ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ

за курсом «Інформатика і Програмування»

студента групи ПА-18-1

Мамедова Артура Шахіновича

кафедра комп’ютерних технологій,

ДНУ 2019/2020 н.р.

# The Task

## General task

To develop an object-oriented program in C++, that manages graphical objects, that display on the screen, in dialog mode. By pressing F1, the program displays a brief help note about all available commands/hotkeys, for example: how to create/move/switch object and so on.

The program must support the following general elements of behavior of graphical objects:

1. Activation/visualization of a graphical object by the user’s choice.
2. Move with/without trace.
3. Reset the original state of the object.
4. Color change.
5. Hiding/showing object.
6. Aggregation – creation of a new object from the already created ones. Aggregation of aggregates shall also be supported.

The program shall provide the user with an opportunity to save the current program state to the selected by user text file on a drive and run the program with the configuration set in file.

Graphical objects are suggested to be generated on the base of the following objects: dot, line, triangle, rectangle, circle, star. For example: ring, star in ring, circle in triangle and so on.

## Individual task:

1. The configuration file is set as a command prompt argument
2. The object motion in automatic mode: by the remembered trace.
3. Color change: by interaction with the other object.
4. Program deformation: by interaction with the other object.
5. Building/memorization of an aggregate object: by deleting objects, of which it consists.

