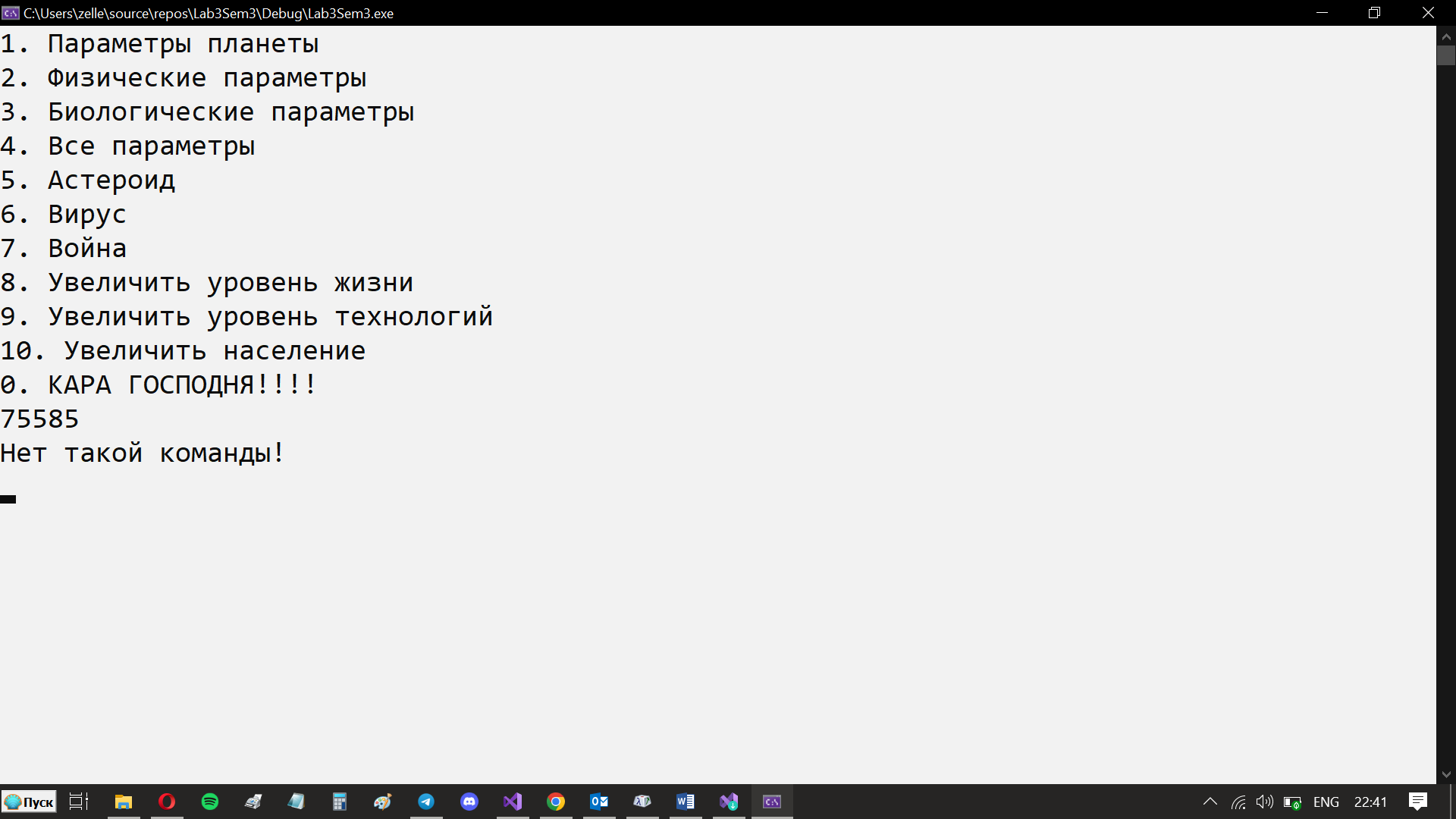












**3. Реализация и тестирование программы**

**3.1. Листинг программы**

#include <iostream>

#include <string>

#include <clocale>

#include <cstring>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <conio.h>

#include <windows.h>

using namespace std;

