**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: Ляшун Дмитрий Сергеевич

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Спроектировать простейший «графический» векторный редактор. Требование к функционалу редактора:

* Создание нового документа;
* Импорт документа из файл;
* 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 №24:**

|  |  |  |
| --- | --- | --- |
| **Фигура №1** | **Фигура №2** | **Фигура №3** |
| 8-угольник | Треугольник | Квадрат |

1. **Описание программы**

Для представления координат вершин фигур опишем структуру **Point** с вещественными полями x и y.

Для реализации классов заданных фигур сперва опишем базовый абстрактный класс **Figure** со следующими полями:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| square | double | Хранит площадь фигуры. |
| type | int | Хранит числовое значение типа фигуры. |

Класс **Figure** будет иметь следующие методы:

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| Figure(int type) | Конструктор, создающий фигуру с типом type. |
| virtual void Output() | Функция вывода фигуры, определяется только в классах-наследниках. |
| int GetType() | Возвращает числовое значение типа фигуры. |
| void SetSquare(const double& sq) | Изменяет значение площади фигуры на sq. |
| double GetSquare() | Возвращает значение площади фигуры. |

Классы **Triangle**, **Square** и **Octagon**, которые представляют треугольник, квадрат и 8-угольник соответственно, будут являться классами-наследниками от класса **Figure**. В них не будут добавлены новые поля и методы, а только переопределены виртуальная функция Output() (перед выводом фигуры будет выводиться также тип фигуры) из класса-предка. Также в каждом из этих классов будет определен свой конструктор по умолчанию, который вызывает конструктор базового класса Figure(int type) со значением type = 1 для квадрата, 2 для треугольника и 3 для 8-угольника.

Для создания фигур опишем класс **Factory** со следующим методом:

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| Figure\* Create(int type, const std::vector<double>& p) | Возвращает указатель на фигуру типа type, созданную по набору данных p. |

Для работы с документом опишем класс **Document** со следующими полями:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| data | std::vector<Figure\*> data | Хранит данные о фигурах документа. |
| adder | Factory | Используется для добавления фигур в документ. |
| undo | std::stack<std::pair<Figure\*,int> > | Хранит информацию о добавлении или удалении фигур при вызове отмены действия. |

В свою очередь в классе **Document** будут реализованы следующие методы:

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| void Insert(Figure\* figure, const int& index) | Процедура вставки фигуры figure в документ на позицию index. |
| void Erase(const int& index) | Процедура удаления фигуры в позиции index из документа. |
| void LoadFromFile(FILE\* file) | Процедура загрузки документа из файла file. |
| void SaveInFile(FILE\* file) | Процедура записи документа в файл file. |
| void Undo() | Процедура отмены последнего действия в документе. |
| void Print() | Процедура печати содержимого документа. |
| void NewDocument(FILE\* file) | Процедура создания нового документа, текущий документ будет сохранен в файл file. |
| bool Empty() | Функция, возвращающая true, если документ пустой, иначе false. |

1. **Руководство по использованию программы**

**oop\_exercise\_07** – исполняемый файл программы.

Взаимодействие пользователя с программой было реализовано посредством интерактивного меню со следующими поддерживаемыми командами:

1. Создание нового пустого документа с сохранением старого на внешнем файле.
2. Импорт документа из файла.
3. Экспорт документа в файл.
4. Добавление фигуры в текущий документ.
5. Удаление фигуры из документа.
6. Печать содержимого документа.
7. Отмена последнего действия в документе (добавление/удаление фигур).
8. Выход из программы.

**Входные данные**

Номера вызываемых команд, позиция проведения операции (для вставки или удаления), информация о равносторонних фигурах (для вставки): для квадрата – координаты противоположных вершин (стороны квадрата параллельны осям координат), для треугольника – координаты вершин, для восьмиугольника – координаты центра и вектора радиуса описанной окружности, имена файлов для чтения и записи данных документа.

**Выходные данные**

В ответ на каждую команду выводится её результат выполнения.

