**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: Хренникова Ангелина Сергеевна

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Цель: Познакомиться с асинхронным программированием, получить практические навыки в параллельной обработке данных и получить практические навыки в синхронизации потоков.

Задание: 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.

1. Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.
2. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.
3. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков.
4. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.
5. 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.

Фигуры: пятиугольник, шестиугольник, восьмиугольник.

1. **Описание программы**

Программа позволяет создавать фигуры: пятиугольник, шестиугольник и восьмиугольник, и записывать их в буфер, размер которого подается при запуске. Когда буфер заполняется, запускаются два обработчика, один из которых печатает данные в стандартный поток, а другой - в указанный файл. Обработчики находятся в разных потоках и действуют асинхронно.

Пятиугольник, шестиугольник и восьмиугольник задаются координатами центральной точкой и одной из точек самой фигуры, остальные точки вычисляются программой. Для каждой фигуры печатаются координаты, центр и площадь. Площадь вычисляется по формуле: , где R - радиус описанной окружности(разность координат центра и одной из точек многоугольника), n - количество углов многоугольника.

1. **Набор тестов**

Тест1: (параметр командной строки: 2) 1 pentagon 0 0 0 6 1 hexagon 0 0 0 6 1 octagon 0 1 0 0 1 pentagon 0 0 0 1 2

1. Создаются две фигуры: пятиугольник и шестиугольник с центром в точки С(0; 0) и точкой А(0; 6). Так как размер буфера 2, то эти две фигуры записываются в файл и печатаются на экран.
2. Создаются две фигуры: восьмиугольник с центром в точке С(0; 1) и точкой А(0; 0) и пятиугольник с центром в точки С(0; 0) и точкой А(0; 1). Так как размер буфера 2, то эти две фигуры записываются в файл и печатаются на экран.
3. Выход из программы.

Тест2: (параметр командной строки: 3) 1 pentagon 0 0 0 6 1 hexagon 0 0 0 6 1 octagon 0 0 0 6 2

1. Создаются три фигуры: пятиугольник, шестиугольник и восьмиугольник с центром в точки С(0; 0) и точкой А(0; 6). Так как размер буфера 3, то эти три фигуры записываются в файл и печатаются на экран.
2. **Результаты выполнения тестов**

Тест1:

1. Add a figure

pentagon

hexagon

octagon

2. Exit

Select menu item: 1

pentagon 0 0 0 6

Select menu item: 1

hexagon 0 0 0 6

You've reached the limit

Pentagon:

(0; 6) (-5.70634; 1.8541) (-3.52671; -4.8541) (3.52671; -4.8541) (5.70634; 1.8541)

Center: (0; 0)

Area: 85.5951

Hexagon:

(0; 6) (-5.19615; 3) (-5.19615; -3) (0; -6) (5.19615; -3) (5.19615; 3)

Center: (0; 0)

Area: 93.5307

Select menu item: 1

octagon 0 1 0 0

Select menu item: 1

pentagon 0 0 0 1

You've reached the limit

Octagon:

(0; 0) (0.707107; 0.292893) (1; 1) (0.707107; 1.70711) (1.11022e-16; 2) (-0.707107; 1.70711) (-1; 1) (-0.707107; 0.292893)

Center: (0; 1)

Area: 2.82843

Pentagon:

(0; 1) (-0.951057; 0.309017) (-0.587785; -0.809017) (0.587785; -0.809017) (0.951057; 0.309017)

Center: (0; 0)

Area: 2.37764

Select menu item: 2

Тест2:

1. Add a figure

pentagon

hexagon

octagon

2. Exit

Select menu item: 1

pentagon 0 0 0 6

Select menu item: 1

hexagon 0 0 0 6

Select menu item: 1

octagon 0 0 0 6

You've reached the limit

Pentagon:

(0; 6) (-5.70634; 1.8541) (-3.52671; -4.8541) (3.52671; -4.8541) (5.70634; 1.8541)

Center: (0; 0)

Area: 85.5951

Hexagon:

(0; 6) (-5.19615; 3) (-5.19615; -3) (0; -6) (5.19615; -3) (5.19615; 3)

Center: (0; 0)

Area: 93.5307

Octagon:

(0; 6) (-4.24264; 4.24264) (-6; 0) (-4.24264; -4.24264) (0; -6) (4.24264; -4.24264) (6; 0) (4.24264; 4.24264)

Center: (0; 0)

Area: 101.823

Select menu item: 2

1. **Листинг программы**

#include <iostream>

#include <fstream>

#include <memory>

#include <vector>

#include <thread>

#include <cmath>

#include <string>

#include <condition\_variable>

#include <mutex>

struct point { //координаты

double x, y;

point(double a, double b) { x = a, y = b; };

point() = default;

};

std::istream& operator>>(std::istream& is, point& p) { //считывание

is >> p.x >> p.y;

return is;

}

std::ostream& operator<< (std::ostream& os, const point& p) { //запись

os << p.x << " " << p.y << std::endl;

return os;

}

struct figure {

virtual point center() const = 0;

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

virtual double area() const = 0;

virtual ~figure() {}

};

struct pentagon : figure { // Пятиугольник

private:

point a0, a1; // Центр и точка

public:

point center() const override { // Центр пятиугольника

return a0;

};

std::ostream& print(std::ostream& os) const override { // Вывод пятиугольника и его характеристик

double c = (sqrt(5) - 1) / 4;

double s = (sqrt(10 + 2 \* sqrt(5))) / 4;

double a2x = a0.x + (a1.x - a0.x) \* c - (a1.y - a0.y) \* s;

double a2y = a0.y + (a1.x - a0.x) \* s + (a1.y - a0.y) \* c;

double a3x = a0.x + (a2x - a0.x) \* c - (a2y - a0.y) \* s;

double a3y = a0.y + (a2x - a0.x) \* s + (a2y - a0.y) \* c;

double a4x = a0.x + (a3x - a0.x) \* c - (a3y - a0.y) \* s;

double a4y = a0.y + (a3x - a0.x) \* s + (a3y - a0.y) \* c;

double a5x = a0.x + (a4x - a0.x) \* c - (a4y - a0.y) \* s;

double a5y = a0.y + (a4x - a0.x) \* s + (a4y - a0.y) \* c;

os << "Pentagon:\n" << "(" << a1.x << "; " << a1.y << ") " <<

"(" << a2x << "; " << a2y << ") " << "(" << a3x << "; " << a3y << ") " <<

"(" << a4x << "; " << a4y << ") " << "(" << a5x << "; " << a5y << ")" << '\n';

os << "Center: " << "("<< a0.x << "; " << a0.y << ")" << '\n' << "Area: " << area() << '\n';

};

double area() const override { // Площадь пятиугольника

double s = (sqrt(10 + 2 \* sqrt(5))) / 4;

double r = sqrt(pow(a1.x - a0.x, 2) + pow(a1.y - a0.y, 2));

double area = 0.5 \* r \* r \* 5 \* s;

return area;

};

pentagon() = default;

pentagon(std::istream& is) { // Ввод пятиугольника

is >> a0 >> a1;

};

};

struct hexagon : figure { // Шестиугольник

private:

point a0, a1;

public:

point center() const override { //центр

return a0;

};

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

double c = 0.5;

double s = sqrt(3) / 2;

double a2x = a0.x + (a1.x - a0.x) \* c - (a1.y - a0.y) \* s;

double a2y = a0.y + (a1.x - a0.x) \* s + (a1.y - a0.y) \* c;

double a3x = a0.x + (a2x - a0.x) \* c - (a2y - a0.y) \* s;

double a3y = a0.y + (a2x - a0.x) \* s + (a2y - a0.y) \* c;

double a4x = a0.x + (a3x - a0.x) \* c - (a3y - a0.y) \* s;

double a4y = a0.y + (a3x - a0.x) \* s + (a3y - a0.y) \* c;

double a5x = a0.x + (a4x - a0.x) \* c - (a4y - a0.y) \* s;

