**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: Почечура Артемий Андреевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

**1.** **Постановка задачи**

**Цель:**

* Получение практических навыков в хороших практиках проектирования структуры классов приложения;

**Порядок выполнения работы**

* 1. Ознакомиться с теоретическим материалом.
  2. Получить у преподавателя вариант задания.
  3. Реализовать задание своего варианта в соответствии с поставленными требованиями.
  4. Подготовить тестовые наборы данных.
  5. Создать репозиторий на GitHub.
  6. Отправить файлы лабораторной работы в репозиторий.
  7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

Вариант: 7;

Фигуры: шестиугольник, восьмиугольник, треугольник.

**2.** **Описание программы**

Программе на вход поступают данные о фигурах – длина её стороны (все фигуры правильные). Эти данные хранятся в форме списка. Из списка мы можем удалять элементы по индексу, распечатать все элементы и отменить последнее действие. Также эти данные можно синхронизировать с файлом, после чего в файле будет содержаться та же информация, что и буфере. Буфер можно синхронизировать с указанным файлом, после чего наш список будет равен списку, который лежит в файле. Поддерживается обработка следующих ошибок: неверная команда, неверные данные для стороны фигуры, неверное имя файла, неверный индекс.

3. **Набор тестов и результаты их исполнения**

Тест 1:

0 - exit programm

1 filename - inload file

2 filename - upload file

3 - input triangle

4 - input hexagon

5 - input pentagon

6 ind - remove element

7 - print all elments

8 - undo

Enter command: 3

Enter side: 2

Enter command: 4

Enter side: 3

Enter command: 5

Enter side: 4

Enter command: 7

Triangle: ((-1; -0.57735), (0; 1.1547), (1; -0.57735))

Hexagon: ((-3; 0), (-1.5; 2.59808), (1.5; 2.59808), (3; 0), (-1.5; -2.59808), (1.5; -2.59808))

Pentagon: ((-3.4026; 0), (0; 3.4026), (3.4026; 0), (-2; -2.75276), (2; -2.75276))

Enter command: 0

Тест 2:

0 - exit programm

1 filename - inload file

2 filename - upload file

3 - input triangle

4 - input hexagon

5 - input pentagon

6 ind - remove element

7 - print all elments

8 - undo

Enter command: 3

Enter side: -1

Incorrect values

Enter command: 4

Enter side: 0

Incorrect values

Enter command: 6

Enter index 1

Index out of range

Enter command: 0

Тест 3:

0 - exit programm

1 filename - inload file

2 filename - upload file

3 - input triangle

4 - input hexagon

5 - input pentagon

6 ind - remove element

7 - print all elments

8 - undo

Enter command: 7

Enter command: 1

Enter filename: qwe

Enter command: 7

Triangle: ((-0.5; -0.288675), (0; 0.57735), (0.5; -0.288675))

Hexagon: ((-2; 0), (-1; 1.73205), (1; 1.73205), (2; 0), (-1; -1.73205), (1; -1.73205))

Pentagon: ((-2.55195; 0), (0; 2.55195), (2.55195; 0), (-1; -2.06457), (1; -2.06457))

Enter command: 0

**4.** **Листинг программы**

Файл main.cpp

// Почечура Артемий, гр. М8О-206Б-20

// ЛР №7, вариант 7

/\* Спроектировать простейший «графический» векторный редактор.

Требование 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;\*/

#include <iostream>

#include <fstream>

#include <exception>

#include <list>

#include <vector>

#include <cmath>

#include <string>

#include <utility>

#include <memory>

#include <stack>

double PI = 3.14159265;

using namespace std;

struct Vertex {

double x = 0;

double y = 0;

void print() {

cout << "(" << x << "; " << y << ")";

}

};

istream& operator>>(istream& in, Vertex& vertex) {

in >> vertex.x >> vertex.y;

return in;

}

ostream& operator<<(ostream& out, const Vertex& vertex) {

out << vertex.x << " " << vertex.y;

return out;

}

enum class type\_figure {

unknown = -1,

Triangle = 0,

Hexagon = 1,

Pentagon = 2,

};

string type\_to\_string(type\_figure type) {

if (type == type\_figure::Triangle) {

return "Triangle";

}

else if (type == type\_figure::Hexagon) {

return "Hexagon";

}

else if (type == type\_figure::Pentagon) {

return "Pentagon";

}

else {

throw runtime\_error("Undefined figure type");

}

}

class Figure {

public:

vector<Vertex> data;

Figure() = default;

Figure(vector<Vertex> vec) : data(vec) {}

void print() {

cout << type\_to\_string(get\_type\_figure()) << ": (";

bool is\_first = true;

for (Vertex& v : data) {

if (!is\_first) {

cout << ", ";

