ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ

за курсом «Інформатика і Програмування»

студента групи МС-17-1

Гетьмана Ігоря Вікторовича

кафедра комп’ютерних технологій, ДНУ

2019/2020 н.р.

Завдання до лабораторної роботи № 3

за курсом "(Об’єктно-орієнтоване) Програмування"

кафедра комп’ютерних технологій, ДНУ

2018/2019

Тема: Наслідування. Поліморфізм.

Постановка задачі

Скласти об’єктно-орієнтовану програму на мові C++, яка в діалоговому режимі керує графічними об’єктами, що відображаються на екрані дисплею.

По натисненню клавіши F1 програма виводить на екран коротку підказку по усіх наявних командах/клавішах; наприклад: як створити об’єкт, як зрушити з місця, як перейти до «наступного» об’єкту тощо.

Програма повинна підтримувати такі загальні елементи поведінки графічних об’єктів:

1. Активізація/візуалізація графічного об’єкту за вибором користувача.

2. Переміщення зі слідом/без.

3. Відновлення початкового стану об’єкту.

4. Зміна кольору.

5. Зміна стану видимий/невидимий.

6. Агрегація, тобто утворення нових об’єктів з вже створених. При агрегації об’єктів забезпечити можливість агрегації агрегатів.

Програма повинна надати користувачеві можливість зберігати поточну конфігурацію програми у вказаний користувачем текстовий файл на диску і завантажувати поточну конфігурацію програми із вказаного користувачем текстового файлу на диску.

За основу програми можна взяти навчальний приклад «Графіка в консолі» (файл *graphics\_in\_console.7z* – додається; див. наприклад, тут: [*http://khizha.dp.ua/library/graphics\_in\_console/*](http://khizha.dp.ua/library/graphics_in_console/)).

Варіант індивідуального завдання:

1.Ім’я конфігураційного файлу:

0. задане у командному рядку при запуску програми.

2. Рух об’єкта в автоматичному режимі:

0. По заданому закону.

3. Зміна кольору:

0. По команді.

4. Деформація образу:

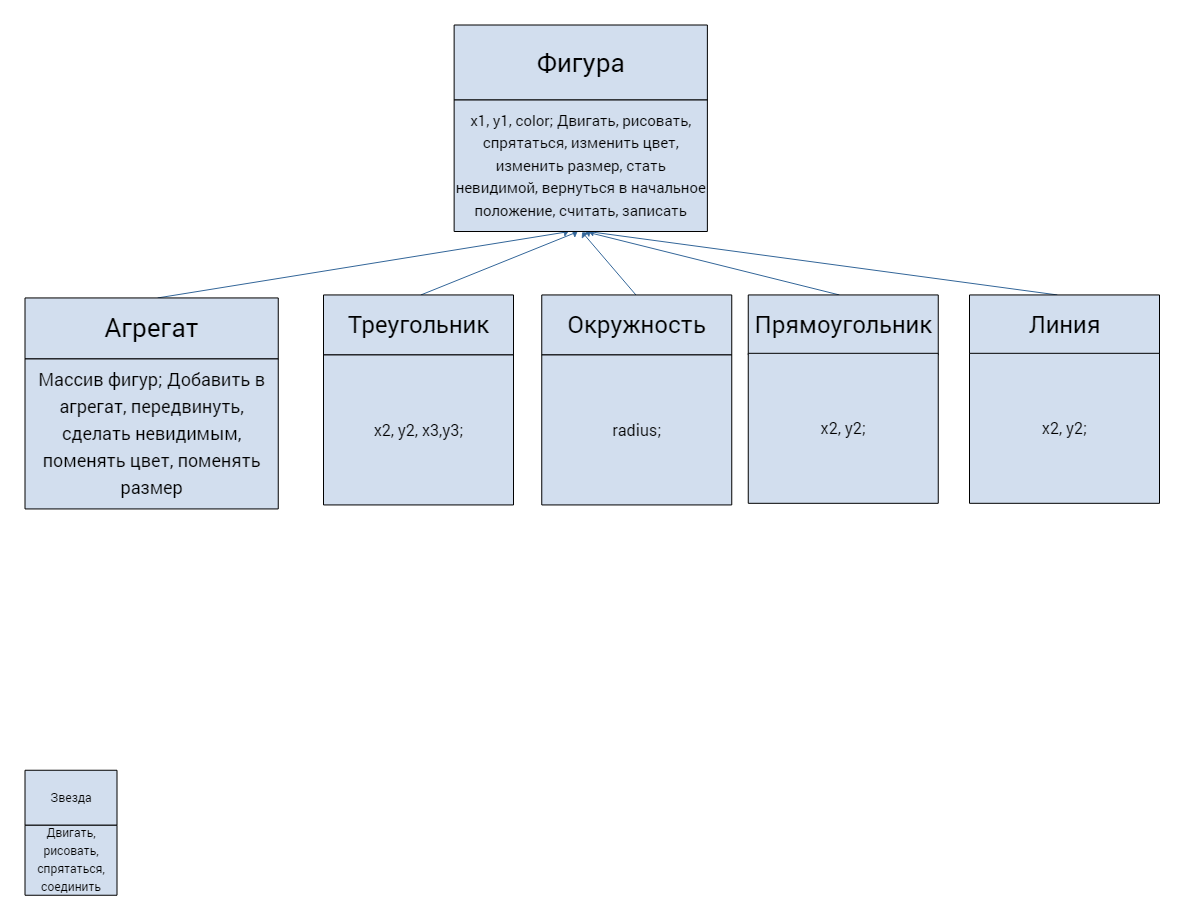
0. По команді.

5. Збирання/запам’ятовування агрегованого образу:

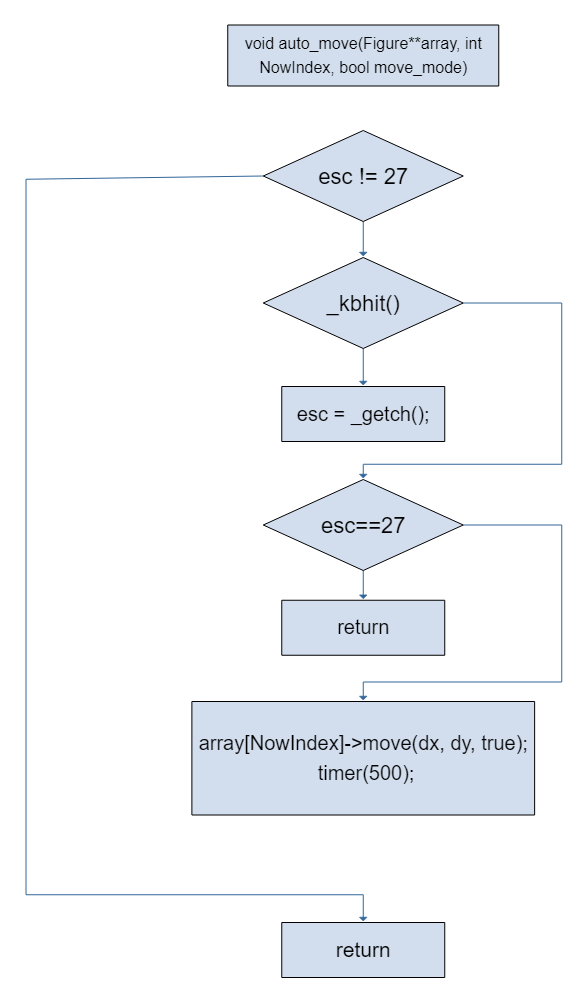
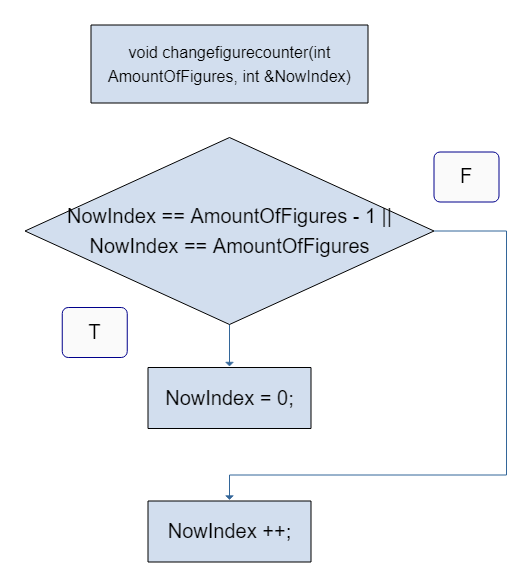
1. Видаленням об’єктів, з яких будується агрегат.

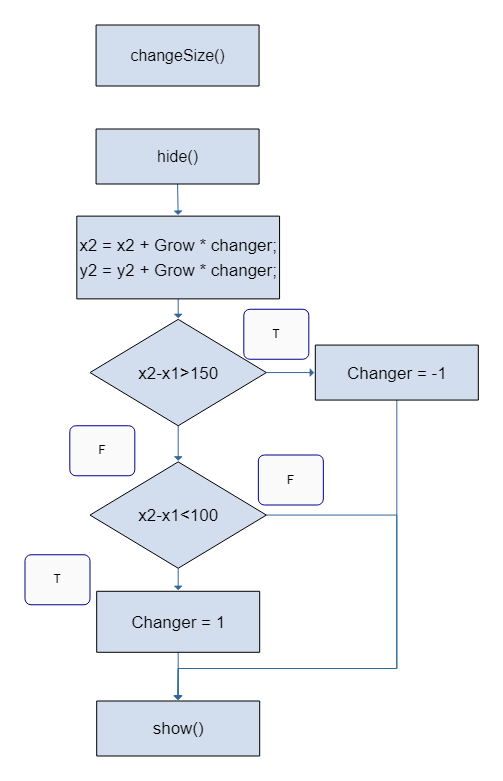
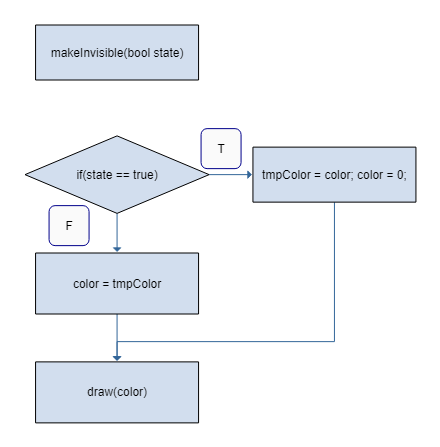
Варіанти графічних образів пропонується згенерувати самостійно на основі таких графічних примітивів: точка, відрізок, трикутник, прямокутник, коло, зірка. Наприклад: кільце; зірка, вписана в коло; коло, вписане в трикутник, сузір’я і та ін. Під час здавання роботи викладачеві слід очікувати на пропозицію додати до програми графічний об’єкт нової форми.