double a5y = a0.y + (a4x - a0.x) \* s + (a4y - a0.y) \* c;

double a6x = a0.x + (a5x - a0.x) \* c - (a5y - a0.y) \* s;

double a6y = a0.y + (a5x - a0.x) \* s + (a5y - a0.y) \* c;

os << "Hexagon:\n" << "(" << a1.x << "; " << a1.y << ") " <<

"(" << a2x << "; " << a2y << ") " << "(" << a3x << "; " << a3y << ") " <<

"(" << a4x << "; " << a4y << ") " << "(" << a5x << "; " << a5y << ") " <<

"(" << a6x << "; " << a6y << ")" << '\n';

os << "Center: " << "("<< a0.x << "; " << a0.y << ")" << '\n' << "Area: " << area() << '\n';

};

double area() const override { //площадь

double area = pow(sqrt(pow(a1.x - a0.x, 2) + pow(a1.y - a0.y, 2)), 2) \* 3 \* sqrt(3) \* 0.5;

return area;

};

hexagon() = default;

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

is >> a0 >> a1;

};

};

struct octagon : figure { //восьмиугольник

private:

point a0, a1;

public:

point center() const override { //центр

return a0;

};

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

double c = sqrt(2) / 2;

double s = sqrt(2) / 2;

double a2x = a0.x + (a1.x - a0.x) \* c - (a1.y - a0.y) \* s;

double a2y = a0.y + (a1.x - a0.x) \* s + (a1.y - a0.y) \* c;

double a3x = a0.x + (a2x - a0.x) \* c - (a2y - a0.y) \* s;

double a3y = a0.y + (a2x - a0.x) \* s + (a2y - a0.y) \* c;

double a4x = a0.x + (a3x - a0.x) \* c - (a3y - a0.y) \* s;

double a4y = a0.y + (a3x - a0.x) \* s + (a3y - a0.y) \* c;

double a5x = a0.x + (a4x - a0.x) \* c - (a4y - a0.y) \* s;

double a5y = a0.y + (a4x - a0.x) \* s + (a4y - a0.y) \* c;

double a6x = a0.x + (a5x - a0.x) \* c - (a5y - a0.y) \* s;

double a6y = a0.y + (a5x - a0.x) \* s + (a5y - a0.y) \* c;

double a7x = a0.x + (a6x - a0.x) \* c - (a6y - a0.y) \* s;

double a7y = a0.y + (a6x - a0.x) \* s + (a6y - a0.y) \* c;

double a8x = a0.x + (a7x - a0.x) \* c - (a7y - a0.y) \* s;

double a8y = a0.y + (a7x - a0.x) \* s + (a7y - a0.y) \* c;

os << "Octagon:\n" << "(" << a1.x << "; " << a1.y << ") " <<

"(" << a2x << "; " << a2y << ") " << "(" << a3x << "; " << a3y << ") " <<

"(" << a4x << "; " << a4y << ") " << "(" << a5x << "; " << a5y << ") " <<

"(" << a6x << "; " << a6y << ") " << "(" << a7x << "; " << a7y << ") " <<

"(" << a8x << "; " << a8y << ") " << '\n';

os << "Center: " << "("<< a0.x << "; " << a0.y << ")" << '\n' << "Area: " << area() << '\n';

};

double area() const override { //площадь

double r = sqrt(pow(a1.x - a0.x, 2) + pow(a1.y - a0.y, 2));

double area = 4 \* r \* r \* (sqrt(2) / 2);

return area;

};

octagon() = default;

octagon(std::istream& is) {

is >> a0 >> a1;

};

};

struct factory {

std::shared\_ptr<figure> new\_figure(std::istream& is);

};

std::shared\_ptr<figure> factory::new\_figure(std::istream& is) {

std::string name;

is >> name;

if (name == "pentagon") {

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

}

else if (name == "hexagon") {

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

}

else if (name == "octagon") {

return std::shared\_ptr<figure>(new octagon(is));

}

else {

throw std::logic\_error("There's no such figure\n");

}

}

struct processor {

virtual void process(std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer) = 0;

