**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 8**

Тема: Асинхронное программирование

Студент: Чернобаев Андрей Александрович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Создать приложение, которое будет считывать из стандартного ввода данные фигур, согласно варианту задания, выводить их характеристики на экран и записывать в файл. Фигуры могут задаваться как своими вершинами, так и другими характеристиками (например, координата центра, количество точек и радиус).

Программа должна:

1. Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;

2. Программа должна создавать классы, соответствующие введенным данных фигур;

3. Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задается параметром командной строки. Например, для буфера размером 10 фигур: **oop\_exercise\_08 10**

4. При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;

5. Обработка должна производиться в отдельном потоке;

6. Реализовать два обработчика, которые должны обрабатывать данные буфера:

a. Вывод информации о фигурах в буфере на экран;

b. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.

7. Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.

8. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.

9. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;

10. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.

11. Реализовать в основном потоке (main) ожидание обработки буфера в потоке-обработчике. Т.е. после отправки буфера на обработку основной поток должен ждать, пока поток обработчик выведет данные на экран и запишет в файл.

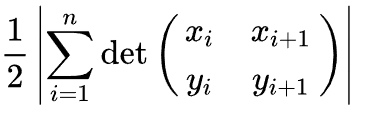
**Варианты заданий (выпуклые равносторонние фигуры вращения):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **Фигура №1** | **Фигура №2** | **Фигура №3** |
| 9. | Треугольник | Квадрат | Прямоугольник |

.

1. Описание программы

Площадь фигур вычисляется с помощью формулы площади Гаусса:



Программа состоит из 10 файлов:

Factory.cpp - реализация класса Factory из Factory.h

Factory.h - реализация фабрики

Figure.h - класс с абстрактным классом фигуры

Состоит из:

* класса Figure
* функции вычисления детерминанта
* функции вычисления площади многоугольника с помощью формулы Гаусса

Handler.cpp - реализация класса Handler из Handler.h

Handler.h - класс Handler

Point.h - заголовочный файл с классом Point, отвечающим за точку

Rectangle.h - заголовочный файл с классом Rectangle, отвечающим за треугольник

Square.h - заголовочный файл с классом Square, отвечающим за квадрат

Triangle.h - заголовочный файл с классом Square, отвечающим за квадрат

1. Набор тестов

из файла **test\_01.txt:**

add

rectangle 0 0 1 0 1 1 0 1

add

square 0 0 2 0 2 2 0 2

add

triangle 0 0 1 1 6 7

add

triangle 0 0 1 1 6 12

add

rectangle 0 0 3 0 3 2 0 2

add

square 0 0 7 0 7 7 0 7

add

rectangle 0 0 4 0 4 3 0 3

exit

Из файла **test\_02.txt:**

add

rectangle 0 0 3 0 3 3 0 3

add

triangle 0 0 1 1 6 12

add

square 0 0 9 0 9 9 0 9

add

triangle 0 0 1 1 6 7

add

rectangle 0 0 5 0 5 7 0 7

add

square 0 0 5 0 5 5 0 5

add

rectangle 0 0 7 0 7 8 0 8

exit

1. Результаты выполнения тестов

|  |  |
| --- | --- |
| input | output |
| add  rectangle 0 0 1 0 1 1 0 1  add  square 0 0 2 0 2 2 0 2  add  triangle 0 0 1 1 6 7  add  triangle 0 0 1 1 6 12  add  rectangle 0 0 3 0 3 2 0 2  add  square 0 0 7 0 7 7 0 7  add  rectangle 0 0 4 0 4 3 0 3  exit | menu  exit  add  was added  was added  was added  was added  was added  was added  was added  rectangle  0 0  1 0  1 1  0 1  area is 1  center is 0.5 0.5  square  0 0  2 0  2 2  0 2  area is 4  center is 1 1  triangle  0 0  1 1  6 7  area is 0.5  center is 2.33333 2.66667  triangle  0 0  1 1  6 12  area is 3  center is 2.33333 4.33333  rectangle  0 0  3 0  3 2  0 2  area is 6  center is 1.5 1  square  0 0  7 0  7 7  0 7  area is 49  center is 3.5 3.5  rectangle  0 0  4 0  4 3  0 3  area is 12  center is 2 1.5 |
| add  rectangle 0 0 3 0 3 3 0 3  add  triangle 0 0 1 1 6 12  add  square 0 0 9 0 9 9 0 9  add  triangle 0 0 1 1 6 7  add  rectangle 0 0 5 0 5 7 0 7  add  square 0 0 5 0 5 5 0 5  add  rectangle 0 0 7 0 7 8 0 8  exit | menu  exit  add  was added  was added  was added  was added  was added  was added  was added  rectangle  0 0  3 0  3 3  0 3  area is 9  center is 1.5 1.5  triangle  0 0  1 1  6 12  area is 3  center is 2.33333 4.33333  square  0 0  9 0  9 9  0 9  area is 81  center is 4.5 4.5  triangle  0 0  1 1  6 7  area is 0.5  center is 2.33333 2.66667  rectangle  0 0  5 0  5 7  0 7  area is 35  center is 2.5 3.5  square  0 0  5 0  5 5  0 5  area is 25  center is 2.5 2.5  rectangle  0 0  7 0  7 8  0 8  area is 56  center is 3.5 4 |

