ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. О.ГОНЧАРА ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА КОМП’ЮТЕРНИХ ТЕХНОЛОГІЙ

# ЗВІТ

# З ЛАБОРАТОРНОЇ РОБОТИ №3

# за курсом «Програмування»

# Варіант № 8

Виконав

студент групи МА-21-1

Шайхулов Іван

Дніпро, 2024

1. Постановка задачі

2. Опис розв’язку

3. Вихідний код програми

4. Опис інтерфейсу програми(робота з користувачем)

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 Q 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.

Загальні вимоги

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

1. Створення довільної кількості графічних об’єктів із фіксованої множини їх різновидів.

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

3. Виконання операцій над поточним (активним) об’єктом  
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/)).

# Опис розв’язку

Завдання було виконано за допомогою основних інструментів Visual Studio 2022 та бібліотеки gdiplus та інших. Вимоги були частково виконані і показані у реалізації. Меню відображає всі функціональності програми.

Програма реалізована як багатомодульна. У header-файлі записано ініціалізацію всіх функцій, в одному з .cpp файлів реалізовано функції, а в іншому основна функція main.

Файл з функцією main() має додання нових фігур (класів) та

Отримані результати виводяться на екран, опісля створюється текстовий файл, ім’я якого програма також отримує з командного рядка. Якщо, в командному рядку ім’я не задане, то програма входить в діалог з користувачем для його отримання. Далі програма зберігає дані, отриманні під час виконання всієї програми, у вихідному файлі.

Вихідний код програми

**Main.cpp**  
#include "graphics.h"

#pragma comment(lib, "Gdiplus.lib")

Grfx::Graphics console\_graphics;

const char MENU = 'q';

const char TRAJ = 'z';

const char TRAJ2 = 'x';

const char SaveToFile = 'f';

const char ReadFromFile = 'l';

const char ChangeObject = 'o';

const char AddObject = 'g';

const char ChangeColor = 'r';

const char ChangeSize = 'm';

const char Hideobject = 'h';

const char Showobject = 'j';

const char ClearScreen = 'c';

const char ScreenSize = 'v';

const char ShapeTrail = 't';

const char UP = 'w';

const char DOWN = 's';

const char LEFT = 'a';

const char RIGHT = 'd';

const int BGCOLOR = 0;

const int COLOR = 2;

const int STEP = 10;

class Shape

{

protected:

int x, y, color, size;

bool drawTrail = false;

void drawPixel(int x, int y, int c) {

console\_graphics.setcolor(c);

console\_graphics.rectangle(x, y, x + 1, y + 1);

}

public:

Shape(int a, int b, int c) : x(a), y(b), color(c), size(1), drawTrail(false) {}

virtual ~Shape() {};

virtual void draw(int c) = 0;

virtual void move(int dx, int dy) = 0;

virtual void setColor(int c) { color = c; }

virtual void setSize(int s) { size = s; }

virtual int getSize() { return size; }

virtual void resize(int delta) { size += delta; }

void show() { draw(color); }

void hide() { draw(BGCOLOR); }

void SetTrail(bool value) { drawTrail = value; }

void toggleTrail() { drawTrail = !drawTrail; }

int getX() { return this->x; }

int getY() { return this->y; }

int getColor() { return this->color; }

bool getDrawTrail() { return this->drawTrail; }

virtual std::string getType() const {

return "Shape";

}

};

class Segment : public Shape

{

int dx, dy;

std::vector<std::pair<int, int>> trail;

public:

Segment(int a, int b, int da, int db, int c) : Shape(a, b, c), dx(da), dy(db) { show(); }

void draw(int c) override {

console\_graphics.setcolor(c);

console\_graphics.line(x, y, x + dx, y + dy);

if (drawTrail) {

for (const auto& point : trail) {

drawPixel(point.first, point.second, c);

}

}

}

int getSize() override {

return this->dx;

}

void resize(int delta) override {

dx += delta;

dy += delta;