**Тестовые примеры**

Список поддерживаемых команд (для вызова необходимо ввести номер):

1 - Создание нового документа

2 - Импорт документа из файла

3 - Экспорт документа в файл

4 - Добавление фигуры

5 - Удаление фигуры

6 - Печать документа

7 - Отмена последнего действия

8 - Выход из программы

> 4

Введите тип добавляемой фигуры (1 - Квадрат, 2 - Треугольник, 3 - Восьмиугольник) > 1

Введите координаты противоположных вершин квадрата > 0 0 2 2

Введите позицию вставки фигуры > 1

Фигура успешно вставлена.

> 4

Введите тип добавляемой фигуры (1 - Квадрат, 2 - Треугольник, 3 - Восьмиугольник) > 2

Введите координаты вершин треугольника > 0 0 2 0 1 2

Введите позицию вставки фигуры > 2

Фигура успешно вставлена.

> 4

Введите тип добавляемой фигуры (1 - Квадрат, 2 - Треугольник, 3 - Восьмиугольник) > 3

Введите координаты центра восьмиугольника и вектора описанной окружности > 4 4 2 2

Введите позицию вставки фигуры > 2

Фигура успешно вставлена.

> 6

Содержимое документа:

Квадрат: ( 0, 0 ), ( 2, 2 ), S = 4

Восьмиугольник: ( 4, 4 ), ( 2, 2 ), S = 22.6274

Треугольник: ( 0, 0 ), ( 2, 0 ), ( 1, 2 ), S = 1.73205

> 3

Введите имя файла для экспорта документа > t1.txt

Документ успешно записан в файл.

> 7

Последнее действие успешно отменено!

> 6

Содержимое документа:

Квадрат: ( 0, 0 ), ( 2, 2 ), S = 4

Треугольник: ( 0, 0 ), ( 2, 0 ), ( 1, 2 ), S = 1.73205

> 7

Последнее действие успешно отменено!

> 6

Содержимое документа:

Квадрат: ( 0, 0 ), ( 2, 2 ), S = 4

> 2

Введите имя файла для импорта документа > t1.txt

Документ успешно загружен из файла.

> 6

Содержимое документа:

Квадрат: ( 0, 0 ), ( 2, 2 ), S = 4

Восьмиугольник: ( 4, 4 ), ( 2, 2 ), S = 22.6274

Треугольник: ( 0, 0 ), ( 2, 0 ), ( 1, 2 ), S = 1.73205

> 1

Вы хотите сохранить старый документ в файл перед созданием нового? (y/n)

> n

Новый документ успешно создан.

> 6

Документ пустой!

> 8

**4. Листинг программы**

Исходный код программы состоит из трех файлов: **figures.hpp** – содержит реализацию классов **Figure**, **Square**, **Triangle**, **Octagon**, **Factory**; **document.hpp** – содержит реализацию класса **Document**; **main.cpp** – основная программа. **CMakeLists.txt** – содержит код для утилиты cmake, которая проводит сборку программы из исходного кода.

**Содержимое файла figures.hpp**

/\* Ляшун Дмитрий Сергеевич, М8О-207Б-19 \*/

#pragma once

#include <iostream>

#include <cmath>

#include <vector>

struct Point {

double x;

double y;

};

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

return os << "( " << p.x << ", " << p.y << " )";

}

bool operator==(const Point& first, const Point& second) {

return (first.x == second.x && first.y == second.y);

}

class Figure {

public :

Figure(int type) : type(type) {}

virtual void Output() = 0;

int GetType() {

return type;

}

void SetSquare(const double& sq) {

square = sq;

}

double GetSquare() {

return square;

}

friend bool Equals(Figure\* first, Figure\* second);

int type;

double square;

};

class Square : public Figure {

public :

Square(const Point& point1, const Point& point2) : Figure(1), point1(point1), point2(point2) {

double square = abs(point1.x - point2.x);

square = pow(square, 2);

SetSquare(square);

}

virtual void Output() override;

friend bool Equals(Figure\* first, Figure\* second);

Point point1, point2;

};

class Triangle : public Figure {

public :