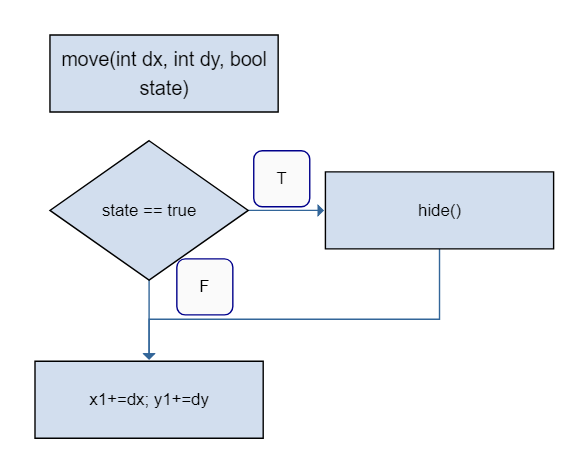
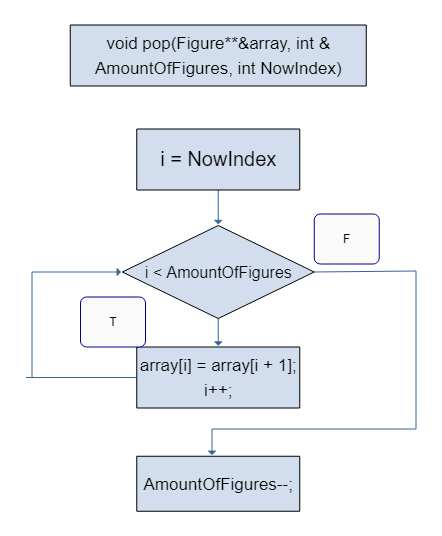
2. Опис розв’язку

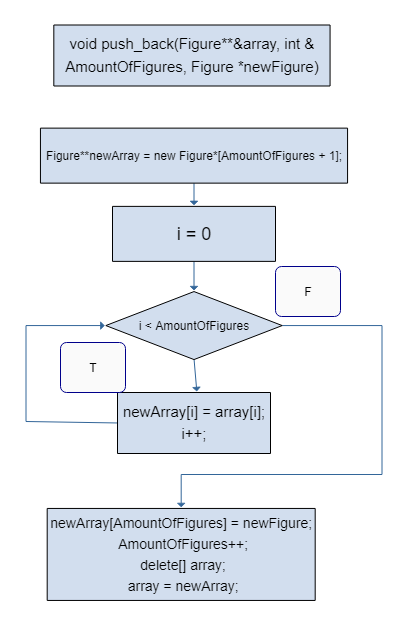


Блок-схеми:



3. Вихідний текст програми розв’язку задачі

Source.cpp

#include<iostream>

#include<fstream>

#include<conio.h>

#include<string>

#include<ctime>

#include"Type of figures.h"

using namespace std;

const char LARROW = 'K';

const char RARROW = 'M';

const char UARROW = 'H';

const char DARROW = 'P';

const char LINE = '1';

const char TRIANGLE = '2';

const char RECTANGLE = '3';

const char CIRCLE = '4';

const char NEWAGREGATE = '9';

const char MoveWithoutHide = 'v';

const char ClearScreen = 'c';

const char AddToAgregate = 'a';

const char CreateAgregate = 's';

const char ChangeColor = 'q';

const char ChangeSize = 'w';

const char ChangeFigureCounter = 'z';

const char AutoMove = 'm';

const char MakeInvisible = 'i';

const char ReturnToDefault = 'r';

const char SaveToFile = 'b';

const char ReadFromFile = 'n';

const char HELP = ';';

const int STEP = 10;

const int TAB = 9;

const int SizeOfArray = 10;

void timer(int ms)

{

int CLOCKS\_PER\_MSEC = CLOCKS\_PER\_SEC / 1000;

clock\_t end\_time = clock() + ms \* CLOCKS\_PER\_MSEC; // время завершения

while (clock() < end\_time)

{

}

}

void push\_back(Figure\*\*&array, int &AmountOfFigures, Figure \*newFigure)

{

Figure\*\*newArray = new Figure\*[AmountOfFigures + 1];

for (int i = 0; i < AmountOfFigures; i++)

{

newArray[i] = array[i];

}

newArray[AmountOfFigures] = newFigure;

AmountOfFigures++;

delete[] array;

array = newArray;

}

void pop(Figure\*\*&array, int &AmountOfFigures, int NowIndex)

{

for (int i = NowIndex; i < AmountOfFigures; i++)

{

array[i] = array[i + 1];

}

AmountOfFigures--;

}

void changefigurecounter(int AmountOfFigures, int &NowIndex)

{

if (NowIndex == AmountOfFigures - 1 || NowIndex == AmountOfFigures)

NowIndex = 0;

else

NowIndex++;

cout << "Now # " << NowIndex << "AofF = " << AmountOfFigures << endl;

}

void help()

{

cout << "1 - Line" << endl;

cout << "2 - Triangle" << endl;

cout << "3 - Rectangle" << endl;

cout << "4 - Circle" << endl;

cout << "9 - Initialize agregate" << endl;

cout << "z - next figure" << endl;

cout << "a - add to agregate" << endl;

cout << "s - create an agregate" << endl;

cout << "w - change size" << endl;

cout << "q - change color" << endl;

cout << "r - return to default" << endl;

cout << "i - hide figure" << endl;

cout << "m - auto move" << endl;

cout << "c - to clean screen" << endl;

cout << "n - load from file" << endl;

cout << "b - load to file" << endl;

cout << "To continue press any key" << endl;

system("PAUSE>NULL");

system("cls");

}

void show\_all(Figure\*\*array, int AmountOfFigures)

{

for (int i = 0; i < AmountOfFigures; i++)

{

array[i]->show\_all();

}

}

bool change\_state(bool state)

{

if (state == true)

state = false;

else

state = true;

return state;

}

void auto\_move(Figure\*\*array, int NowIndex, bool move\_mode)

{

char esc=' ';

while(esc!=27)

{

if (\_kbhit())

esc = \_getch();

if (esc == 27)

return;

array[NowIndex]->move(+STEP, +STEP, true);

timer(500);

if (\_kbhit())

esc = \_getch();

if (esc == 27)

return;

array[NowIndex]->move(+STEP, -STEP, true);

timer(500);

if (\_kbhit())

esc = \_getch();

if (esc == 27)

return;

array[NowIndex]->move(-STEP, -STEP, true);

timer(500);

if (\_kbhit())

esc = \_getch();

if (esc == 27)

return;

array[NowIndex]->move(-STEP, +STEP, true);

timer(500);

}

}

int int\_type\_of\_figure(char \*TypeOfFigure)

{

if (strcmp(TypeOfFigure, "l") == 1)

return 1;

else if (strcmp(TypeOfFigure, "triangle") == 1)

return 2;

else

return -1;

}

void read\_file(Figure\*\*array, int &AmountOfFigures, int&NowIndex, char\*argv[], int argc)//problem

{

ifstream file\_in;

string FileNameRead;

FileNameRead = argv[1];

cout << FileNameRead << endl;

file\_in.open(FileNameRead);

char TypeOfFigure[10];

int Type;

file\_in >> TypeOfFigure;

for (int i = 0; file\_in.eof()!= true; i++)

{

Type = int\_type\_of\_figure(TypeOfFigure);

switch (Type)

{

case 1:

array[i]->read(file\_in);

break;

default:

break;

}

}

file\_in.close();

}

void write\_file(Figure\*\*array, int AmountOfFigures, char \*argv[], int argc)

{

ofstream file\_out;

string FileNameRead;

FileNameRead = argv[1];

file\_out.open(FileNameRead);

for (int i = 0; i < AmountOfFigures; i++)

{

array[i]->write(file\_out);

}

file\_out.close();

cout << "Figures saved to " << FileNameRead << endl;

}

int main(int argc, char\*argv[])

{

int counter = 0;

int NowIndex = -1;

int Amount\_of\_figures = 0;

int num\_of\_agregate = -1;

bool state = true;

bool move\_mode = false;

bool hide\_mode = false;

char key;

Figure\*\*array = new Figure\*[Amount\_of\_figures];

Agregate agr[SizeOfArray];

while (key = \_getch(), key != 27)

{

if (key == '\xe0'&& Amount\_of\_figures != 0)

{

key = \_getch();

switch (key)

{

case LARROW:

array[NowIndex]->move(-STEP, 0, state);

break;

case RARROW:

array[NowIndex]->move(+STEP, 0, state);

break;

case UARROW:

array[NowIndex]->move(0, -STEP, state);

break;

case DARROW:

array[NowIndex]->move(0, +STEP, state);

break;

default:

{};

}

}

else

{

switch (key)

{

case MoveWithoutHide:

state = change\_state(state);

break;

case AutoMove:

cout << "To stop auto move press Escape" << endl;

timer(700);

system("cls");

show\_all(array, Amount\_of\_figures);

auto\_move(array, NowIndex, move\_mode);

cout << "Auto move is stopped" << endl;

timer(500);

system("cls");

show\_all(array, Amount\_of\_figures);

break;

case MakeInvisible:

hide\_mode = change\_state(hide\_mode);

array[NowIndex]->makeInvisible(hide\_mode);

break;

case ReturnToDefault:

array[NowIndex]->returnToDefault();

break;

case LINE:

push\_back(array, Amount\_of\_figures, new Line);

NowIndex++;

break;

case TRIANGLE:

push\_back(array, Amount\_of\_figures, new Triangle);

NowIndex++;

break;

case RECTANGLE:

push\_back(array, Amount\_of\_figures, new Rectangle);

NowIndex++;

break;

case CIRCLE:

push\_back(array, Amount\_of\_figures, new Circle);

NowIndex++;

break;

case NEWAGREGATE:

cout << "Agregate initialized" << endl;

num\_of\_agregate++;

break;

case AddToAgregate:

if (num\_of\_agregate == -1)

cout << "Initialize an Agregate with 9" << endl;

else

{

agr[num\_of\_agregate].AddToAgregate(array[NowIndex]);

pop(array, Amount\_of\_figures, NowIndex);

cout << "Figure # " << NowIndex << " added to agr" << endl;

}

break;

case CreateAgregate:

if (num\_of\_agregate == -1)

{

cout << "Add 2 figures first" << endl;

timer(700);

system("cls");

show\_all(array, Amount\_of\_figures);

}

else

{

push\_back(array, Amount\_of\_figures, &agr[num\_of\_agregate]);

cout << "Agregate created" << endl;

cout << "Press any key to continue" << endl;

system("PAUSE>NULL");

system("cls");

show\_all(array, Amount\_of\_figures);

}

break;

case ChangeFigureCounter:

changefigurecounter(Amount\_of\_figures, NowIndex);

show\_all(array, Amount\_of\_figures);

break;

case ChangeColor:

array[NowIndex]->changeColor();

break;

case ChangeSize:

array[NowIndex]->changeSize();

break;

case ClearScreen:

system("cls");

show\_all(array, Amount\_of\_figures);

break;

case SaveToFile:

write\_file(array, Amount\_of\_figures, argv, argc);

break;

case ReadFromFile:

read\_file(array, Amount\_of\_figures, NowIndex, argv, argc);

break;

case ';':

help();

show\_all(array, Amount\_of\_figures);

break;

default:

{

};

}

}

}

return 0;

}

Figure.h

#pragma once

const int Grow = 10;

class Figure

{

protected:

char TypeOfFigure[15] = "Figure";

int x1; int y1;

int def\_x1=200; int def\_y1=200;

int color; int tmpColor;

int def\_Color = 2;

public:

Figure();

Figure(int x1, int y1, int color);

virtual void move(int dx, int dy, bool state);

virtual void changeColor();

virtual void changeSize();

virtual void makeInvisible(bool state);

virtual void returnToDefault();

virtual void show\_all();

virtual void read(std::ifstream &in);

virtual void write(std::ofstream &out);

protected:

virtual void draw(int color);

void show();

void hide();

};

Figure.cpp

#include"Figure.h"

#include"graphics.h"

#include<fstream>

using namespace Grfx;

Graphics gr;

Figure::Figure()

{

x1 = 200;

y1 = 200;

color = 2;

}

Figure::Figure(int x1, int y1, int color)

{

this->x1 = x1;

this->y1 = y1;

this->color = color;

}

void Figure::draw(int color)

{

gr.setcolor(color);

}

void Figure::show()

{

draw(color);

}

void Figure::hide()

{

draw(0);

}

void Figure::move(int dx, int dy, bool state)

{

if (state == true)

hide();

else

;

x1 += dx;

y1 += dy;

show();

}

void Figure::changeColor()

{

gr.setcolor(color + 1);

draw(color);

}

void Figure::changeSize()

{

}

void Figure::makeInvisible(bool state)

{

if (state == true)

{

tmpColor = color;

color = 0;

}

else

color = tmpColor;

draw(color);

}

void Figure::returnToDefault()

{

hide();

this->x1 = def\_x1;

this->y1 = def\_y1;

this->color = def\_Color;

show();

}

void Figure::show\_all()

{

show();

}

void Figure::write(std::ofstream & out)

{

out << this->TypeOfFigure;

out << ' ';

out << this->x1;

out << ' ';

out << this->y1;

out << ' ';

out << this->color;

out << std::endl;

}

void Figure::read(std::ifstream &in)

{

in >> this->TypeOfFigure;

in >> this->x1;

in >> this->y1;

in >> this->color;

}

Line.h

#pragma once

#include"Figure.h"

class Line :public Figure

{

protected:

char TypeOfFigure[10] = "line";

int x2;

int y2;

public:

Line();

Line(int x2, int y2, int color);

void move(int dx, int dy, bool state);

void write(std::ofstream &out);

void read(std::ifstream &in);

protected:

void show();

void draw(int color);

void changeColor();

void changeSize();

void returnToDefault();

};

Line.cpp

#include "Line.h"

#include"graphics.h"

#include<fstream>

using namespace Grfx;