}

void setColor(int c) override {

color = c;

}

void move(int dx, int dy) override {

hide();

if (drawTrail) {

trail.push\_back(std::make\_pair(x, y));

}

x += dx;

y += dy;

show();

}

void setSize(int s) override {

double ratio = static\_cast<double>(s) / getSize();

dx = static\_cast<int>(dx \* ratio);

dy = static\_cast<int>(dy \* ratio);

}

std::string getType() const override {

return "Segment";

}

};

class Star : public Shape

{

int innerRadius;

int outerRadius;

std::vector<std::pair<int, int>> trail; // Trail coordinates

public:

Star(int a, int b, int inner, int outer, int c) : Shape(a, b, c), innerRadius(inner), outerRadius(outer) { show(); }

void setColor(int c) override {

color = c;

}

int getSize() override {

return innerRadius;

}

void resize(int delta) override {

innerRadius += delta;

outerRadius += delta;

}

void draw(int c) override {

console\_graphics.setcolor(c);

// Draw the star using lines

for (int i = 0; i < 360; i += 36) {

double angle1 = i \* M\_PI / 180;

double angle2 = (i + 36) \* M\_PI / 180;

int x1 = static\_cast<int>(x + outerRadius \* cos(angle1));

int y1 = static\_cast<int>(y + outerRadius \* sin(angle1));

int x2 = static\_cast<int>(x + innerRadius \* cos(angle2));

int y2 = static\_cast<int>(y + innerRadius \* sin(angle2));

console\_graphics.line(x1, y1, x2, y2);

}

if (getDrawTrail()) {

// Draw the trail for stars

for (const auto& point : trail) {

drawPixel(point.first, point.second, c);

}

}

}

void move(int dx, int dy) override {

hide();

if (getDrawTrail()) {

trail.push\_back(std::make\_pair(x, y));

}

x += dx;

y += dy;

show();

}

void setSize(int s) override {

double ratio = static\_cast<double>(s) / getSize();

innerRadius = static\_cast<int>(innerRadius \* ratio);

outerRadius = static\_cast<int>(outerRadius \* ratio);

}

std::string getType() const override {

return "Star";

}

protected:

void drawTrailPixels(int c) {

for (const auto& point : trail) {

drawPixel(point.first, point.second, c);

}

}

};

class MyRectangle : public Shape

{

int width, height;

std::vector<std::pair<int, int>> trail;

public:

MyRectangle(int a, int b, int w, int h, int c) : Shape(a, b, c), width(w), height(h) { show(); }

void draw(int c) override {

console\_graphics.setcolor(c);

console\_graphics.rectangle(x, y, x + width, y + height);

if (drawTrail) {

for (const auto& point : trail) {

drawPixel(point.first, point.second, c);

}

}

}

void move(int dx, int dy) override {

hide();

if (drawTrail) {

trail.push\_back(std::make\_pair(x, y));

}

x += dx;

y += dy;

show();

}

void setSize(int s) override {

double ratio = static\_cast<double>(s) / getSize();

width = static\_cast<int>(width \* ratio);

height = static\_cast<int>(height \* ratio);

}

};

class Circle : public Shape

{

int radius;

std::vector<std::pair<int, int>> trail;

public:

Circle(int a, int b, int r, int c) : Shape(a, b, c), radius(r) { show(); }

std::string getType() const override {

return "Circle";

}

void setColor(int c) override {

color = c;

}

void draw(int c) override {

console\_graphics.setcolor(c);

console\_graphics.circle(x, y, radius);

if (drawTrail) {

for (const auto& point : trail) {

drawPixel(point.first, point.second, c);

}

}

}

void resize(int delta) override {

radius += delta;

}

void move(int dx, int dy) override {

hide();

if (drawTrail) {

trail.push\_back(std::make\_pair(x, y));

}

x += dx;

y += dy;

show();

}

int getSize() override

{

return this->radius;

