**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: Терво Александр Александрович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Спроектировать простейший «графический» векторный редактор.

Требование к функционалу редактора:

• создание нового документа;

• импорт документа из файла;

• экпорт документа в файл;

• создание графического примитива (согласно варианту задания);

• удаление графического примитива;•отображение документа на экране (печать перечня графических объектов и иххарактеристик в std::cout);

• реализовать операцию undo, отменяющую последнее сделанное действие.

Должно действовать для операций добавления/удаления фигур.

Требование 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 1. Квадрат, прямоугольник, треугольник.

1. Описание программы

Создадим абстрактный класс IFigure, чтобы наследовать от него классы других фигур и иметь возможность хранить разные фигуры вместе. Каждая фигура — шаблонный полиморфный класс, поддерживающий методы родителя. Шаблон фигуры — скалярный тип. Из входных данных трудно определить, какую фигуру вводит пользователь, поэтому каждая фигура будет иметь свой уникальный идентификатор. Класс TFactory читает аргументы фигур и создаёт их, возвращает указатели на эти фигуры. Шаблонный класс TDocument хранит список указателей на фигуры и стек с действиями. Действия — наследники абстракного класса IAction, поддерживающие общий метод PerformAction. Шаблон документа — тоже скалярный тип, то есть можно задать этот тип сразу для всех фигур документа. Для чтения из файла и записи в файл используются системные вызовы fread и fwrite.

Программа обрабатывает команды пользователя до окончания ввода и поддерживает следующие команды:

• «n» — создание нового документа;

• «o имя\_файла» — загрузить документ из файла;

• «s имя\_файла» — сохранить документ в файл;

• «+ позиция тип\_фигуры аргументы\_фигуры» — добавить фигуру с заданными аргументами на заданную позицию;

• «- позиция» — удалить фигуру на заданной позиции;

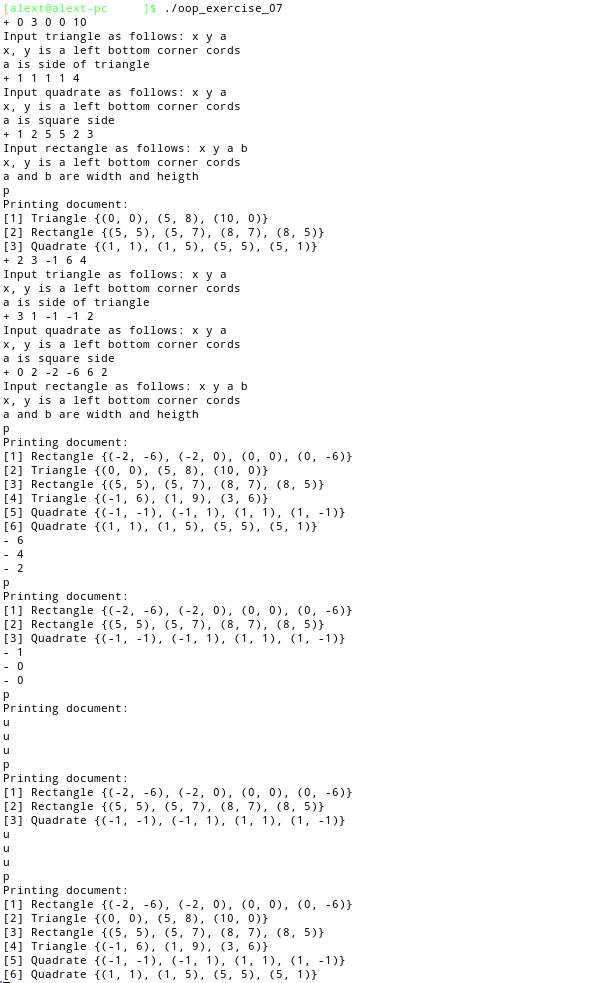
• «p» — вывести весь документ;

• «u» — отменить предыдущее изменение;

