МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВЦИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЬЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №8

по курсу объектно-ориентированное программирование I семестр, 2019/20 уч. год

Студент *Артемьев Дмитрий Иванович, группа М8О-206Б-18*

Преподаватель *Журавлёв Андрей Андреевич*

Условие

Создать приложение, которое будет считывать из стандартного ввода данные фигур, согласно варианту задания, выводить их характеристики на экран и записывать их в файл.

Программа должна:

1. Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;
2. Создавать классы, соответсвующие введённым данным фигур;
3. Содержать внутренний буфер, в который помещаются фигуры, размер буфера задаётся аргументом командной строки;
4. При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;
5. Обработка должна производиться в отдельном потоке;
6. Два обработчкика: вывод буфера на экран и в файл;
7. Оба обработчика должны обрабатывать каждый введённый буфер;
8. В программе должно быть ровно два потока: один основной, второй для обработчиков;
9. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быить реализован как отдельный подписчик;
10. Реализовать в основном потоке ожидание буфера обработки в потоке-обработчике.

Описание программы

Исходный код лежит в 8 файлах:

1. src/main.cpp: основная интерактивная программа, возможность работать со стеком
2. include/figure.hpp: описание класса фигур, перегрузка операторов считывания
3. include/figure.cpp: реализация класса фигур, операторов считывания
4. include/triangle.hpp: описание класса треугольника
5. include/triangle.cpp: реализация класса треугольника
6. include/square.hpp: описание класса квадрата

7. include/square.cpp: реализация класса квадрата
8. include/rectangle.hpp: описание класса прямоугольника
9. include/rectangle.cpp: реализация класса прямоугольника
10. include/point.hpp: описание класса точки
11. include/point.cpp: реализация класса точки
12. include/processor.hpp: описание класса обработчика
13. include/processor.cpp: реализация класса обработчика
14. include/subscriber.hpp: описание класса "подписчика"
15. include/subscriber.cpp: реализация класса "подписчика"

Дневник отладки

Нужно аккуратно работать с потоками.

Недочёты

Не реализована возможность адекватного выхода на CTRL-D.

Выводы

Я приобрёл базовые навыки написания многопоточных программ на языке C++, получил опыт в использовании mutex и conditional variable. В добавок, научился использовать Continuous Integration.

Исходный код

main.cpp

```
#include <iostream>
#include <memory>
#include <vector>
#include <thread>
#include "processor.hpp"
#include "subscriber.hpp"
#include "figure.hpp"
#include "triangle.hpp"
#include "square.hpp"
#include "rectangle.hpp"

std::shared_ptr<Figure> getFigure(std::istream& is) {
    Figure* fig;
    is >> fig;
    return std::shared_ptr<Figure>(fig);
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cout << "Usage: " << argv[0] << " <buffer size>\n";
        return 1;
    }
    const int buf_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(buf_size);

    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));

    while(true) {
        std::unique_lock<std::mutex> guard(sub.mtx);
        std::string cmd;
        std::cin >> cmd;
        if (cmd == "add") {
            try {
```

```
                buffer->push_back(getFigure(std::cin));
            }
            catch (std::logic_error& e) {
                std::cout << e.what() << "\n";
                continue;
            }
            if (buffer->size() == buf_size) {
                sub.buffer = buffer;
                sub.cv.notify_all();
                sub.cv.wait(guard, [&]() { return sub.buffer == nullptr; });
                buffer->clear();
            }
        }
        else if (cmd == "exit"){
            break;
        }
        else {
            std::cout << "Wrong command!\n";
        }
    }
    sub.end = true;
    sub.cv.notify_all();
    sub_thread.join();

    return 0;
}
```

figure.hpp

```cpp
#pragma once

#include <iostream>

#include <point.hpp>

struct Figure {
    virtual Point figureCenter() const = 0;
    virtual double figureArea() const = 0;
    virtual void printPoints(std::ostream& ) const = 0;

    virtual ~Figure() = default;
};

std::ostream& operator<< (std::ostream& out, Figure const& fig);
std::istream& operator>> (std::istream& in, Figure*& fig);
```

