**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: Попов Илья Павлович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата: 16.12

Оценка:

Москва, 2021

# 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 № 22.

Фигуры: 5 - угольник 6 - угольник 8 - угольник

# Описание программы

В программе реализованы функции сохранения фигур (квадрата, прямоугольника и трапеции) в файл, загрузки из файла и отмены последнего добавления / удаления фигуры в файл.

Программа состоит из 16 файлов:

1. figure.h - содержит реализацию родительского класса фигур;

pentagon.cpp и pentagon.h — реализация класса 5-угольника;

hexagon.cpp и hexagon.h — реализация класса 6-угольника;

octagon.cpp и octagon.h — реализация класса 8-угольника;

1. factory.cpp и factory.h — класс для создания графических примитивов фигур ;
2. document.cpp и document.h — реализация класса, содержащего непосредственно методы работы с файлом;
3. editor.cpp и editor.h — реализация класса, необходимого согласованной работы document и command;
4. command.cpp и command.h — реализация класса, реализующего перемещение по истории изменения документа, необходимое для операции undo;
5. main.cpp - файл с взаимодействием с пользователем.

Пользователю доступны команды:

1. create <path> — создание файла;
2. save — сохранить хранящиеся в document фигуры созданный ранее файл
3. add <hexagon, pentagon trapezoid> — добавить фигуру
4. remove <index> — удалить фигуру по индексу
5. print — печать document
6. undo — отмена последнего действия
7. load <path> — загрузить данные из указанного файла

# Набор тестов

## Тест №1 (демонстрирует работу программы при корректно введенных входных данных)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Available commands:

create <path> - create a new file

save - save data to a file

add <hexagon, pentagon trapezoid>

remove <index> - remove a figure by index

print

undo - undo the last action

load <path> - load data from a file

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

create test1.txt

Create success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

add pentagon

side = 5

Add success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

add hexagon

side = 6

Add success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

add octagon

side = 8

Add success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print

Pentagon

Side = 5

Area:43.0119

Hexagon

Side = 6

Area:93.5307

Octagon

Side = 8

Area:309.019

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

remove 0

Remove success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print

Hexagon

Side = 6

Area:93.5307

Octagon

Side = 8

Area:309.019

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

undo

Undo success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print

Pentagon

Side = 5

Area:43.0119

Hexagon

Side = 6

Area:93.5307

Octagon

Side = 8

Area:309.019

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

save

Save success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

create test2.txt

Create success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

add pentagon

side = 55

Add success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print

Pentagon

Side = 55

Area:5204.44

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

save

Save success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

load test1.txt

Load success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print

Pentagon

Side = 5

Area:43.0119

Hexagon

Side = 6

Area:93.5307

Octagon

Side = 8

Area:309.019

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

exit

## Тест №2 (демонстрирует работу программы при некорректно введенных входных данных)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Available commands:

create <path> - create a new file

save - save data to a file

add <hexagon, pentagon trapezoid>

remove <index> - remove a figure by index

print

undo - undo the last action

load <path> - load data from a file

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

create test1.txt

Create success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dsafsd

Error input!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

add fdsf

There is no such figure!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

add pentagon

side = sfd

Pentagon input error!

Repeat input!

side = 3

Add success!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

remove dsfsd

Index error!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

remove 333

Index error!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

undo

Undo success

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

undo

History is empty!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

load safaf

File is not opened!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

exit

# Результаты выполнения тестов

Представлены выше, с целью упростить прочтение.