1. Листинг программы

Из файла **Factory.cpp**

std::shared\_ptr<Figure> Factory::figure\_create(std::istream& is) {

std::string name;

is >> name;

if ( name == "rectangle" ) {

return std::shared\_ptr<Figure> (new Rectangle(is));

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

return std::shared\_ptr<Figure> (new Triangle(is));

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

return std::shared\_ptr<Figure> (new Square(is));

} else {

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

}

}

std::shared\_ptr<Figure> Factory::figure\_create\_file(std::ifstream& is) {

std::string name;

is >> name;

if ( name == "rectangle" ) {

return std::shared\_ptr<Figure> (new Rectangle(is));

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

return std::shared\_ptr<Figure> (new Triangle(is));

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

return std::shared\_ptr<Figure> (new Square(is));

} else {

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

}

}

Из файла **Factory.h**

#ifndef OOP7\_FACTORY\_H

#define OOP7\_FACTORY\_H

#include <memory>

#include <iostream>

#include <fstream>

#include <string>

#include "Square.h"

#include "Rectangle.h"

#include "Triangle.h"

class Factory {

public:

static std::shared\_ptr<Figure> figure\_create(std::istream& is);

static std::shared\_ptr<Figure> figure\_create\_file(std::ifstream& is);

};

#include "Factory.cpp"

#endif //OOP7\_FACTORY\_H

Из файла **Figure.h**

#ifndef OOP7\_FIGURE\_H

#define OOP7\_FIGURE\_H

#include <iostream>

#include <vector>

#include <fstream>

#include "Point.h"

double get\_det(const double x1, const double y1,

const double x2, const double y2){

return x1 \* y2 - x2 \* y1;

}

double gauss\_area(const std::vector<Point>& nodes){

double out = 0.0;

auto count = nodes.size();

for(size\_t i = 0; i < (count - 1); i++){

out += get\_det(static\_cast<double>(nodes[i].x),static\_cast<double>(nodes[i].y),

static\_cast<double>(nodes[i+1].x),static\_cast<double>(nodes[i+1].y));

}

out += get\_det(static\_cast<double>(nodes[count-1].x),static\_cast<double>(nodes[count-1].y),

static\_cast<double>(nodes[0].x),static\_cast<double>(nodes[0].y));

return fabs(0.5 \* out);

}

class Figure {

public:

virtual Point center() const = 0;

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

virtual void print\_file(std::ofstream&) const = 0 ;

virtual double area() const = 0;

};

#endif //OOP7\_FIGURE\_H

Из файла **Handler.cpp**

Handler::Handler(size\_t max\_count\_){

max\_count = max\_count\_;

running = true;

thread = std::thread(print,this);

}

Handler::~Handler(){

running = false;

cv.notify\_one();

thread.join();

}

void Handler::print(Handler\* h) {

while(h->running){

std::unique\_lock<std::mutex> ul(h->mutex);

h->cv.wait(ul,[h](){

return h->running == false || h->is\_full();

});

for(auto& el : h->handlers){

el(h->figures);

}

h->figures.clear();

ul.unlock();

h->cv.notify\_one();

}

}

void Handler::add\_to\_handler(std::function<void(std::vector<std::shared\_ptr<Figure>>&)>&& func){

handlers.push\_back(func);

}

bool Handler::is\_full(){

return figures.size() == max\_count;

}

void Handler::push(std::shared\_ptr<Figure> el){

figures.push\_back(el);

std::unique\_lock<std::mutex> ul(mutex);

if (is\_full()){

cv.notify\_one();

cv.wait(ul,[this](){

return figures.empty();

});

}

}

Из файла **Handler.h**

#ifndef OOP8\_HANDLER\_H

#define OOP8\_HANDLER\_H

#include <condition\_variable>

#include <mutex>

#include <shared\_mutex>

#include <thread>

#include <functional>

#include <atomic>

#include <vector>

#include <iostream>

#include <memory>

#include "Figure.h"

class Handler{

public:

private:

size\_t max\_count = 0;

std::mutex mutex;

std::thread thread;

std::condition\_variable cv;

std::vector<std::shared\_ptr<Figure>> figures;

std::vector<std::function<void(std::vector<std::shared\_ptr<Figure>>&)>> handlers;

public:

bool running;

Handler(size\_t max\_count\_);

~Handler();

static void print(Handler\* h);

void add\_to\_handler(std::function<void(std::vector<std::shared\_ptr<Figure>>&)>&& func);

bool is\_full();

void push(std::shared\_ptr<Figure> el);

};

#include "Handler.cpp"

#endif //OOP8\_HANDLER\_H

Из файла **main.cpp**

#include <iostream>

#include <string>

#include <cmath>

#include "Handler.h"

#include "Factory.h"

const int ARGUMENT\_ERROR = -1;

void menu(){

std::cout << "menu" << std::endl;

std::cout << "exit" << std::endl;

std::cout << "add" << std::endl;

