**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 7**

Тема: Проектирование структуры классов

Студент: Айрапетова Евгения Ашотовна

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Спроектировать простейший «графический» векторный редактор.

Требования к функционалу редактора:

* создание нового документа
* импорт документа из файла
* экспорт документа в файл
* создание графического примитива (согласно варианту задания)
* удаление графического примитива
* отображение документа на экране (печать перечня графических объектов и их характеристик в std::cout)
* реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

Требования к реализации:

* Создание графических примитивов необходимо вынести в отдельный класс – Factory.
* Сделать упор на использовании полиморфизма при работе с фигурами;
* Взаимодействие с пользователем (ввод команд) реализовать в функции main;

Вариант 34: квадрат, прямоугольник, трапеция

1. Описание программы

Для решения данной задачи был реализован виртуальный класс Figure и классы-наследники Square (для квадрата), Rectangle (для прямоугольника) и Trapezoid (для трапеции). Фигуры создаются в специальном классе Factory, а указатели на созданные фигуры хранятся в векторе. Для реализации undo (отмены последнего действия) каждое состояние вектора кладется в стек в структуре Originator. То есть для того, чтобы выполнить undo, нужно просто взять последнее состояние из стека.

Для работы с документами реализован класс Document, который позволяет сохранять данные в файл и загружать их из файла.

Также реализован интерфейс для работы с пользователем, в котором есть классификация ошибок (каждый раз, когда команда была выполнена без ошибки, выводится ОК). Для пользователя также выводится список команд, которые могут быть использованы.

1. Набор тестов

Тест №1 (три корректные фигуры – два квадрата и прямоугольник):

3   
0 0 0 0 2 2 2 2 0  
0 0 0 0 3 3 3 3 0  
1 0 0 0 3 4 3 4 0

Тест №2 (две корректные фигуры – трапеция и прямоугольник):

2  
2 0 0 3 0 2 1 1 1  
1 0 0 0 5 3 5 3 0

1. Результаты выполнения тестов

Тест №1:

No:1 - It's square  
Figure coordinates:  
A: (0,0)  
B: (0,2)  
C: (2,2)  
D: (2,0)  
  
No:2 - It's square  
Figure coordinates:  
A: (0,0)  
B: (0,3)  
C: (3,3)  
D: (3,0)  
  
No:3 - It's rectangle  
Figure coordinates:  
A:(0,0)  
B:(0,3)  
C:(4,3)  
D:(4,0)

Тест №2:

No:1 - It's trapezoid  
Figure coordinates:  
A:(0,0)  
B:(3,0)  
C:(2,1)  
D:(1,1)  
  
No:2 - It's rectangle  
Figure coordinates:  
A:(0,0)  
B:(0,5)  
C:(3,5)  
D:(3,0)

1. Листинг программы

// Айрапетова Евгения М8О-206Б-19

// Вариант 34: квадрат, прямоугольник, трапеция

#include <iostream>

#include <fstream>

#include <memory>

#include <stack>

#include <vector>

#include <string>

using namespace std;

using vertex\_t = pair<double, double>;

istream& operator>> (istream& input, vertex\_t& v) {

input >> v.first >> v.second;

return input;

}

ostream& operator<< (ostream& output, const vertex\_t v) {

output << "(" << v.first << "," << v.second << ")" << '\n';

return output;

}

double lenght(vertex\_t a, vertex\_t b) {

return sqrt((b.first - a.first) \* (b.first - a.first) + ((b.second - a.second) \* (b.second - a.second)));

}

int CorrectInput(vertex\_t& a, vertex\_t& b, vertex\_t& c, vertex\_t& d, int fig) {

if (((abs(a.first - c.first) < numeric\_limits<double>::epsilon()) &&

(abs(a.second - c.second) < numeric\_limits<double>::epsilon())) ||

((abs(b.first - d.first) < numeric\_limits<double>::epsilon()) &&

(abs(b.second - d.second) < numeric\_limits<double>::epsilon()))) {

return 1;

}

double len\_ab = lenght(a, b);

double len\_bc = lenght(b, c);

double len\_cd = lenght(c, d);

double len\_da = lenght(d, a);

double len\_ac = lenght(a, c);

double len\_bd = lenght(b, d);

if (fig == 1) {

if (abs(len\_cd - len\_ab) < numeric\_limits<double>::epsilon() &&

abs(len\_cd - len\_bc) < numeric\_limits<double>::epsilon() &&

abs(len\_cd - len\_da) < numeric\_limits<double>::epsilon() &&

abs(len\_bd - len\_ac) < numeric\_limits<double>::epsilon()) {

return 0;