#pragma warning(suppress : 4996)

class Planet

{

private:

string name = "None";

protected:

int weight = NULL;

int radius = NULL;

public:

Planet() {};

~Planet() {};

Planet(string n, int m, int r) : name(n), weight(m), radius(r){}

string GetName()

{

return name;

}

int GetWeight()

{

return weight;

}

int GetRadius()

{

return radius;

}

void SetName(string Name)

{

name = Name;

}

void SetWeight(int Weight)

{

weight = Weight;

}

void SetRadius(int Radius)

{

radius = Radius;

}

void PlanetPrint()

{

cout << "Имя: " << name << endl;

cout << "Масса: " << weight << endl;

cout << "Радиус: " << radius << endl;

}

};

class Physics : virtual public Planet

{

protected:

int technic = NULL;

double g\_acceler;

public:

Physics(){}

~Physics() {};

double GetGAcceler()

{

g\_acceler = 6.67 \* GetWeight() / pow(GetRadius(), 2);

return g\_acceler;

}

int GetTechnic()

{

return technic;

}

void SetTechnic(int any)

{

technic = any;

}

string IncTechnic()

{

int time\_qol = rand() % 3;

if (technic + time\_qol > 10) technic = 10;

else technic = technic + time\_qol;

return "Уровень технологий вырос\n+" + to\_string(time\_qol) + " ед.\n";

}

void PhysPrint()

{

cout << "Ускорение свободного падения: " << GetGAcceler() << endl;

cout << "Технологии: " << technic << endl;

}

};

class Biology : virtual public Planet

{

protected:

int quality\_of\_life = 0;

int population = NULL;

public:

Biology() {};

~Biology() {};

int GetQOLife()

{

return quality\_of\_life;

}

int GetPopulation()

{

return population;

}

void SetQOLife(int any)

{

quality\_of\_life = any;

}

void SetPopulation(int Population)

{

population = Population;

}

void BioPrint()

{

cout << "Уровень жизни: " << quality\_of\_life << endl;

cout << "Население: " << population << endl;

}

string IncPop()

{

int time\_pop = population / (round(log10(population))\*5);

population += population / (round(log10(population)) \* 5);

return "Население выросло\n+" + to\_string(time\_pop) + " ед населения.\n";

}

string IncQOLife()

{

int time\_qol = rand() % 3;

if (quality\_of\_life + time\_qol > 10) quality\_of\_life = 10;

else quality\_of\_life = quality\_of\_life + time\_qol;

return "Уровень жизни вырос\n+" + to\_string(time\_qol) + " ед.\n";

}

};

class Space : public Biology, public Physics

{

private:

int asteroid = 0;

public:

Space(){}

~Space() {}

int GetAst()

{

return asteroid;

}

string Asteroid(int m, int v, int S)

{

if ((1.0 \* m\*v\*v/2 > GetGAcceler() \* m \* S) && ((rand() % 100) >= 70))

{

int die\_pop = (rand() % 1000);

int time\_population = GetPopulation() - die\_pop;

SetPopulation(time\_population);

int low\_technic = abs(rand() % 5);

int time\_technic = GetTechnic() - low\_technic;

SetTechnic(time\_technic);

int low\_quality\_of\_life = abs(rand() % 5);

int time\_quality\_of\_life = GetQOLife() - low\_quality\_of\_life;

SetQOLife(time\_quality\_of\_life);

string answer = "Астероид упал на Планету \n-" + to\_string(die\_pop) + " ед. населения\n-" + to\_string(low\_technic) + " ед. науки\n-" + to\_string(low\_quality\_of\_life) + " ед. качества жизни\n";

asteroid++;

return answer;

}

else

{

string answer = "Астероид пролетел мимо Планеты";

return answer;

}

}

string War()

{

int random = rand() % 10;

if (GetQOLife() > 9)

{

return "Война не возможна, Люди слишком благоразумны, Милорд";