Graphics grLine;

const int length = 100;

int Lchanger = 1;

Line::Line(): Figure()

{

x2 = x1 + length;

y2 = x1 + length;

show();

}

Line::Line(int x2, int y2, int color)

{

this->x2 = x2;

this->y2 = y2;

this->color = color;

}

void Line::show()

{

draw(color);

}

void Line::draw(int color)

{

grLine.setcolor(color);

grLine.line(x1, y1, x2, y2);

}

void Line::changeColor()

{

this->color = color + 1;

show();

}

void Line::changeSize()

{

hide();

this->x2 = x2 + Grow \* Lchanger;

this->y2 = y2 + Grow \* Lchanger;

if (x2 - x1 > 150)

Lchanger = -1;

else if (x2 - x1 < 100)

Lchanger = 1;

show();

}

void Line::returnToDefault()

{

hide();

x1 = def\_x1;

y1 = def\_y1;

x2 = x1 + 100;

y2 = y1 + 100;

color = def\_Color;

show();

}

void Line::move(int dx, int dy, bool state)

{

if (state == true)

hide();

else

;

x1 += dx;

y1 += dy;

x2 += dx;

y2 += dy;

show();

}

void Line::write(std::ofstream & out)

{

out << this->TypeOfFigure;

out << ' ';

out << this->x1;

out << ' ';

out << this->y2;

out << ' ';

out << this->x2;

out << ' ';

out << this->y2;

out << std::endl;

}

void Line::read(std::ifstream & in)

{

in >> this->TypeOfFigure;

in >> this->x1;

in >> this->y1;

in >> this->color;

in >> this->x2;

in >> this->y2;

}

Triangle.h

#pragma once

#include"Line.h"

class Triangle : public Figure

{

protected:

char TypeOfFigure[10] = "Triangle";

int x2,x3;

int y2,y3;

public:

Triangle();

Triangle(int x2, int y2, int x3, int y3, int color);

void move(int dx, int dy, bool state);

protected:

void show();

void draw(int color);

void changeColor();

void changeSize();

void returnToDefault();

};

Triangle.cpp

#include "Triangle.h"

#include"graphics.h"

using namespace Grfx;

Graphics grTriangle;

const int length = 100;

int Tchanger = 1;

Triangle::Triangle():Figure()

{

x2 = x1 + length;

y2 = y1 + length;

x3 = x1 - length;

y3 = y1 + length;

show();

}

void Triangle::show()

{

draw(color);

}

void Triangle::draw(int color)

{

grTriangle.setcolor(color);

grTriangle.triangle(x1, y1, x2, y2, x3, y3);

}

void Triangle::changeColor()

{

this->color = color + 1;

show();

}

void Triangle::changeSize()

{

hide();

this->y1 = y1 - Grow \* Tchanger;

this->x2 = x2 + Grow \* Tchanger;

this->y2 = y2 + Grow \* Tchanger;

this->x3 = x3 - Grow \* Tchanger;

this->y3 = y3 + Grow \* Tchanger;

if (x2 - x1 > 140)

Tchanger = -1;

else if (x2 - x1 < 100)

Tchanger = 1;

show();

}

void Triangle::returnToDefault()

{

hide();

x1 = def\_x1;

y1 = def\_y1;

x2 = x1 + 50;

y2 = y1 + 50;

x3 = x1 - 50;

y3 = y1 + 50;

color = def\_Color;

show();

}

void Triangle::move(int dx, int dy, bool state)

{

if (state == true)

hide();

else

;

x1 += dx;

y1 += dy;

x2 += dx;

y2 += dy;

x3 += dx;

y3 += dy;

show();

}

Rectangle.h

#pragma once

#include"Figure.h"

class Rectangle:public Figure

{

protected:

char TypeOfFigure[12] = "Rectangle";

int x2;

int y2;

public:

Rectangle();

Rectangle(int x2, int y2, int color);

protected:

void show();

void draw(int color);

void move(int dx, int dy, bool state);

void changeColor();

void changeSize();

};

Rectangle.cpp

#include "Rectangle.h"

#include"graphics.h"

using namespace Grfx;

Graphics grRectangle;

const int length = 150;

int Rchanger = 1;

Rectangle::Rectangle()

{

x2 = x1 + 150;

y2 = y1 + 150;

show();

}

Rectangle::Rectangle(int x2, int y2, int color)

{

}

void Rectangle::show()

{

draw(color);

}

void Rectangle::draw(int color)

{

grRectangle.setcolor(color);

grRectangle.rectangle(x1, y1, x2, y2);

}

void Rectangle::move(int dx, int dy, bool state)

{

if (state == true)

hide();

else

;

x1 += dx;

y1 += dy;

x2 += dx;

y2 += dy;

show();

}

void Rectangle::changeColor()

{

this->color = color + 1;

show();

}

void Rectangle::changeSize()

{

hide();

this->x1 = x1 - Grow\*Rchanger;

this->y1 = y1 - Grow \* Rchanger;

this->x2 = x2 + Grow \* Rchanger;

this->y2 = y2 + Grow \* Rchanger;

if (x2 - x1 > 200)

Rchanger = -1;

else if (x2 - x1 < 100)

Rchanger = 1;

show();

}

Point.h

#pragma once

#include "Figure.h"

class Point :public Figure

{

protected:

char TypeOfFigure[10] = "Point";

int r;

public:

Point();

Point(int x1, int y1, int color, int r);

void move(int dx, int dy, bool state);

protected:

void show();

void draw(int color);

void changeColor();

void changeSize();

};

Point.cpp

#include "Point.h"

#include"graphics.h"

using namespace Grfx;

Graphics grPoint;

const int radius = 7;

Point::Point()

{

r = radius;

show();

}

Point::Point(int x1, int y1, int color, int r)

{

}

void Point::move(int dx, int dy, bool state)

{

if (state == true)

hide();

else

;

x1 += dx;

y1 += dy;

show();

}

void Point::show()

{

draw(color);

}

void Point::draw(int color)

{

grPoint.setcolor(color);

grPoint.circle(x1, y1, r);

}

void Point::changeColor()

{

this->color = color + 1;

show();

}

void Point::changeSize()

{

;

}

Agregate.h

pragma once

#include "Figure.h"

const int N = 20;

class Agregate: public Figure

{

protected:

Figure\*\*agregate = new Figure\*[N];

int AmountOfFiguresInAgregate = 0;

public:

Agregate();

void AddToAgregate(Figure\*figure);

void makeInvisible(bool state);

void show\_all();

protected:

void move(int dx, int dy, bool state);

void changeColor();

void changeSize();

};

Agregate.cpp

#include "Agregate.h"

Agregate::Agregate()

{

Figure\*\*agregate = new Figure\*[N];

//AmountOfFiguresInAgregate = 0;

}

void Agregate::AddToAgregate(Figure \* figure)

{

agregate[AmountOfFiguresInAgregate] = figure;

AmountOfFiguresInAgregate++;

}

void Agregate::makeInvisible(bool state)

{

for (int i = 0; i < AmountOfFiguresInAgregate; i++)

agregate[i]->makeInvisible(state);

}

void Agregate::show\_all()

{

for (int i = 0; i < AmountOfFiguresInAgregate; i++)

agregate[i]->show\_all();

}

void Agregate::move(int dx, int dy, bool state)

{

for (int i = 0; i < AmountOfFiguresInAgregate; i++)

agregate[i]->move(dx, dy, state);

}

void Agregate::changeColor()

{

for (int i = 0; i < AmountOfFiguresInAgregate; i++)

agregate[i]->changeColor();

}

void Agregate::changeSize()

{

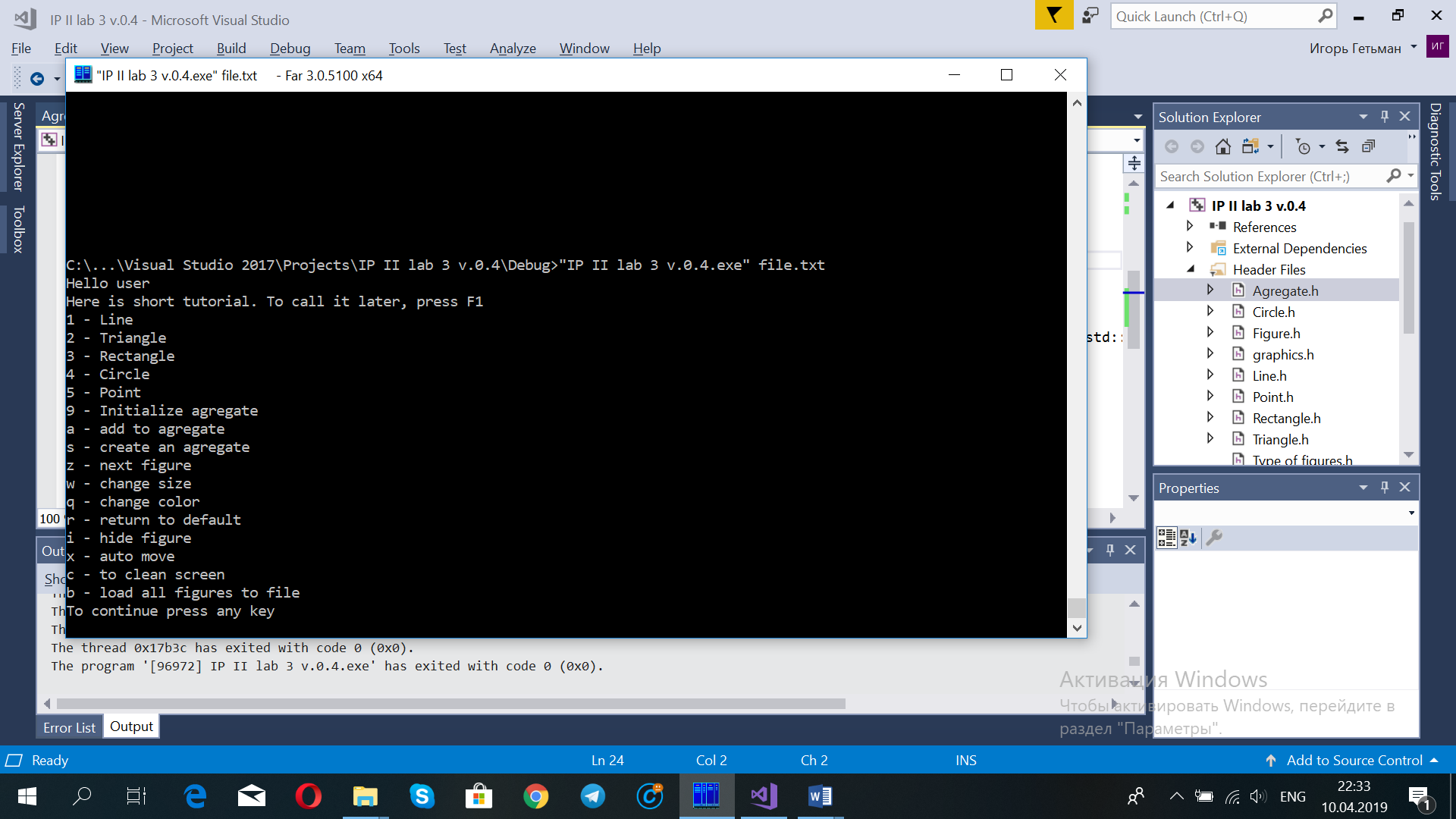
for (int i = 0; i < AmountOfFiguresInAgregate; i++)

agregate[i]->changeSize();