}

void setSize(int s) override {

double ratio = static\_cast<double>(s) / getSize();

radius = static\_cast<int>(radius \* ratio);

}

};

class Square : public Shape

{

int side;

std::vector<std::pair<int, int>> trail;

public:

Square(int a, int b, int s, int c) : Shape(a, b, c), side(s) { show(); }

void setColor(int c) override {

color = c;

}

int getSize() override {

return this->side;

}

std::string getType() const override {

return "Square";

}

void draw(int c) override {

console\_graphics.setcolor(c);

console\_graphics.rectangle(x, y, x + side, y + side);

if (drawTrail) {

for (const auto& point : trail) {

drawPixel(point.first, point.second, c);

}

}

}

void resize(int delta) override {

side += delta;

}

void move(int dx, int dy) override {

hide();

if (drawTrail) {

trail.push\_back(std::make\_pair(x, y));

}

x += dx;

y += dy;

show();

}

void setSize(int s) override {

double ratio = static\_cast<double>(s) / getSize();

side = static\_cast<int>(side \* ratio);

}

};

void menu(const std::vector<Shape\*>& objects) {

for (const auto& obj : objects)

{

obj->hide();

}

std::cout << "Выберите действие:" << std::endl;

std::cout << SaveToFile << " - Сохранить в файл" << std::endl;

std::cout << ReadFromFile << " - Прочитать из файла" << std::endl;

std::cout << TRAJ << " - Запомнить траекторию" << std::endl;

std::cout << TRAJ2 << " - Двигатся по траектории" << std::endl;

//std::cout << Composite << " - Создать агрегат" << std::endl;

std::cout << ChangeObject << " - Изменить объект" << std::endl;

std::cout << AddObject << " - Добавить объект" << std::endl;

std::cout << Hideobject << " - Скрыть объект" << std::endl;

std::cout << Showobject << " - Показать объект" << std::endl;

std::cout << ChangeSize << " - Изменить размер объекта" << std::endl;

std::cout << ChangeColor << " - Изменить цвет объекта" << std::endl;

std::cout << ClearScreen << " - Очистить экран" << std::endl;

std::cout << ScreenSize << " - Изменить размер экрана" << std::endl;

std::cout << ShapeTrail << " - Отобразить/скрыть траекторию объекта" << std::endl;

std::cout << UP << " - Двигаться вверх" << std::endl;

std::cout << DOWN << " - Двигаться вниз" << std::endl;

std::cout << LEFT << " - Двигаться влево" << std::endl;

std::cout << RIGHT << " - Двигаться вправо" << std::endl;

system("pause");

system("cls");

for (const auto& obj : objects)

{

obj->show();

}

}

void clearConsoleLine(int line) {

COORD cursorPosition;

cursorPosition.X = 0;

cursorPosition.Y = line;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), cursorPosition);

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);

DWORD written;

FillConsoleOutputCharacter(GetStdHandle(STD\_OUTPUT\_HANDLE), ' ', csbi.dwSize.X, cursorPosition, &written);

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), cursorPosition);

}

void addObject(std::vector<Shape\*>& objects) {

std::cout << "Выберите тип создаваемого объекта (1 - Segment, 2 - Circle, 3 - Square, 4 - Star): ";

char choice;

std::cin >> choice;

clearConsoleLine(0);

std::cin.ignore(32767, '\n');

switch (choice) {

case '1':

objects.push\_back(new Segment(200, 200, 100, 100, COLOR));

break;

case '2':

objects.push\_back(new Circle(300, 300, 50, COLOR));

break;

case '3':

objects.push\_back(new Square(400, 400, 50, COLOR));

break;

case '4':

objects.push\_back(new Star(500, 400, 30, 20, COLOR));

break;

default:

std::cout << "Введен некорректный номер." << std::endl;

clearConsoleLine(0);

std::cin.ignore(32767, '\n');

break;