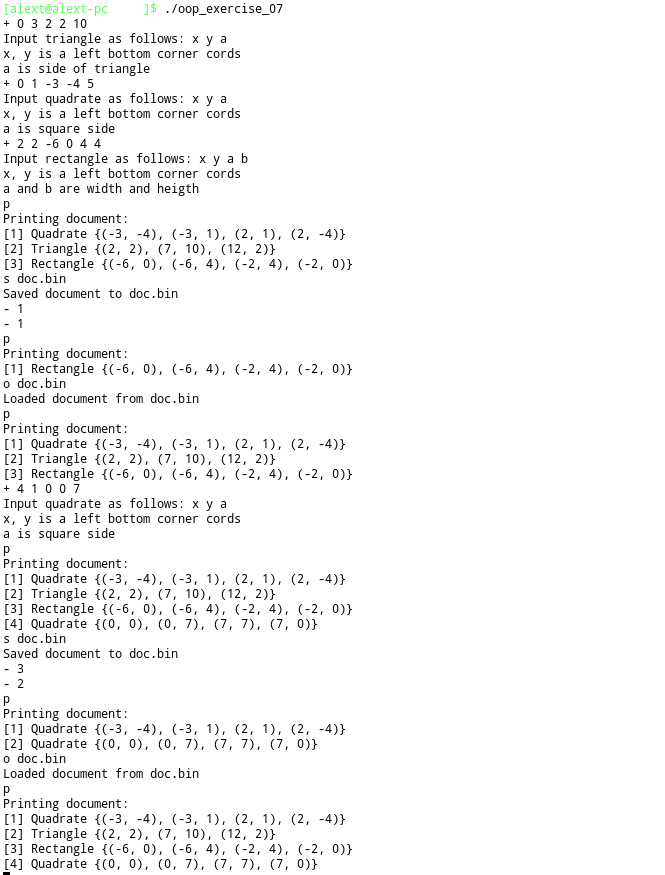
• «h» — вывести справочную информацию.

1. Набор тестов и результаты их выполнения

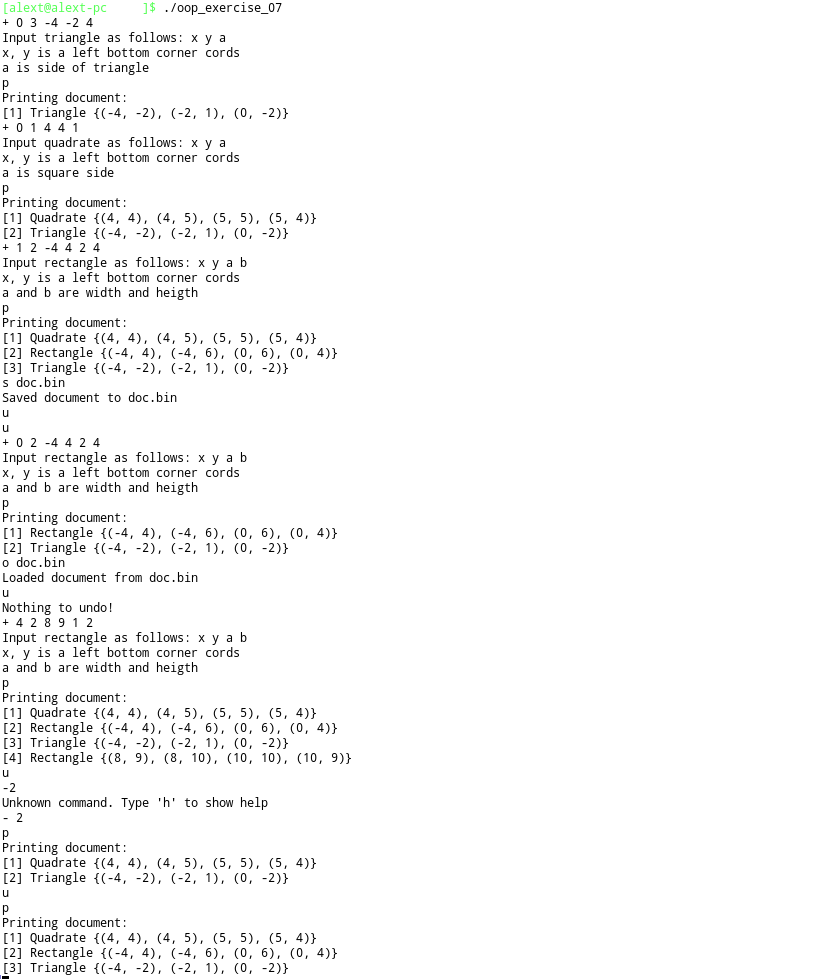
Тест 1. Проверка функций вставки, удаления и отмены предыдущего действия.