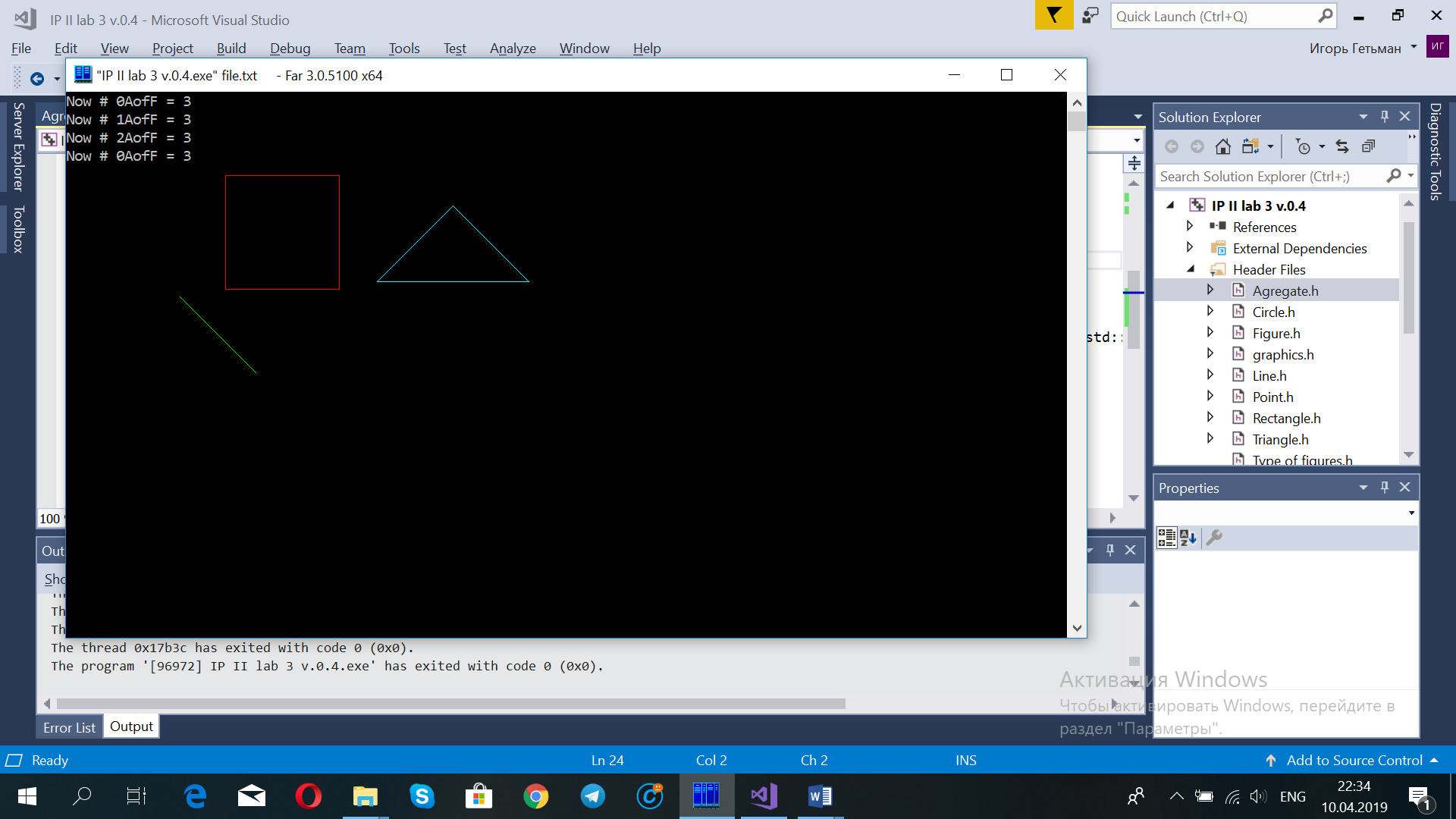
}

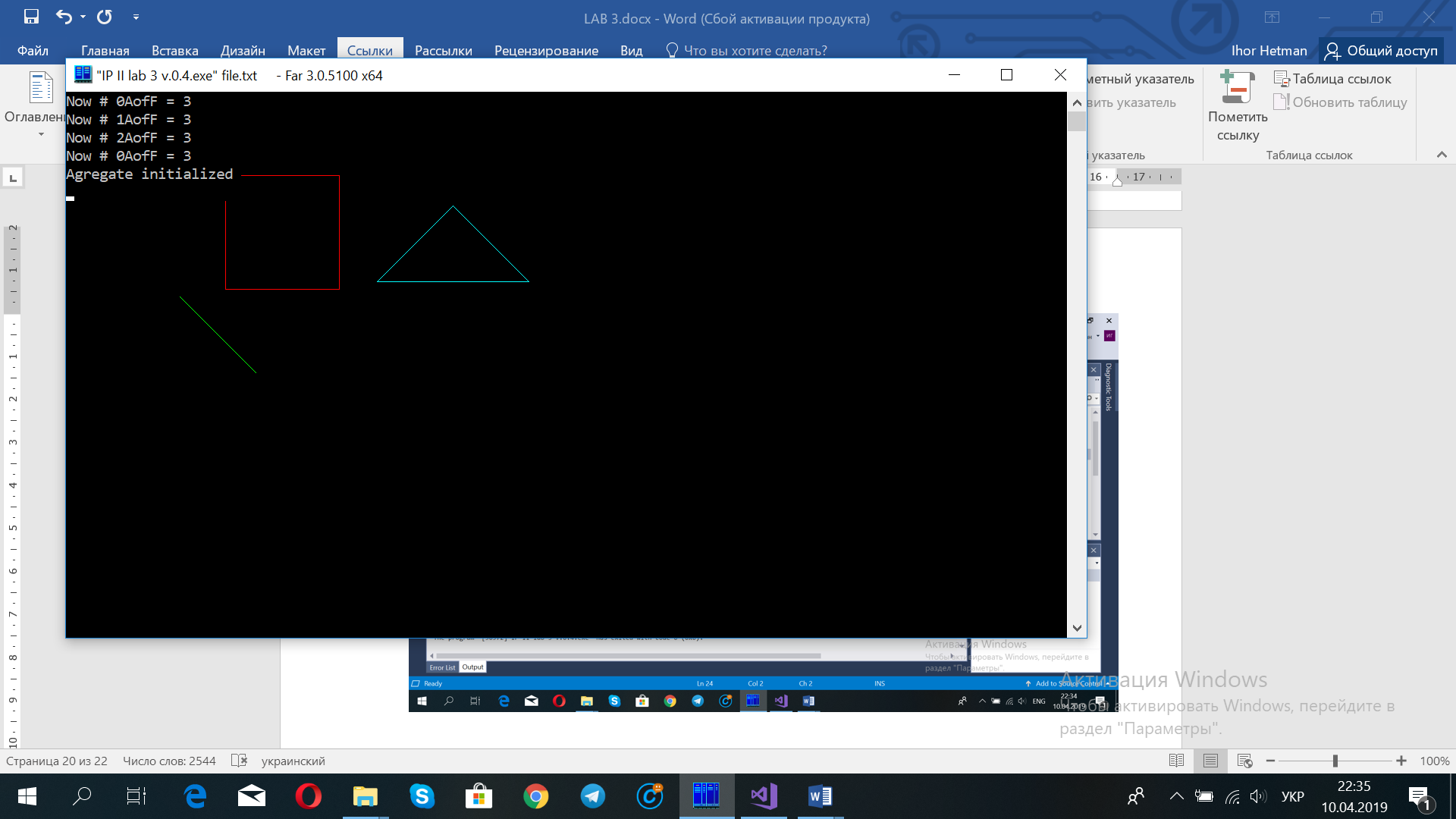
4. Опис інтерфейсу (керівництво користувача)

При запуску програми, користувача зустрічає довідка



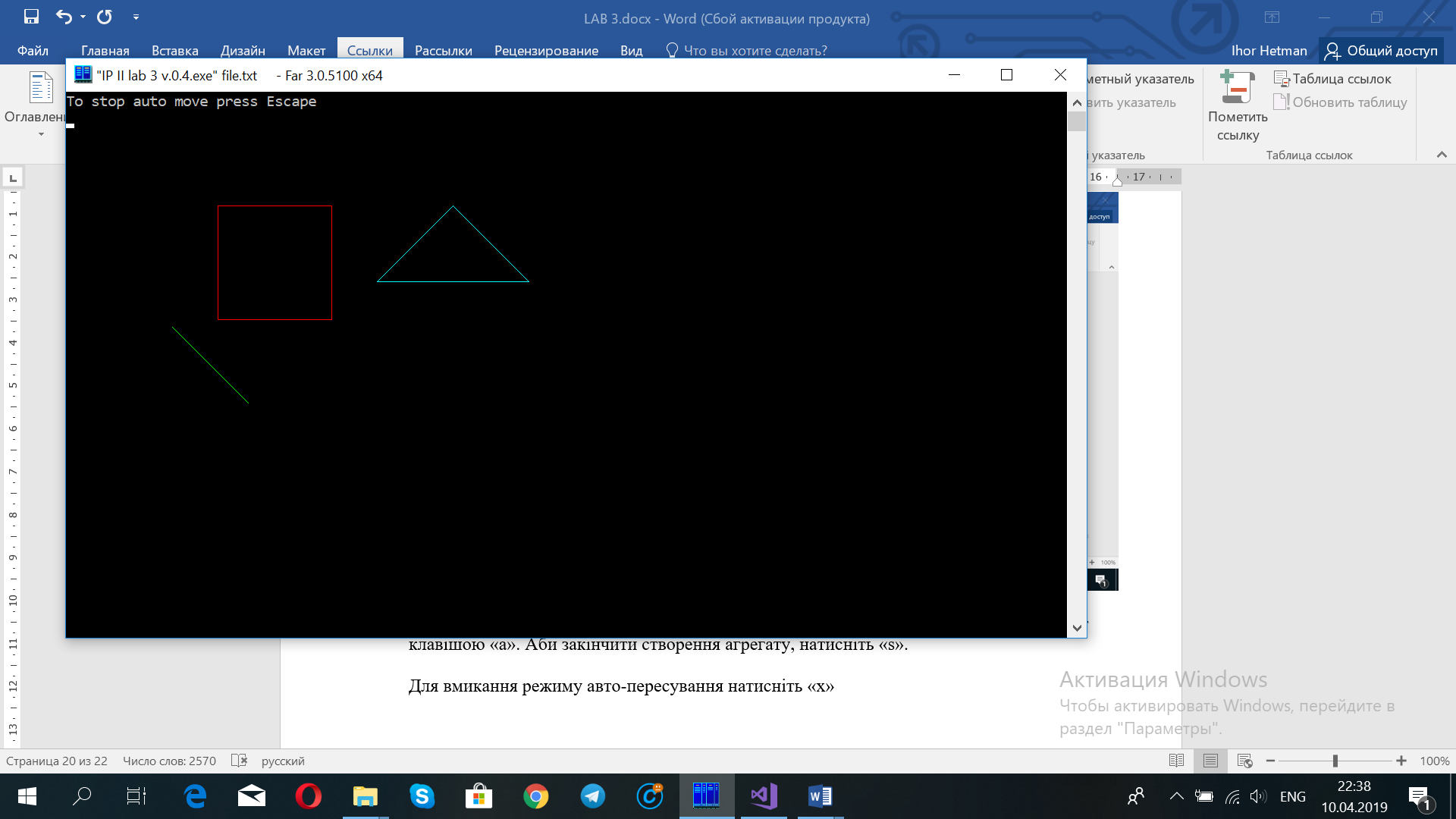
Користувач може створювати фігури, натискаючи на відповідні клавіші. А також перемикатись між ними