figure.cpp

```cpp
#include <vector>
#include <algorithm>

#include <figure.hpp>
#include <triangle.hpp>
#include <rectangle.hpp>
#include <square.hpp>

std::ostream& operator<< (std::ostream& out, Figure const& fig) {
    fig.printPoints(out);
    return out;
}

void orderPoints(std::vector<Point>& pts);
Figure* whatFigure(std::vector<Point>& pts);

std::istream& operator>> (std::istream& in, Figure*& fig) {
    std::vector<Point> pts;

    while (true) {
        char c = in.get();
        while (c == ' ' || c == '\t' || c == '\n')
            c = in.get();
        in.unget();

        if (((c < '0') || (c > '9')) && c != '-') {
            break;
        }
        else {
            Point pt;
            in >> pt;
            pts.push_back(pt);
        }
    }

    fig = whatFigure(pts);
    return in;
}

Figure* whatFigure(std::vector<Point>& pts) {
    Figure* ptr = nullptr;
```

```
    if (pts.size() == 3) {
        ptr = new Triangle(pts);
    }
    else if (pts.size() == 4) {
        if (dist(pts[0], pts[2]) == dist(pts[1], pts[3]) &&
            dist(pts[0], pts[1]) == dist(pts[2], pts[3]) &&
            dist(pts[1], pts[2]) == dist(pts[0], pts[3])) {
            if (dist(pts[0], pts[1]) == dist(pts[1], pts[2])) {
                ptr = new Square(pts);
            }
            else {
                ptr = new Rectangle(pts);
            }
        }
    }

    return ptr;
}
```

triangle.hpp

```cpp
#pragma once

#include <iostream>
#include <vector>

#include <figure.hpp>

class Triangle : public Figure{
public:
    Triangle(std::vector<Point> const& );

    Point figureCenter() const override;
    double figureArea() const override;
    void printPoints(std::ostream& ) const override;

    ~Triangle() = default;
protected:
    static const int NUM = 3;
    Point pts[NUM];
};
```

triangle.cpp

```cpp
#include <triangle.hpp>
#include <algorithm>

Triangle::Triangle(std::vector<Point> const& fig) {
    for (int i = 0; i < NUM; i++) {
        pts[i] = fig[i];
    }
}

Point Triangle::figureCenter() const {
    return (pts[0] + pts[1] + pts[2]) / NUM;
}



double Triangle::figureArea() const {
    return std::abs((
                    (pts[1].x - pts[0].x) * (pts[3].y - pts[0].y) -
                    (pts[1].y - pts[0].y) * (pts[3].x - pts[0].x)
                    ).v) / 2;
}

void Triangle::printPoints(std::ostream& out) const {
    out << "Triangle: " << std::endl;
    for (Point const& pt : pts) {
        out << pt << std::endl;
    }
}
```

square.hpp

```cpp
#pragma once

#include <figure.hpp>
#include <rectangle.hpp>

class Square : public Rectangle {
public:
    Square(std::vector<Point> const& fig) : Rectangle(fig) {};
    void printPoints(std::ostream& out) const override;
    ~Square() = default;
};
```

square.cpp

```cpp
#include <square.hpp>

void Square::printPoints(std::ostream& out) const {
    out << "Square: " << std::endl;
    for (Point const& pt : pts) {
        out << pt << std::endl;
    }
}
```

rectangle.hpp

```cpp
#pragma once

#include <iostream>
#include <vector>

#include <figure.hpp>

class Rectangle : public Figure {
public:
    Rectangle(std::vector<Point> const& );

    Point figureCenter() const override;
    double figureArea() const override;
    void printPoints(std::ostream& ) const override;

    ~Rectangle() = default;
protected:
    static const int NUM = 4;
    Point pts[NUM];
};
```

rectangle.cpp

```cpp
#include <rectangle.hpp>
#include <algorithm>

Rectangle::Rectangle(std::vector<Point> const& fig) {
    for (int i = 0; i < NUM; i++) {
        pts[i] = fig[i];
    }
}

Point Rectangle::figureCenter() const {
    return (pts[0] + pts[1] + pts[2] + pts[3]) / 4;
}

double Rectangle::figureArea() const {
    return std::abs(((pts[2].x - pts[0].x) * (pts[1].y - pts[0].y)).v);
}

void Rectangle::printPoints(std::ostream& out) const {
    out << "Rectangle: " << std::endl;
    for (Point const & pt : pts) {
        out << pt << std::endl;
    }
}
```

point.hpp

```cpp
#pragma once

#include <iostream>

struct my_double {
    double v;
    my_double() {};
    my_double(double x) : v(x) {};
};

struct Point {
    my_double x, y;
    Point() {};
    Point(double x, double y) : x(my_double(x)), y(my_double(y)) {};
};

std::istream& operator>> (std::istream& , Point& );
std::ostream& operator<< (std::ostream& , Point const& );
Point operator+ (Point const& , Point const& );
Point operator- (Point const& , Point const& );
Point operator* (Point const& , double const& );
Point operator/ (Point const& , double const& );

my_double operator+ (my_double const& , my_double const& );
my_double operator- (my_double const& , my_double const& );
my_double operator* (my_double const& , my_double const& );
my_double operator/ (my_double const& , my_double const& );
bool operator<  (my_double const& a, my_double const& b);
bool operator>  (my_double const& a, my_double const& b);
bool operator== (my_double const& a, my_double const& b);

bool operator< (Point const& , Point const& );
bool operator> (Point const& , Point const& );
bool operator<= (Point const& , Point const& );
bool operator>= (Point const& , Point const& );
bool operator== (Point const& , Point const& );

my_double dist(const Point& , const Point& );
```