Тест 2. Проверка функций вставки, удаления, чтения из файла и записи в файл.



Тест 3. Комбинация тестов 1 и 2.



1. Листинг программы

**document.hpp:**

#ifndef DOCUMENT\_HPP

#define DOCUMENT\_HPP

#include <list>

#include <stack>

#include "factory.hpp"

template<class SCALAR\_TYPE>

class TDocument {

private:

struct IAction;

using figure\_pointer = std::shared\_ptr<IFigure>;

using action\_pointer = std::shared\_ptr<IAction>;

using const\_iterator = std::list< figure\_pointer >::const\_iterator;

std::list< figure\_pointer > FiguresList;

std::stack< action\_pointer > ActionStack;

struct IAction {

virtual void PerformAction(TDocument & fact) = 0;

virtual ~IAction() {}

};

class TDeleteAction : public IAction {

private:

size\_t DeletePos;

public:

TDeleteAction(const size\_t & pos) : DeletePos(pos) {}

void PerformAction(TDocument & fact) override {

fact.Delete(DeletePos);

}

};

class TAddAction : public IAction {

private:

size\_t AddPos;

figure\_pointer AddFigure;

public:

TAddAction(const size\_t & pos, const figure\_pointer & fig) : AddPos(pos), AddFigure(fig) {}

void PerformAction(TDocument & fact) override {

fact.AddFigure(AddPos, AddFigure);

}

};

public:

void CreateNew() {

while (!ActionStack.empty()) {

ActionStack.pop();

}

FiguresList.clear();

}

void LoadFromFile(FILE\* in) {

CreateNew();

size\_t n;

fread(&n, sizeof(size\_t), 1, in);

for (size\_t i = 0; i < n; ++i) {

unsigned int type;

fread(&type, sizeof(unsigned int), 1, in);

if (type == QUAD\_TYPE\_ID) {

FiguresList.push\_back(TFactory< SCALAR\_TYPE, TQuad<SCALAR\_TYPE> >::Read(in));

} else if (type == RECTANGLE\_TYPE\_ID) {

FiguresList.push\_back(TFactory< SCALAR\_TYPE, TRectangle<SCALAR\_TYPE> >::Read(in));

} else if (type == TRIANGLE\_TYPE\_ID) {

FiguresList.push\_back(TFactory< SCALAR\_TYPE, TTriangle<SCALAR\_TYPE> >::Read(in));

}

}

}

void SaveToFile(FILE\* out) {

size\_t n = FiguresList.size();

fwrite(&n, sizeof(size\_t), 1, out);

for (const\_iterator it = FiguresList.begin(); it != FiguresList.end(); ++it) {

(\*it)->Write(out);

}

}

void Add(const size\_t & pos, const unsigned int & figureID) {

if (figureID == QUAD\_TYPE\_ID) {

AddFigure(pos, TFactory<SCALAR\_TYPE, TQuad<SCALAR\_TYPE> >::CreateFigure());

} else if (figureID == RECTANGLE\_TYPE\_ID) {

AddFigure(pos, TFactory<SCALAR\_TYPE, TRectangle<SCALAR\_TYPE> >::CreateFigure());

} else if (figureID == TRIANGLE\_TYPE\_ID) {

AddFigure(pos, TFactory<SCALAR\_TYPE, TTriangle<SCALAR\_TYPE> >::CreateFigure());

}

}

void AddFigure(const size\_t & pos, const figure\_pointer & fig) {

if (pos > FiguresList.size()) {

FiguresList.push\_back(fig);

TDeleteAction\* delAct = new TDeleteAction(FiguresList.size());

ActionStack.push(action\_pointer(delAct));