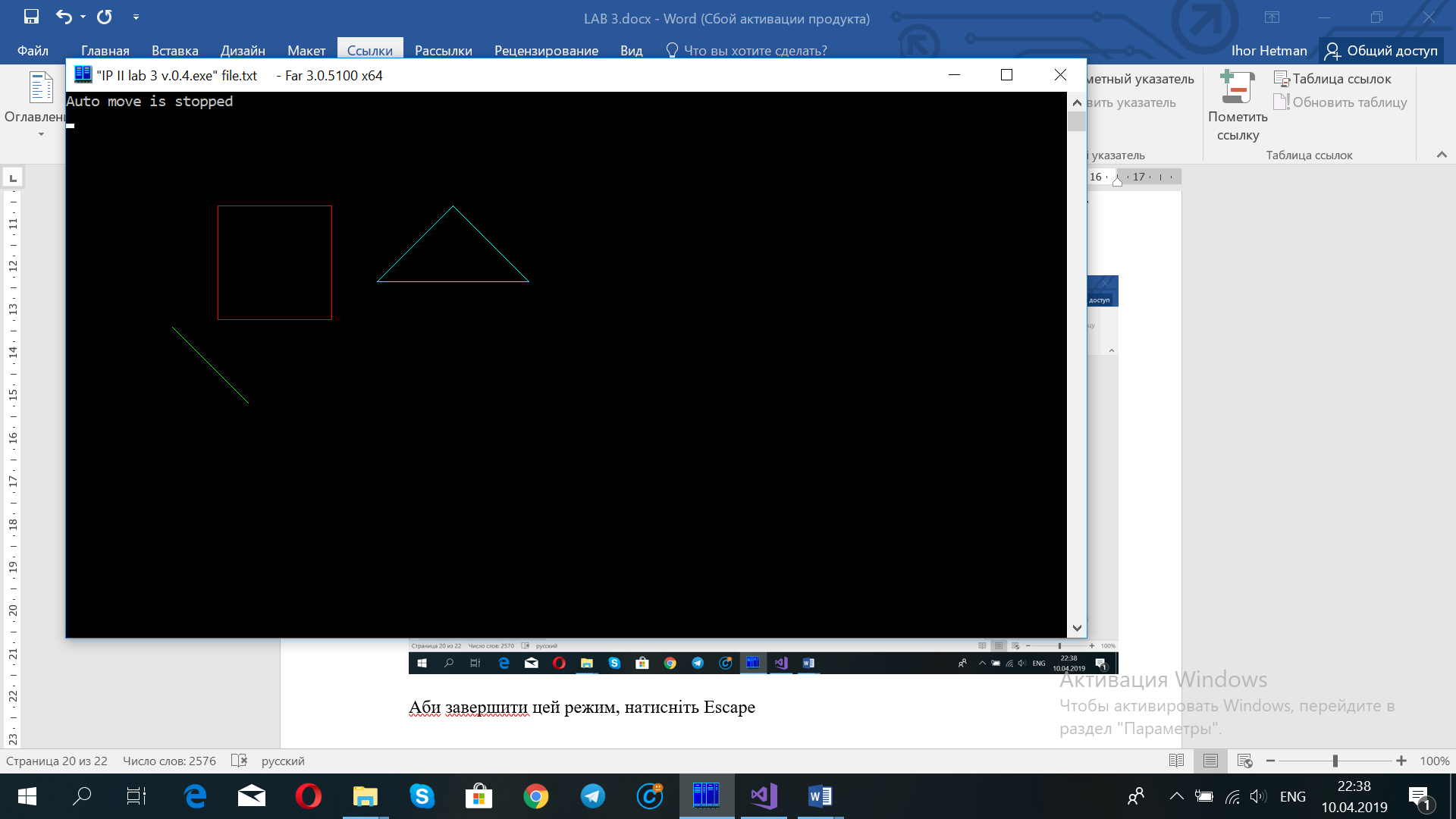


Аби створити агрегат, його необхідно спочатку ініціалізувати з допомогою 9

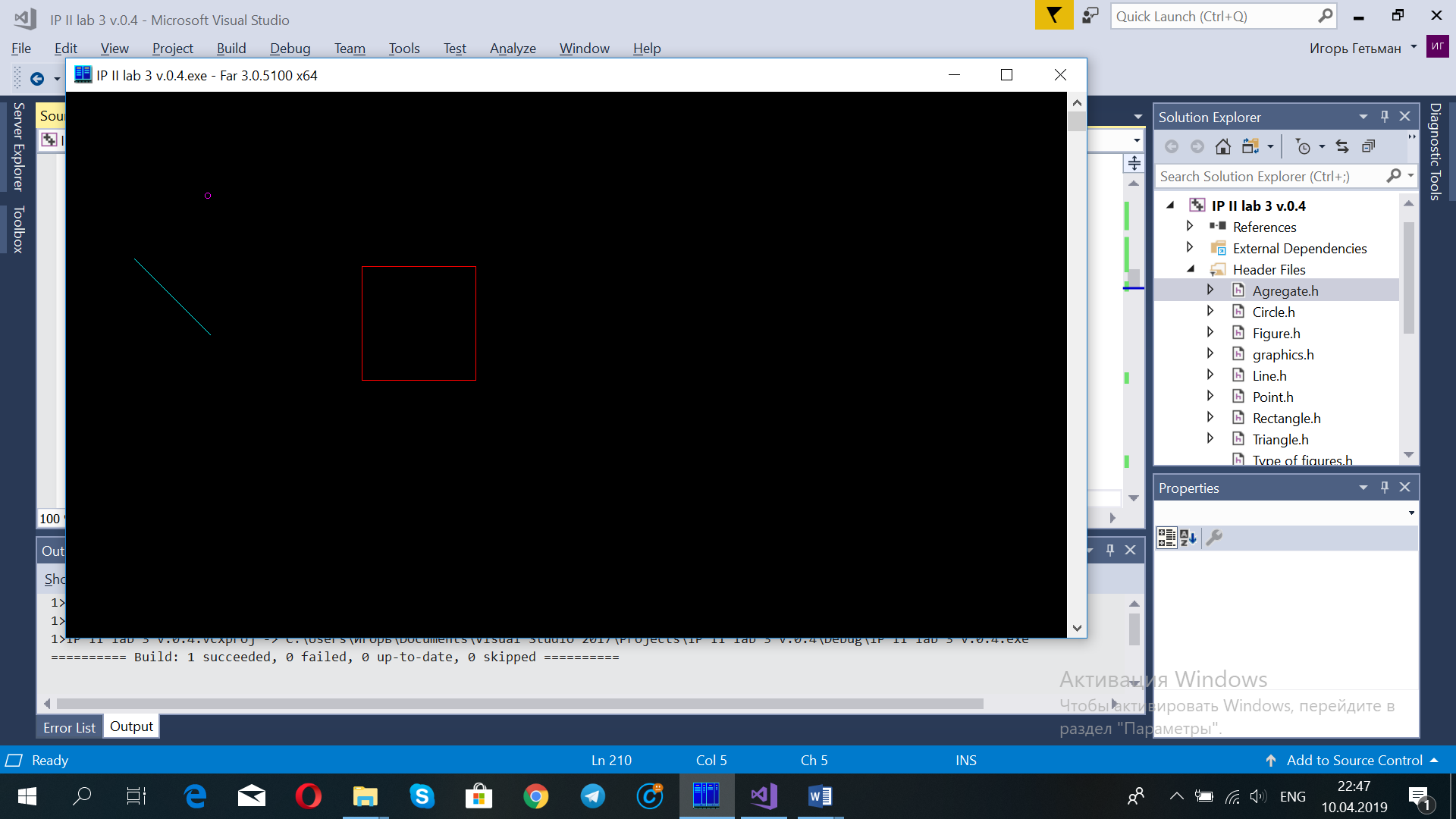
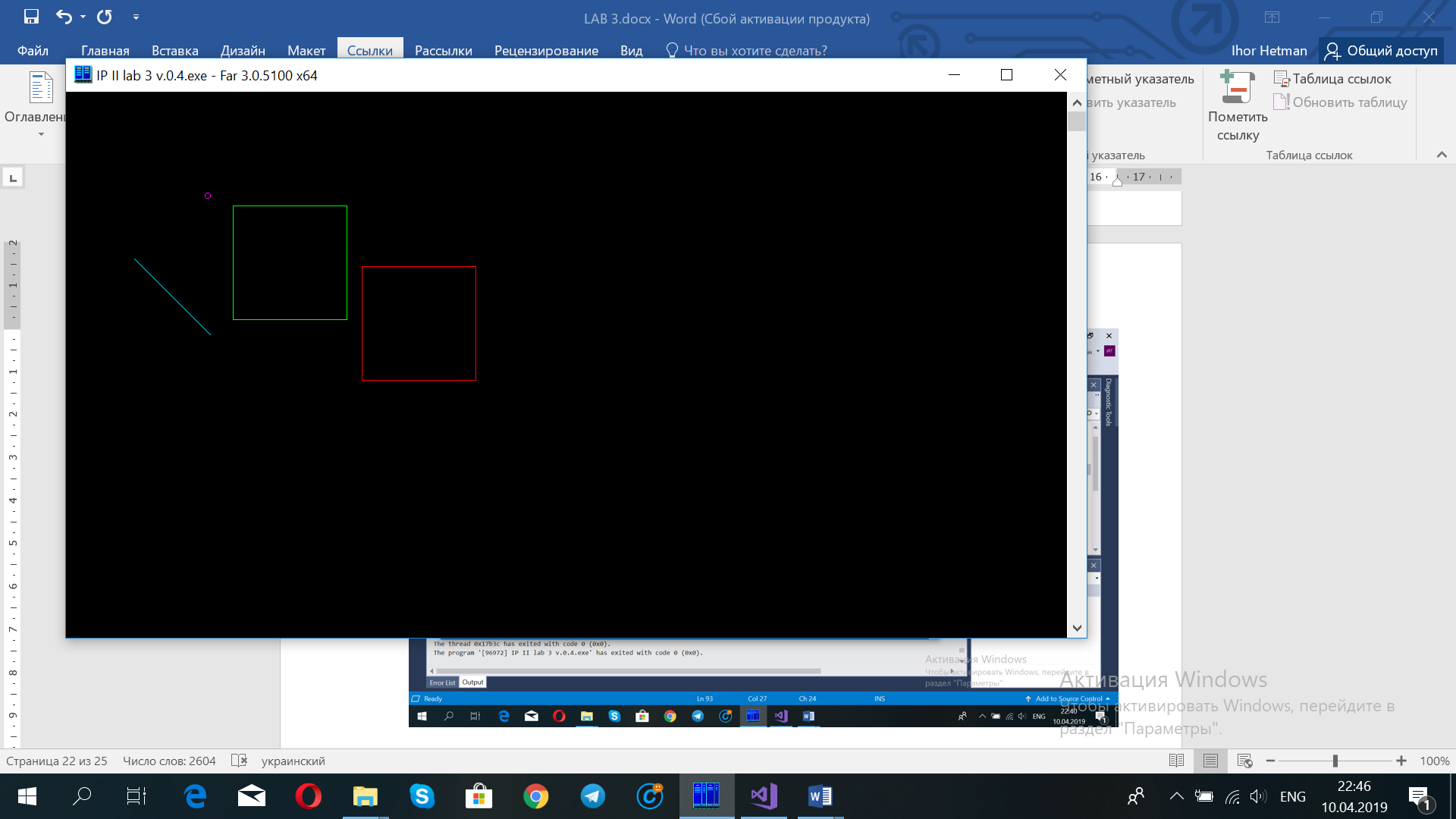
Далі по черзі, перемикаючись між фігурами, натискаючи «z», додаємо фігури у агрегат клавішою «а». Аби закінчити створення агрегату, натисніть «s».

Для вмикання режиму авто-пересування натисніть «x»