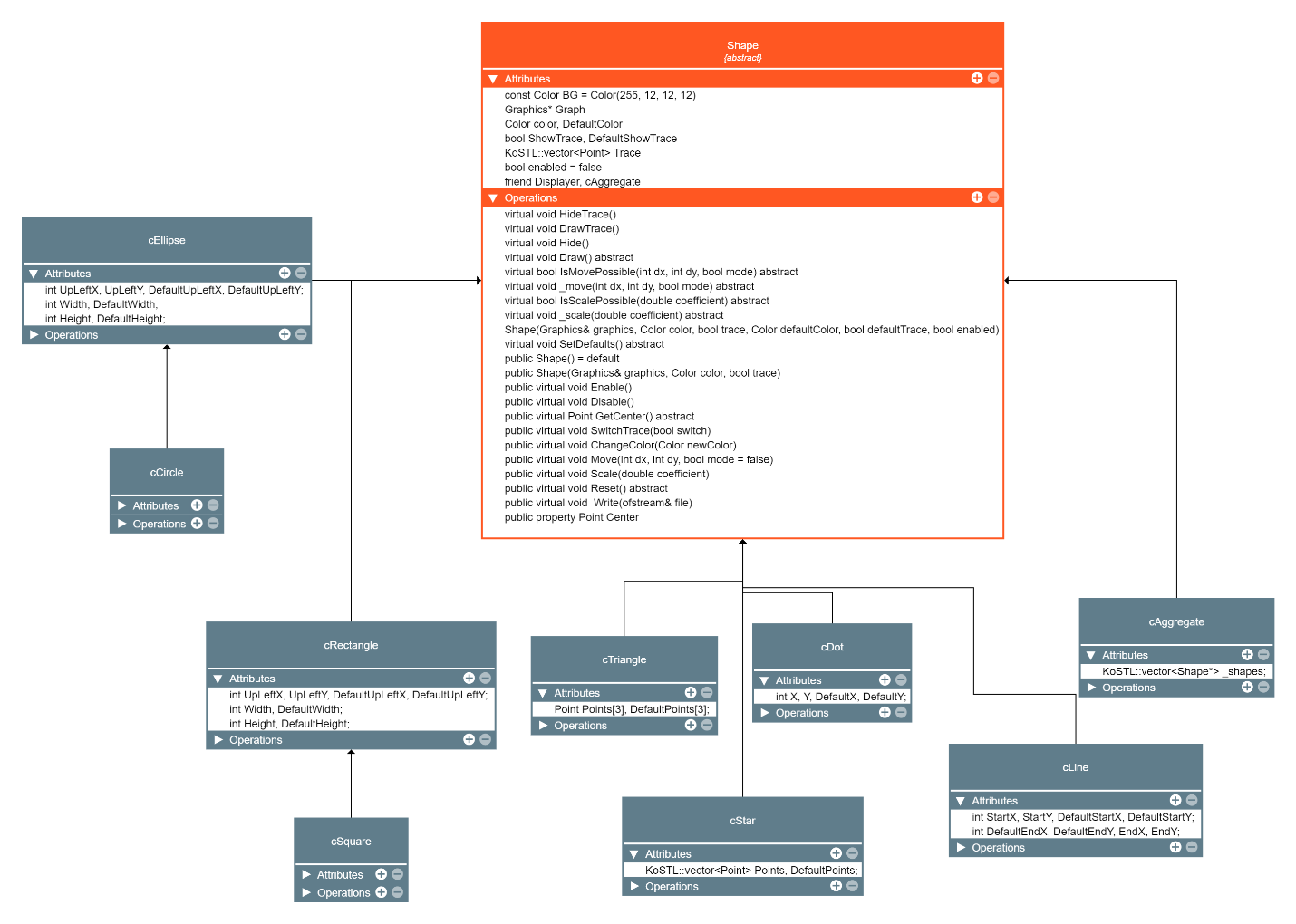
# Solution description

As a graphical engine for drawing the shapes I decided to use **GDI+**.

The graphical objects that can be drawn:

* **Square**
* **Circle**
* **Triangle**
* **Rectangle**
* **Dot**
* **Line**
* **Ellipse**
* **Star**
* **Aggregate**

As for program structure: I have an abstract class **Shape**, that other graphical objects inherit from. Then I have a class **Displayer**, that manages all the graphical objects on the screen. Inherently, **Displayer** is a **KoSTL::vector** of pointers to shapes and all the methods of **Displayer** just call the appropriate methods for each pointer in it. And also, I have a class **Factory**, that helps to assemble shapes in runtime and restore them from file.

Here is the scheme of inheritance:

# The Code

Here are the main classes the program consists of (to make code more clear I omitted the implementation of methods):

## Class Shape

class Shape abstract {

protected:

    const Color BG = Color(255, 12, 12, 12);

    Graphics\* Graph;

    Color color, DefaultColor;

    bool ShowTrace, DefaultShowTrace;

    KoSTL::vector<Point> Trace;

    bool enabled = false;

    virtual void HideTrace();

    virtual void DrawTrace();

    virtual void Hide();

    virtual void Draw() abstract;

    virtual bool IsMovePossible(int dx, int dy, bool mode) abstract;

    virtual void \_move(int dx, int dy, bool mode) abstract;

    virtual bool IsScalePossible(double coefficient) abstract;

    virtual void \_scale(double coefficient) abstract;

    Shape(Graphics& graphics, Color color, bool trace, Color defaultColor, bool defaultTrace, bool enable);

    virtual void SetDefaults() abstract;

    friend class Displayer;

    friend class cAgregate;

public:

    Shape() = default;

    Shape(Graphics& graphics, Color color, bool trace);

    virtual void Enable();

    virtual void Disable();

    virtual Point GetCenter() abstract;

    virtual void SwitchTrace(bool switcher);

    virtual void ChangeColor(Color newColor);

    virtual void Move(int dx, int dy, bool mode = false);

    virtual void Scale(double coefficient);

    virtual void Reset() abstract;

    virtual void Write(ofstream& file);

    \_\_declspec(property(get = GetCenter)) Point Center;

};

## Class Aggregate

(I accidently misspelled the name of the class, so now it has only one “g”)

class cAgregate : public Shape {

protected:

    KoSTL::vector<Shape\*> \_shapes;

    void Draw() override;

    bool IsMovePossible(int dx, int dy, bool mode) override;

    void \_move(int dx, int dy, bool mode) override;

    bool IsScalePossible(double coefficient) override;

    void \_scale(double coefficient) override;

public:

    void SetDefaults() override;

    cAgregate(Graphics& graphics, KoSTL::vector<Shape\*>& shapes, Color color, bool trace);

    cAgregate(Graphics& graphics, KoSTL::vector<Shape\*>& shapes, Color color, bool trace, Color defaultColor, bool defaultTrace, bool enabled);

    void ChangeColor(Color newColor) override;

    void Reset() override;

    void Write(ofstream& file) override;

    void Enable() override;

    void Disable() override;

    Point GetCenter() override;

    ~cAgregate();

};

## Class Displayer

class Displayer {

private:

    Graphics\* graphics;