} else {

size\_t cur = 0;

const\_iterator it = FiguresList.begin();

while (cur < pos) {

++cur;

++it;

}

FiguresList.insert(it, fig);

TDeleteAction\* delAct = new TDeleteAction(pos + 1);

ActionStack.push(action\_pointer(delAct));

}

}

void Delete(const size\_t & pos) {

if (FiguresList.empty()) {

std::cout << "Nothing to delete!" << std::endl;

return;

}

if (pos > FiguresList.size()) {

TAddAction\* addAct = new TAddAction(FiguresList.size() - 1, FiguresList.back());

ActionStack.push(action\_pointer(addAct));

FiguresList.pop\_back();

} else {

size\_t cur = 1;

const\_iterator it = FiguresList.begin();

while (cur < pos) {

++cur;

++it;

}

TAddAction\* addAct = new TAddAction(cur - 1, \*it);

ActionStack.push(action\_pointer(addAct));

FiguresList.erase(it);

}

}

void Undo() {

if (ActionStack.empty()) {

std::cout << "Nothing to undo!" << std::endl;

} else {

ActionStack.top()->PerformAction(\*this);

ActionStack.pop();

ActionStack.pop();

}

}

template<class U>

friend std::ostream & operator << (std::ostream & of, const TDocument<U> & fact) {

TDocument::const\_iterator it = fact.FiguresList.begin();

for (size\_t i = 0; i < fact.FiguresList.size(); ++i) {

of << "[" << i + 1 << "] ";

(\*it)->Print(of);

of << std::endl;

++it;

}

return of;

}

};

#endif /\* DOCUMENT\_HPP \*/

**factory.hpp:**

#ifndef FACTORY\_HPP

#define FACTORY\_HPP

#include <memory>

#include "rectangle.hpp"

#include "quad.hpp"

#include "triangle.hpp"

template <class T, class FIGURE>

class TFactory;

template <class T>

class TFactory< T, TQuad<T> > {

public:

static std::shared\_ptr<IFigure> CreateFigure() {

std::pair<T, T> curCords;

T curSide;

std::cout << "Input quadrate as follows: x y a" << std::endl;

std::cout << "x, y is a left bottom corner cords" << std::endl;

std::cout << "a is square side" << std::endl;

std::cin >> curCords.first >> curCords.second >> curSide;

TQuad<T>\* q = new TQuad<T>(curCords, curSide);

return std::shared\_ptr<IFigure>(q);

}

static std::shared\_ptr<IFigure> Read(FILE\* in) {

std::pair<T, T> curCords;

T curSide;

fread(&curCords.first, sizeof(T), 1, in);

fread(&curCords.second, sizeof(T), 1, in);

fread(&curSide, sizeof(T), 1, in);

TQuad<T>\* q = new TQuad<T>(curCords, curSide);

return std::shared\_ptr<IFigure>(q);

}

};

template <class T>

class TFactory< T, TRectangle<T> > {

public:

static std::shared\_ptr<IFigure> CreateFigure() {

std::pair<T, T> curCords;

T curHeight, curWidth;

std::cout << "Input rectangle as follows: x y a b" << std::endl;

std::cout << "x, y is a left bottom corner cords" << std::endl;

std::cout << "a and b are width and heigth" << std::endl;

std::cin >> curCords.first >> curCords.second >> curHeight >> curWidth;

TRectangle<T>\* rect = new TRectangle<T>(curCords, curHeight, curWidth);

return std::shared\_ptr<IFigure>(rect);

}

static std::shared\_ptr<IFigure> Read(FILE\* in) {

std::pair<T, T> curCords;

T curHeight, curWidth;

fread(&curCords.first, sizeof(T), 1, in);

fread(&curCords.second, sizeof(T), 1, in);

fread(&curHeight, sizeof(T), 1, in);

fread(&curWidth, sizeof(T), 1, in);

TRectangle<T>\* rect = new TRectangle<T>(curCords, curHeight, curWidth);

return std::shared\_ptr<IFigure>(rect);

}

};

template <class T>

class TFactory< T, TTriangle<T> > {

public:

static std::shared\_ptr<IFigure> CreateFigure() {

std::pair<T, T> curCords;