}

else if ((6 < GetQOLife() <= 9) && (random >= 7))

{

int die\_pop = round((GetPopulation() / 3) % 100 \* ((GetTechnic()+1) \* 2));

int time\_pop = GetPopulation() - die\_pop;

SetPopulation(time\_pop);

if (die\_pop == 0) return "Война закончилась! \nПотерь нет!\n";

else return "Война закончилась! \nМы понесли потери: \n-" + to\_string(die\_pop) + "ед. населения\n";

}

else if ((3 < GetQOLife() <= 6) && (random >= 4))

{

int die\_pop = round((GetPopulation() / 3) % 150 \* ((GetTechnic()+1) \* 3));

int time\_pop = GetPopulation() - die\_pop;

SetPopulation(time\_pop);

if (die\_pop == 0) return "Война закончилась! \nПотерь нет!\n";

else return "Война закончилась! \nМы понесли потери: \n-" + to\_string(die\_pop) + " ед. населения\n";

}

else if ((1 < GetQOLife() <= 3) && (random >= 1))

{

int die\_pop = round((GetPopulation() / 3) % 200 \* ((GetTechnic()+1) \* 4));

int time\_pop = GetPopulation() - die\_pop;

SetPopulation(time\_pop);

if (die\_pop == 0) return "Война закончилась! \nПотерь нет!\n";

else return "Война закончилась! \nМы понесли потери: \n-" + to\_string(die\_pop) + "ед. населения\n";

}

else if (0<= GetQOLife()<=1)

{

int die\_pop = round((GetPopulation() / 3) % 250 \* ((GetTechnic()+1) \* 5));

int time\_pop = GetPopulation() - die\_pop;

SetPopulation(time\_pop);

if (die\_pop == 0) return "Война закончилась! \nПотерь нет!\n";

else return "Война закончилась!\nМы понесли потери: \n-" + to\_string(die\_pop) + "ед. населения\n";

}

else

{

return "Войны не случилось, Массонская ложе все решило за вас\n";

}

}

string Virus()

{

int total = 0;

if (GetPopulation() <= 0)

{

return "Людей нет, все вымерли\n";

}

else if (technic == 0 && quality\_of\_life == 0)

{

int last\_pop = GetPopulation();

for (size\_t i = 0; i < log10(last\_pop); i++)

{

int time\_pop = exp(i) \* (rand() % ((i+1) \* (i+1)));

total += time\_pop;

if (time\_pop > population)

{

total -= (time\_pop - population);

cout << i + 1 << "волна: " << endl;

cout << "-" + to\_string(population) + "ед. населения" << endl;

Sleep(1000);

SetPopulation(0);

break;

}

else

{

cout << i + 1 << "волна: " << endl;

cout << "-" + to\_string(time\_pop) + "ед. населения" << endl;

Sleep(1000);

SetPopulation(last\_pop - time\_pop);

}

if (GetPopulation() <= 0)

{

return "Людей нет, все вымерли\n";

}

}

if (GetPopulation() <= 0)

{

SetPopulation(0);

return "Людей нет, все вымерли\n";

}

else return "Вирус закончился\nПотери:\n" + to\_string(total) + " ед. населения\n";

}

else

{

int last\_pop = GetPopulation();

for (size\_t i = 0; i < log10(last\_pop); i++)

{

int time\_pop = exp(i) \* (rand() % (i + 1) + 1) - (technic \* quality\_of\_life \* 5) % ((rand() % (i + 1))+1);

total += time\_pop;

if (time\_pop > population)

{

total -= (time\_pop - population);

cout << "-" + to\_string(population) + "ед. населения" << endl;

Sleep(1000);

SetPopulation(0);

break;

}

else

{

cout << "-" + to\_string(time\_pop) + "ед. населения" << endl;

Sleep(1000);

SetPopulation(last\_pop - time\_pop);

}

if (GetPopulation() <= 0)

{

return "Людей нет, все вымерли\n";

}

}

if (GetPopulation() <= 0)

{

SetPopulation(0);

return "Людей нет, все вымерли\n";

