Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа**

**по курсу «ООП»**

**Тема:**

**Основы работы с коллекциями: итераторы.**

|  |  |
| --- | --- |
| Студент: | Симонов С.Я. |
| Группа: | М8О-206Б-18 |
| Преподаватель: | Журавлев А.А. |
| Вариант: | 21 |
| Оценка: |  |
| Дата: |  |

Москва

2019

**1. Код на C++:**

command.cpp

#include "command.hpp"

void InsertCommand::UnExecute() {

doc\_->RemoveLast();

}

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

doc\_ = doc;

}

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

doc\_ = doc;

figure\_ = newFigure;

index\_ = newIndex;

}

void DeleteCommand::UnExecute() {

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

}

command.hpp

#ifndef OOP7\_COMMAND\_HPP

#define OOP7\_COMMAND\_HPP

#include "document.hpp"

struct Acommand {

virtual ~Acommand() = default;

virtual void UnExecute() = 0;

protected:

std::shared\_ptr<document> doc\_;

};

struct InsertCommand : public Acommand {

public:

void UnExecute() override;

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

};

struct DeleteCommand : public Acommand {

public:

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

void UnExecute() override;

private:

std::shared\_ptr<figure> figure\_;

uint32\_t index\_;

};

#endif

document.cpp

#include "document.hpp"

void document::Print() const {

{

if (buffer\_.empty()) {

std::cout << "Buffer is empty\n";

}

for (auto elem : buffer\_) {

elem->print(std::cout);

}

}

}

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

buffer\_.push\_back(ptr);

}

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

name\_ = newName;

}

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

std::ofstream fout;

fout.open(filename);

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

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

}

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

for (auto elem : buffer\_) {

elem->printFile(fout);

}

}

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

std::ifstream fin;

fin.open(filename);

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

throw std::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;

}

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

return buffer\_[index];

}

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

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

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

}

buffer\_[index] = nullptr;

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

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

}

buffer\_.pop\_back();

}

std::string document::GetName() {

return this->name\_;

}

size\_t document::Size() {

return buffer\_.size();

}

void document::RemoveLast() {

if (buffer\_.empty()) {

throw std::logic\_error("Document is empty");

}

buffer\_.pop\_back();

}

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

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

}

document.hpp

#ifndef OOP7\_DOCUMENT\_HPP

#define OOP7\_DOCUMENT\_HPP

#include <fstream>

#include <cstdint>

#include <memory>

#include <string>

#include <algorithm>

#include "figure.hpp"

#include <vector>

#include "factory.hpp"

struct document {

public:

void Print() const ;

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

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

void Rename(const std::string& newName);

void Save (const std::string& filename) const;

void Load(const std::string& filename);

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

void Erase(uint32\_t index);

std::string GetName();

size\_t Size();

private:

friend class InsertCommand;

friend class DeleteCommand;

factory factory\_;

std::string name\_;

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

void RemoveLast();

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

};

#endif

editor.cpp

#include "editor.hpp"

void editor::PrintDocument() {

if (doc\_ == nullptr) {

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

return;

}

doc\_->Print();

}

void editor::CreateDocument(std::string &newName) {

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

}

bool editor::DocumentExist() {

return doc\_ != nullptr;

}

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

if (doc\_ == nullptr) {

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

return;

}

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

doc\_->Insert(newFigure);

history\_.push(command);

}

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

if (doc\_ == nullptr) {

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

return;

}

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

std::cout << "Out of bounds\n";

return;

}

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

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

doc\_->Erase(index);

history\_.push(command);

}

void editor::SaveDocument() {

if (doc\_ == nullptr) {

std::cout << "No document!\nNot ";

return;

}

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

doc\_ ->Save(saveName);

}

void editor::LoadDocument(std::string &name) {

try {

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

doc\_->Load(name);

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

history\_.pop();

}

} catch(std::logic\_error& e) {

std::cout << e.what();

}

}

void editor::Undo() {

if (history\_.empty()) {

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

}

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

lastCommand->UnExecute();

history\_.pop();

}

editor.hpp

#ifndef OOP7\_EDITOR\_HPP

#define OOP7\_EDITOR\_HPP

#include "figure.hpp"

#include "document.hpp"

#include <stack>

#include "command.hpp"

struct editor {

private:

std::shared\_ptr<document> doc\_;

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

public:

~editor() = default;

void PrintDocument();

void CreateDocument(std::string& newName);

bool DocumentExist();

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

{

}

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

void DeleteInDocument(uint32\_t index);

void SaveDocument();

void LoadDocument(std::string& name);

void Undo();

