ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ № 3

«Поліморфізм»

з курсу «Об’єктно-орієнтоване програмування»

студентки групи ПА-21-2

Кондратьєвої Дар’ї Олександрівни

Кафедра комп’ютерних технологій, ДНУ

2023/2024 н.р.

Варіант № 11

1. **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. 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:
   1. переміщення зі слідом/без;
   2. збереження і відновлення стану об’єкту;
   3. зміна кольору;
   4. зміна розміру;
   5. зміна стану видимий/невидимий.
4. Агрегація, тобто утворення нових об’єктів з вже створених. При агрегації об’єктів забезпечити наступні можливості:
   1. агрегація агрегатів;
   2. можливість виконувати над агрегатами ті ж дії, що і над «звичайними» об’єктами.

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.

Етапи виконання лабораторної роботи

1) Описання класів, які будуть покладені в основу розробки програми (фрейми, схема ієрархії). Описання призначення класів, обґрунтування їх необхідності.

2) Реалізація основних функцій програми в текстовому режимі консолі (візуалізація об’єктів і зміна їх стану відображується текстом).

3) Реалізація всіх функцій програми в графічному режимі (графічний режим може бути реалізований в консолі – див. приклад згаданий вище).

4) Тестування функцій/режимів програми і документування тестів у звіті. Паралельно із створенням програмного проекту відбувається написання звіту (див. Вимоги до звіту в окремому файлі).

**Індивідуальні варіанти завдань**

11. 01001

1. Ім’я конфігураційного файлу:

1. задане у командному рядку при запуску програми.
2. користувач вказує в діалозі під час роботи програми.
3. Рух об’єкта в автоматичному режимі:
4. По заданому закону.
5. По запам’ятованій траєкторії.
6. Зміна кольору об’єкта:
7. По команді.
8. Під впливом іншого об’єкта.
9. Деформація об’єкта:
10. По команді.
11. Під впливом іншого об’єкта.
12. Збирання/запам’ятовування агрегованого образу:
13. Дублюванням об’єктів, з яких будується агрегат.
14. Видаленням об’єктів, з яких будується агрегат.
15. **Опис розв'язку:**

Запускаючи програму, користувач отримає повідомлення «Press K to print help»

При натисканні літери К, виведеться підказочка з можливими діями:

K: Show help

Arrow keys: Move

R: Change color

Z: Increase size

X: Decrease size

Num1-Num9: Switch objects

T: Create Circle

D: Create Square

L: Create Line

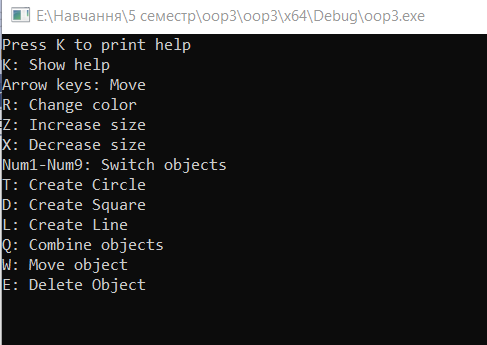
Q: Combine objects

W: Move object

E: Delete Object

Not enough elements to combine.

По натисканню клавіші К - виводиться меню:



Для реалізації в коді пишемо так:

if (event.key.code == sf::Keyboard::K) {

std::cout << "K: Show help \nArrow keys: Move\nR: Change color\n Z: Increase size\n X: Decrease size\n Num1-Num9: Switch objects\n T: Create Circle\n D: Create Square\n L: Create Line\n Q: Combine objects\n W: Move object \n E: Delete Object" << std::endl;

}

По натисканню клавіші Up/ Down - фігура рухається вгору/вниз.

По натисканню клавіші Right/Left - фігура рухається вправо/вліво.

Для реалізації, в коді пишемо так:

else if (event.key.code == sf::Keyboard::Up) {

elements[activeElement]->move(0, -5);

}

else if (event.key.code == sf::Keyboard::Left) {

elements[activeElement]->move(-5, 0);

}

else if (event.key.code == sf::Keyboard::Down) {

elements[activeElement]->move(0, 5);

}

else if (event.key.code == sf::Keyboard::Right) {

elements[activeElement]->move(5, 0);

}

По натисканню клавіші R - бере поточний колір і змінює його на новий колір.

else if (event.key.code == sf::Keyboard::R) {

sf::Color currentColor = elements[activeElement]->getColor();

sf::Color newColor;

…

По натисканню клавіші Z - воно збільшує розміри. Тобто бере старий розмір, і додає +1.

