**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: Почечура Артемий Андреевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

**1.** **Постановка задачи**

**Цель:**

* Знакомство с асинхронным программированием;
* Получение практических навыков в параллельной обработке данных;
* Получение практических навыков в синхронизации потоков;

**Порядок выполнения работы**

1. Ознакомиться с теоретическим материалом.
2. Получить у преподавателя вариант задания.
3. Реализовать задание своего варианта в соответствии с поставленными требованиями.
4. Подготовить тестовые наборы данных.
5. Создать репозиторий на GitHub.
6. Отправить файлы лабораторной работы в репозиторий.
7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

Вариант: 3;

Фигуры: прямоугольник, трапеция, ромб.

**2.** **Описание программы**

Сначала мы указываем предельное количество фигур, при котором происходит выгрузка данных в файл. Далее на вход программе поступает информация о фигурах – координаты их вершин. Координаты проверяются на то, можно ли из данных точек составить фигуру, для которой мы вводим вершины. Когда количество фигур достигает предельного значения, создаётся файл, в который записываются все фигуры. Затем буфер очищается, и мы снова можем записывать фигуры, пока опять не достигнем предельного значения. Если завершить программу до того, как значение будет достигнуто, то запишутся все те фигуры, которые были в буфере на момент выхода из программы. Запись каждый раз происходит в новый файл.

3. **Набор тестов и результаты их исполнения**

Тест 1:

Enter max size of doc: 2

0 - exit programm

1 - input Rhombus

2 - input Trapezoid

3 - input Rectangle

1

Enter four pair of coord vertex of rhombus: 0 0 0 1 1 1 1 0

3

Enter four pair of coord vertex of rectangle: 0 0 0 1 1 1 1 0

Add in document log1:

Rhombus: ((0; 0), (0; 1), (1; 1), (1; 0))

Rectangle: ((0; 0), (0; 1), (1; 1), (1; 0))

0

Тест 2:

Enter max size of doc: 0

Wrong values

Enter max size of doc: ad

Wrong values

Enter max size of doc: 1

0 - exit programm

1 - input Rhombus

2 - input Trapezoid

3 - input Rectangle

1

Enter four pair of coord vertex of rhombus: 0 0 1 1 2 1 3 0

Is not a rhombus

2

Enter four pair of coord vertex of trapezoid: 0 0 0 1 1 1 1 0

Is not a trapezoid

0

Тест 3:

Enter max size of doc: 2

0 - exit programm

1 - input Rhombus

2 - input Trapezoid

3 - input Rectangle

1

Enter four pair of coord vertex of rhombus: 0 0 0 1 1 1 1 0

2

Enter four pair of coord vertex of trapezoid: 0 0 1 1 2 1 3 0

Add in document log1:

Rhombus: ((0; 0), (0; 1), (1; 1), (1; 0))

Trapezoid: ((0; 0), (1; 1), (2; 1), (3; 0))

3

Enter four pair of coord vertex of rectangle: 0 0 0 1 1 1 1 0

0

Add in document log2:

Rectangle: ((0; 0), (0; 1), (1; 1), (1; 0))

**4.** **Листинг программы**

Файл main.cpp

// Почечура Артемий, гр. М8О-206Б-20

// ЛР №8, вариант 3

// Фигуры: квадрат, прямоугольник, трапеция

/\* 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.

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.\*/

#include <iostream>

#include <fstream>

#include <exception>

#include <list>

#include <vector>

#include <cmath>

#include <string>

#include <utility>

#include <memory>

#include <future>

#include <mutex>

#include <condition\_variable>

double PI = 3.14159265;

using namespace std;

struct Vertex {

double x = 0;

double y = 0;

void print() {

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

}

};

Vertex operator- (Vertex a, Vertex b) {

return { a.x - b.x, a.y - b.y };

}

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

in >> vertex.x >> vertex.y;

return in;

}

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

out << vertex.x << " " << vertex.y;

return out;

}

enum class type\_figure {

unknown = -1,

Rhombus = 0,

Trapezoid = 1,

Rectangle = 2,

};

string type\_to\_string(type\_figure type) {

if (type == type\_figure::Rhombus) {

return "Square";