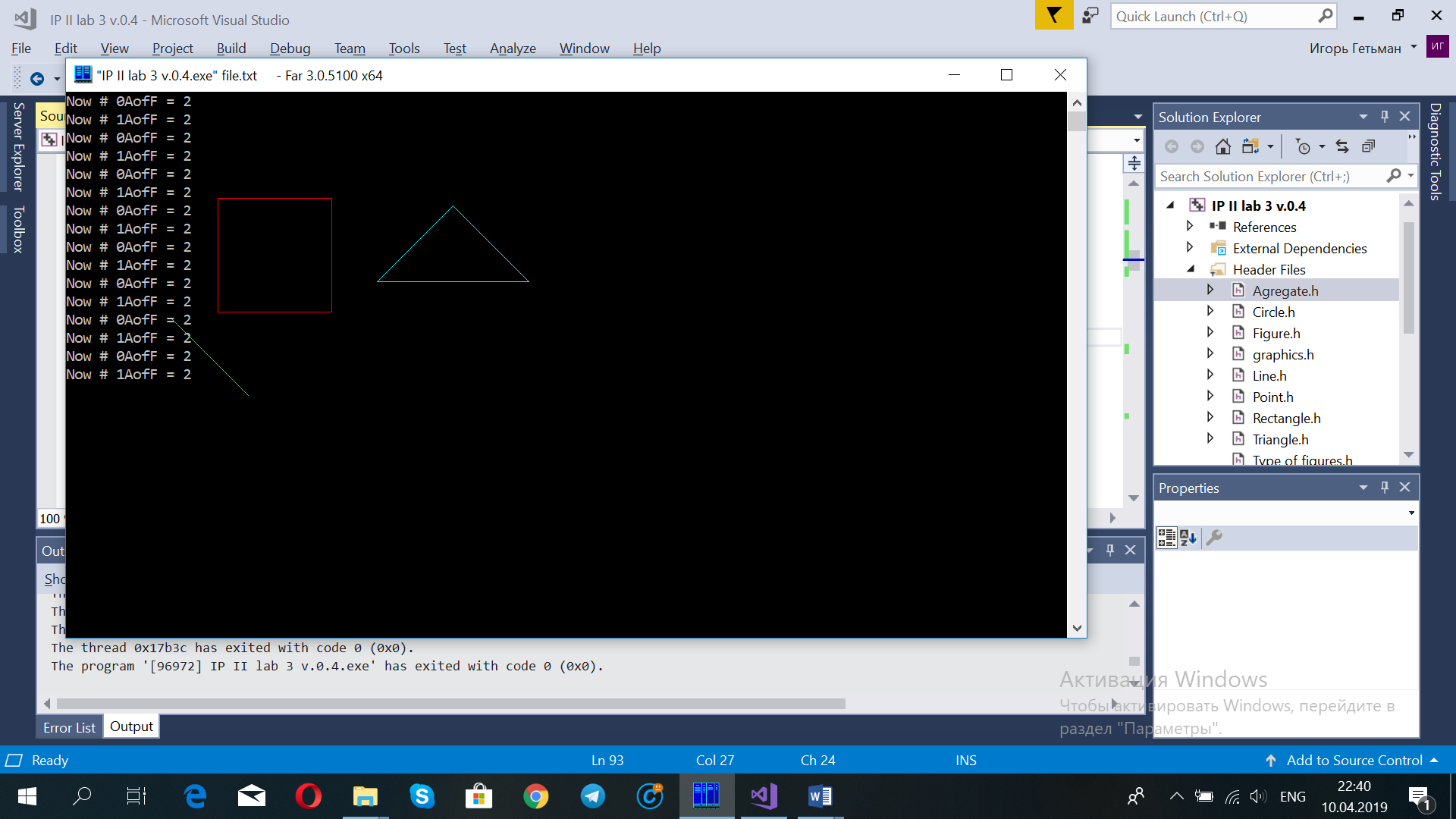


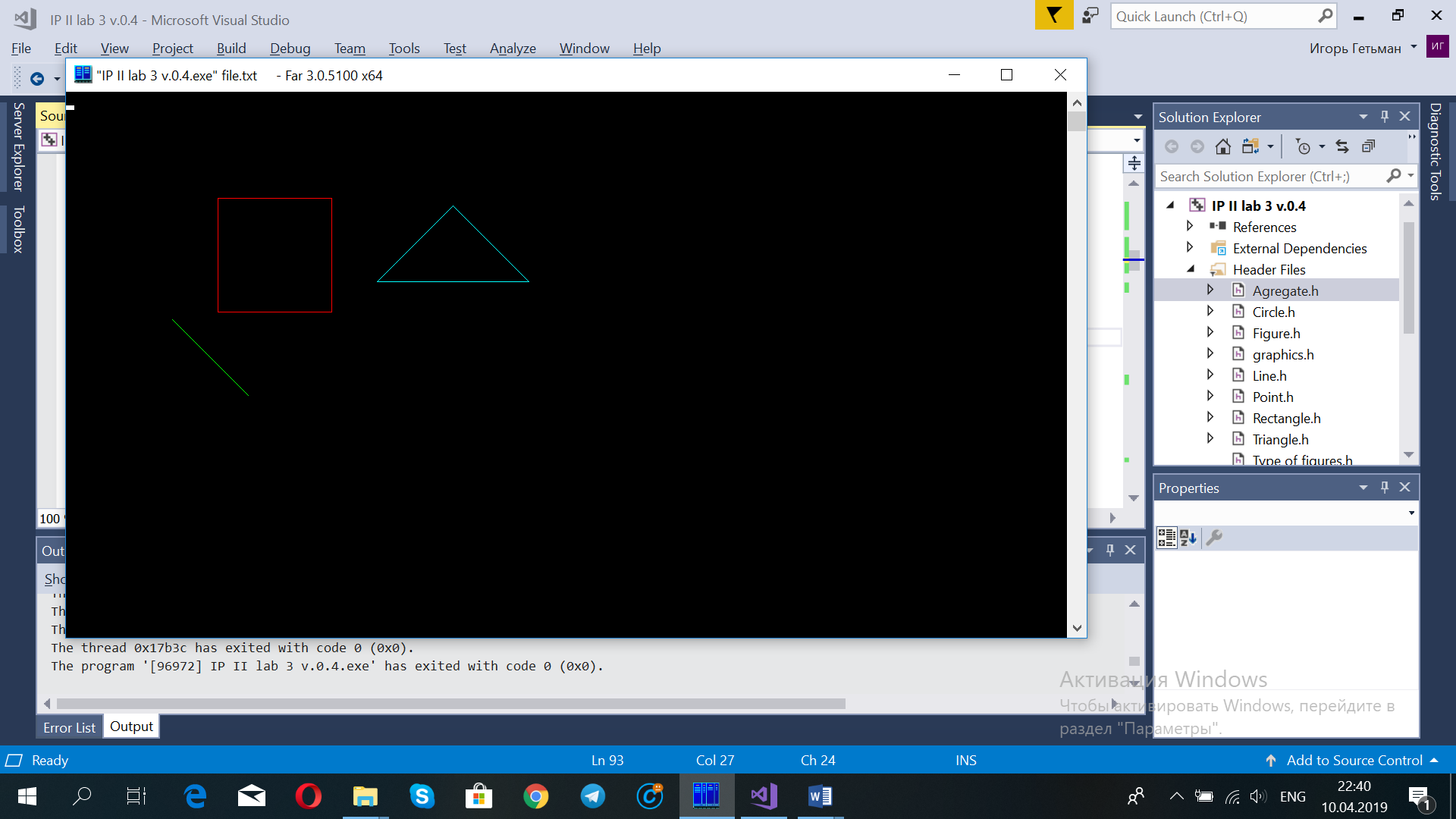
Аби завершити цей режим, натисніть Escape

Щоб приховати обрану фігуру, натисніть «і»

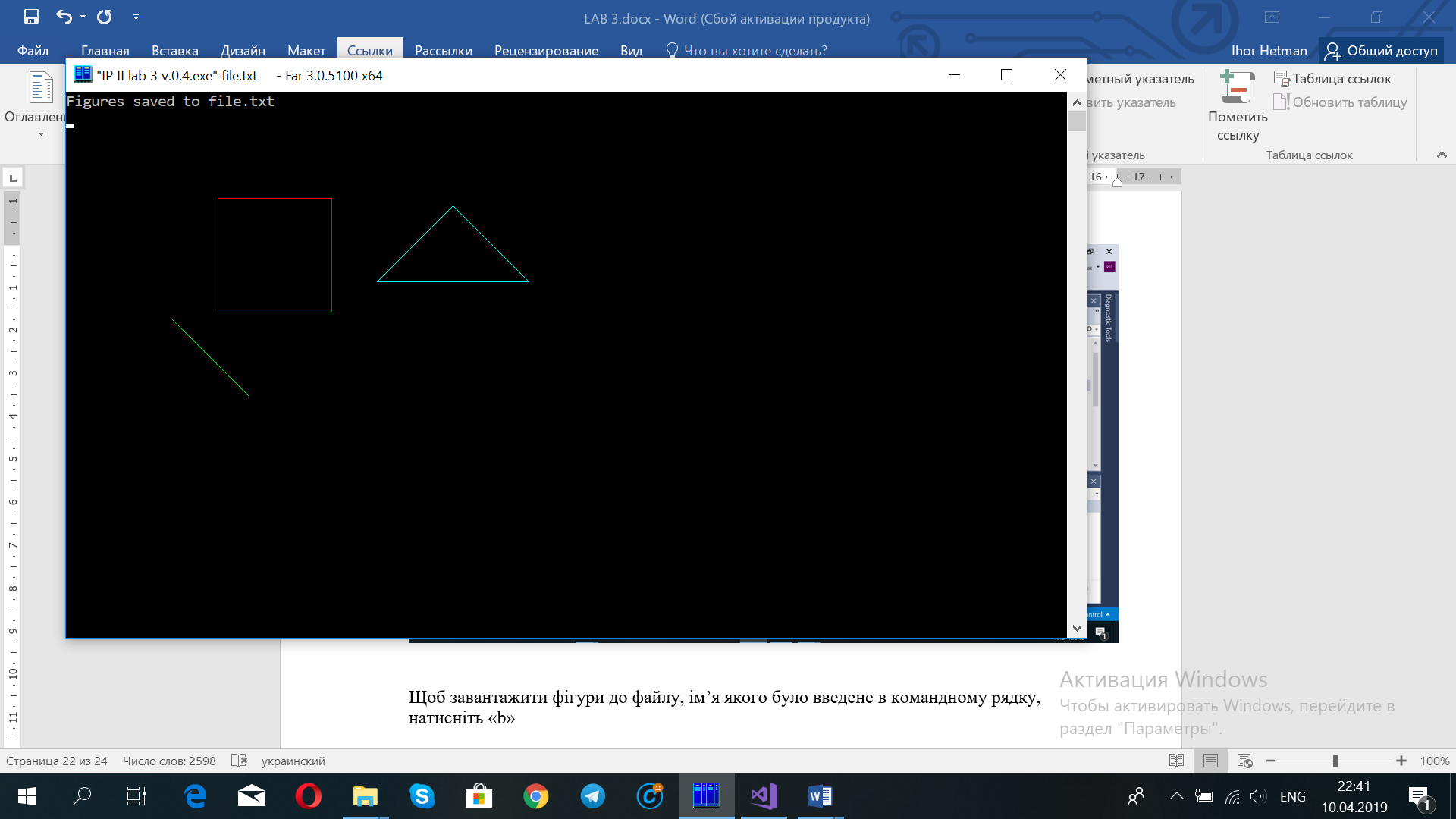


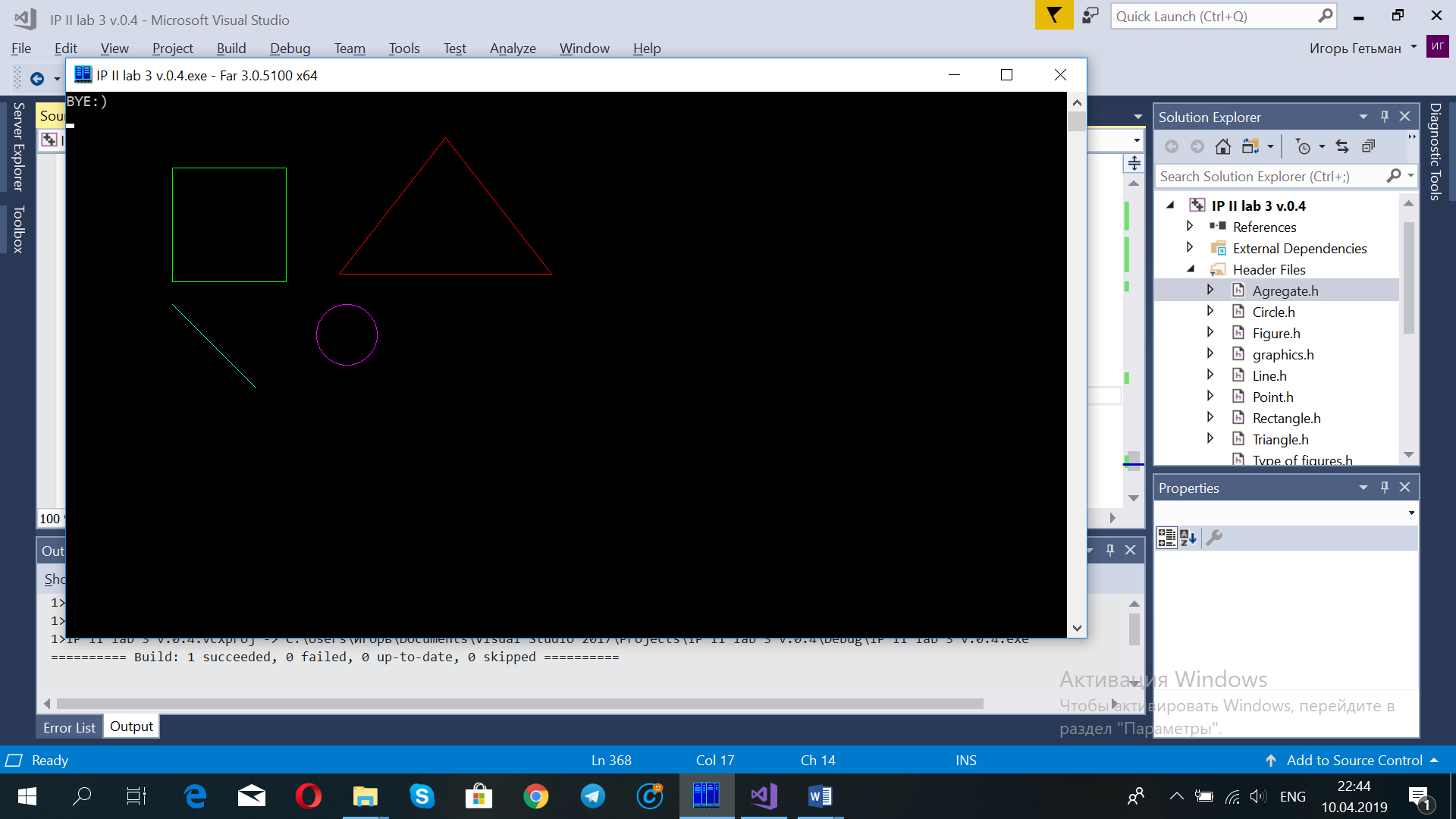
Аби її відновити, знову натисніть «і»