А по натисканню клавіші Х - зменшує розміри на -1.

else if (event.key.code == sf::Keyboard::Z) {

elements[activeElement]->setSize(elements[activeElement]->getSize() + 1);

}

else if (event.key.code == sf::Keyboard::X) {

int newSize = std::max(1, elements[activeElement]->getSize() - 1);

elements[activeElement]->setSize(newSize);

}

По натисканню клавіші Т - створює новий елемент на заданих координатах в коді(круг, лінію або ромб).

else if (event.key.code == sf::Keyboard::T) {

elements.push\_back(std::make\_unique<CircularElement>(100, 100, 50));

activeElement = elements.size() - 1;

}

По натисканню клавіші L - створюється лінія.

else if (event.key.code == sf::Keyboard::L) {

elements.push\_back(std::make\_unique<LineElement>(100, 100,100, 10));

activeElement = elements.size() - 1;

}

По натисканню клавіші D - створюється ромбик.

else if (event.key.code == sf::Keyboard::D) {

elements.push\_back(std::make\_unique<DiamondElement>(200, 200, 50));

activeElement = elements.size() - 1;

}

Якщо ми натиснемо Q, то якщо у нас елементів більше ніж 2, то тоді, відбуваються всі ці дії вже обьєднаних фігур.

Якщо елементів менше ніж 2, то виводиться відповідне повідомлення

else if (event.key.code == sf::Keyboard::Q) {

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

std::unique\_ptr<GraphicalElement> combinedElement = std::make\_unique<CombinedElement>(std::move(elements[elements.size() - 2]), std::move(elements[elements.size() - 1]));

elements.erase(elements.end() - 2, elements.end());

elements.push\_back(std::move(combinedElement));

activeElement = elements.size() - 1;

}

else {

std::cout << "Not enough elements to combine.\n";

}

По натисканню клавіші Е - видаляється елемент.

else if (event.key.code == sf::Keyboard::E) {

if (!elements.empty()) {

elements.erase(elements.begin() + activeElement);

if (activeElement >= elements.size()) {

activeElement = elements.size() - 1;

}

}

}

По натисканню клавіші W - змінюється положення фігури на 70 пікселей по руху квадрату.

else if (event.key.code == sf::Keyboard::W) {

for (int i = 0; i < 4; ++i) {

int moveAmount = 70;

switch (i) {

case 0: // Move to the right

elements[activeElement]->move(moveAmount, 0);

break;

case 1: // Move down

elements[activeElement]->move(0, moveAmount);

break;

case 2: // Move to the left

elements[activeElement]->move(-moveAmount, 0);

break;

case 3: // Move up

elements[activeElement]->move(0, -moveAmount);

break;

}

…

1. **Вихідний текст програми (з необхідними коментарями):**

#include <SFML/Graphics.hpp>

#include <iostream>

#include <memory>

#include <cstdlib>

#include <ctime>

#include <Windows.h>

class GraphicalElement {

public:

virtual void draw(sf::RenderWindow& window) = 0;

virtual void move(int x, int y) = 0;

virtual void setColor(sf::Color color) = 0;

virtual void setSize(int size) = 0;

virtual void setSizeWithDimensions(int length, int width) = 0; // Уникальное имя

virtual void setVisible(bool visible) = 0;

virtual sf::Color getColor() = 0;

virtual int getSize() = 0;

};

class LineElement : public GraphicalElement {

public:

LineElement(int x, int y, float length, float width) : line(sf::Vector2f(length, width)) {

line.setPosition(x, y);

line.setFillColor(sf::Color::Transparent);

line.setOutlineColor(sf::Color::White);

line.setOutlineThickness(1);

}

void draw(sf::RenderWindow& window) override {

window.draw(line);

}

void move(int x, int y) override {

line.move(x, y);

}

void setColor(sf::Color color) override {

// Не используется для линии

}

void setSize(int size) override {

line.setSize(sf::Vector2f(size, 1));

}

void setSizeWithDimensions(int length, int width) override {

line.setSize(sf::Vector2f(length, width));

}

void setVisible(bool visible) override {

line.setOutlineColor(visible ? sf::Color::White : sf::Color::Transparent);