};

struct stream\_processor : processor {

void process(std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer) override;

};

struct file\_processor : processor {

void process(std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer) override;

private:

int cnt = 0;

};

void stream\_processor::process(std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer) {

for (auto figure : \*buffer) {

figure->print(std::cout);

}

}

void file\_processor::process(std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer) {

std::ofstream fout;

fout.open(std::to\_string(cnt) + ".txt");

cnt++;

if (!fout.is\_open()) {

std::cout << "can't open\n";

return;

}

for (auto figure : \*buffer) {

figure->print(fout);

}

}

struct subscriber {

void operator()();

std::vector<std::shared\_ptr<processor>> processors;

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

std::mutex mtx;

std::condition\_variable cond\_var;

bool stop = false;

};

void subscriber::operator()() {

for (;;) {

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

cond\_var.wait(lock, [&] { return (buffer != nullptr || stop); });

if (stop) {

break;

}

for (auto elem : processors) {

elem->process(buffer);

}

buffer = nullptr;

cond\_var.notify\_all();

}

}

void help() {

std::cout <<

"1. Add a figure\n"

"\tpentagon\n"

"\thexagon\n"

"\toctagon\n"

"2. Exit\n";

}

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

if (argc != 2) {

std::cout << "2 arguments needed\n";

return 1;

}

int buffer\_size = std::stoi(argv[1]);

std::shared\_ptr<std::vector<std::shared\_ptr<figure>>> buffer = std::make\_shared<std::vector<std::shared\_ptr<figure>>>();

buffer->reserve(buffer\_size);

factory factory;

subscriber sub;

sub.processors.push\_back(std::make\_shared<stream\_processor>());

sub.processors.push\_back(std::make\_shared<file\_processor>());

std::thread sub\_thread(std::ref(sub));

help();

int command;

std::cout << "Select menu item: ";

std::cin >> command;

while (command > 0 && command < 2) {

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

if (command == 1) {

try {

buffer->push\_back(factory.new\_figure(std::cin));

}

catch (std::logic\_error& e) {

std::cout << e.what() << '\n';

continue;

}

if (buffer->size() == buffer\_size) {

std::cout << "You've reached the limit\n";

sub.buffer = buffer;

sub.cond\_var.notify\_all();

sub.cond\_var.wait(locker, [&]() { return sub.buffer == nullptr; });

buffer->clear();

}

}

std::cout << "Select menu item: ";

std::cin >> command;

}

sub.stop = true;

sub.cond\_var.notify\_all();

sub\_thread.join();

return 0;

}

1. **Вывод**

Во время выполнения данной лабораторной работы я познакомилась с асинхронным программированием, получила практические навыки в параллельной обработке данных и получила практические навыки в синхронизации потоков.

Асинхронные вычисления - форма, при которой допускается продолжение работы до окончания передачи данных. Асинхронные события происходят независимо от основного потока исполнения. Во многих случаях асинхронность дает дает хороший выигрыш по времени, так как многие тяжеловесные вызовы можно выносить в отдельные потоки и продолжать вычисления. Однако появляется дополнительная ответственность, так как нужно следить за выполнением уже каждого процесса.

**Список литературы**

1. Асинхронность в С++/Электронный ресурс/Режим доступа:<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=2ahUKEwibzoaIjMzmAhWzwMQBHQIoBZwQFjABegQIBBAB&url=https%3A%2F%2Fotus.ru%2Fnest%2Fpost%2F337%2F&usg=AOvVaw1DAXgMK0jgQ2FyE-xkjB8j> (дата обращения: 19.12.2020).

2. Асинхронное программирование/Электронный ресурс/Режим доступа:

[https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&cad=rja&uact=8&ved=2ahUKEwiSt6bKiczmAhWlwsQBHVxlAsYQFjAHegQIBRAB&url=https%3A%2F%2Fproglib.io%2Fp%2Fasynchrony%2F&usg=AOvVaw3dtozhGdW9hvtf3I-cIYV4](https://proglib.io/p/asynchrony/) (дата обращения: 19.12.2020).