}

else return "Вирус закончился\nПотери:\n" + to\_string(total) + " ед. населения\n";

}

}

void KARA\_GOSPODNYA()

{

for (size\_t i = 0; i < 1000; i++)

{

cout << "ЭТО КАРА ГОСПОДНЯ, РАСКАЙТЕСЬ, СМЕРТНЫЕ!!!!!!!!!!!!!!!\n";

}

exit(0);

}

void Print()

{

cout << "Имя: " << GetName() << endl;

cout << "Масса: " << GetWeight() << endl;

cout << "Радиус: " << GetRadius() << endl;

cout << "Население: " << GetPopulation() << endl;

cout << "Уровень Жизни: " << GetQOLife() << endl;

cout << "Уровень технологий: " << GetTechnic() << endl;

cout << "Ускорение свободного падения: " << GetGAcceler() << endl;

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

system("chcp 1251 >> null");

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

cout << "Настройте параметры планеты:" << endl << endl;

string n;

int m, r, tech, g\_ac, pop, qol, ast, al;

int m\_as, v\_as, S\_as;

cout << "Введите данные: \n\n";

cout << "Имя планеты: \n";

cin >> n;

cout << "Масса планеты (10.000<=m<=1.000.000): \n";

string val;

cin >> val;

try {

m = stoi(val);

if (m < 10000 || m > 1000000) {

throw out\_of\_range("Неверный диапазон");

}

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\nm = 1.000.000" << endl;

m = 1000000;

}

catch (const out\_of\_range& e) {

cout << "m = 1.000.000" << endl;

m = 1000000;

}

cout << "Радиус планеты (100<=r<=10.000): \n";

cin >> val;

try {

r = stoi(val);

if (r < 100 || r > 10000) {

throw out\_of\_range("Неверный диапазон");

}

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\nr = 10.000" << endl;

r = 10000;

}

catch (const out\_of\_range& e) {

cout << "r = 10.000" << endl;

r = 10000;

}

Planet Earth(n, m, r);

cout << "Нажмите Enter..." << endl;

\_getch();

system("cls");

Physics PhEarth;

cout << "Настройте физические параметры планеты: " << endl << endl;

cout << "Технологии (0 <= technic <= 10): \n";

cin >> val;

try {

tech = stoi(val);

if (tech < 0 || tech > 10) {

throw out\_of\_range("Неверный диапазон");

}

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\ntech = 0" << endl;

tech = 0;

}

catch (const out\_of\_range& e) {

cout << "tech = 0" << endl;

tech = 0;

}

cout << "Нажмите Enter..." << endl;

\_getch();

system("cls");

PhEarth.SetName(Earth.GetName());

PhEarth.SetRadius(Earth.GetRadius());

PhEarth.SetWeight(Earth.GetWeight());

PhEarth.SetTechnic(tech);

Biology BioEarth;

cout << "Настройте биологические параметры планеты: " << endl << endl;

cout << "Качество жизни (0 <= quality\_of\_life <= 10):\n";

cin >> val;

try {

qol = stoi(val);

if (qol < 0 || qol > 10) {

throw out\_of\_range("Неверный диапазон");

}

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\nqol = 0" << endl;

qol = 0;

}

catch (const out\_of\_range& e) {

cout << "qol = 0" << endl;

qol = 0;

}

cout << "Население (0 < population < 1000000): \n";

cin >> val;

try {

pop = stoi(val);

if (pop < 0 || pop > 1000000) {

throw out\_of\_range("Неверный диапазон");

}

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\npop = 1.000.000" << endl;

pop = 1000000;

}

catch (const out\_of\_range& e) {

cout << "pop = 1.000.000" << endl;

pop = 1000000;

}

cout << "Нажмите Enter..." << endl;

\_getch();

system("cls");

BioEarth.SetQOLife(qol);

BioEarth.SetPopulation(pop);

Space Cosmos;

Cosmos.SetPopulation(BioEarth.GetPopulation());

Cosmos.SetQOLife(BioEarth.GetQOLife());