}

sf::Color getColor() override {

return line.getOutlineColor();

}

int getSize() override {

return static\_cast<int>(line.getSize().x);

}

private:

sf::RectangleShape line;

};

class CircularElement : public GraphicalElement {

public:

CircularElement(int x, int y, int radius) : circle(radius) {

circle.setPosition(x, y);

}

void draw(sf::RenderWindow& window) override {

window.draw(circle);

}

void move(int x, int y) override {

circle.move(x, y);

}

void setColor(sf::Color color) override {

circle.setFillColor(color);

}

void setSize(int size) override {

circle.setRadius(size);

}

void setSizeWithDimensions(int length, int width) override {

// Для круга можно рассмотреть вариант, когда width игнорируется

circle.setRadius(length);

}

void setVisible(bool visible) override {

circle.setFillColor(visible ? sf::Color::White : sf::Color::Transparent);

}

sf::Color getColor() override {

return circle.getFillColor();

}

int getSize() override {

return circle.getRadius();

}

private:

sf::CircleShape circle;

};

class DiamondElement : public GraphicalElement {

public:

DiamondElement(int x, int y, int size) {

diamond.setPointCount(4);

diamond.setRadius(size);

diamond.setPosition(x, y);

}

void draw(sf::RenderWindow& window) override {

window.draw(diamond);

}

void move(int x, int y) override {

diamond.move(x, y);

}

void setColor(sf::Color color) override {

diamond.setFillColor(color);

}

void setSize(int size) override {

diamond.setRadius(size);

}

void setSizeWithDimensions(int length, int width) override {

diamond.setRadius(sqrt(length \* length + width \* width) / 2);

}

void setVisible(bool visible) override {

diamond.setFillColor(visible ? sf::Color::White : sf::Color::Transparent);

}

sf::Color getColor() override {

return diamond.getFillColor();

}

int getSize() override {

return static\_cast<int>(2 \* diamond.getRadius());

}

private:

sf::CircleShape diamond;

};

class CombinedElement : public GraphicalElement {

public:

CombinedElement(std::unique\_ptr<GraphicalElement> elem1, std::unique\_ptr<GraphicalElement> elem2)

: elem1(std::move(elem1)), elem2(std::move(elem2)) {

size = (this->elem1->getSize() + this->elem2->getSize()) / 2;

}

void draw(sf::RenderWindow& window) override {

elem1->draw(window);

elem2->draw(window);

}

void move(int x, int y) override {

elem1->move(x, y);

elem2->move(x, y);

}

void setColor(sf::Color color) override {

elem1->setColor(color);

elem2->setColor(color);

}

void setSize(int size) override {

elem1->setSize(size);

elem2->setSize(size);

this->size = size;

}

void setSizeWithDimensions(int length, int width) override {

int newSize = (elem1->getSize() + elem2->getSize()) / 2;

elem1->setSize(newSize);

elem2->setSize(newSize);

this->size = newSize;

}

void setVisible(bool visible) override {

elem1->setVisible(visible);

elem2->setVisible(visible);

}

sf::Color getColor() override {

return elem1->getColor();

}

int getSize() override {

return size;

}

private:

std::shared\_ptr<GraphicalElement> elem1;

std::shared\_ptr<GraphicalElement> elem2;

int size;

};

int main() {

AllocConsole();

FILE\* dummy;

freopen\_s(&dummy, "CONOUT$", "w", stdout);

std::srand(std::time(0));

sf::RenderWindow window(sf::VideoMode(800, 800), "Daria");

std::vector<std::unique\_ptr<GraphicalElement>> elements;

elements.push\_back(std::make\_unique<CircularElement>(100, 100, 50));

int activeElement = 0;

std::cout << "Press K to print help" << std::endl;;

while (window.isOpen()) {

sf::Event event;

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed)

window.close();

if (event.type == sf::Event::KeyPressed) {

if (event.key.code == sf::Keyboard::K) {

std::cout << "K: Show help\nArrow keys: Move\nR: Change color\nZ: Increase size\nX: Decrease size\nNum1-Num9: Switch objects\nT: Create Circle\nD: Create Square\nL: Create Line\nQ: Combine objects\nW: Move object \nE: Delete Object" << std::endl;

}

else if (event.key.code == sf::Keyboard::Up) {

elements[activeElement]->move(0, -5);

}

else if (event.key.code == sf::Keyboard::Left) {

elements[activeElement]->move(-5, 0);

