Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа**

**по курсу «ООП»**

**Тема:**

**Асинхронное программирование.**

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

Москва

2019

**1. Код на C++:**

figure.hpp

#ifndef D\_FIGURE\_HPP\_

#define D\_FIGURE\_HPP\_

#include <iostream>

#include "point.hpp"

struct figure {

virtual point center() const = 0;

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

virtual double area() const = 0;

virtual ~figure() {}

};

#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 operator+ (point p1, point p2) {

point p;

p.x = p1.x + p2.x;

p.y = p1.y + p2.y;

return p;

}

point& operator/ (point& p, int num) {

p.x = p.x / num;

p.y = p.y / num;

return p;

}

point.hpp

#ifndef D\_POINT\_HPP\_

#define D\_POINT\_HPP\_

#include <iostream>

struct point {

double x, y;

};

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

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

point operator+ (point p1, point p2);

point& operator/ (point& p, int num);

#endif

pentagon.cpp

#include <iostream>

#include <cmath>

#include "pentagon.hpp"

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

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

}

point pentagon::center() const {

point result;

result = a1 + a2 + a3 + a4 + a5;

result = result / 5;

return result;

}

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

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

}

double pentagon::area() const {

double s1 = (a2.x \* a3.y - a1.x \* a3.y - a2.x \* a1.y - a3.x \* a2.y + a1.x \* a2.y + a1.y \* a3.x) / 2;

double s2 = (a3.x \* a4.y - a1.x \* a4.y - a3.x \* a1.y - a4.x \* a3.y + a1.x \* a3.y + a1.y \* a4.x) / 2;

double s3 = (a4.x \* a5.y - a1.x \* a5.y - a4.x \* a1.y - a5.x \* a4.y + a1.x \* a4.y + a1.y \* a5.x) / 2;

double result = s1 + s2 + s3;

if (result >= 0) {

return result;

} else {

return -result;

}

}

pentagon.hpp

#ifndef D\_PENTAGON\_HPP\_

#define D\_PENTAGON\_HPP\_

#include <iostream>

#include "figure.hpp"

struct pentagon : public figure

{

pentagon(std::istream& is);

point center() const override;

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

double area() const override;

private:

point a1, a2, a3, a4, a5;

};

#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 {

point result;

result = a1 + a2 + a3 + a4;

result = result / 4;

return result;

}

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;

}

suba.hpp

#ifndef D\_SUB\_HPP\_

#define D\_SUB\_HPP\_ 1

#include <memory>

#include <vector>

#include "figure.hpp"

#include "rhombus.hpp"

#include "pentagon.hpp"

#include "hexagon.hpp"

class sub {

public:

virtual void Print(std::vector<std::shared\_ptr<figure>>& v) = 0;

};

#endif // D\_SUB\_HPP\_

rhombus.hpp

#ifndef D\_RHOMBUS\_HPP\_

#define D\_RHOMBUS\_HPP\_

#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);

point center() const override;

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

double area() const override;

private:

point a1, a2, a3, a4;

};

#endif

rhombus.cpp

heagon.cpp

#include "hexagon.hpp"

#include <iostream>

#include <cmath>

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

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

}

point hexagon::center() const {

point result;

result = (a1 + a2 + a3 + a4 + a5 + a6);

result = result / 6;

return result;

}

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

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

}

double hexagon::area() const {

double s1 = (a2.x \* a3.y - a1.x \* a3.y - a2.x \* a1.y - a3.x \* a2.y + a1.x \* a2.y + a1.y \* a3.x) / 2;

double s2 = (a3.x \* a4.y - a1.x \* a4.y - a3.x \* a1.y - a4.x \* a3.y + a1.x \* a3.y + a1.y \* a4.x) / 2;

double s3 = (a4.x \* a5.y - a1.x \* a5.y - a4.x \* a1.y - a5.x \* a4.y + a1.x \* a4.y + a1.y \* a5.x) / 2;

double s4 = (a5.x \* a6.y - a1.x \* a6.y - a5.x \* a1.y - a6.x \* a5.y + a1.x \* a5.y + a1.y \* a6.x) / 2;

double result = s1 + s2 + s3 + s4;

if (result >= 0) {

return result;

} else {

return -result;

}

}

factory.hpp

#ifndef D\_FACTORY\_HPP\_

#define D\_FACTORY\_HPP\_

#include <iostream>

#include <string>

#include <memory>

#include "figure.hpp"

#include "pentagon.hpp"

#include "hexagon.hpp"

#include "rhombus.hpp"

class factory {

public:

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

int command;

std::cin >> command;

if (command == 3) {

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

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

return f;

} else if (command == 2) {

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

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

return f;

} else if (command == 1) {

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

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

return f;

} else {

throw std::logic\_error("ERROR\n");

}

}

};