    KoSTL::vector<Shape\*> Shapes;

    void drawer();

    size\_t sensitivity;

public:

    int GetSize();

    size\_t GetSensitivity();

    void SetSensitivity(size\_t sensitivity);

    \_\_declspec(property(get = GetSize)) int Size;  // Thanks Microsoft for this sugar

    \_\_declspec(property(get = GetSensitivity, put = SetSensitivity)) size\_t Sensitivity;

    Displayer(Graphics\* graphics);

    void Add(int shape, Color color = Color(255, 255, 255, 255), bool trace = false, KoSTL::vector<int> coordinates = {});

    void Erase(int);

    void FollowTrace(int index);

    void Init();

    void Move(int dx, int dy, bool mode, int index);

    void ChangeColor(Color newColor, int index);

    void Scale(double coefficient, int index);

    void Reset(int index);

    void SwitchTrace(bool switcher, int index);

    void Disable(int index);

    void Enable(int index);

    void DrawField(std::string file\_path = "", bool redraw\_shapes = false);

    void Blink(int index);

    void Read(string& file);

    void Write(string& file);

    ~Displayer();

    friend class Factory;

};

## Class Factory

class Factory {

private:

    Displayer\* disp;

    Graphics\* graph;

public:

    Factory(Displayer\* disp, Graphics\* graph) {

        this->disp = disp;

        this->graph = graph;

    };

    Shape\* MakeShape(int shape, Color color = DefaultColor, bool trace = false, KoSTL::vector<int> shapes\_for\_vector = {});

    Shape\* RestoreShape(int shape, KoSTL::vector<int> params);

};

## Main program

int main(int argc, char\*\* argv) {

std::system("cls");

GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

HWND hWnd = GetConsoleWindow();

HDC hdc = GetDC(hWnd);

Graphics graphics(hdc);

string file = "";

if (argc > 1)

file = argv[1];

Displayer disp(&graphics);

disp.Sensitivity = 15;

if(argc >= 1)

disp.Read(file);

disp.DrawField(file);

int index = -1, figure\_number = disp.Size, shape\_number = 0;

while (true) {

char switcher = \_getch();

switch (switcher) {

case 'w':

disp.Move(0, -10, false, index); break;

case 'a':

disp.Move(-10, 0, false, index); break;

case 's':

disp.Move(0, 10, false, index); break;

case 'd':

disp.Move(10, 0, false, index); break;

case '+': case '=':

AddShape(shape\_number, {}, figure\_number, disp, index);

break;

case'f':

disp.FollowTrace(index);

break;

case '-':

DeleteShape(disp, index, figure\_number);

break;

case 'c':

ChangeColorShape(disp, index);

break;

case 'r':

disp.Reset(index); break;

case 't':

disp.SwitchTrace(true, index); break;

case 'e':

disp.Scale(0.5, index); break;

case 'q':

disp.Scale(2, index); break;

case 'T':

disp.SwitchTrace(false, index); break;

case 'O':

if (!gdiplusToken)

GdiplusShutdown(gdiplusToken);

return 0;

case 'h':

disp.Disable(index);

break;

case 'R':

if (argc > 1) disp.Write(file);

else cout << (char)7;

break;

case'H':

disp.Enable(index); break;

case '>': case '.':

GoToNextShape(index, figure\_number, disp); break;

case '<': case ',':

GoToPrevStep(index, figure\_number, disp); break;

case char(0):

if (\_getch() == char(59)) {

ShowHelp();

system("pause");

system("cls");

disp.Init();

disp.DrawField(file);

}

break;

default: break;

}

}

if (!gdiplusToken)