}

}

int switchObject(const std::vector<Shape\*>& objects) {

if (objects.size() >= 2) {

int obj = 0;

std::cout << "Введите номер объекта, с которым вы хотите работать( всего объектов активно: " << objects.size() << ")";

std::cin >> obj;

clearConsoleLine(0);

std::cin.ignore(32767, '\n');

std::cin.clear();

while (obj < 0 || obj > objects.size())

{

std::cout << "Пожалуйста, введите корректный номер объекта";

std::cin >> obj;

clearConsoleLine(0);

std::cin.ignore(32767, '\n');

std::cin.clear();

}

return obj - 1;

}

return int(objects.size() - 1);

}

void resizeObject(Shape\*& object) {

std::cout << "Введите новый размер для текущего объекта: ";

int newSize;

std::cin >> newSize;

clearConsoleLine(0);

std::cin.ignore(32767, '\n');

if (std::cin.fail()) {

std::cin.clear();

std::cin.ignore(32767, '\n');

std::cout << "Не удалось записать. Попробуйте еще раз"; std::cin >> newSize;

clearConsoleLine(0);

std::cin.ignore(32767);

}

else {

object->hide();

object->resize(object->getSize() + newSize);

}

}

void Recolor(Shape\*& object)

{

std::cout << "Выберите цвет 1 - Красный, 2 - Синий, 3 - Фиолетовый, 4 - Белый ";

int color;

std::cin >> color;

clearConsoleLine(0);

std::cin.ignore(32767, '\n');

switch (color)

{

case 1:

object->setColor(4);

break;

case 2:

object->setColor(1);

break;

case 3:

object->setColor(5);

break;

case 4:

object->setColor(7);

break;

default:

Recolor(object);

break;

}

}

void SFile(const std::vector<Shape\*> objects) {

std::string filename;

std::cin.clear();

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::cout << "Введите путь к файлу: ";

std::getline(std::cin, filename);

clearConsoleLine(0);

std::cin.ignore(32767, '\n');

std::cin.clear();

std::ofstream file(filename);

if (file.is\_open()) {

for (const auto& obj : objects) {

file << obj->getType() << " "

<< obj->getX() << " "

<< obj->getY() << " "

<< obj->getSize() << " "

<< obj->getColor() << " \n";

}

file.close();

}

SetConsoleCP(866);

SetConsoleOutputCP(866);

}

void RFile(std::vector<Shape\*>& objects) {

for (const auto& obj : objects)

{

obj->hide();

}

std::string filename;

std::cin.clear();

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::cout << "Введите путь к файлу: ";

std::getline(std::cin, filename);

clearConsoleLine(0);

std::cin.clear();

std::ifstream file(filename);

if (file.is\_open()) {

std::string line;

while (std::getline(file, line)) {

std::istringstream iss(line);

std::string objectType;

iss >> objectType;

int x, y, size, color;

iss >> x >> y >> size >> color;

if (objectType == "Segment") {

objects.push\_back(new Segment(x, y, size, size, color));

}

else if (objectType == "Circle") {

objects.push\_back(new Circle(x, y, size, color));

}

else if (objectType == "Square") {

objects.push\_back(new Square(x, y, size, color));

}

else if (objectType == "Star") {

objects.push\_back(new Star(x, y, size, 2 \* size / 3, color));

}

}

file.close();

}

for (const auto& obj : objects)

{

obj->show();

}

}

void trajectory(std::vector <int>& t, const char step)

{

if (step == UP) { t.push\_back(0); return; }

if (step == DOWN) { t.push\_back(1); return; }

if (step == LEFT) { t.push\_back(2); return; }

if (step == RIGHT) { t.push\_back(3); return; }

}

char trajectory2(const std::vector <int>& t, const int i)

{

if (t[i] == 0) return UP;

if (t[i] == 1) return DOWN;

if (t[i] == 2) return LEFT;

if (t[i] == 3) return RIGHT;

return \_getch();