T curSide;

std::cout << "Input triangle as follows: x y a" << std::endl;

std::cout << "x, y is a left bottom corner cords" << std::endl;

std::cout << "a is side of triangle" << std::endl;

std::cin >> curCords.first >> curCords.second >> curSide;

TTriangle<T>\* tri = new TTriangle<T>(curCords, curSide);

return std::shared\_ptr<IFigure>(tri);

}

static std::shared\_ptr<IFigure> Read(FILE\* in) {

std::pair<T, T> curCords;

T curSide;

fread(&curCords.first, sizeof(T), 1, in);

fread(&curCords.second, sizeof(T), 1, in);

fread(&curSide, sizeof(T), 1, in);

TTriangle<T>\* tri = new TTriangle<T>(curCords, curSide);

return std::shared\_ptr<IFigure>(tri);

}

};

#endif

**figure.hpp:**

#ifndef FIGURE\_HPP

#define FIGURE\_HPP

#include <iostream>

#include <tuple>

class IFigure {

public:

virtual void Print(std::ostream & of) = 0;

virtual void Write(FILE\* out) = 0;

virtual ~IFigure() {}

};

template <class T1, class T2>

std::ostream & operator << (std::ostream & out, const std::pair<T1, T2> & p) {

out << '(' << p.first << ", " << p.second << ')';

return out;

}

#endif

**quad.hpp:**

#ifndef QUAD\_HPP

#define QUAD\_HPP

#include "figure.hpp"

const unsigned int QUAD\_TYPE\_ID = 1;

template <class T>

class TQuad : public IFigure {

private:

std::pair<T, T> Cords;

T Side;

public:

TQuad() : Cords(), Side() {}

TQuad(const std::pair<T, T> & xy, const T & l) : Cords(xy), Side(l) {}

void Write(FILE\* out) override {

fwrite(&QUAD\_TYPE\_ID, sizeof(unsigned int), 1, out);

fwrite(&Cords.first, sizeof(T), 1, out);

fwrite(&Cords.second, sizeof(T), 1, out);

fwrite(&Side, sizeof(T), 1, out);

}

void Print(std::ostream & of) override {

of << \*this;

}

template <class U>

friend std::ostream & operator << (std::ostream & of, const TQuad<U> & q) {

of << "Quadrate {";

of << std::pair<U, U>(q.Cords.first, q.Cords.second) << ", ";

of << std::pair<U, U>(q.Cords.first, q.Cords.second + q.Side) << ", ";

of << std::pair<U, U>(q.Cords.first + q.Side, q.Cords.second + q.Side) << ", ";

of << std::pair<U, U>(q.Cords.first + q.Side, q.Cords.second);

of << "}";

return of;

}

};

#endif

**rectangle.hpp:**

#ifndef RECTANGLE\_HPP

#define RECTANGLE\_HPP

#include "figure.hpp"

const unsigned int RECTANGLE\_TYPE\_ID = 2;

template <class T>

class TRectangle : public IFigure {

private:

/\* Cords of left bottom corner, height and width \*/

std::pair<T, T> Cords;

T Height, Width;

public:

TRectangle() : Cords(), Height(), Width() {}

TRectangle(const std::pair<T, T> & xy, const T & h, const T & w) : Cords(xy), Height(h), Width(w) {}

void Print(std::ostream & of) override {

of << \*this;

}

void Write(FILE\* out) override {

fwrite(&RECTANGLE\_TYPE\_ID, sizeof(unsigned int), 1, out);

fwrite(&Cords.first, sizeof(T), 1, out);

fwrite(&Cords.second, sizeof(T), 1, out);

fwrite(&Height, sizeof(T), 1, out);

fwrite(&Width, sizeof(T), 1, out);

}

template <class U>

friend std::ostream & operator << (std::ostream & of, const TRectangle<U> & rect) {

of << "Rectangle {";

of << std::pair<U, U>(rect.Cords.first, rect.Cords.second) << ", ";

of << std::pair<U, U>(rect.Cords.first, rect.Cords.second + rect.Height) << ", ";

of << std::pair<U, U>(rect.Cords.first + rect.Width, rect.Cords.second + rect.Height) << ", ";

of << std::pair<U, U>(rect.Cords.first + rect.Width, rect.Cords.second);

of << "}";

return of;