};

#endif

factory.cpp

#include "factory.hpp"

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

std::string name;

is >> name;

if ( name == "pentagon" ) {

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

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

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

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

return std::shared\_ptr<figure> ( new rhombus(is));

} else {

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

}

}

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

std::string name;

is >> name;

if ( name == "pentagon" ) {

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

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

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

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

return std::shared\_ptr<figure> ( new rhombus(is));

} else {

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

}

}

factory.hpp

#ifndef OOP7\_FACTORY\_HPP

#define OOP7\_FACTORY\_HPP

#include <memory>

#include <iostream>

#include <fstream>

#include "hexagon.hpp"

#include "rhombus.hpp"

#include "pentagon.hpp"

#include <string>

struct factory {

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

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

};

#endif

figure.hpp

#ifndef OOP7\_FIGURE\_HPP

#define OOP7\_FIGURE\_HPP

#include <iostream>

#include "point.hpp"

#include <fstream>

struct figure {

virtual point center() const = 0;

virtual void print(std::ostream&) const = 0 ;

virtual void printFile(std::ofstream&) const = 0 ;

virtual double area() const = 0;

virtual ~figure() = default;

};

#endif

hexagon.cpp

#include "hexagon.hpp"

point hexagon::center() const {

double x,y;

x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x) / 6;

y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y) / 6;

point p(x,y);

return p;

}

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

os << "a1 = " << a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " << a4 << " a5 = " << a5 << " a6 = " << a6 << "\n";

}

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

of << "a1 = " << a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " << a4 << " a5 = " << a5 << " a6 = " << a6 << "\n";

}

double hexagon::area() const {

return (-0.5) \* ((a1.x\*a2.y + a2.x\*a3.y + a3.x\*a4.y + a4.x\*a5.y + a5.x\*a6.y + a6.x\*a1.y) - ( a1.y\*a2.x + a2.y\*a3.x + a3.y\*a4.x + a4.y\*a5.x + a5.y\*a6.x + a6.y\*a1.x ));

}

hexagon::hexagon(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;

}

hexagon::hexagon(std::ifstream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;

}

hexagon.hpp

#ifndef D\_HEXAGON\_HPP\_

#define D\_HEXAGON\_HPP\_

#include <iostream>

#include "figure.hpp"

struct hexagon : figure

{

point center() const override ;

void print(std::ostream&) const override ;

void printFile(std::ofstream&) const override ;

double area() const override ;

hexagon() = default;

hexagon(std::istream& is);

hexagon(std::ifstream& is);

private:

point a1, a2, a3, a4, a5, a6;

};

#endif

main.cpp

#include <iostream>

#include "factory.hpp"

#include "editor.hpp"

void help() {

std::cout << "1 - help\n"

"2 - создать\n"

"3 - загрузить\n"

"4 - сохранить\n"

"5 - добавить фигуру\n"

"6 - удалить\n"

"7 - печать\n"

"8 - назад\n"

"0 - выход\n";

}

void create(editor& edit) {

std::string tmp;

std::cout << "Введите название нового документа\n";

std::cin >> tmp;

edit.CreateDocument(tmp);

std::cout << "Документ создан\n";

}

void load(editor& edit) {

std::string tmp;

std::cout << "Введите название файла\n";

std::cin >> tmp;

try {

edit.LoadDocument(tmp);

std::cout << "Документ загружен\n";

} catch (std::runtime\_error& e) {

std::cout << e.what();

}

}

void save(editor& edit) {

std::string tmp;

try {

edit.SaveDocument();

std::cout << "Документ сохранен\n";

} catch (std::runtime\_error& e) {

std::cout << e.what();

}

}

void add(editor& edit) {

factory fac;

try {

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

edit.InsertInDocument(newElem);

} catch (std::logic\_error& e) {

std::cout << e.what() << '\n';

}

std::cout << "Ok\n";

}

void remove(editor& edit) {

uint32\_t index;

std::cout << "Введите индекс фигуры которую хотите удалить\n";

std::cin >> index;

try {

edit.DeleteInDocument(index);

std::cout << "Ok\n";

} catch (std::logic\_error& err) {

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

}

}

int main() {

editor edit;

int command;

std::cout << "1 - help" << "\n"

<< "2 - создать файл" << "\n"

<< "3 - загрузить" << "\n"

<< "4 - сохранить" << "\n"

<< "5 - добавить фигуру" << "\n"

<< "6 - удалить фигуру из файла" << "\n"

<< "7 - печать" << "\n"