#endif //D\_FACTORY\_HPP\_

subscriber.hpp

#ifndef D\_SUBSCRIBER\_HPP\_

#define D\_SUBSCRIBER\_HPP\_

#include <iostream>

#include <string>

#include <fstream>

#include "subs.hpp"

class PrintC : public sub {

public:

void Print(std::vector<std::shared\_ptr<figure>>& v) override {

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

v[i]->print(std::cout);

}

}

};

class PrintF : public sub {

private:

int count = 1;

public:

void Print(std::vector<std::shared\_ptr<figure>>& v) override {

std::string filename = "";

filename = "file\_" + std::to\_string(count) + ".txt";

count += 1;

std::ofstream file(filename);

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

v[i]->print(file);

}

}

};

#endif //D\_SUBSCRIBER\_HPP\_

main.cpp

#include <iostream>

#include <vector>

#include <memory>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include "factory.hpp"

#include "subscriber.hpp"

#include "rhombus.hpp"

#include "pentagon.hpp"

#include "hexagon.hpp"

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

if (argc != 2) {

std::cout << "./lab8 N\n";

return 0;

}

unsigned int BufSize = std::atoi(argv[1]);

std::vector<std::shared\_ptr<figure>> f;

int command;

factory factory;

bool done = false;

std::condition\_variable rd;

std::condition\_variable hd;

std::mutex mutex;

int in = 1;

std::vector<std::shared\_ptr<sub>> s;

s.push\_back(std::make\_shared<PrintC>());

s.push\_back(std::make\_shared<PrintF>());

std::thread sub([&]() {

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

while (!done) {

rd.wait(sub\_lock);

if (done) {

hd.notify\_all();

break;

}

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

s[i]->Print(f);

}

in++;

f.resize(0);

hd.notify\_all();

}

});

while(command != 0) {

std::cin >> command;

if (command != 0) {

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

for (unsigned int i = 0; i < BufSize; i++) {

f.push\_back(factory.FigureCreate(std::cin));

if (f.size() == BufSize) {

std::cout << "Buffer is full!\n";

}

}

rd.notify\_all();

hd.wait(main\_lock);

} else {

return 0;

}

}

done = true;

rd.notify\_all();

sub.join();

return 0;

return 0;

}

hexagon.hpp

#ifndef D\_HEXAGON\_H\_

#define D\_HEXAGON\_H\_

#include <iostream>

#include "figure.hpp"

struct hexagon : figure

{

hexagon(std::istream& is);

point center() const override;

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

double area() const override;

private:

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

};

#endif

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

https://github.com/keoni02032/oop\_exercise\_08

**3. Набор testcases:**

1

1

1 1 1 1 1 1 1 1

2

1 1 2 2 3 3 4 4 5 5

3

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

0

**4.Результаты выполнения программы:**

**sergey@sergey-RedmiBook-14:~/labs/OOP/lab8/lab8v1$ make   
[100%] Built target lab8   
sergey@sergey-RedmiBook-14:~/labs/OOP/lab8/lab8v1$ ./lab8 3   
1   
1   
rhombus:   
1 1 1 1 1 1 1 1   
2   
pentagon:   
1 1 2 2 3 3 4 4 5 5       
3   
hexagon:   
1 1 2 2 3 3 4 4 5 5 6 6   
Buffer is full!   
a1 = 1 1 a2 = 1 1 a3 = 1 1 a4 = 1 1   
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   
0   
terminate called without an active exception   
Аварийный останов (стек памяти сброшен на диск)   
sergey@sergey-RedmiBook-14:~/labs/OOP/lab8/lab8v1$ ls   
CMakeCache.txt  cmake\_install.cmake  factory.hpp  file\_1.txt   hexagon.hpp  main.cpp  pentagon.cpp  point.cpp  report.docx  rhombus.hpp     subs.hpp   
CMakeFiles      CMakeLists.txt       figure.hpp   hexagon.cpp  lab8         Makefile  pentagon.hpp  point.hpp  rhombus.cpp  subscriber.hpp  test\_01.test   
sergey@sergey-RedmiBook-14:~/labs/OOP/lab8/lab8v1$ cat file\_1.txt    
a1 = 1 1 a2 = 1 1 a3 = 1 1 a4 = 1 1   
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   
sergey@sergey-RedmiBook-14:~/labs/OOP/lab8/lab8v1$**

**5. Объяснение результатов работы программы:**

При запуске программы необходимо нажать любую цифру. После ввода цифры предоставляется выбор из трех фигур: при нажатии «1» необходимо ввести координаты точек ромба, при нажатии «2» нужно вводить координаты пятиугольника, при нажатии «3» вводятся координаты шестиугольника. Так же пред запуском программы необходимо через пробел относительно ./lab8 ввести объем буфера.

**6. Вывод:**

В данной лабораторной работе я ознакомился с темой асинхронного программирования, также получил практические навыки в параллельной обработке данных и ,кроме того, освоил функции по синхронизации потоков.