point.cpp

```
#include <iostream>
#include <cmath>
#include <point.hpp>

std::istream& operator>> (std::istream& in, Point& p) {
    return in >> p.x.v >> p.y.v;
}

std::ostream& operator<< (std::ostream& out, Point const& p) {
    return out << "{"  << p.x.v << ' ' << p.y.v << "} ";
}

Point operator+ (Point const& a, Point const& b) {
    return Point{a.x.v + b.x.v, a.y.v + b.y.v};
}

Point operator- (Point const& a, Point const& b) {
    return Point{a.x.v - b.x.v, a.y.v - b.y.v};
}

Point operator* (Point const& a, double const& b) {
    return Point{a.x.v * b, a.y.v * b};
}

Point operator/ (Point const& a, double const& b) {
    return Point{a.x.v / b, a.y.v / b};
}



my_double operator+ (my_double const& a, my_double const& b) {
    return my_double{a.v + b.v};
}

my_double operator- (my_double const& a, my_double const& b) {
    return my_double{a.v - b.v};
}

my_double operator* (my_double const& a, my_double const& b) {
    return my_double{a.v * b.v};
}
```

```
my_double operator/ (my_double const& a, my_double const& b) {
    return my_double{a.v / b.v};
}



const double EPS = 1e-9;

bool operator< (my_double const& a, my_double const& b) {
    return b.v - a.v > EPS;
}

bool operator> (my_double const& a, my_double const& b) {
    return a.v - b.v > EPS;
}

bool operator== (my_double const& a, my_double const& b) {
    return !(a.v < b.v) && !(a.v > b.v);
}

bool operator<  (Point const& a, Point const& b) {
    return (a.x.v < b.x.v) || (a.x.v == b.x.v && a.y.v < b.y.v);
}

bool operator>  (Point const& a, Point const& b) {
    return (a.x.v > b.x.v) || (a.x.v == b.x.v && a.y.v > b.y.v);
}

bool operator== (Point const& a, Point const& b) {
    return (a.x.v == b.x.v) && (a.y.v == b.y.v);
}

bool operator<= (Point const& a, Point const& b) {
    return a == b || a < b;
}

bool operator>= (Point const& a, Point const& b) {
    return a == b || a > b;
}

my_double dist(const Point& a, const Point& b) {
```

```
    return my_double(sqrt((a.x.v - b.x.v) * (a.x.v - b.x.v) +
                          (a.y.v - b.y.v) * (a.y.v - b.y.v)));
}
```

processor.hpp

```cpp
#pragma once

#include <iostream>
#include <memory>
#include <vector>
#include <thread>
#include <mutex>
#include "figure.hpp"


class processor {
public:
    virtual void process(std::shared_ptr<std::vector<std::shared_ptr<Figure>>> buffer) =
};

class stream_processor : public processor {
public:
    void process(std::shared_ptr<std::vector<std::shared_ptr<Figure>>> buffer) override;
};

class file_processor : public processor {
public:
    void process(std::shared_ptr<std::vector<std::shared_ptr<Figure>>> buffer) override;
private:
    int counter = 0;
};
```

processor.cpp

```cpp
#include <fstream>
#include "processor.hpp"

void stream_processor::process(std::shared_ptr<std::vector<std::shared_ptr<Figure>>> buf
    std::cout << "PRINT CONSOLE" << std::endl;
    for (const auto& figure : *buffer) {
        std::cout << (*figure);
    }
}


void file_processor::process(std::shared_ptr<std::vector<std::shared_ptr<Figure>>> buffe
    std::cout << "PRINT FILE" << std::endl;
    std::ofstream osf;
    osf.open(std::to_string(counter) + ".txt");
    ++counter;
    if (!osf.is_open()) {
        std::cout << "Error opening file\n";
        return;
    }
    for (const auto& figure : *buffer) {
        figure->printPoints(osf);
    }
}
```

subscriber.hpp

```cpp
#pragma once

#include <memory>
#include <iostream>
#include <condition_variable>
#include <thread>
#include <vector>
#include <mutex>
#include "figure.hpp"
#include "processor.hpp"


class subscriber {
public:
    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 cv;
    bool end = false;
};
```

subscriber.cpp

```cpp
#include "subscriber.hpp"

void subscriber::operator()() {
    while (true) {
        std::unique_lock<std::mutex> lock(mtx);
        std::cout << "SUBSCRIBER IS WORKING" << std::endl;
        cv.wait(lock, [&] { return (buffer != nullptr || end); });
        if (end) {
            break;
        }
        for (const auto& processor_elem : processors) {
            processor_elem->process(buffer);
        }
        buffer = nullptr;
        cv.notify_all();
    }
}
```