<< "8 - назад" << "\n"

<< "0 - выход" << "\n";

while (true) {

std::cin >> command;

if (command == 1) {

help();

} else if (command == 2) {

create(edit);

} else if (command == 3) {

load(edit);

} else if (command == 4) {

save(edit);

} else if (command == 0) {

break;

} else if (command == 5) {

add(edit);

} else if (command == 6) {

remove(edit);

} else if (command == 7) {

edit.PrintDocument();

} else if (command == 8) {

try {

edit.Undo();

} catch (std::logic\_error& e) {

std::cout << e.what();

}

} else {

std::cout << "ERROR\n";

}

}

return 0;

}

pentagon.cpp

#include "pentagon.hpp"

#include <cmath>

#include "point.hpp"

point pentagon::center() const {

double x,y;

x = (a1.x + a2.x + a3.x + a4.x + a5.x) / 5;

y = (a1.y + a2.y + a3.y + a4.y + a5.y) / 5;

point p(x,y);

return p;

}

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

os << "a1 = " << a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " << a4 << " a5 = " << a5 << '\n';

}

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

of << "a1 = " << a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " << a4 << " a5 = " << a5 << '\n';

}

double pentagon::area() const {

return (-0.5) \* ((a1.x\*a2.y + a2.x\*a3.y + a3.x\*a4.y + a4.x\*a5.y + a5.x\*a1.y) - ( a1.y\*a2.x + a2.y\*a3.x + a3.y\*a4.x + a4.y\*a5.x + a5.y\*a1.x ));

}

pentagon::pentagon(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5;

}

pentagon::pentagon(std::ifstream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5;

}

pentagon.hpp

#ifndef OOP\_PENTAGON\_HPP

#define OOP\_PENTAGON\_HPP

#include "figure.hpp"

struct pentagon : figure{

private:

point a1,a2,a3,a4,a5;

public:

point center() const override;

void print(std::ostream&) const override;

void printFile(std::ofstream&) const override;

double area() const override;

pentagon() = default;

pentagon(std::istream& is);

pentagon(std::ifstream& is);

};

#endif

point.cpp

#include "point.hpp"

std::istream& operator >> (std::istream& is,point& p ) {

return is >> p.x >> p.y;

}

std::ostream& operator << (std::ostream& os,const point& p) {

return os << p.x <<' '<< p.y;

}

point.hpp

#ifndef OOP7\_POINT\_H

#define OOP7\_POINT\_H

#include <iostream>

struct point {

double x, y;

point (double a,double b) { x = a, y = b;};

point() = default;

};

std::istream& operator >> (std::istream& is,point& p );

std::ostream& operator << (std::ostream& os,const point& p);

#endif

rhombus.cpp

#include <iostream>

#include <cmath>

#include "rhombus.hpp"

rhombus::rhombus(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4;

double str1, str2, str3, str4;

str1 = sqrt((a1.x - a2.x) \* (a1.x - a2.x) + (a1.y - a2.y) \* (a1.y - a2.y));

str2 = sqrt((a2.x - a3.x) \* (a2.x - a3.x) + (a2.y - a3.y) \* (a2.y - a3.y));

str3 = sqrt((a3.x - a4.x) \* (a3.x - a4.x) + (a3.y - a4.y) \* (a3.y - a4.y));

str4 = sqrt((a4.x - a1.x) \* (a4.x - a1.x) + (a4.y - a1.y) \* (a4.y - a1.y));

if (str1 != str2 || str2 != str3 || str3 != str4) {

throw std::logic\_error("Is not rhombus");

}

}

rhombus::rhombus(const point& a1, const point& a2, const point& a3, const point& a4) {

double str1, str2, str3, str4;

str1 = sqrt((a1.x - a2.x) \* (a1.x - a2.x) + (a1.y - a2.y) \* (a1.y - a2.y));

str2 = sqrt((a2.x - a3.x) \* (a2.x - a3.x) + (a2.y - a3.y) \* (a2.y - a3.y));

str3 = sqrt((a3.x - a4.x) \* (a3.x - a4.x) + (a3.y - a4.y) \* (a3.y - a4.y));

str4 = sqrt((a4.x - a1.x) \* (a4.x - a1.x) + (a4.y - a1.y) \* (a4.y - a1.y));

if (str1 != str2 || str2 != str3 || str3 != str4) {

throw std::logic\_error("Is not rhombus");

}

}

point rhombus::center() const {

double x, y;

x = (a1.x + a2.x + a3.x + a4.x) / 4;

y = (a1.y + a2.y + a3.y + a4.y) / 4;

point p(x, y);

return p;

