**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: Айрапетова Евгения Ашотовна

Группа: 80-206

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.

.

1. Описание программы

Для решения данной задачи был реализован виртуальный класс Figure и классы-наследники Square (для квадрата), Rectangle (для прямоугольника) и Trapezoid (для трапеции). Фигуры записываются в виде их типа и координат вершин. Вершины проверяются на корректность и записываются в вектор.

Перед началом ввода фигур на вход поступает размер буфера, при накоплении которого запускаются обработчики, а буфер очищается. Обработка производится в классе Document. При переполнении буфера основной поток замораживается, пока не произойдет запись.

Обработчик тестов и классы фигур соответствуют предыдущей лабораторной работе.

1. Набор тестов

4  
Square 0 0 0 1 1 1 1 0  
Rectangle 1 1 1 3 3 3 3 1  
Trapezoid 0 0 3 4 6 4 9 0  
Square -1 0 0 1 1 0 0 -1  
Square 0 0 0 1 1 1 1 0  
Rectangle 1 1 1 3 3 3 3 1  
Trapezoid 0 0 3 4 6 4 9 0  
Square -1 0 0 1 1 0 0 -1  
Square 0 0 0 1 1 1 1 0  
Rectangle 1 1 1 3 3 3 3 1  
Trapezoid 0 0 3 4 6 4 9 0  
Square -1 0 0 1 1 0 0 -1

1. Результаты выполнения тестов

1:  
Square:  
(0, 0), (0, 1), (1, 1), (1, 0)  
Rectangle:  
(1, 1), (1, 3), (3, 3), (3, 1)  
Trapezoid:  
(0, 0), (3, 4), (6, 4), (9, 0)  
Square:  
(-1, 0), (0, 1), (1, 0), (0, -1)  
2:  
Square:  
(0, 0), (0, 1), (1, 1), (1, 0)  
Rectangle:  
(1, 1), (1, 3), (3, 3), (3, 1)  
Trapezoid:  
(0, 0), (3, 4), (6, 4), (9, 0)  
Square:  
(-1, 0), (0, 1), (1, 0), (0, -1)  
3:  
Square:  
(0, 0), (0, 1), (1, 1), (1, 0)  
Rectangle:  
(1, 1), (1, 3), (3, 3), (3, 1)  
Trapezoid:  
(0, 0), (3, 4), (6, 4), (9, 0)  
Square:  
(-1, 0), (0, 1), (1, 0), (0, -1)

1. Листинг программы  
   // Айрапетова Евгения М8О-206Б-19

// Вариант 34: квадрат, прямоугольник, трапеция

#include <iostream>

#include <fstream>

#include <exception>

#include <list>

#include <vector>

#include <string>

#include <utility>

#include <memory>

#include <future>

#include <condition\_variable>

using namespace std;

struct Vertex {

double x = 0;

double y = 0;

void Print() const {

cout << "(" << x << ", " << y << ")";

}

};

istream& operator>>(istream& input, Vertex& vertex) {

input >> vertex.x >> vertex.y;

return input;

}

ostream& operator<<(ostream& output, const Vertex& vertex) {

output << vertex.x << " " << vertex.y;

return output;

}

enum class FigureType {

Square,

Rectangle,

Trapezoid

};

string TypeToString(FigureType type) {

switch (type) {

case FigureType::Trapezoid:

return "Trapezoid";

case FigureType::Square:

return "Square";

case FigureType::Rectangle:

return "Rectangle";

default:

throw runtime\_error("Undefined figure type");

}

}

class Figure {

public:

Figure() = default;

Figure(vector<Vertex> vec) : stats\_(vec) {}

void Print() const {

cout << TypeToString(GetType()) << ":\n ";

bool is\_first = true;

for (const Vertex& v : stats\_) {

if (!is\_first) {

cout << ", ";

}

is\_first = false;

v.Print();

}

cout << "\n";

}

virtual FigureType GetType() const = 0;

virtual ~Figure() {}

friend istream& operator>>(istream& input, Figure& rb);

friend ostream& operator<<(ostream& output, const Figure& rb);

protected:

vector<Vertex> stats\_;

};

istream& operator>>(istream& input, Figure& figure) {

for (Vertex& v : figure.stats\_) {

input >> v;

}

return input;

}

ostream& operator<<(ostream& output, const Figure& figure) {

output << static\_cast<int>(figure.GetType()) << " ";

for (const Vertex& v : figure.stats\_) {

output << v << " ";

}

return output;

}

class Square : public Figure {

public:

Square() : Figure(vector<Vertex>(4)) {}

Square(Vertex a, Vertex b, Vertex c, Vertex d) : Figure({ a, b, c ,d }) {}

virtual FigureType GetType() const override { return FigureType::Square; }

};

class Rectangle : public Figure {

public:

Rectangle() : Figure(vector<Vertex>(4)) {}

Rectangle(Vertex a, Vertex b, Vertex c, Vertex d) : Figure({ a, b, c, d }) {}

virtual FigureType GetType() const override {

return FigureType::Rectangle; }

};

class Trapezoid : public Figure {

public:

Trapezoid() : Figure(vector<Vertex>(4)) {}

Trapezoid(Vertex a, Vertex b, Vertex c, Vertex d) : Figure({ a, b, c, d }) {}

virtual FigureType GetType() const override { return FigureType::Trapezoid; }

};

class Document {

public:

Document() {

fut\_con = async([&]() {Logger(); });

}

void Set\_size(size\_t s) { critical\_size = s; }

~Document() {

production\_stopped = true;

cv\_consumption.notify\_all();

fut\_con.get();

}

void Export() { //запись в файл

static int i = 0;

++i;

ofstream out("log" + to\_string(i) + ".txt");

for (const auto& ptr : data) {

out << \*ptr << "\n";

}

out << "\n";

}

void Add(shared\_ptr<Figure> figure\_ptr) {

std::unique\_lock<std::mutex> lock(m);

cv\_production.wait(lock, [&] { return data.size() < critical\_size; });

data.push\_back(move(figure\_ptr));

cv\_consumption.notify\_all();

}

void Print() const {

static int i = 0;

++i;

cout << i << ":\n";

for (const auto& ptr : data) {

ptr->Print();

}

cout << "\n";

}

private:

list<shared\_ptr<Figure>> data;

size\_t critical\_size = 3;

mutex m;

condition\_variable cv\_production;

condition\_variable cv\_consumption;

future<void> fut\_con;

bool production\_stopped = false;

void Logger() {

while (!production\_stopped) {

std::unique\_lock<std::mutex> lock(m);

cv\_consumption.wait(lock, [&] { return data.size() >= critical\_size || production\_stopped; });

if (!data.empty()) {

Print();

Export();

data.clear();

}

cv\_production.notify\_all();

}

}

};

double lenght(Vertex a, Vertex b) {

return sqrt((b.x - a.x) \* (b.x - a.x) + ((b.y - a.y) \* (b.y - a.y)));

}

int CorrectInput(Vertex& a, Vertex& b, Vertex& c, Vertex& d, int fig) {

if (((abs(a.x - c.x) < numeric\_limits<double>::epsilon()) &&

(abs(a.y - c.y) < numeric\_limits<double>::epsilon())) ||

((abs(b.x - d.x) < numeric\_limits<double>::epsilon()) &&

(abs(b.y - d.y) < numeric\_limits<double>::epsilon()))) {

throw runtime\_error("ERROR INPUT\n");

}

double len\_ab = lenght(a, b);

double len\_bc = lenght(b, c);

double len\_cd = lenght(c, d);

double len\_da = lenght(d, a);

double len\_ac = lenght(a, c);

double len\_bd = lenght(b, d);

if (fig == 1) {

if (abs(len\_cd - len\_ab) < numeric\_limits<double>::epsilon() &&

abs(len\_cd - len\_bc) < numeric\_limits<double>::epsilon() &&

abs(len\_cd - len\_da) < numeric\_limits<double>::epsilon() &&

abs(len\_bd - len\_ac) < numeric\_limits<double>::epsilon()) {

return 0;

}

throw runtime\_error("ERROR INPUT\n");

}

//проверка корректности прямоугольника: равенство противоположных сторон и диагоналей

if (fig == 2) {

if (abs(len\_ab - len\_cd) < numeric\_limits<double>::epsilon() && abs(len\_bc - len\_da) < numeric\_limits<double>::epsilon() &&

abs(len\_ac - len\_bd) < numeric\_limits<double>::epsilon()) {

return 0;

}

throw runtime\_error("ERROR INPUT\n");

}

//проверка корректности равнобедренной трапеции: равенство боковых сторон и диагоналей

if (fig == 3) {

if (abs(len\_ab - len\_cd) < numeric\_limits<double>::epsilon() &&

abs(len\_ac - len\_bd) < numeric\_limits<double>::epsilon()) {

return 0;

}

throw runtime\_error("ERROR INPUT\n");

}

return 0;

}

void InputError() {

throw runtime\_error("ERROR INPUT\n");

}

int main() {

try {

Document doc;

string cmd;

size\_t size = 3;

cin >> size;

doc.Set\_size(size);

while (cin >> cmd) {

if (cmd == "Square") {

Vertex a, b, c, d;

if (!(cin >> a >> b >> c >> d)) {

InputError();

}

CorrectInput(a, b, c, d, 1);

doc.Add(make\_shared<Square>(a, b, c, d));

}

else if (cmd == "Rectangle") {

Vertex a, b, c, d;

if (!(cin >> a >> b >> c >> d)) {

InputError();

}

CorrectInput(a, b, c, d, 2);

doc.Add(make\_shared<Rectangle>(a, b, c, d));

}

else if (cmd == "Trapezoid") {

Vertex a, b, c, d;

if (!(cin >> a >> b >> c >> d)) {

InputError();

}

CorrectInput(a, b, c, d, 3);

doc.Add(make\_shared<Trapezoid>(a, b, c, d));

}

else {

cout << cmd << " is not a command\n";

}

}

}

catch (exception & ex) {

cerr << ex.what();

}

return 0;

}

1. Вывод

В процессе выполнения данной лабораторной работы я научилась работать с потоками и распараллеливать обработку данных.

Список литературы

1. Справочник по языку C++ [Электронный ресурс]. URL: <https://en.cppreference.com/w/> (дата обращения: 18.01.2020).
2. “Распараллеливание задач” [Электронный ресурс]. URL: <https://habr.com/ru/post/104078/> (дата обращения: 19.01.2020).