# Листинг программы

## figure.h

#ifndef \_FIGURE\_H\_

#define \_FIGURE\_H\_

#include <iostream>

#include <fstream>

#include <cmath>

#include <string>

using namespace std;

struct figure {

virtual void print(ostream&) const = 0 ;

virtual void printFile(ofstream&) const = 0 ;

virtual double area() const = 0;

virtual ~figure() = default;

virtual bool is\_fig\_num(const string& s) {

bool point = false;

for (int i = 0; i < s.length(); ++i) {

if (s[i] == '.') {

if ((i == 0 || i == s.length() - 1) || point) {

return false;

}

else {

point = true;

}

}

else if (s[i] < '0' || s[i] > '9') { return false; }

}

return true;

}

};

#endif

## pentagon.cpp

#include "pentagon.h"

void pentagon::print(ostream& os) const {

os << "Pentagon\n";

os << "Side = " << side;

os << "\nArea:" << area() << '\n';

}

void pentagon::printFile(ofstream &of) const {

of << "pentagon\n"<< side << '\n';

}

double pentagon::area() const {

long double cur = (double)pow(side, 2) / (double)4;

long double res = cur \* sqrt(25 + 10 \* sqrt(5));

return res;

}

pentagon::pentagon(istream& is) {

string str;

cout << "side = "; is >> str; cout << endl;

while (true) {

if (is\_fig\_num(str)) {

side = stod(str);

break;

}

cout << "Pentagon input error!\n";

cout << "Repeat input!\n";

cout << "side = "; is >> str; cout << endl;

}

}

pentagon::pentagon(ifstream& is) {

is >> side;

}

## pentagon.h

#ifndef \_PENTAGON\_H\_

#define \_PENTAGON\_H\_

#include "figure.h"

using namespace std;

struct pentagon : figure{

private:

double side;

public:

void print(ostream&) const override ;

void printFile(ofstream&) const override ;

double area() const override ;

pentagon() = default;

pentagon(istream& is);

pentagon(ifstream& is);

};

#endif

## hexagon.cpp

#include "hexagon.h"

void hexagon::print(ostream& os) const {

os << "Hexagon\n";

os << "Side = " << side;

os << "\nArea:" << area() << '\n';

}

void hexagon::printFile(ofstream& of) const {

of << "hexagon\n" << side << '\n';

}

double hexagon::area() const{

long double res = (double)((3 \* sqrt(3)) / 2) \* pow(side, 2);

return res;

}

hexagon::hexagon(istream& is) {

string str;

cout << "side = "; is >> str; cout << endl;

while (true) {

if (is\_fig\_num(str)) {

side = stod(str);

break;

}

cout << "Hexagon input error!\n";

cout << "Repeat input!\n";

cout << "side = "; is >> str; cout << endl;

}

}

hexagon::hexagon(ifstream& is) {

is >> side;

}

## hexagon.h

#ifndef \_HEXAGON\_H\_

#define \_HEXAGON\_H\_

#include "figure.h"

using namespace std;

struct hexagon : figure {

private:

double side;

public:

void print(ostream&) const override ;

void printFile(ofstream&) const override ;

double area() const override ;

hexagon() = default;

hexagon(istream& is);

hexagon(ifstream& is);

};

#endif

## octagon.cpp

#include "octagon.h"

void octagon::print(ostream& os) const {

os << "Octagon\n";

os << "Side = " << side;

os << "\nArea:" << area() << '\n';

}

void octagon::printFile(ofstream& of) const {

of << "octagon\n" << side << '\n';

}

double octagon::area() const {

long double res = (2 + 2 \* sqrt(2)) \* pow(side, 2);

return res;

}

octagon::octagon(istream& is) {

string str;

cout << "side = "; is >> str; cout << endl;

while (true) {

if (is\_fig\_num(str)) {

side = stod(str);

break;

}

cout << "Octagon input error!\n";

cout << "Repeat input!\n";

cout << "side = "; is >> str; cout << endl;

}

}

octagon::octagon(ifstream& is) {

is >> side;

}

## octagon.h

#ifndef \_OCTAGON\_H\_

#define \_OCTAGON\_H\_

#include "figure.h"

using namespace std;

struct octagon : figure {

private:

double side;

public:

void print(ostream&) const override;

void printFile(ofstream&) const override;

double area() const override;

octagon() = default;

octagon(istream& is);

octagon(ifstream& is);