}

void rhombus::print(std::ostream& os) const {

os << "a1 = " << a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " << a4 << "\n";

}

double rhombus::area() const {

point v\_1;

v\_1.x = a1.x - a3.x;

v\_1.y = a1.y - a3.y;

point v\_2;

v\_2.x = a2.x - a4.x;

v\_2.y = a2.y - a4.y;

double result = 0.5 \* (sqrt(v\_1.x \* v\_1.x + v\_1.y \* v\_1.y) \* sqrt(v\_2.x \* v\_2.x + v\_2.y \* v\_2.y));

return result;

}

void rhombus::printFile(std::ofstream &of) const {

of << "a1 = "<< a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " << a4 << '\n';

}

rhombus::rhombus(std::ifstream& is) {

is >> a1 >> a2 >> a3 >> a4;

}

rhombus.hpp

#ifndef D\_RHOMBUS\_HPP\_

#define D\_RHOMBUS\_HPP\_ 1

#include <iostream>

#include "figure.hpp"

struct rhombus : figure

{

rhombus(std::istream& is);

rhombus(const point& a1, const point& a2, const point& a3, const point& a4);

rhombus() = default;

rhombus(std::ifstream& is);

point center() const override;

void print(std::ostream& os) const override;

void printFile(std::ofstream&) const override ;

double area() const override;

private:

point a1, a2, a3, a4;

};

#endif

**2. Ссылка на репозиторий в GitHub:**

https://github.com/keoni02032/oop\_exercise\_07

**3. Набор testcases:**

test\_01.test

2

kek.txt

5

rhombus

1 1 1 1 1 1 1 1

5

pentagon

1 1 2 2 3 3 4 4 5 5

5

hexagon

1 1 2 2 3 3 4 4 5 5 6 6

4

8

8

8

8

0

test\_02.test

3

kek.txt

7

5

pentagon

1 1 2 2 3 3 4 4 5 5

5

hexagon

1 1 2 2 3 3 4 4 5 5 6 6

7

4

8

8

8

7

0

**4.Результаты выполнения программы:**

**test\_01.result**

**1 - help**

**2 - создать файл**

**3 - загрузить**

**4 - сохранить**

**5 - добавить фигуру**

**6 - удалить фигуру из файла**

**7 - печать**

**8 - назад**

**0 - выход**

**2**

**Введите название нового документа**

**kek.txt**

**Документ создан**

**5**

**rhombus**

**1 1 1 1 1 1 1 1**

**Ok**

**5**

**pentagon**

**1 1 2 2 3 3 4 4 5 5**

**Ok**

**5**

**hexagon**

**1 1 2 2 3 3 4 4 5 5 6 6**

**Ok**

**4**

**Документ сохранен**

**8**

**8**

**8**

**8**

**History is empty**

**0**

**test\_02.result**

**1 - help   
2 - создать файл   
3 - загрузить   
4 - сохранить   
5 - добавить фигуру   
6 - удалить фигуру из файла   
7 - печать   
8 - назад   
0 - выход   
3   
Введите название файла   
kek.txt   
There is no such figure   
Документ загружен   
7   
Buffer is empty   
5   
pentagon   
1 1 2 2 3 3 4 4 5 5    
Ok   
5   
hexagon   
1 1 2 2 3 3 4 4 5 5 6 6    
Ok   
7   
a1 = 1 1 a2 = 2 2 a3 = 3 3 a4 = 4 4 a5 = 5 5   
a1 = 1 1 a2 = 2 2 a3 = 3 3 a4 = 4 4 a5 = 5 5 a6 = 6 6   
4   
Документ сохранен   
8   
8   
8   
History is empty   
7   
Buffer is empty   
6   
Введите индекс фигуры которую хотите удалить   
0   
Out of bounds   
Ok   
0**

**5. Объяснение результатов работы программы:**

При вводе комады «2» – происходит создание файла, «1» – команда помощи по вводу команд, «3» – загрузить уже созданный файл, «4» – сохранить введеные фигуры, «5» – добавление фигуры (по заданию необходимо ввести одну из следующих фигур: «rhombus», «pentagon», «hexagon»), «6» – удаление фигуры из файла по индексу, «7» – напечатать все фигуры которые находятся в данный момент в буфере, «8» – откатить программу на одно действие, «0» – выход из программы.

**6. Вывод:**

В данной лабораторной работе, я получил хороший опыт работы в проектировании структуры классов приложения. Реализовал выгрузку и загрузку в файл.