}

int main(int argc, char\*\* argv) {

if (argc != 2){

std::cout << "buffer size error" << std::endl;

return ARGUMENT\_ERROR;

}

if (std::stol(argv[1]) <= 0){

std::cout << "buffer size error" << std::endl;

return ARGUMENT\_ERROR;

}

const size\_t buffer\_size = std::stol(argv[1]);

Handler handler(buffer\_size);

handler.add\_to\_handler([](std::vector<std::shared\_ptr<Figure>>& figures) {

for (const auto& figure : figures) {

figure->print(std::cout);

}

});

static int file\_index = 0;

handler.add\_to\_handler([](std::vector<std::shared\_ptr<Figure>>& figures) {

std::ofstream file(std::to\_string(file\_index) + ".txt");

for (const auto& items : figures) {

items->print(file);

}

file.close();

file\_index++;

});

menu();

std::string cmd;

while (true) {

std::cin >> cmd;

if (cmd == "menu") {

menu();

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

break;

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

handler.push(Factory::figure\_create(std::cin));

std::cout << "was added" << std::endl;

}

}

return 0;

}

Из файла **Point.h**

#ifndef OOP7\_POINT\_H

#define OOP7\_POINT\_H

#include <iostream>

class Point {

public:

double x, y;

Point() = default;

Point (double f,double s) {

x = f;

y = s;

}

};

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;

}

#endif //OOP7\_POINT\_H

Из файла **Rectangle.h**

#ifndef OOP7\_RECTANGLE\_H

#define OOP7\_RECTANGLE\_H

#include <cmath>

#include "Point.h"

#include "Figure.h"

class Rectangle : public Figure {

public:

Point a1, a2, a3, a4;

Rectangle(std::istream& is) {

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

}

Point center() const override{

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 print(std::ostream& os) const override{

os << "rectangle" << std::endl;

os << a1 << std::endl;

os << a2 << std::endl;

os << a3 << std::endl;

os << a4 << std::endl;

os << "area is " << area() << std::endl;

os << "center is " << center() << std::endl;

}

void print\_file(std::ofstream& of) const override{

of << "rectangle" << std::endl;

of << a1 << std::endl;

of << a2 << std::endl;

of << a3 << std::endl;

of << a4 << std::endl;

of << "area is " << area() << std::endl;

of << "center is " << center() << std::endl;

}

double area() const override{

std::vector<Point> points = {a1,a2,a3,a4};

return gauss\_area(points);

}

};

#endif //OOP7\_RECTANGLE\_H

Из файла **Square.h**

#ifndef OOP7\_SQUARE\_H

#define OOP7\_SQUARE\_H

#include <cmath>

#include "Point.h"

#include "Figure.h"

class Square : public Figure {

public:

Point a1, a2, a3, a4;

Square(std::istream &is) {

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

}

Point center() const override{

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 print(std::ostream &os) const override{

os << "square" << std::endl;

os << a1 << std::endl;

os << a2 << std::endl;

os << a3 << std::endl;

os << a4 << std::endl;

os << "area is " << area() << std::endl;

os << "center is " << center() << std::endl;

}

void print\_file(std::ofstream &of) const override{

of << "square" << std::endl;

of << a1 << std::endl;

of << a2 << std::endl;

of << a3 << std::endl;

of << a4 << std::endl;

of << "area is " << area() << std::endl;

of << "center is " << center() << std::endl;

}

double area() const override{

std::vector<Point> points = {a1,a2,a3,a4};

return gauss\_area(points);

}

};

#endif //OOP7\_SQUARE\_H

Из файла **Triangle.h**

#ifndef OOP7\_TRIANGLE\_H

#define OOP7\_TRIANGLE\_H

#include <cmath>

#include <iostream>

#include "Point.h"

#include "Figure.h"

class Triangle : public Figure{

public:

Point a1, a2, a3;

Triangle(std::istream& is) {

is >> a1 >> a2 >> a3;

}

Point center() const override{

double x,y;

x = (a1.x + a2.x + a3.x) / 3;

y = (a1.y + a2.y + a3.y) / 3;

Point p(x,y);

return p;

}

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

os << "triangle " << std::endl;

os << a1 << std::endl;

os << a2 << std::endl;

os << a3 << std::endl;

os << "area is " << area() << std::endl;

os << "center is " << center() << std::endl;

}

void print\_file(std::ofstream &of) const override{

of << "triangle " << std::endl;

of << a1 << std::endl;

of << a2 << std::endl;

of << a3 << std::endl;

of << "area is " << area() << std::endl;

of << "center is " << center() << std::endl;

}

double area() const override{

std::vector<Point> points = {a1,a2,a3};

return gauss\_area(points);

}

};

#endif //OOP7\_TRIANGLE\_H

1. Выводы

Я научился на практике использовать многопоточное и асинхронное программирование в С++, повторил лямбда-функции.

1. Список использованных источников

1 std::thread [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/thread/thread

2 Templates [Электронный ресурс].

URL: https://www.cplusplus.com/doc/oldtutorial/templates/

3 std::function [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/utility/functional/function

4 std::logic\_error [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/error/logic\_error

5 std::mutex [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/thread/mutex