}

else if (type == type\_figure::Trapezoid) {

return "Trapezoid";

}

else if (type == type\_figure::Rectangle) {

return "Rectangle";

}

else {

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

}

}

class Figure {

public:

vector<Vertex> data;

Figure() = default;

Figure(vector<Vertex> vec) : data(vec) {}

void print() {

cout << type\_to\_string(get\_type\_figure()) << ": (";

bool is\_first = true;

for (Vertex& v : data) {

if (!is\_first) {

cout << ", ";

}

is\_first = false;

v.print();

}

cout << ")\n";

}

virtual type\_figure get\_type\_figure() {

return type\_figure::unknown;

}

};

class Rhombus : public Figure {

public:

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

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

Rhombus(vector<Vertex> v) {

if (v.size() != 4) {

throw invalid\_argument("Incorrect number of vertex of rhombus");

}

data = v;

}

virtual type\_figure get\_type\_figure() { return type\_figure::Rhombus; }

};

class Rectangle : public Figure {

public:

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

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

Rectangle(vector<Vertex> v) {

if (v.size() != 4) {

throw invalid\_argument("Incorrect number of vertex of Rectangle");

}

data = v;

}

virtual type\_figure get\_type\_figure() { return type\_figure::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 }) {}

Trapezoid(vector<Vertex> v) {

if (v.size() != 4) {

throw invalid\_argument("Incorrect number of vertex of Trapezoid");

}

data = v;

}

virtual type\_figure get\_type\_figure() { return type\_figure::Trapezoid; }

};

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

for (Vertex& v : figure.data) {

input >> v;

}

return input;

}

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

output << static\_cast<int>(figure.get\_type\_figure()) << " ";

for (const Vertex& v : figure.data) {

output << v << " ";

}

return output;

}

enum class type\_of\_action {

unknown,

add,

remove,

};

struct Action {

type\_of\_action type = type\_of\_action::unknown;

shared\_ptr<Figure> figure\_ptr;

};

class Factory {

public:

list<shared\_ptr<Figure>> docs;

size\_t max\_size;

mutex mut;

condition\_variable cv\_prod;

condition\_variable cv\_cons;

future<void> fut;

bool prod\_stopped = false;

Factory(size\_t mx) {

max\_size = mx;

fut = async([&]() {logger(); });

}

Factory() {

max\_size = 3;

fut = async([&]() {logger(); });

}

~Factory() {

prod\_stopped = true;

cv\_cons.notify\_all();

fut.get();

}

void inload() {

static int i = 0;

i++;

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

if (!out) {

throw invalid\_argument("Incorrect filename");

}

for (auto& ptr : docs) {

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

}

out << "\n";

}

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

unique\_lock<mutex> lock(mut);

cv\_prod.wait(lock, [&] { return docs.size() < max\_size; });

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

cv\_cons.notify\_all();

}

void print() {

cout << "Add in document log";

static int i = 0;

++i;

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

for (const auto& ptr : docs) {

ptr->print();

}

}

void logger() {

while (!prod\_stopped) {

unique\_lock<mutex> lock(mut);

cv\_cons.wait(lock, [&] { return docs.size() >= max\_size || prod\_stopped; });

if (!docs.empty()) {

print();

inload();

docs.clear();

}

cv\_prod.notify\_all();

}

}

};

void help() {

cout << "0 - exit programm\n";

cout << "1 - input Rhombus\n";

cout << "2 - input Trapezoid\n";

cout << "3 - input Rectangle\n";

}

double side\_2pow(Vertex a, Vertex b) {

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

}

double cross(Vertex a, Vertex b) {

return a.x \* b.y - b.x \* a.y;

}

bool is\_rhombus(vector<Vertex>& coord) {

double a = side\_2pow(coord[0], coord[1]);

double b = side\_2pow(coord[1], coord[2]);

double c = side\_2pow(coord[2], coord[3]);

double d = side\_2pow(coord[3], coord[0]);

if ((a == b) && (b == c) && (c == d) && (d == a)) {

return true;