}

else if (event.key.code == sf::Keyboard::Down) {

elements[activeElement]->move(0, 5);

}

else if (event.key.code == sf::Keyboard::Right) {

elements[activeElement]->move(5, 0);

}

else if (event.key.code == sf::Keyboard::R) {

sf::Color currentColor = elements[activeElement]->getColor();

sf::Color newColor;

if (currentColor == sf::Color::Red) {

newColor = sf::Color::White;

}

else if (currentColor == sf::Color::White) {

newColor = sf::Color::Blue;

}

else if (currentColor == sf::Color::Blue) {

newColor = sf::Color::Green;

}

else if (currentColor == sf::Color::Green) {

newColor = sf::Color::Red;

}

else {

newColor = sf::Color::White;

}

elements[activeElement]->setColor(newColor);

}

else if (event.key.code == sf::Keyboard::Z) {

elements[activeElement]->setSize(elements[activeElement]->getSize() + 1);

}

else if (event.key.code == sf::Keyboard::X) {

int newSize = std::max(1, elements[activeElement]->getSize() - 1);

elements[activeElement]->setSize(newSize);

}

else if (event.key.code == sf::Keyboard::T) {

elements.push\_back(std::make\_unique<CircularElement>(100, 100, 50));

activeElement = elements.size() - 1;

}

else if (event.key.code == sf::Keyboard::L) {

elements.push\_back(std::make\_unique<LineElement>(100, 100,100, 10));

activeElement = elements.size() - 1;

}

else if (event.key.code == sf::Keyboard::D) {

elements.push\_back(std::make\_unique<DiamondElement>(200, 200, 50));

activeElement = elements.size() - 1;

}

}

else if (event.key.code >= sf::Keyboard::Num1 && event.key.code <= sf::Keyboard::Num9) {

int index = event.key.code - sf::Keyboard::Num1;

if (index < elements.size()) {

activeElement = index;

}

}

else if (event.key.code == sf::Keyboard::Q) {

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

std::unique\_ptr<GraphicalElement> combinedElement = std::make\_unique<CombinedElement>(std::move(elements[elements.size() - 2]), std::move(elements[elements.size() - 1]));

elements.erase(elements.end() - 2, elements.end());

elements.push\_back(std::move(combinedElement));

activeElement = elements.size() - 1;

}

else {

std::cout << "Not enough elements to combine.\n";

}

}

else if (event.key.code == sf::Keyboard::E) {

if (!elements.empty()) {

elements.erase(elements.begin() + activeElement);

if (activeElement >= elements.size()) {

activeElement = elements.size() - 1;

}

}

}

else if (event.key.code == sf::Keyboard::W) {

for (int i = 0; i < 4; ++i) {

int moveAmount = 70;

switch (i) {

case 0: // Move to the right

elements[activeElement]->move(moveAmount, 0);

break;

case 1: // Move down

elements[activeElement]->move(0, moveAmount);

break;

case 2: // Move to the left

elements[activeElement]->move(-moveAmount, 0);

break;

case 3: // Move up

elements[activeElement]->move(0, -moveAmount);

break;

}

sf::sleep(sf::milliseconds(200));

window.clear();

for (auto& element : elements) {

element->draw(window);

}

window.display();

}

}

}

window.clear();

for (auto& element : elements) {

element->draw(window);

}

window.display();