}

return 1;

}

//проверка коректности прямоугольника: равенсво противоположных сторон и диагоналей

if (fig == 2) {

if (abs(len\_ab - len\_cd) < numeric\_limits<double>::epsilon() && abs(len\_bc - len\_da) < numeric\_limits<double>::epsilon() &&

abs(len\_ac - len\_bd) < numeric\_limits<double>::epsilon()) {

return 0;

}

return 1;

}

//проверка коректности равнобедренной трапеции: равенсво боковых строн и диагоналей

if (fig == 3) {

if (abs(len\_ab - len\_cd) < numeric\_limits<double>::epsilon() &&

abs(len\_ac - len\_bd) < numeric\_limits<double>::epsilon()) {

return 0;

}

return 1;

}

return 0;

}

class Figure {

public:

virtual void Set(istream& is) = 0;

virtual ostream& Print(ostream& output) = 0;

virtual ~Figure() {};

virtual void Save(ofstream& os) const = 0;

virtual void Load(ifstream& is) = 0;

virtual size\_t GetId() = 0;

};

enum figure\_t {

square = 0,

rectangle = 1,

trapezoid = 2

};

class Square : public Figure {

public:

Square() {};

Square(vertex\_t A, vertex\_t B, vertex\_t C, vertex\_t D) {

a = A; b = B; c = C; d = D;

};

void Set(istream& is) override {

is >> a >> b >> c >> d;

if (CorrectInput(a, b, c, d, 1) == 1) {

throw logic\_error("It`s not a square");

}

id = 0;

}

size\_t GetId() override {

return id;

}

ostream& Print(ostream& output) {

//output << "It's square\n";

output << "A: " << a << '\n' << "B: " << b << '\n' << "C: " << c << '\n' << "D: " << d << '\n';

return output;

}

void Save(ofstream& output) const {

output << id << " ";

output << a.first << ' ' << a.second << ' ' << b.first << ' ' << b.second << ' '

<< c.first << ' ' << c.second << ' ' << d.first << ' ' << d.second << '\n';

}

void Load(ifstream& input) override {

input >> a.first >> a.second >> b.first >> b.second >>

c.first >> c.second >> d.first >> d.second;

id = 0;

}

private:

size\_t id;

vertex\_t a;

vertex\_t b;

vertex\_t c;

vertex\_t d;

};

class Rectangle : public Figure {

public:

Rectangle() {};

Rectangle(vertex\_t A, vertex\_t B, vertex\_t C, vertex\_t D) {

a = A; b = B; c = C; d = D;

};

void Set(istream& is) override {

is >> a >> b >> c >> d;

if (CorrectInput(a, b, c, d, 2) == 1) {

throw logic\_error("It`s not a square");

}

id = 1;

}

size\_t GetId() {

return id;

}

ostream& Print(ostream& output) {

output << "A:" << a << '\n' << "B:" << b << '\n' << "C:" << c << '\n' << "D:" << d << '\n';

return output;

}

void Save(ofstream& output) const {

output << id << " ";

output << a.first << ' ' << a.second << ' ' << b.first << ' ' << b.second << ' '

<< c.first << ' ' << c.second << ' ' << d.first << ' ' << d.second << '\n';

}

void Load(ifstream& input) override {

input >> a.first >> a.second >> b.first >> b.second >>

c.first >> c.second >> d.first >> d.second;

id = 1;

}

private:

size\_t id;

vertex\_t a;

vertex\_t b;

vertex\_t c;

vertex\_t d;

};

class Trapezoid : public Figure {

public:

Trapezoid() {};

Trapezoid(vertex\_t A, vertex\_t B, vertex\_t C, vertex\_t D) {

a = A; b = B; c = C; d = D;

};

void Set(istream& is) override {

is >> a >> b >> c >> d;

if (CorrectInput(a, b, c, d, 3) == 1) {

throw logic\_error("It`s not a square");

}

id = 2;

}

size\_t GetId() {

return id;

}

ostream& Print(ostream& output) {

//output << "It's trapezoid\n";

output << "A:" << a << '\n' << "B:" << b << '\n' << "C:" << c << '\n' << "D:" << d << '\n';

return output;

}

void Save(ofstream& output) const {

output << id << " ";

output << a.first << ' ' << a.second << ' ' << b.first << ' ' << b.second << ' '

<< c.first << ' ' << c.second << ' ' << d.first << ' ' << d.second << '\n';

}

void Load(ifstream& input) override {

input >> a.first >> a.second >> b.first >> b.second >>

c.first >> c.second >> d.first >> d.second;

id = 2;