Аби очистити екран від системних повідомлень, натисніть «с» 

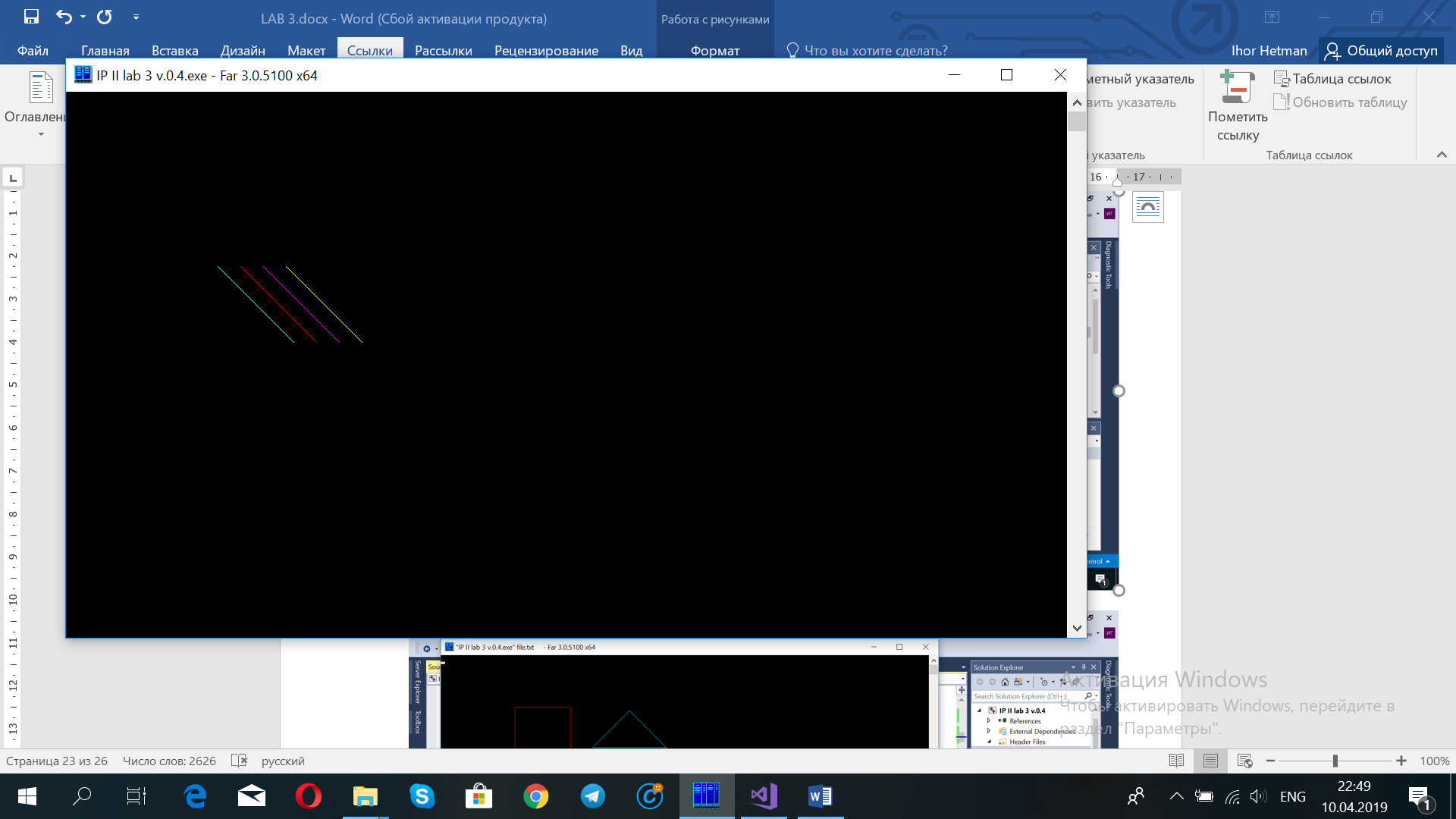


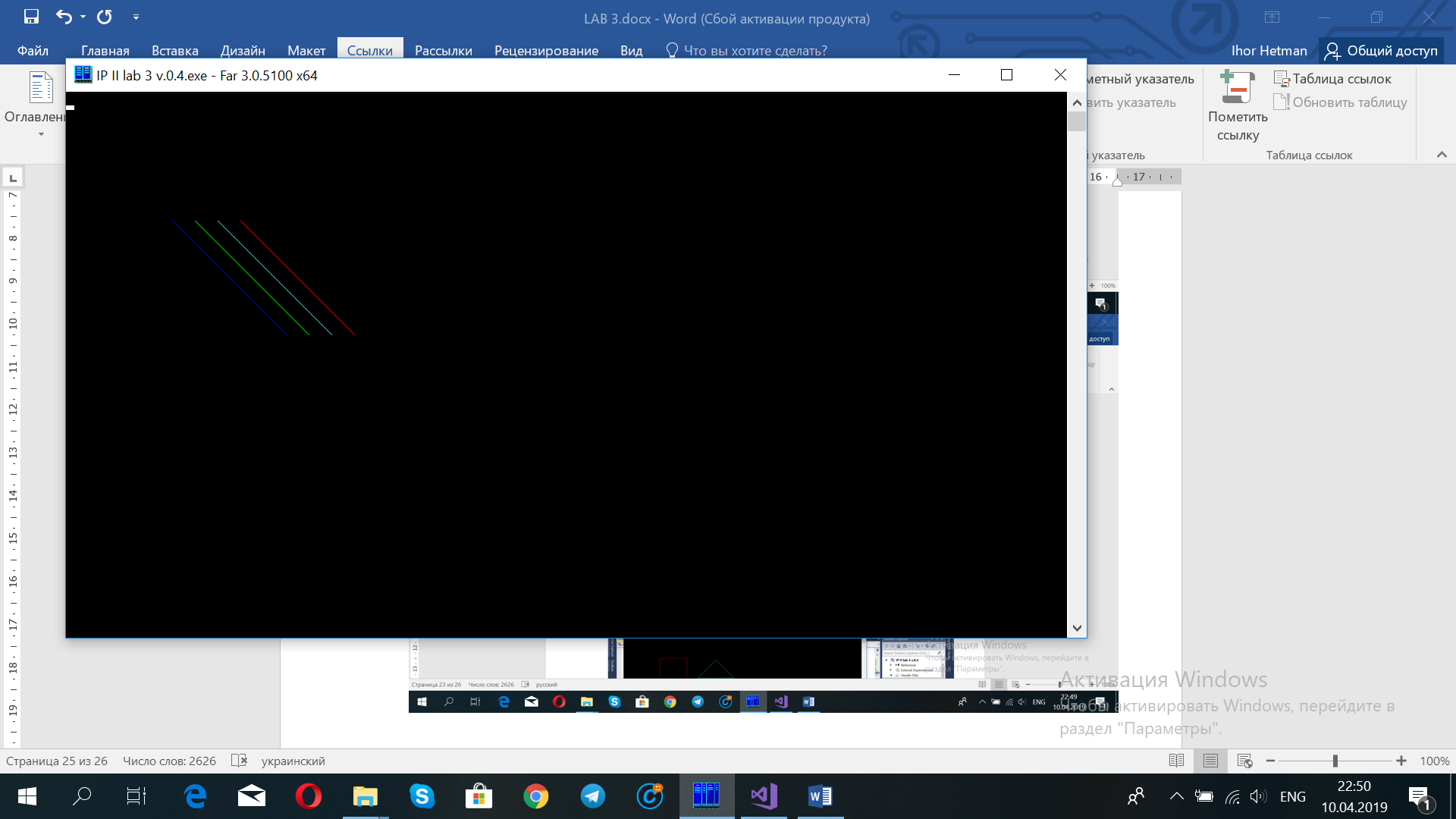
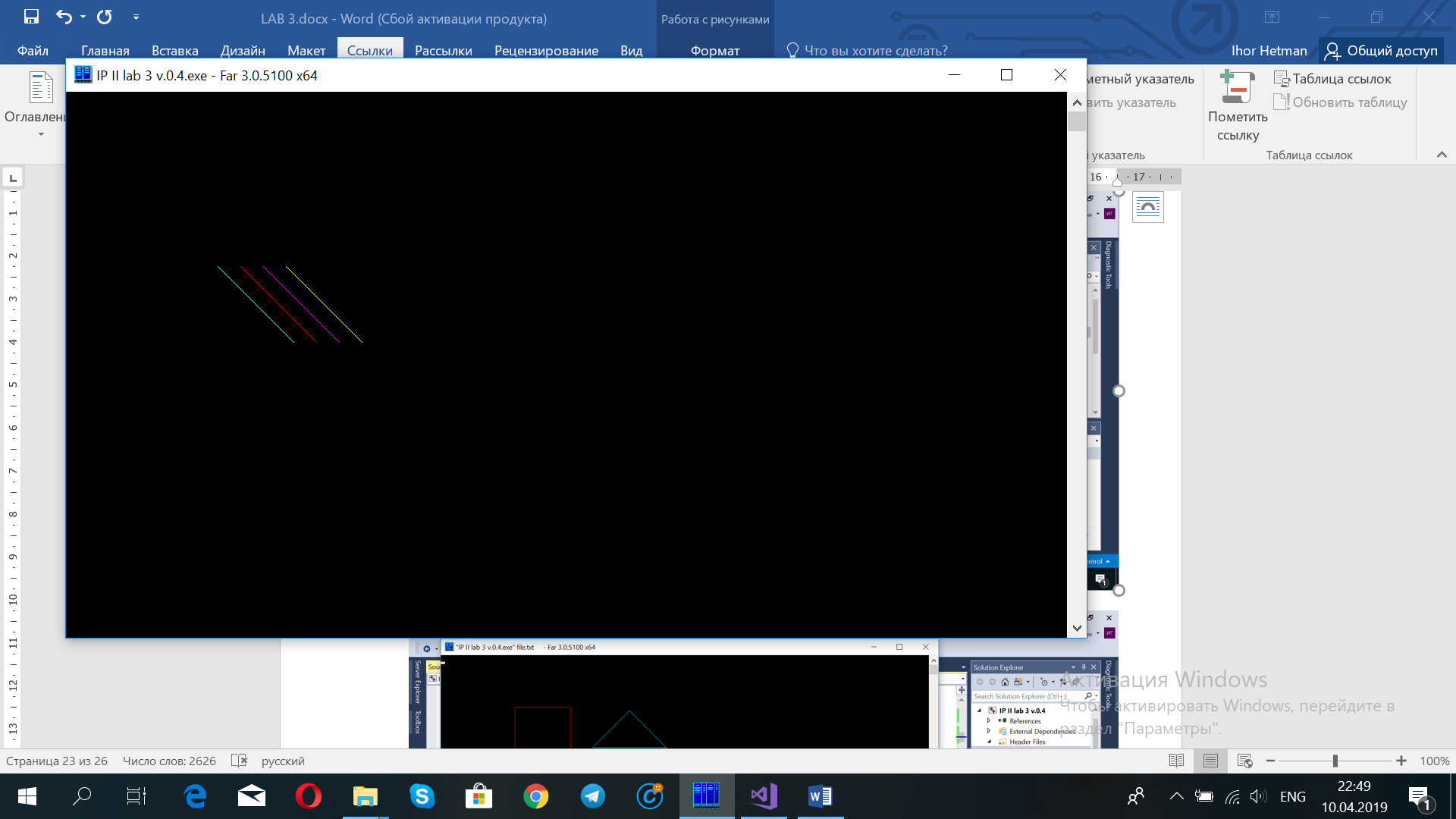
Щоб завантажити фігури до файлу, ім’я якого було введене в командному рядку, натисніть «b»