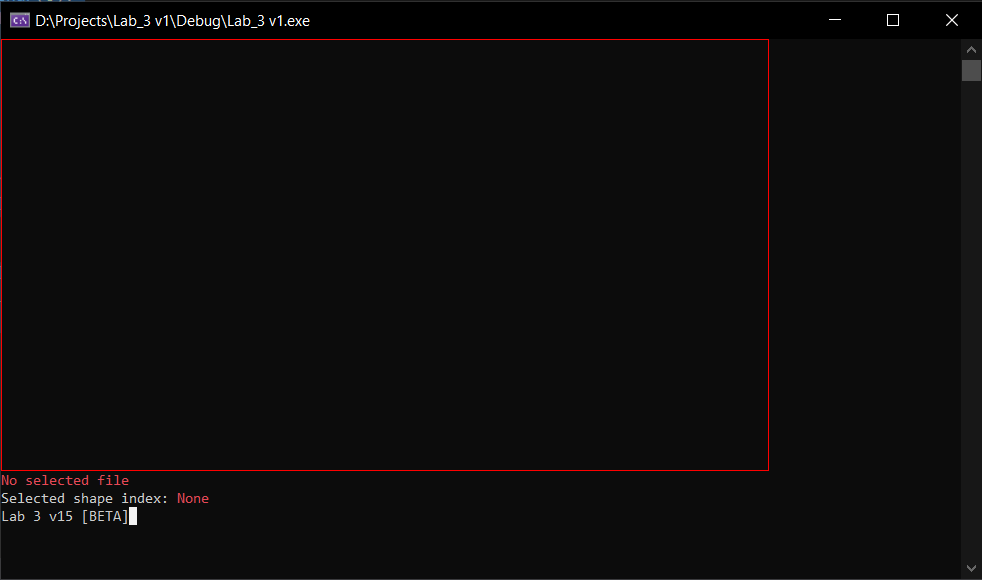
GdiplusShutdown(gdiplusToken);

return 0;

}

# Interface description

Interface is pretty simple.

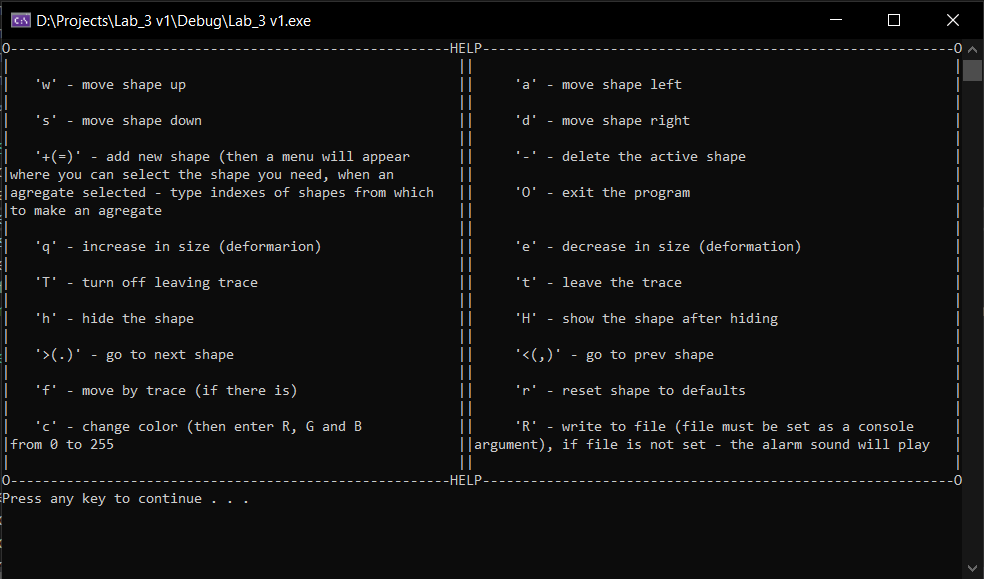


There is a field (highlighted with the red lines), which graphical objects can’t cross. Also, there are several inscriptions that show the selected shape’s index, selected file and version of the program.

Main commands and keys:

* “w” – move graphical object upward
* “s” – move graphical object downward
* “a” – move graphical object leftward
* “d” – move graphical object rightward
* “c” – change graphical object’s color
* “+” or “=” – add new graphical object (than a menu will appear in which you can select the necessary object)
* “-“ – delete the current selected object
* “O” – exit the program
* “q” – increase the size of the selected object
* “e” – decrease the size of the selected object
* “t” – turn on trace
* “T” – turn off trace
* “h” – hide the object
* “H” – show the object
* “>” or “.” – go to next object
* “<” or “,” – go to previous object
* “f” – automatic movement following the trace if there is
* “r” – reset shape to the original state
* “R” – write the program state to a file
* “F1” – show help

Here is the help message:



As for reading from or writing to a file, it is possible **only in the case** when it’s set as argument of a command prompt.

# Examples