}

private:

size\_t id;

vertex\_t a;

vertex\_t b;

vertex\_t c;

vertex\_t d;

};

struct Factory { // создание графических примитивов

static shared\_ptr<Figure> create(figure\_t t) {

switch (t) {

case figure\_t::square:

return make\_shared<Square>();

case figure\_t::rectangle:

return make\_shared<Rectangle>();

case figure\_t::trapezoid:

return make\_shared<Trapezoid>();

}

}

};

struct Memento {

vector<shared\_ptr<Figure>> state;

vector<size\_t> id;

Memento() {};

Memento(vector<shared\_ptr<Figure>> other) {

state = other;

};

};

struct Originator { // структура, хранящая стек состояний

stack<Memento> mementos;

void createMemento(vector<shared\_ptr<Figure>> state) {

mementos.emplace(state);

}

vector<shared\_ptr<Figure>> restore() {

if (!mementos.empty()) {

vector<shared\_ptr<Figure>> result = move(mementos.top().state);

mementos.pop();

return result;

}

else throw logic\_error("undo stack empty");

}

};

class Document {

public:

void DSave(vector<shared\_ptr<Figure>>& f, string& file) {

ofstream os(file);

if (!os) {

cerr << "INCORRECT PATH\n";

return;

}

os << f.size() << ' ';

for (int j = 0; j < f.size(); ++j) {

f[j]->Save(os);

}

}

void DLoad(vector<shared\_ptr<Figure>>& figures, string& file) {

ifstream is(file);

int i;

shared\_ptr<Figure> fig = nullptr;

if (!is) {

cerr << "INCORRECT PATH\n";

return;

}

int s;

is >> s;

for (int j = 0; j < s; ++j) {

is >> i;

fig = Factory::create((figure\_t)i);

fig->Load(is);

figures.push\_back(move(fig));

}

}

};

void Usage() {

cout << "Enter choice:\n"

"a - add\n"

"r - remove\n"

"u - undo\n"

"s - save in file\n"

"l - load out file\n"

"p - print all figures\n"

"c - print usage\n\n"

"e - exit\n";

cout << "Types of figures:\n"

"0 - square\n"

"1 - rectangle\n"

"2 - trapezoid\n";

}

int main() {

Originator origin;

char choice = ' ';

int index;

int i;

int idx;

string path = "";

Document doc;

vector<shared\_ptr<Figure>> figures;

shared\_ptr<Figure> fig = nullptr;

Usage();

while (choice != 'e') {

cin >> choice;

switch (choice) {

case 'a':

cin >> i;

fig = Factory::create((figure\_t)i);

try {

fig->Set(cin);

}

catch (logic\_error) { cerr << "ERROR INPUT\n"; break; }

origin.createMemento(figures);

figures.push\_back(move(fig));

cout << "OK\n";

break;

case 'r':

cout << "Enter index: ";

cin >> index;

if (index >= 0 && index < figures.size()) {

origin.createMemento(figures);

figures.erase(figures.begin() + index);

cout << "OK\n";

break;

}

else { cerr << "BORDER OVERLAY\n"; break; }

case 'u':

figures = origin.restore();

cout << "OK\n";

break;

case 's':

cout << "Enter path: ";

cin >> path;

doc.DSave(figures, path);

cout << "Ok\n";

break;

case 'l':

cout << "Enter path: ";

cin >> path;

figures.clear();

doc.DLoad(figures, path);

origin.createMemento(figures);

cout << "OK\n";

break;

case 'p':

idx = 0;

for (int j = 0; j < figures.size(); ++j) {

cout << "No:" << ++idx << " - ";

if (figures[j]->GetId() == 0) {

cout << "It's square\n"; }

if (figures[j]->GetId() == 1) {

cout << "It's rectangle\n"; }

if (figures[j]->GetId() == 2) {

cout << "It's trapezoid\n"; }

cout << "Figure coordinates:\n";

figures[j]->Print(cout);

}

break;

case 'c':

Usage();

break;

case 'e':

break;

default:

cerr << "ERROR COMMAND\n";

break;

}

}

return 0;

}

1. Вывод

Данная лабораторная работа научила меня работать с графическими примитивами, а также я попрактиковалась в работе с документами.

Реализовав откат к предыдущим действиям, я усовершенствовала свои навыки работы со стеком. Однако, также можно сделать вывод, что такой способ довольно затратен по памяти.

Список литературы

1. Справочник по языку C++ [Электронный ресурс]. URL: <https://en.cppreference.com/w/> (дата обращения: 10.01.2021).