Triangle(const Point& point1, const Point& point2, const Point& point3) : Figure(2), point1(point1), point2(point2), point3(point3) {

double side = sqrt(pow(point1.x - point2.x, 2) + pow(point1.y - point2.y, 2));

double square = sqrt(3) \* pow(side, 2) / 4;

SetSquare(square);

}

virtual void Output() override;

friend bool Equals(Figure\* first, Figure\* second);

Point point1, point2, point3;

};

class Octagon : public Figure {

public :

Octagon(const Point& center, const double& vec\_x, const double& vec\_y) : Figure(3), center(center), vec\_x(vec\_x), vec\_y(vec\_y) {

double side = pow(vec\_x, 2) + pow(vec\_y, 2);

double square = 4.0 \* sin(acos(-1) / 4.0) \* side;

SetSquare(square);

}

virtual void Output() override;

friend bool Equals(Figure\* first, Figure\* second);

Point center;

double vec\_x, vec\_y;

};

class Factory {

public:

Figure\* Create(int type, const std::vector<double>& p) {

Figure\* ans;

if (type == 1) {

ans = new Square({p[0], p[1]}, {p[2], p[3]});

}

else if (type == 2) {

ans = new Triangle({p[0], p[1]}, {p[2], p[3]}, {p[4], p[5]});

}

else if (type == 3) {

ans = new Octagon({p[0], p[1]}, p[2], p[3]);

}

return ans;

}

};

void Square::Output() {

std::cout << "Квадрат: ";

std::cout << point1 << ", " << point2 << ", S = " << GetSquare() << std::endl;

}

void Triangle::Output() {

std::cout << "Треугольник: ";

std::cout << point1 << ", " << point2 << ", " << point3 << ", S = " << GetSquare() << std::endl;

}

void Octagon::Output() {

std::cout << "Восьмиугольник: ";

std::cout << center << ", ( " << vec\_x << ", " << vec\_y << " )" << ", S = " << GetSquare() << std::endl;

}

bool Equals(Figure\* first, Figure\* second) {

if (first->type != second->type) {

return false;

}

if (first->type == 1) {

Square\* f = dynamic\_cast<Square\*>(first);

Square\* s = dynamic\_cast<Square\*>(second);

return (f->point1 == s->point1 && f->point2 == s->point2);

}

else if (first->type == 2) {

Triangle\* f = dynamic\_cast<Triangle\*>(first);

Triangle\* s = dynamic\_cast<Triangle\*>(second);

return (f->point1 == s->point1 && f->point2 == s->point2 && f->point3 == s->point3);

}

else if (first->type == 3) {

Octagon\* f = dynamic\_cast<Octagon\*>(first);

Octagon\* s = dynamic\_cast<Octagon\*>(second);

return (f->center == s->center && f->vec\_x == s->vec\_x && f->vec\_y == s->vec\_y);

}

}

**Содержимое файла document.hpp**

/\* Ляшун Дмитрий Сергеевич, М8О-207Б-19 \*/

#pragma once

#include <vector>

#include <stack>

#include <stdio.h>

#include <cstdlib>

#include "figures.hpp"

enum OPERATION\_TYPE { INSERT, ERASE };

class Document {

public:

void Insert(Figure\* figure, const int& index);

void Erase(const int& index);

void LoadFromFile(FILE\* file);

void SaveInFile(FILE\* file);

void Undo();

void Print();

void NewDocument(FILE\* file);

bool Empty() {

return data.empty();

}

~Document() {

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

delete data[i];

}

while (!undo.empty()) {

std::pair<Figure\*, int> t = undo.top();

if (t.second == INSERT) {

delete t.first;

}

undo.pop();

}

}

private:

std::vector<Figure\*> data;

Factory adder;

std::stack<std::pair<Figure\*,int> > undo;

};

void Document::Insert(Figure\* figure, const int& index) {

if (index < 0 || index > data.size()) {

throw(std::runtime\_error("Ошибка! Указана некорректная позиция удаления!"));

}

if (data.empty()) {

data.push\_back(figure);

}

else {

data.insert(data.begin() + index, figure);

}

undo.push(std::make\_pair(figure, ERASE));