}

is\_first = false;

v.print();

}

cout << ")\n";

}

virtual type\_figure get\_type\_figure() {

return type\_figure::unknown;

}

};

class Triangle : public Figure {

public:

Triangle() : Figure(vector<Vertex>(3)) {}

Triangle(Vertex a, Vertex b, Vertex c) : Figure({ a, b, c}) {}

Triangle(vector<Vertex> v) {

if (v.size() != 3) {

throw invalid\_argument("Incorrect number of vertex of square");

}

data = v;

}

virtual type\_figure get\_type\_figure() { return type\_figure::Triangle; }

};

class Pentagon : public Figure {

public:

Pentagon() : Figure(vector<Vertex>(5)) {}

Pentagon(Vertex a, Vertex b, Vertex c, Vertex d, Vertex e) : Figure({ a, b, c, d, e }) {}

Pentagon(vector<Vertex> v) {

if (v.size() != 5) {

throw invalid\_argument("Incorrect number of vertex of pentagon");

}

data = v;

}

virtual type\_figure get\_type\_figure() { return type\_figure::Pentagon; }

};

class Hexagon : public Figure {

public:

Hexagon() : Figure(vector<Vertex>(6)) {}

Hexagon(Vertex a, Vertex b, Vertex c, Vertex d, Vertex e, Vertex f) : Figure({ a, b, c, d, e, f }) {}

Hexagon(vector<Vertex> v) {

if (v.size() != 6) {

throw invalid\_argument("Incorrect number of vertex of hexagon");

}

data = v;

}

virtual type\_figure get\_type\_figure() { return type\_figure::Hexagon; }

};

istream& operator>>(istream& input, Figure& figure) {

for (Vertex& v : figure.data) {

input >> v;

}

return input;

}

ostream& operator<<(ostream& output, Figure& figure) {

output << static\_cast<int>(figure.get\_type\_figure()) << " ";

for (const Vertex& v : figure.data) {

output << v << " ";

}

return output;

}

enum class type\_of\_action {

unknown,

add,

remove,

};

struct Action {

type\_of\_action type = type\_of\_action::unknown;

shared\_ptr<Figure> figure\_ptr;

};

class Docs {

public:

Docs() = default;

list<shared\_ptr<Figure>> data;

stack<Action> act;

void upload(string& filename) {

ofstream out(filename);

if (!out) {

throw invalid\_argument("Incorrect filename");

}

for (const auto& ptr : data) {

out << \*ptr << "\n";

}

}

void inload(const string& filename) {

data.clear();

while (act.size() != 0) {

act.pop();

}

ifstream in(filename);

if (!in) {

throw invalid\_argument("Incorrect filename");

}

int type\_int;

while (in >> type\_int) {

type\_figure type = (type\_figure)type\_int;

shared\_ptr<Figure> figure\_ptr;

if (type == type\_figure::Triangle) {

figure\_ptr = make\_shared<Triangle>();

}

else if (type == type\_figure::Hexagon) {

figure\_ptr = make\_shared<Hexagon>();

}

else if (type == type\_figure::Pentagon) {

figure\_ptr = make\_shared<Pentagon>();

}

else {

throw runtime\_error("Undefined figure type");

}

in >> \*figure\_ptr;

data.push\_back(move(figure\_ptr));

}

}

void add(shared\_ptr<Figure> figure\_ptr) {

Action tmp;

tmp.type = type\_of\_action::add;

tmp.figure\_ptr = figure\_ptr;

data.push\_back(move(figure\_ptr));

act.push(tmp);

}

void remove(size\_t idx) {

if (idx >= data.size() || idx < 0) {

throw out\_of\_range("Index out of range");

}

Action tmp;

tmp.type = type\_of\_action::remove;

auto it = next(data.begin(), idx);

tmp.figure\_ptr = \*it;

data.erase(it);

act.push(tmp);

}

void print() {

for (auto& ptr : data) {

ptr->print();

}

}

void undo() {

if (act.size() == 0) {

throw out\_of\_range("Nothing undo");

}

Action tmp = act.top();

act.pop();

if (tmp.type == type\_of\_action::unknown) {

return;

}

else if (tmp.type == type\_of\_action::add) {

data.pop\_back();

}

else if (tmp.type == type\_of\_action::remove) {

data.push\_back(tmp.figure\_ptr);

}

else {

throw runtime\_error("Undefined action type");

}

};

};

void help() {

cout << "0 - exit programm\n";

cout << "1 filename - inload file\n";

cout << "2 filename - upload file\n";

cout << "3 - input triangle\n";

cout << "4 - input hexagon\n";

cout << "5 - input pentagon\n";

cout << "6 ind - remove element\n";

cout << "7 - print all elments\n";

cout << "8 - undo\n";