}

int main() {

std::vector<Shape\*> objects;

objects.push\_back(new Segment(200, 200, 100, 100, COLOR));

std::vector <int> t;

setlocale(LC\_ALL, "russian");

int iter = 0, i = 0;

bool tr1 = false, tr2 = false;

char c = 0;

while (c != 27)

{

/\*if (GetAsyncKeyState(VK\_LEFT) & 0x8000) objects.at(iter)->move(-STEP, 0);

if (GetAsyncKeyState(VK\_RIGHT) & 0x8000) objects.at(iter)->move(STEP, 0);

if (GetAsyncKeyState(VK\_UP) & 0x8000) objects.at(iter)->move(0, -STEP);

if (GetAsyncKeyState(VK\_DOWN) & 0x8000) objects.at(iter)->move(0, STEP);\*/

if (tr1)

{

trajectory(t, c);

}

if (tr2)

{

c = trajectory2(t, i);

if (i == t.size() - 1) {

trajectory2(t, i);

tr2 = false;

i = 0;

c = \_getch();

}

else i++;

}

else

{

c = \_getch();

}

switch (c)

{

case UP:

objects.at(iter)->move(0, -STEP);

break;

case DOWN:

objects.at(iter)->move(0, STEP);

break;

case LEFT:

objects.at(iter)->move(-STEP, 0);

break;

case RIGHT:

objects.at(iter)->move(STEP, 0);

break;

case MENU:

menu(objects);

break;

case ChangeColor:

Recolor(objects.at(iter));

break;

case Showobject:

objects.at(iter)->show();

break;

case ChangeSize:

resizeObject(objects.at(iter));

break;

case TRAJ:

if (tr1 == false && t.size() != 0) { tr1 = true; t.clear(); break; }

if (tr1 == false) { tr1 = true; break; }

tr1 = false;

break;

case TRAJ2:

if (tr2 == false) { tr2 = true; break; }

tr2 = false;

break;

case SaveToFile:

SFile(objects);

break;

case ReadFromFile:

RFile(objects);

break;

case Hideobject:

objects.at(iter)->hide();

break;

case ClearScreen:

console\_graphics.cls();

break;

case ShapeTrail:

if (!objects.empty()) {

objects.at(iter)->toggleTrail();

}

break;

case ScreenSize:

std::cout << console\_graphics.hSize() << ' ' << console\_graphics.vSize() << std::endl;

break;

case AddObject:

addObject(objects);

iter++;

break;

case ChangeObject:

iter = switchObject(objects);

break;

default:

std::cin.clear();

c = \_getch();

break;

}

if (!objects.empty()) {

objects.at(iter)->show();

}

}

for (Shape\* obj : objects) {

delete obj;

}

return 0;

}  
**Source.cpp**  
//

// A.L. Khizha, 2016-10-22

//

#include "graphics.h"

using namespace Grfx;

Graphics::Graphics()

{

// Èíèöèàëèçàöèÿ ãðàôèêè

// Initialize GDI+.

Gdiplus::GdiplusStartupInput gdiplusStartupInput;

Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

hWnd = GetConsoleWindow();

hDC = GetDC(hWnd);

gr = new Gdiplus::Graphics(hDC); // Remarks (https://msdn.microsoft.com/en-us/library/ms536160(v=vs.85).aspx):

// When you use this constructor to create a Graphics::Graphics object,

// make sure that the Graphics::Graphics object is deleted or goes out of scope

// before the device context is released

// Â ÷àñòíîñòè, ýòî çàñòàâëÿåò ïîìåñòèòü àâòîìàòè÷åñêîå ñîçäàíèå îáúåêòà ãðàôèêè

// âíóòü ôóíêöèè - ëèáî ñîçäàâàòü/óíè÷òîæàòü îáúåêò äèíàìè÷åñêè.