}

void Document::Erase(const int& index) {

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

throw(std::runtime\_error("Ошибка! Указана некорректная позиция удаления!"));

}

undo.push(std::make\_pair(data[index], INSERT));

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

}

void Document::LoadFromFile(FILE\* file) {

int n;

fscanf(file, "%d", &n);

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

delete data[i];

}

while (!undo.empty()) {

std::pair<Figure\*, int> t = undo.top();

if (t.second == INSERT) {

delete t.first;

}

undo.pop();

}

data.resize(n);

int type;

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

fscanf(file, "%d", &type);

std::vector<double> figuresData;

if (type == 1) {

figuresData.resize(4);

}

else if (type == 2) {

figuresData.resize(6);

}

else if (type == 3) {

figuresData.resize(4);

}

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

fscanf(file, "%lf", &figuresData[j]);

}

data[i] = adder.Create(type, figuresData);

}

}

void Document::SaveInFile(FILE\* file) {

int size = data.size();

fprintf(file, "%d\n", size);

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

fprintf(file, "%d ", data[i]->type);

if (data[i]->type == 1) {

Square\* sq = dynamic\_cast<Square\*>(data[i]);

fprintf(file, "%lf ", sq->point1.x);

fprintf(file, "%lf ", sq->point1.y);

fprintf(file, "%lf ", sq->point2.x);

fprintf(file, "%lf\n", sq->point2.y);

}

else if (data[i]->type == 2) {

Triangle\* tr = dynamic\_cast<Triangle\*>(data[i]);

fprintf(file, "%lf ", tr->point1.x);

fprintf(file, "%lf ", tr->point1.y);

fprintf(file, "%lf ", tr->point2.x);

fprintf(file, "%lf ", tr->point2.y);

fprintf(file, "%lf ", tr->point3.x);

fprintf(file, "%lf\n", tr->point3.y);

}

else if (data[i]->type == 3) {

Octagon\* oc = dynamic\_cast<Octagon\*>(data[i]);

fprintf(file, "%lf ", oc->center.x);

fprintf(file, "%lf ", oc->center.y);

fprintf(file, "%lf ", oc->vec\_x);

fprintf(file, "%lf\n", oc->vec\_y);

}

}

}

void Document::Undo() {

if (undo.empty()) {

return;

}

std::pair<Figure\*, int> oper = undo.top();

undo.pop();

if (oper.second == ERASE) {

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

if (Equals(oper.first, data[i])) {

data.erase(data.begin() + i);

return;

}

}

}

else if (oper.second == INSERT) {

data.push\_back(oper.first);

}

}

void Document::Print() {

if (data.empty()) {

std::cout << "Документ пустой!" << std::endl;

return;

}

std::cout << "Содержимое документа: " << std::endl;

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

data[i]->Output();

}

}

void Document::NewDocument(FILE\* file) {

if (file != NULL) {

SaveInFile(file);

}

data.clear();

while (!undo.empty()) {

undo.pop();

}

}

**Содержимое файла main.cpp**

/\* Ляшун Дмитрий Сергеевич, М8О-207Б-19 \*/

/\* Задание: Спроектировать простейший «графический» векторный редактор. Требование к функционалу редактора: создание нового документа; импорт документа из файла; экспорт документа в файл; создание графического примитива; удаление графического примитива; отображение документа на экране; реализовать операцию undo, отменяющую последнее сделанное действие.

Вариант №24: Фигуры - 8-угольник, треугольник, квадрат. \*/

#include "figures.hpp"

#include "document.hpp"

#include <iostream>

#include <stdio.h>

#include <string>

using namespace std;

int main() {

Document work;

Factory adder;

int oper;

cout << "Список поддерживаемых команд (для вызова необходимо ввести номер): " << endl;

cout << "1 - Создание нового документа" << endl;

cout << "2 - Импорт документа из файла" << endl;

cout << "3 - Экспорт документа в файл" << endl;

cout << "4 - Добавление фигуры" << endl;

cout << "5 - Удаление фигуры" << endl;

cout << "6 - Печать документа" << endl;

cout << "7 - Отмена последнего действия" << endl;

cout << "8 - Выход из программы" << endl;

while (true) {

cout << "> ";

cin >> oper;

switch (oper) {

case 1:

{

if (!work.Empty()) {

cout << "Вы хотите сохранить старый документ в файл перед созданием нового? (y/n)" << endl << "> ";

char ans;

cin >> ans;

if (ans == 'y') {

cout << "Введите имя файла для записи документа > ";

string name;

cin >> name;

FILE\* output = fopen(name.c\_str(), "w");

if (output == NULL) {

cout << "Произошла ошибка при открытии файла!" << endl;

}

else {

work.NewDocument(output);

cout << "Новый документ успешно создан." << endl;

fclose(output);

}

}

else if (ans == 'n') {

work.NewDocument(NULL);

cout << "Новый документ успешно создан." << endl;

}

else {

cout << "Новый документ успешно создан." << endl;

}

}

else {

cout << "Ошибка! Документ пустой!" << endl;

}

break;

}

case 2:

{

cout << "Введите имя файла для импорта документа > ";

string name;

cin >> name;

FILE\* input = fopen(name.c\_str(), "r");

if (input == NULL) {

cout << "Произошла ошибка при открытии файла!" << endl;

}

else {

work.LoadFromFile(input);

cout << "Документ успешно загружен из файла." << endl;

fclose(input);

}

break;

}

case 3:

{

cout << "Введите имя файла для экспорта документа > ";

string name;

cin >> name;

FILE\* output = fopen(name.c\_str(), "w");

if (output == NULL) {

cout << "Произошла ошибка при открытии файла!" << endl;

}

else {

work.SaveInFile(output);

cout << "Документ успешно записан в файл." << endl;

fclose(output);

}

break;

}

case 4:

{

cout << "Введите тип добавляемой фигуры (1 - Квадрат, 2 - Треугольник, 3 - Восьмиугольник) > ";

int type;

cin >> type;

vector<double> data;

if (type == 1) {

cout << "Введите координаты противоположных вершин квадрата > ";

data.resize(4);

}

else if (type == 2) {

cout << "Введите координаты вершин треугольника > ";

data.resize(6);

}

else if (type == 3) {

cout << "Введите координаты центра восьмиугольника и вектора описанной окружности > ";

data.resize(4);

}

else {

cout << "Ошибка! Введен неправильный тип фигуры!" << endl;

break;

}

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

cin >> data[i];

}

Figure\* figure = adder.Create(type, data);

cout << "Введите позицию вставки фигуры > ";

int pos;

cin >> pos;

--pos;

try {

work.Insert(figure, pos);

cout << "Фигура успешно вставлена." << endl;

}

catch (std::exception& e) {

cout << e.what() << endl;

}

break;

}

case 5:

{

cout << "Введите позицию удаляемой фигуры > ";

int pos;

cin >> pos;

--pos;

try {

work.Erase(pos);

cout << "Фигура успешно удалена." << endl;

}

catch (std::exception& e) {

cout << e.what() << endl;

}

break;

}

case 6:

{

work.Print();

break;

}

case 7:

{

work.Undo();

cout << "Последнее действие успешно отменено!" << endl;

break;

}

case 8:

return 0;

default:

cout << "Ошибка! Неправильный ввод!" << endl;

}

}

}

**Содержимое файла CMakeLists.txt**

cmake\_minimum\_required(VERSION 2.8) # Указание необходимой версии CMake

set(SOURCES main.cpp) # присвоение переменной SOURCES значения main.cpp

add\_executable(oop\_exercise\_07 ${SOURCES}) # Компиляция исполняемого файла oop\_exercise\_07

**5. Выводы**

В результате выполнения лабораторной работы я научился проектировать собственные структуры классов. Также я закрепил полученные знания ООП в C++: использования наследования, полиморфизма и т. д.

**Список литературы**

1. Справочник по языку C++ [Электронный ресурс]. URL: [http://www.cplusplus.com/](about:blank) (дата обращения: 10.12.2020).

2. Archived Stack Overflow Documentation [Электронный ресурс]. URL: <https://www.riptutorial.com/> (дата обращения 10.12.2020).