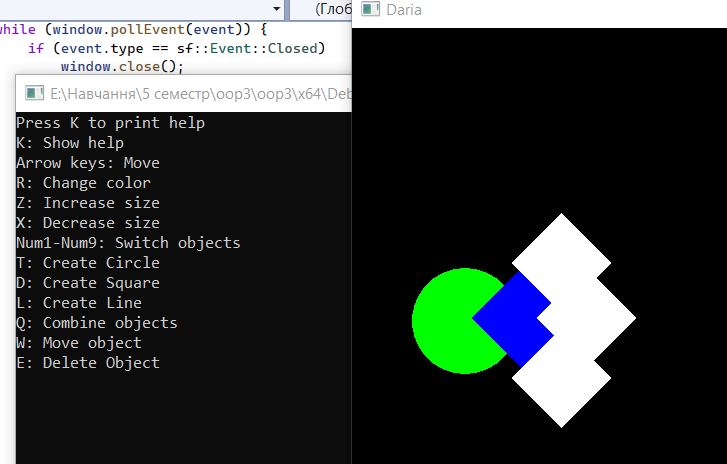
}

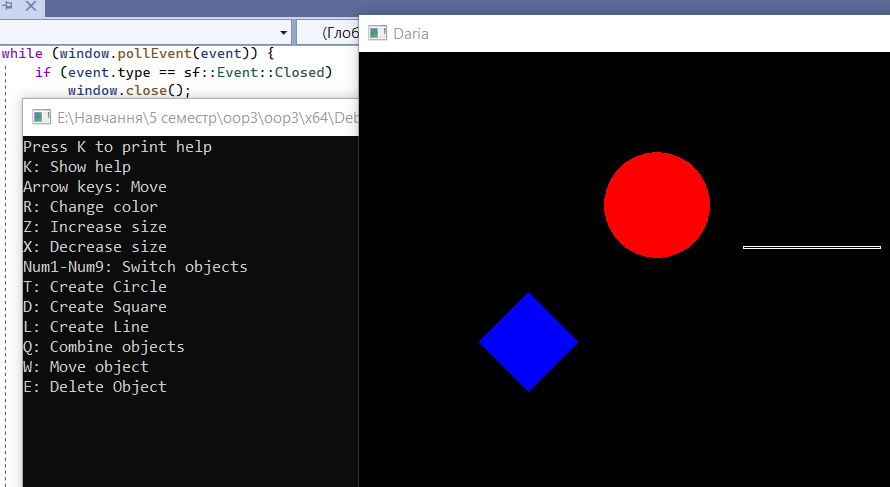
}

return 0;

}

1. **Опис тестових прикладів та скріншоти їх реалізації:**

****

****

1. **Аналіз результатів:**

Запускаючи програму, користувач отримає повідомлення «Press K to print help»

При натисканні літери К, виведеться підказочка з можливими діями:

K: Show help

Arrow keys: Move

R: Change color

Z: Increase size

X: Decrease size

Num1-Num9: Switch objects

T: Create Circle

D: Create Square

L: Create Line

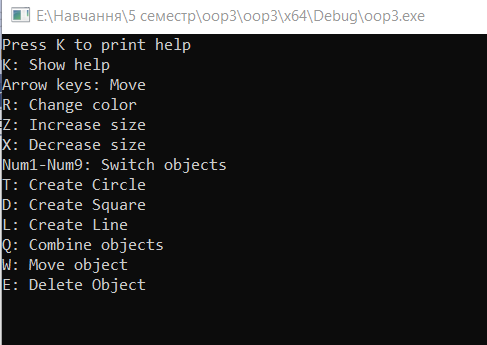
Q: Combine objects

W: Move object

E: Delete Object

Not enough elements to combine.

По натисканню клавіші К - виводиться меню:



По натисканню клавіші Up/ Down - фігура рухається вгору/вниз.

По натисканню клавіші Right/Left - фігура рухається вправо/вліво.

По натисканню клавіші R - бере поточний колір і змінює його на новий колір.

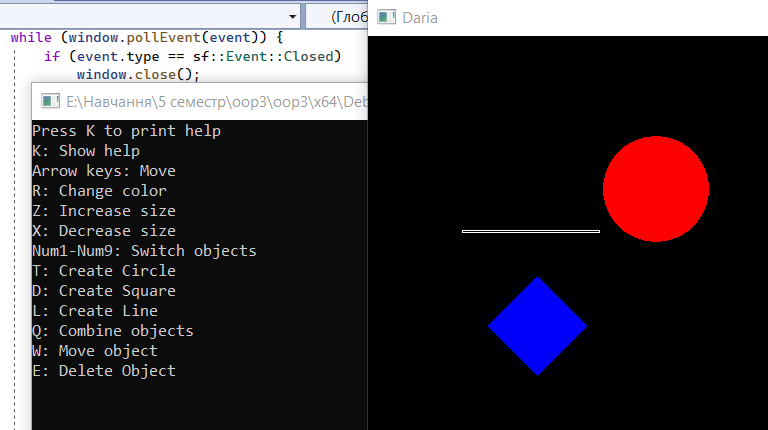
По натисканню клавіші Z - воно збільшує розміри. Тобто бере старий розмір, і додає +1.

А по натисканню клавіші Х - зменшує розміри на -1.