}

};

#endif

**triangle.hpp:**

#ifndef TRIANGLE\_HPP

#define TRIANGLE\_HPP

#include "figure.hpp"

#include <cmath>

const unsigned int TRIANGLE\_TYPE\_ID = 3;

template <class T>

class TTriangle : public IFigure {

private:

std::pair<T, T> Cords;

T Side;

public:

TTriangle() : Cords(), Side() {}

TTriangle(const std::pair<T, T> & xy, const T & l) : Cords(xy), Side(l) {}

void Write(FILE\* out) override {

fwrite(&TRIANGLE\_TYPE\_ID, sizeof(unsigned int), 1, out);

fwrite(&Cords.first, sizeof(T), 1, out);

fwrite(&Cords.second, sizeof(T), 1, out);

fwrite(&Side, sizeof(T), 1, out);

}

void Print(std::ostream & of) override {

of << \*this;

}

template <class U>

friend std::ostream & operator << (std::ostream & of, const TTriangle<U> & tri) {

of << "Triangle {";

of << std::pair<T, T>(tri.Cords.first, tri.Cords.second) << ", ";

of << std::pair<T, T>(tri.Cords.first + 0.5 \* tri.Side, tri.Cords.second + tri.Side \* (std::pow(3, 0.5) / 2)) << ", ";

of << std::pair<T, T>(tri.Cords.first + tri.Side, tri.Cords.second);

of << "}";

return of;

}

};

#endif

**main.cpp:**

#include "document.hpp"

int main() {

TDocument<int> doc;

std::string s;

while (std::cin >> s) {

if (s == "n") {

doc.CreateNew();

std::cout << "Created new document" << std::endl;

} else if (s == "o") {

std::cin >> s;

FILE\* in = fopen(s.c\_str(), "rb");

if (in == NULL) {

std::cout << "No such file on directory" << std::endl;

} else {

doc.LoadFromFile(in);

std::cout << "Loaded document from " << s << std::endl;

fclose(in);

}

} else if (s == "s") {

std::cin >> s;

FILE\* out = fopen(s.c\_str(), "wb");

if (out == NULL) {

std::cout << "No such file on directory" << std::endl;

} else {

doc.SaveToFile(out);

std::cout << "Saved document to " << s << std::endl;

fclose(out);

}

} else if (s == "+") {

size\_t pos;

unsigned short type;

std::cin >> pos >> type;

doc.Add(pos, type);

} else if (s == "-") {

size\_t pos;

std::cin >> pos;

doc.Delete(pos);

} else if (s == "p") {

std::cout << "Printing document:" << std::endl;

std::cout << doc;

} else if (s == "u") {

doc.Undo();

} else if (s == "h") {

std::cout << "\'n\' - create new document" << std::endl;

std::cout << "\'o\' - open document" << std::endl;

std::cout << "\'s\' - save document" << std::endl;

std::cout << "\'+\' - add a figure" << std::endl;

std::cout << "\'-\' - remove a figure" << std::endl;

std::cout << "\'p\' - print document" << std::endl;

std::cout << "\'u\' - undo changes" << std::endl;

std::cout << "\'h\' - show this message" << std::endl;

} else {

std::cout << "Unknown command. Type \'h\' to show help" << std::endl;

}

}

return 0;

}

1. Выводы

В ходе выполнения лабораторной работы я спроектировал систему классов для графического редактора, изучил различные способы записи структуры классов в файл и реализовал один из них. В жизни важно организовать рабочее пространство, это касается и программирования. Правильная структура программы и классов улучшает читаемость кода и облгегчает его дальнейшую поддержку.

1. Список литературы

1. fread — C++ Reference — Cplusplus.com

URL: <http://www.cplusplus.com/reference/cstdio/fread/>

2. fwrite — C++ Reference — Cplusplus.com

URL: http://www.cplusplus.com/reference/cstdio/fwrite/