};

#endif

## factory.cpp

#include "factory.h"

shared\_ptr<figure> factory::FigureCreate(istream &is) {

string name;

is >> name;

if (name == "hexagon" ) {

return shared\_ptr<figure> ( new hexagon(is));

} else if (name == "pentagon") {

return shared\_ptr<figure> ( new pentagon(is));

} else if (name == "octagon") {

return shared\_ptr<figure> ( new octagon(is));

} else {

throw logic\_error("There is no such figure!\n");

}

}

shared\_ptr<figure> factory::FigureCreateFile(ifstream &is) {

string name;

is >> name;

if (name == "hexagon" ) {

return shared\_ptr<figure> (new hexagon(is));

} else if (name == "pentagon") {

return shared\_ptr<figure> (new pentagon(is));

} else if (name == "octagon") {

return shared\_ptr<figure> (new octagon(is));

} else {

throw logic\_error("There is no such figure!\n");

}

}

## factory.h

#ifndef \_FACTORY\_H\_

#define \_FACTORY\_H\_

#include <memory>

#include <iostream>

#include <fstream>

#include <string>

#include "hexagon.h"

#include "pentagon.h"

#include "octagon.h"

struct factory {

shared\_ptr<figure> FigureCreate(istream& is);

shared\_ptr<figure> FigureCreateFile(ifstream& is);

};

#endif

## document.cpp

#include "document.h"

void document::Print() const {

if (buffer\_.empty()) {

cout << "Buffer is empty!\n";

}

for (auto elem : buffer\_) {

elem->print(cout);

}

}

void document::Insert(shared\_ptr<figure> &ptr) {

buffer\_.push\_back(ptr);

}

void document::Rename(const string &newName) {

name\_ = newName;

}

void document::Save(const string &filename) const {

ofstream fout;

fout.open(filename);

if (!fout.is\_open()) {

throw runtime\_error("File is not opened!\n");

}

fout << buffer\_.size() << '\n';

for (auto elem : buffer\_) {

elem->printFile(fout);

}

}

void document::Load(const string &filename) {

ifstream fin;

fin.open(filename);

if (!fin.is\_open()) {

throw runtime\_error("File is not opened!\n");

}

size\_t size;

fin >> size;

buffer\_.clear();

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

buffer\_.push\_back(factory\_.FigureCreateFile(fin));

}

name\_ = filename;

}

shared\_ptr<figure> document::GetFigure(uint32\_t index) {

return buffer\_[index];

}

void document::Erase(uint32\_t index) {

if ( index >= buffer\_.size()) {

throw logic\_error("Out of bounds!\n");

}

buffer\_[index] = nullptr;

for (; index < buffer\_.size() - 1; ++index) {

buffer\_[index] = buffer\_[index + 1];

}

buffer\_.pop\_back();

}

string document::GetName() {

return this->name\_;

}

size\_t document::Size() {

return buffer\_.size();

}

void document::RemoveLast() {

if (buffer\_.empty()) {

throw logic\_error("Document is empty!");

}

buffer\_.pop\_back();

}

void document::InsertIndex(shared\_ptr<figure> &newFigure, uint32\_t index) {

buffer\_.insert(buffer\_.begin() + index, newFigure);

}

## document.h

#ifndef \_DOCUMENT\_H\_

#define \_DOCUMENT\_H\_

#include <cstdint>

#include <memory>

#include <string>

#include <algorithm>

#include <vector>

#include "figure.h"

#include "factory.h"

struct document {

public:

void Print() const ;

document(string& newName): name\_(newName), factory\_(), buffer\_(0) {};

void Insert(shared\_ptr<figure>& ptr);

void Rename(const string& newName);

void Save(const string& filename) const;

void Load(const string& filename);

shared\_ptr<figure> GetFigure(uint32\_t index);

void Erase(uint32\_t index);

string GetName();

size\_t Size();

private:

friend class InsertCommand;

friend class DeleteCommand;

factory factory\_;

string name\_;

vector<shared\_ptr<figure>> buffer\_;

void RemoveLast();

void InsertIndex(shared\_ptr<figure>& newFigure, uint32\_t index);