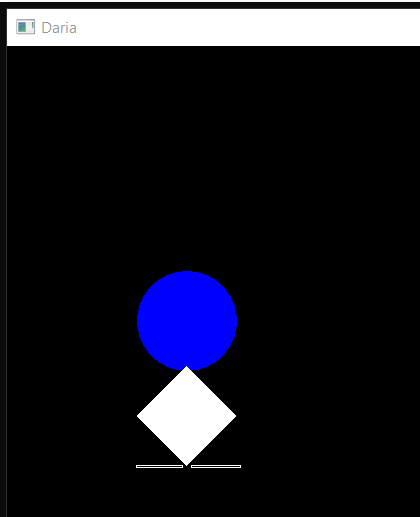
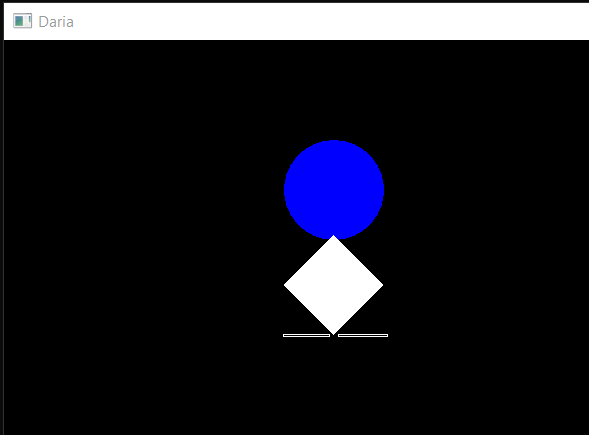
По натисканню клавіші Т - створює новий елемент на заданих координатах в коді(круг, лінію або ромб).

По натисканню клавіші L - створюється лінія.

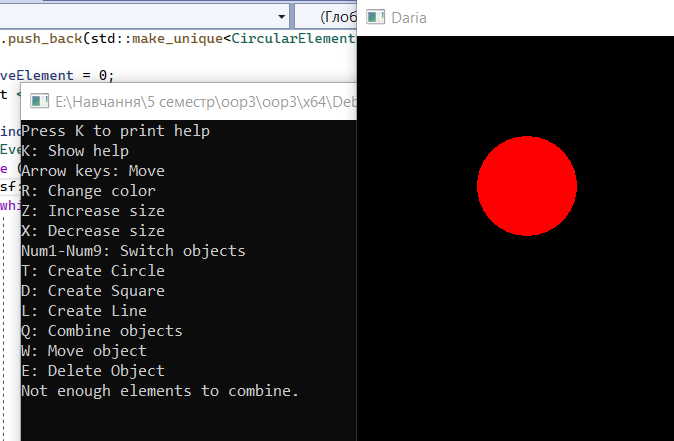
По натисканню клавіші D - створюється ромбик.



Якщо ми натиснемо Q, то якщо у нас елементів більше ніж 2, то тоді, відбуваються всі ці дії вже обьєднаних фігур:



Якщо елементів менше ніж 2, то виводиться відповідне повідомлення:



По натисканню клавіші Е - видаляється елемент.

По натисканню клавіші W - змінюється положення фігури на 70 пікселей по руху квадрату.

1. **Висновки:**

Під час виконання цієї лабораторної роботи №3 «Поліморфізм» я:

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

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

Програма підтримує такі загальні елементи поведінки графічних об’єктів:

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

1. Активізація/візуалізація графічного об’єкту за вибором користувача.
2. Виконання операцій над поточним (активним) об’єктом:
   1. переміщення без сліду;
   2. збереження і відновлення стану об’єкту;
   3. зміна кольору;
   4. зміна розміру;
   5. зміна стану видимий/невидимий.
3. Агрегація, тобто утворення нових об’єктів з вже створених. При агрегації об’єктів забезпечила наступні можливості:
   1. агрегація агрегатів;
   2. можливість виконувати над агрегатами ті ж дії, що і над «звичайними» об’єктами.

**Індивідуальні варіанти завдань**

11. 01001

1. Ім’я конфігураційного файлу:

1. задане у командному рядку при запуску програми.

2.Рух об’єкта в автоматичному режимі:

1. По запам’ятованій траєкторії.

3.Зміна кольору об’єкта:

1. По команді. (літера R)

4.Деформація об’єкта:

1. По команді.

5.Збирання/запам’ятовування агрегованого образу:

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