// Èíà÷å âîçíèêàåò îøèáêà âî âðåìÿ âûïîëíåíèÿ ïðè ïîïûòêå GdiplusShutdown

Gdiplus::Color blackColor(255, 0, 0, 0);

gr->Clear(blackColor);

// 2017-04-07 12:21 alkhizha

windowSize();

}

Graphics::~Graphics()

{

delete gr;

Gdiplus::GdiplusShutdown(gdiplusToken);

ReleaseDC(hWnd, hDC);

}

void Graphics::setcolor(int c)

{

BYTE byRed = 0, byGreen = 0, byBlue = 0;

byRed = 255 \* (c & 0x4);

byGreen = 255 \* (c & 0x2);

byBlue = 255 \* (c & 0x1);

color = Gdiplus::Color(255, byRed, byGreen, byBlue);

}

void Graphics::line(int x, int y, int x2, int y2)

{

Gdiplus::Pen pen(color);

gr->DrawLine(&pen, x, y, x2, y2);

}

void Graphics::circle(int x, int y, int r)

{

Gdiplus::Pen pen(color);

gr->DrawEllipse(&pen, x - r, y - r, r, r);

}

void Graphics::rectangle(int x, int y, int x2, int y2)

{

Gdiplus::Pen pen(color);

gr->DrawLine(&pen, x, y, x, y2);

gr->DrawLine(&pen, x, y, x2, y);

gr->DrawLine(&pen, x2, y, x2, y2);

gr->DrawLine(&pen, x, y2, x2, y2);

}

// 2017-04-01 11:50 alkhizha

void Graphics::cls()

{

gr->Clear(Gdiplus::Color(0, 0, 0, 0));

}

// 2017-04-01 11:50 alkhizha

void Graphics::windowSize()

{

// Get a bounding rectangle for the clipping region.

Gdiplus::Rect boundRect;

gr->GetVisibleClipBounds(&boundRect);

boundRect.GetSize(&sz);

}

int Graphics::hSize() { return sz.Width; }

int Graphics::vSize() { return sz.Height; }

**Graphics.h**  
//

// A.L. Khizha, 2016-10-22

//

#ifndef \_GRAPHICS\_

#define \_GRAPHICS\_

#define \_USE\_MATH\_DEFINES

#include <windows.h>

#include <limits>

#include <math.h>

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

#include <vector>

#include <conio.h>

#include <objidl.h>

#include <gdiplus.h>

namespace Grfx

{

class Graphics

{

HWND hWnd;

HDC hDC;

Gdiplus::Color color;

Gdiplus::Graphics\* gr;

ULONG\_PTR gdiplusToken;

Gdiplus::Size sz;

public:

Graphics();

~Graphics();

void setcolor(int c);

void line(int x, int y, int x2, int y2);

void circle(int x, int y, int r);

void rectangle(int x, int y, int x2, int y2);

// 2017-04-01 11:50 alkhizha

void cls();

// 2017-04-01 11:50 alkhizha

void windowSize();

int hSize();

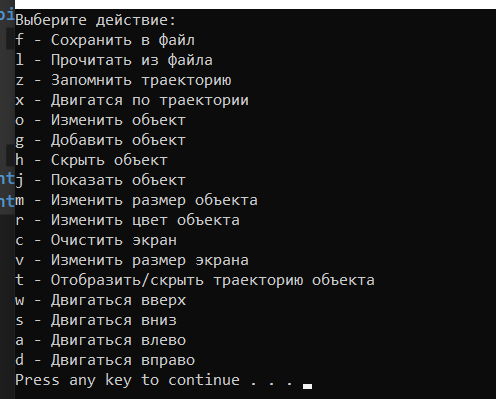
int vSize();

}; // class Graphics

}; // namespace Grfx

#endif

Опис інтерфейсу програми(діалог з користувачем)



Діалог з користувачем йде у вигляді меню, де він може побачити всі функції програми та зберегти ці дані у файл та зчитати. Графічне вікно є інтерфейсом для роботи з примітивами. Mожливо створювати та прцювати з різними фігурами. Наприклад: змінювати колір, деформувати, рухати