};

#endif

## editor.cpp

#include "editor.h"

void editor::PrintDocument() {

if (doc\_ == nullptr) {

cout << "No document!\n\n";

return;

}

doc\_->Print();

}

void editor::CreateDocument(string &newName) {

doc\_ = make\_shared<document>(newName);

}

bool editor::DocumentExist() {

return doc\_ != nullptr;

}

void editor::InsertInDocument(shared\_ptr<figure> &newFigure) {

if (doc\_ == nullptr) {

cout << "No document!\n\n";

return;

}

shared\_ptr<Acommand> command = shared\_ptr<Acommand>(new InsertCommand(doc\_));

doc\_->Insert(newFigure);

history\_.push(command);

cout << "Add success!\n\n";

}

void editor::DeleteInDocument(uint32\_t index) {

if (doc\_ == nullptr) {

cout << "No document!\n\n";

return;

}

if (index >= doc\_->Size()) {

cout << "Index error!\n\n";

return;

}

shared\_ptr<figure> tmp = doc\_->GetFigure(index);

shared\_ptr<Acommand> command = shared\_ptr<Acommand>(new DeleteCommand(tmp,index,doc\_));

doc\_->Erase(index);

history\_.push(command);

cout << "Remove success\n\n";

}

void editor::SaveDocument() {

if (doc\_ == nullptr) {

cout << "No document!\n\n";

return;

}

string saveName = doc\_->GetName();

doc\_ ->Save(saveName);

cout << "Save success\n\n";

}

void editor::LoadDocument(string &name) {

try {

doc\_ = make\_shared<document>(name);

doc\_->Load(name);

while (!history\_.empty()){

history\_.pop();

}

} catch(logic\_error& e) {

cout << e.what();

}

}

void editor::Undo() {

if (history\_.empty()) {

throw logic\_error("History is empty!\n\n");

}

shared\_ptr<Acommand> lastCommand = history\_.top();

lastCommand->UnExecute();

history\_.pop();

}

## editor.h

#ifndef \_D\_EDITOR\_H\_

#define \_D\_EDITOR\_H\_

#include <stack>

#include "command.h"

struct editor {

private:

shared\_ptr<document> doc\_;

stack<shared\_ptr<Acommand>> history\_;

public:

~editor() = default;

void PrintDocument();

void CreateDocument(string& newName);

bool DocumentExist();

editor() : doc\_(nullptr), history\_() {}

void InsertInDocument(shared\_ptr<figure>& newFigure);

void DeleteInDocument(uint32\_t index);

void SaveDocument();

void LoadDocument(string& name);

void Undo();

};

#endif

## command.cpp

#include "command.h"

void InsertCommand::UnExecute() {

doc\_->RemoveLast();

}

InsertCommand::InsertCommand(shared\_ptr<document> &doc) {

doc\_ = doc;

}

DeleteCommand::DeleteCommand(shared\_ptr<figure> &newFigure, uint32\_t newIndex, shared\_ptr<document> &doc) {

doc\_ = doc;

figure\_ = newFigure;

index\_ = newIndex;

}

void DeleteCommand::UnExecute() {

doc\_->InsertIndex(figure\_,index\_);

}

## command.h

#ifndef \_COMMAND\_H\_

#define \_COMMAND\_H\_

#include "document.h"

struct Acommand {

public:

virtual ~Acommand() = default;

virtual void UnExecute() = 0;

protected:

shared\_ptr<document> doc\_;

};

struct InsertCommand : public Acommand {

public:

void UnExecute() override;

InsertCommand(shared\_ptr<document>& doc);

};

struct DeleteCommand : public Acommand {

public:

DeleteCommand(shared\_ptr<figure>& newFigure, uint32\_t newIndex,shared\_ptr<document>& doc);

void UnExecute() override;

private:

shared\_ptr<figure> figure\_;

uint32\_t index\_;

};

#endif

## main.cpp

//Попов Илья Павлович

//М80-206Б-20

//Лабораторная работа №7

//Вариант № 22.