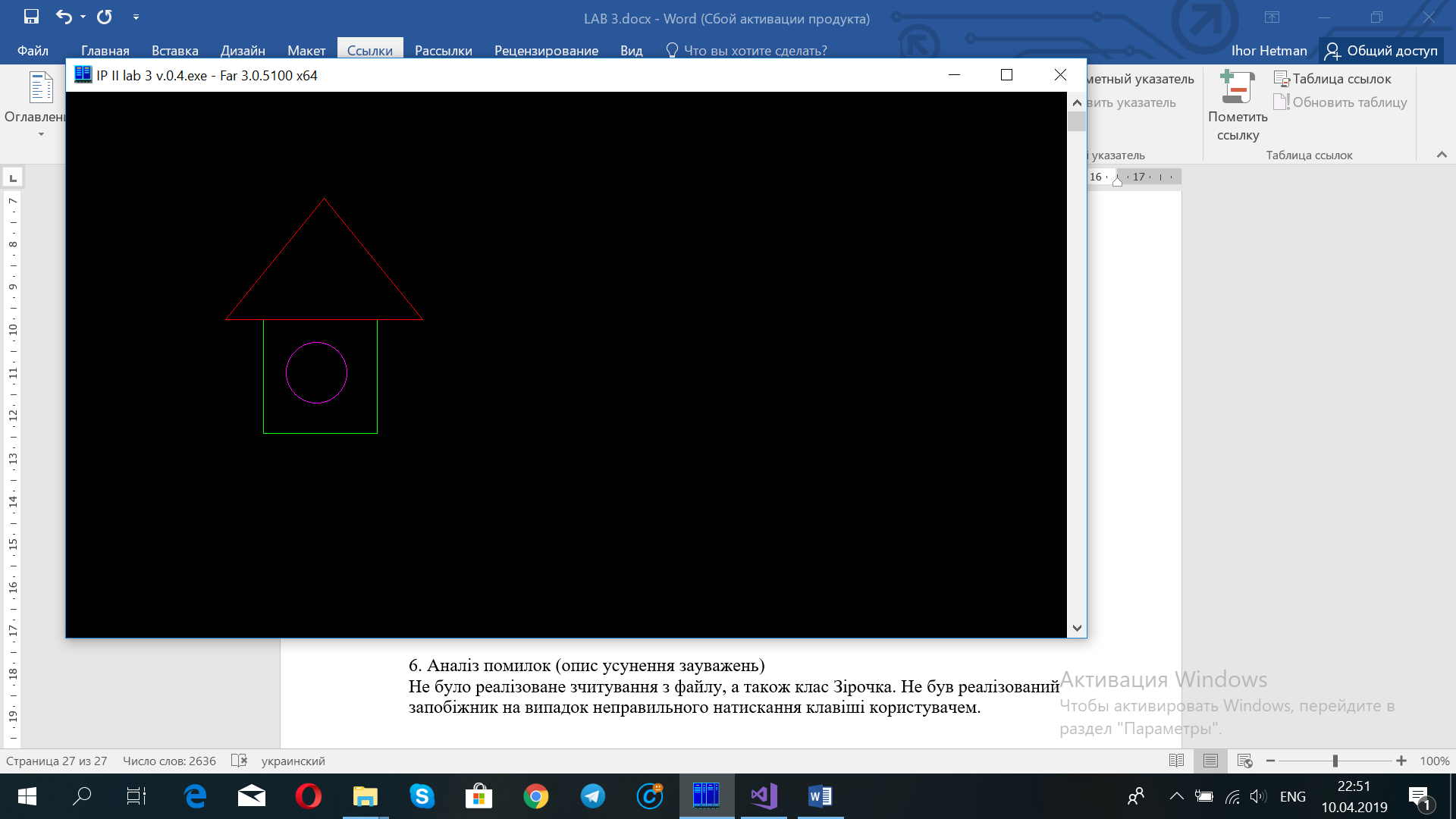


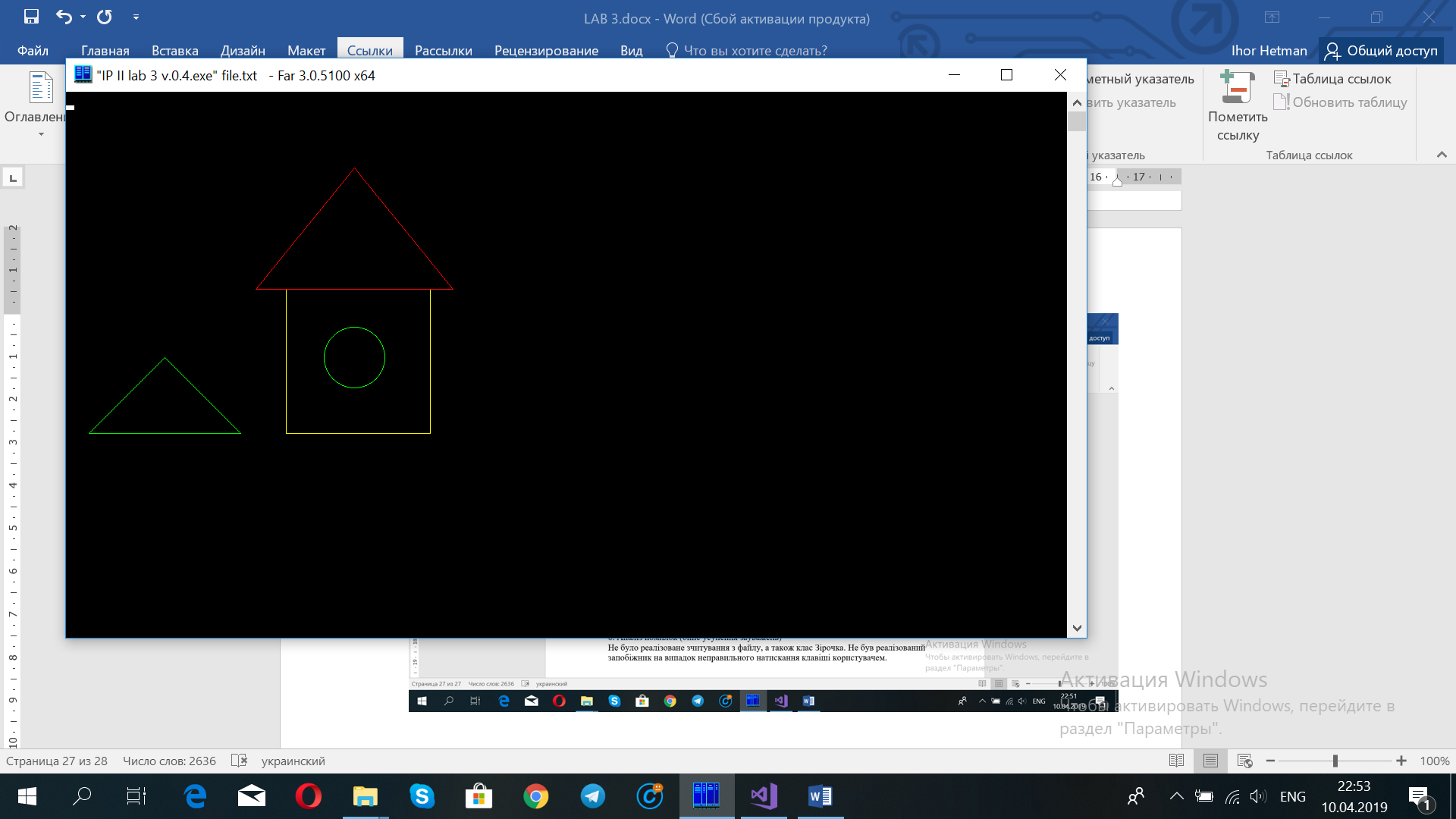
Щоб вийти з програми, натисніть Escape

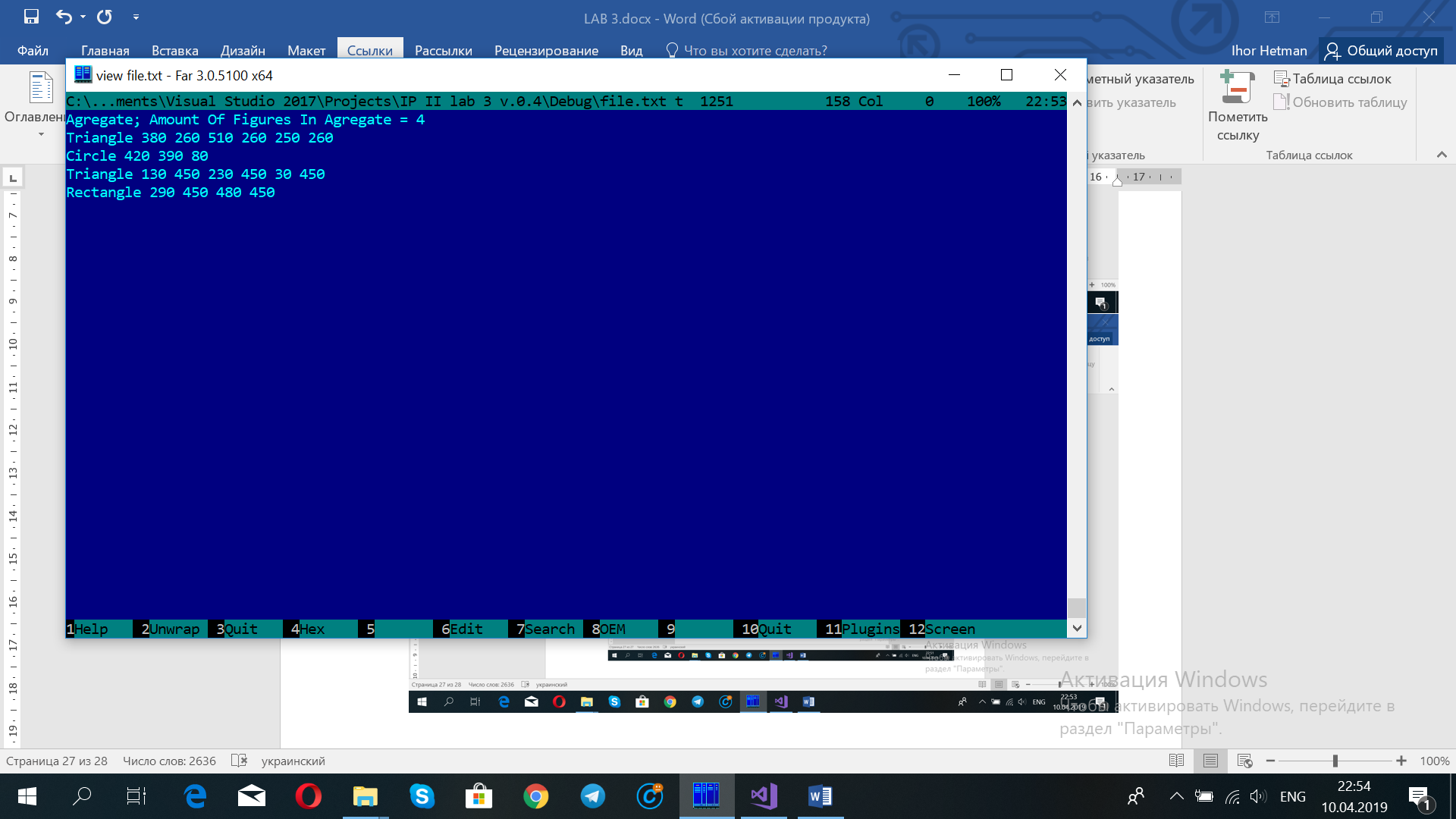
5. Опис тестових прикладів











6. Аналіз помилок (опис усунення зауважень)

Не було реалізоване зчитування з файлу, а також клас Зірочка. Не був реалізований запобіжник на випадок неправильного натискання клавіші користувачем.