Cosmos.SetTechnic(PhEarth.GetTechnic());

Cosmos.SetWeight(m);

Cosmos.SetRadius(r);

Cosmos.SetName(n);

int k = 1;

while (k != 0)

{

cout << "1. Параметры планеты" << endl;

cout << "2. Физические параметры" << endl;

cout << "3. Биологические параметры" << endl;

cout << "4. Все параметры" << endl;

cout << "5. Астероид" << endl;

cout << "6. Вирус" << endl;

cout << "7. Война" << endl;

cout << "8. Увеличить уровень жизни" << endl;

cout << "9. Увеличить уровень технологий" << endl;

cout << "10. Увеличить население" << endl;

cout << "0. КАРА ГОСПОДНЯ!!!!" << endl;

cin >> val;

try {

k = stoi(val);

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\nk = 1" << endl;

k = 1;

}

switch (k)

{

case 1:

{

Earth.PlanetPrint();

\_getch();

break;

}

case 2:

{

PhEarth.PhysPrint();

\_getch();

break;

}

case 3:

{

BioEarth.BioPrint();

\_getch();

break;

}

case 4:

{

Cosmos.Print();

\_getch();

break;

}

case 5:

system("cls");

cout << "Введите параметры астероида: " << endl;

cout << "Масса (0 <= масса астероида <= 1.000.000):\n";

cin >> val;

try {

m\_as = stoi(val);

if (m\_as < 0 || m\_as > 1000000) {

throw out\_of\_range("Неверный диапазон");

}

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\nмасса астероида = 1000000" << endl;

m\_as = 1000000;

}

catch (const out\_of\_range& e) {

cout << "m\_as = 1.000.000" << endl;

m\_as = 1000000;

}

cout << "Скорость (0 <= скорость астероида <= 1.000.000):\n";

cin >> val;

try {

v\_as = stoi(val);

if (v\_as < 0 || v\_as > 1000000) {

throw out\_of\_range("Неверный диапазон");

}

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\nскорость астероида = 1000000" << endl;

v\_as = 1000000;

}

catch (const out\_of\_range& e) {

cout << "v\_as = 1.000.000" << endl;

v\_as = 1000000;

}

cout << "Расстоние до планеты (0 <= расстояние до планеты <= 1.000.000):\n";

cin >> val;

try {

S\_as = stoi(val);

if (S\_as < 0 || S\_as > 1000000) {

throw out\_of\_range("Неверный диапазон");

}

}

catch (const invalid\_argument& e) {

cout << "Ошибка: неверный аргумент.\nрасстояние до планеты = 1.000" << endl;

S\_as = 1000;

}

catch (const out\_of\_range& e) {

cout << "S\_as = 1.000" << endl;

S\_as = 1000;

}

cout << Cosmos.Asteroid(m\_as,v\_as,S\_as) << endl;

PhEarth.SetTechnic(Cosmos.GetTechnic());

BioEarth.SetPopulation(Cosmos.GetPopulation());

BioEarth.SetQOLife(Cosmos.GetQOLife());

\_getch();

break;

case 6:

cout << Cosmos.Virus() << endl;

BioEarth.SetPopulation(Cosmos.GetPopulation());

\_getch();

break;

case 7:

cout << Cosmos.War() << endl;

BioEarth.SetPopulation(Cosmos.GetPopulation());

\_getch();

break;

case 9:

cout << PhEarth.IncTechnic() << endl;

\_getch();

Cosmos.SetTechnic(PhEarth.GetTechnic());

break;

case 8:

cout << BioEarth.IncQOLife() << endl;

\_getch();

Cosmos.SetQOLife(BioEarth.GetQOLife());

break;

case 10:

cout << BioEarth.IncPop() << endl;

\_getch();

Cosmos.SetPopulation(BioEarth.GetPopulation());

break;

case 0:

Cosmos.KARA\_GOSPODNYA();

default:

cout << "Нет такой команды!" << endl;

\_getch();

break;

}

system("cls");

}

return 0;

}

**3.2. Тестирование программы**