//Фигуры: 5 - угольник 6 - угольник 8 - угольник

//Взаимодействуем с документом(document.h) происходит посредством editor.h, в котором лежат ссылки на документ(вектор фигур) и стек команд

//Перенаправление записи и вывода из файла происходят в factory.h

//Команды отслеживаются в command.h, что необходимо для реализации undo

/\*

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 "factory.h"

#include "editor.h"

void usage() {

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Available commands:\n";

cout << "create <path> - create a new file\n";

cout << "save - save data to a file\n";

cout << "add <hexagon, pentagon trapezoid>\n";

cout << "remove <index> - remove a figure by index\n";

cout << "print\n";

cout << "undo - undo the last action\n";

cout << "load <path> - load data from a file\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;

}

bool is\_number(const string& s) {

bool point = false;

for (int i = 0; i < s.length(); ++i) {

if (s[i] == '-' && i == 0) {

continue;

}

else if (s[i] == '.') {

if ((i == 0 || i == s.length() - 1) || point) {

return false;

}

else {

point = true;

}

}

else if (s[i] < '0' || s[i] > '9') { return false; }

}

return true;

}

void create(editor& edit) {

string tmp;

cin >> tmp;

edit.CreateDocument(tmp);

cout << "Create success\n\n";

}

void load(editor& edit) {

string tmp;

cin >> tmp;

try {

edit.LoadDocument(tmp);

cout << "Load success\n\n";

} catch (runtime\_error& e) {

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

}

}

void save(editor& edit) {

string tmp;

try {

edit.SaveDocument();

} catch (runtime\_error& err) {

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

}

}

void add(editor& edit) {

factory fac;

try {

shared\_ptr<figure> newElem = fac.FigureCreate(cin);

edit.InsertInDocument(newElem);

} catch (logic\_error& err) {

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

}

}

void remove(editor& edit) {

try {

string str;

cin >> str;

if (!is\_number(str)) {

cout << "Index error!\n\n";

}

else {

uint32\_t index = stoi(str);

edit.DeleteInDocument(index);

}

} catch (logic\_error& err) {

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

}

}

int main() {

usage();

editor edit;

string cmd;

while (true) {

cin >> cmd;

if (cmd == "help") {

usage();

} else if (cmd == "create") {

create(edit);

} else if (cmd == "load") {

load(edit);

} else if (cmd == "save") {

save(edit);

} else if (cmd == "exit") {

break;

} else if (cmd == "add") {

add(edit);

} else if (cmd == "remove") {

remove(edit);

} else if (cmd == "print") {

edit.PrintDocument();

} else if (cmd == "undo") {

try {

edit.Undo();

cout << "Undo success\n\n";

} catch (logic\_error& err) {

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

}

} else {

cout << "Error input!\n\n";

}

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

return 0;

}

# Вывод

Почти все серьезные коммерческие проекты используют работу с файлами, в парадигме объектно-ориентированного программирования реализация этого механизма представлена довольно просто и ознакомиться с ней должен каждый программист.

Также в ходе выполнения лабораторной работы была решена непростая задача — реализована функция undo, которая отменяет последнее удаление и добавление фигуры. Данную задачу удалось решить путем реализации класса command, который хранит историю(стек команд) изменения document.

# ЛИТЕРАТУРА

1. Работа с файлами язык программирования С++ [Электронный ресурс].

URL:<http://cppstudio.com/post/446/> (дата обращения: 16.12.2021).

1. Базовый файловый вывод и ввод в С++ [Электронный ресурс].

URL:<https://ravesli.com/urok-212-bazovyj-fajlovyj-vvod-vyvod/> (дата обращения: 16.12.2021).