}

return false;

}

bool is\_rectangle(vector<Vertex>& coord) {

double a = side\_2pow(coord[0], coord[1]);

double b = side\_2pow(coord[1], coord[2]);

double c = side\_2pow(coord[2], coord[3]);

double d = side\_2pow(coord[3], coord[0]);

if ((a == c) && (b == d)) {

for (int i = 0; i < 3; i++) {

if (((coord[(i + 1) % 4].x - coord[i % 4].x) \* (coord[(i + 2) % 4].x - coord[(i + 1) % 4].x) +

(coord[(i + 1) % 4].y - coord[i % 4].y) \* (coord[(i + 2) % 4].y - coord[(i + 1) % 4].y)) != 0) {

return false;

}

}

return true;

}

return false;

}

bool is\_trapezoid(vector<Vertex>& coord) {

double s1 = cross(coord[0] - coord[1], coord[2] - coord[3]);

double s2 = cross(coord[1] - coord[2], coord[0] - coord[3]);

return (s1 != s2 && (s1 == 0 || s2 == 0));

}

int main() {

int mx;

cout << "Enter max size of doc: ";

cin >> mx;

Factory doc(mx);

int cmd = -1;

help();

while (cmd != 0) {

try {

cin >> cmd;

if (cmd == 1) { // Rhombus

cout << "Enter four pair of coord vertex of rhombus: ";

vector<Vertex> v(4);

for (int i = 0; i < 4; i++) {

if(!(cin >> v[i])){

cin.clear();

cin.ignore(1e9, '\n');

throw runtime\_error("Incorrect values");

}

}

if (is\_rhombus(v)) {

doc.add(make\_shared<Rhombus>(v));

}

else {

cout << "Is not a rhombus\n";

}

}

else if (cmd == 2) { // Trapezoid

cout << "Enter four pair of coord vertex of trapezoid: ";

vector<Vertex> v(4);

for (int i = 0; i < 4; i++) {

if(!(cin >> v[i])){

cin.clear();

cin.ignore(1e9, '\n');

throw runtime\_error("Incorrect values");

}

}

if (is\_trapezoid(v)) {

doc.add(make\_shared<Trapezoid>(v));

}

else {

cout << "Is not a trapezoid\n";

}

}

else if (cmd == 3) { // Rectangle

cout << "Enter four pair of coord vertex of rectangle: ";

vector<Vertex> v(4);

for (int i = 0; i < 4; i++) {

if(!(cin >> v[i])){

cin.clear();

cin.ignore(1e9, '\n');

throw runtime\_error("Incorrect values");

}

}

if (is\_rectangle(v)) {

doc.add(make\_shared<Rectangle>(v));

}

else {

cout << "Is not a rectangle\n";

}

}

else if (cmd == 0) {

break;

}

else {

throw runtime\_error("Wrong command");

}

}

catch (runtime\_error& ex) {

cout << ex.what() << "\n";

}

catch (invalid\_argument& ex) {

cout << ex.what() << "\n";

}

catch (out\_of\_range& ex) {

cout << ex.what() << "\n";

}

}

return 0;

}

**5.** **Выводы**

В ходе лабораторной работы были реализованы классы Rhombus, Trapezoid, Rectangle для работы с фигурами, класс Factory для асинхронной работы с файлами. По итогам выполнения работы я усовершенствовал навыки работы с файлами и геометрическими фигурами. Взаимодействие с пользователем происходит посредством интерактивного меню, в котором пользователь выбирает команды и вводит данные, если это нужно.

ЛИТЕРАТУРА

1. Уроки программирования на языке С++ [Электронный ресурс]. URL:<https://ravesli.com/uroki-cpp> (дата обращения 29.11.2021).

2. The Compiler Generator Coco/R for C++. Documentation [Электронный ресурс]. URL:<https://ssw.jku.at/Research/Projects/Coco/#Docu> (дата обращения 29.11.2021).

3. cpp.reference, справочник по спецификации языка C++ [Электронный ресурс]. URL:<https://en.cppreference.com/w/> (дата обращения 02.12.2021).