}

double side\_2pow(Vertex a, Vertex b) {

return (a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y);

}

bool check\_vertex(const vector<Vertex>& v) {

for (int i = 0; i < v.size(); i++) {

if (side\_2pow(v[i % v.size()], v[(i + 1) % v.size()]) != side\_2pow(v[(i + 1) % v.size()], v[(i + 2) % v.size()])) {

return false;

}

}

return true;

}

int main() {

Docs doc;

int cmd = -1;

help();

while (cmd != 0) {

try {

cout << "Enter command: ";

cin >> cmd;

if (cmd == 1) { // inload file

cout << "Enter filename: ";

string filename;

cin >> filename;

doc.inload(filename);

}

else if (cmd == 2) { // upload file

cout << "Enter filename: ";

string filename;

cin >> filename;

doc.upload(filename);

}

else if (cmd == 3) { // Triangle

cout << "Enter side: ";

double side;

if(!(cin >> side) || side<=0){

cin.clear();

cin.ignore(1e9, '\n');

throw runtime\_error("Incorrect values");

}

vector<Vertex> v(3);

double R = side / (2 \* sin(PI / 3.0));

double r = side / (2 \* tan(PI / 3.0));

v[0].x = -side / 2; v[0].y = -r;

v[1].x = 0; v[1].y = R;

v[2].x = side / 2; v[2].y = -r;

doc.add(make\_shared<Triangle>(v));

}

else if (cmd == 4) { // Hexagon

cout << "Enter side: ";

double side;

if(!(cin >> side) || side<=0){

cin.clear();

cin.ignore(1e9, '\n');

throw runtime\_error("Incorrect values");

}

vector<Vertex> v(6);

double R = side / (2 \* sin(PI / 6.0));

double r = side / (2 \* tan(PI / 6.0));

v[0].x = -R; v[0].y = 0;

v[1].x = -1 \* side / 2; v[1].y = r;

v[2].x = side / 2; v[2].y = r;

v[3].x = R; v[3].y = 0;

v[4].x = -1 \* side / 2; v[4].y = -1 \* r;

v[5].x = side / 2; v[5].y = -1 \* r;

doc.add(make\_shared<Hexagon>(v));

}

else if (cmd == 5) { // Pentagon

cout << "Enter side: ";

int side;

if(!(cin >> side) || side<=0){

cin.clear();

cin.ignore(1e9, '\n');

throw runtime\_error("Incorrect values");

}

double R = side / (2 \* sin(PI / 5.0));

double r = side / (2 \* tan(PI / 5.0));

vector<Vertex> v(5);

v[0].x = -R; v[0].y = 0;

v[1].x = 0; v[1].y = R;

v[2].x = R; v[2].y = 0;

v[3].x = -1 \* side / 2; v[3].y = -1 \* r;

v[4].x = side / 2; v[4].y = -1 \* r;

doc.add(make\_shared<Pentagon>(v));

}

else if (cmd == 6) { // remove

cout << "Enter index ";

size\_t idx;

cin >> idx;

doc.remove(idx);

}

else if (cmd == 7) { // print

doc.print();

}

else if (cmd == 8) { // undo

doc.undo();

}

else if (cmd == 0) {

break;

}

else {

throw runtime\_error("Wrong command");

}

}

catch (runtime\_error& ex) {

cout << ex.what() << "\n";

}

catch (invalid\_argument& ex) {

cout << ex.what() << "\n";

}

catch (out\_of\_range& ex) {

cout << ex.what() << "\n";

}

}

return 0;

}

**5.** **Выводы**

В ходе лабораторной работы были реализованы классы Triangle, Hexagon, Pentagon для работы с фигурами, класс Docs для работы с файлами. По итогам выполнения работы я обрёл навыки работы с файлами и попрактиковался в работе с геометрическими фигурами. Взаимодействие с пользователем происходит посредством интерактивного меню, в котором пользователь выбирает команды и вводит данные, если это нужно.

ЛИТЕРАТУРА

1. Уроки программирования на языке С++ [Электронный ресурс]. URL:<https://ravesli.com/uroki-cpp> (дата обращения 29.11.2021).

2. The Compiler Generator Coco/R for C++. Documentation [Электронный ресурс]. URL:<https://ssw.jku.at/Research/Projects/Coco/#Docu> (дата обращения 29.11.2021).

3. cpp.reference, справочник по спецификации языка C++ [Электронный ресурс]. URL:<https://en.cppreference.com/w/> (дата обращения 02.12.2021).