

**Московский авиационный институт
(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 8

Тема: Асинхронное программирование

Студент: Семенов Илья

Преподаватель: Журавлев А.А.

Дата:

Оценка:

Москва, 2019

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.

Программа должна

- Осуществлять ввод из стандартного ввода данных фигур
- Создавать классы, соответствующие введенным данным фигур
- Программа должна содержать внутренний буфер, в который помещаются фигуры. Размер буфера задается параметром командной строки.
- При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться
- Обработка производится в отдельном потоке
- Должны быть реализованы два обработчика, которые должны обрабатывать данные буфера: один из них выводит данные на экран, другой в файл с уникальным именем
- Оба обработчика должны обрабатывать каждый введенный буфер
- В программе должно быть два потока

Вариант задания 20:

- Фигуры – Трапеция, Ромб, Пятиугольник

2. Репозиторий github

https://github.com/ilya89099/oop_exercise_08/

3. Описание программы

Реализован класс Publisher, содержащий оператор (), код которого запускается в отдельном потоке. Так же этот класс содержит необходимые средства синхронизации – mutex и condition_variable, использующая этот mutex. Так же реализована фабрика, позволяющая конструировать фигуры из стандартного(и не только) потока ввода.

4. Набор testcases

Для тестов используются:

- Модуль юнит тестов библиотеки boost, подключенная через CMake с помощью CTest.

- Небольшой bash скрипт, тестирующий вывод собранной программы

test.cpp(boost)

```
#define BOOST_TEST_DYN_LINK

#define BOOST_TEST_MODULE figures
#include "../src/Figures/Trapeze.h"
#include "../src/Figures/Rhombus.h"
#include "../src/Figures/Pentagon.h"
#include "../src/Processors/ConsoleProcessor.h"
#include "../src/Processors/FileProcessor.h"
#include <sstream>
#include <memory>
#include <boost/test/unit_test.hpp>
BOOST_AUTO_TEST_SUITE(figures)
BOOST_AUTO_TEST_CASE(trapeze) {
    {
        Point p1 = {0,0}, p2 = {1, 2}, p3 = {4, 5}, p4 = {10,0};
        BOOST_CHECK_THROW(Trapeze(p1,p2,p3,p4), std::logic_error);
    }
    {
        Point p1 = {0,0}, p2 = {0, 4}, p3 = {4, 4}, p4 = {10,0};
        BOOST_CHECK_NO_THROW(Trapeze(p1,p2,p3,p4));
    }
    {
        Point p1 = {0,0}, p2 = {0, 0}, p3 = {0, 0}, p4 = {0,0};
        BOOST_CHECK_NO_THROW(Trapeze(p1,p2,p3,p4));
    }
}
BOOST_AUTO_TEST_CASE(rhombus) {
    {
        Point p1 = {0,0}, p2 = {1, 2}, p3 = {4, 5}, p4 = {10,0};
        BOOST_CHECK_THROW(Rhombus(p1,p2,p3,p4), std::logic_error);
    }
    {
        Point p1 = {0,0}, p2 = {3, 4}, p3 = {8, 4}, p4 = {5,0};
        BOOST_CHECK_NO_THROW(Rhombus(p1,p2,p3,p4));
    }
    {
        Point p1 = {0,0}, p2 = {0, 0}, p3 = {0, 0}, p4 = {0,0};
        BOOST_CHECK_NO_THROW(Rhombus(p1,p2,p3,p4));
    }
}
BOOST_AUTO_TEST_CASE(pentagon) {
    {
        Point p1 = {0,0}, p2 = {1, 2}, p3 = {4, 5}, p4 = {10,0}, p5 = {-5,
-5};
        BOOST_CHECK_NO_THROW(Pentagon(p1,p2,p3,p4,p5));
    }
}
```

```

}
BOOST_AUTO_TEST_CASE(console_processor) {
    std::stringstream buffer;
    std::streambuf * old = std::cout.rdbuf(buffer.rdbuf());
    std::vector<std::shared_ptr<Figure>> v;

    v.push_back(std::make_shared<Pentagon>(Point{1,2},Point{3,4},Point{5,6},Point{7,8},Point{9,10}));
    v.push_back(std::make_shared<Rhombus>(Point{0,0}, Point{3, 4}, Point{8, 4}, Point{5,0}));
    v.push_back(std::make_shared<Trapeze>(Point{0,0}, Point{0, 4}, Point{4, 4}, Point{6,0}));
    ConsoleProcessor proc;
    proc.Process(v);
    std::string output = "Pentagon, p1: 1 2, p2: 3 4, p3: 5 6, p4: 7 8, p5: 9 10\n"
                        "Rhombus, p1: 0 0, p2: 3 4, p3: 8 4, p4: 5 0\n"
                        "Trapeze, p1: 0 0, p2: 4 4, p3: 0 4, p4: 6 0\n";
    BOOST_CHECK_EQUAL(output, buffer.str());
    std::cout.rdbuf(old);
}
BOOST_AUTO_TEST_SUITE_END()

```

test.sh

```

#!/bin/bash
prog="$1"
echo $prog

"$prog" 3 < test_01.test > tmp_test_file

if diff tmp_test_file test_01.result && diff Buffer_1
test_01.buffer1.result ; then
    echo "test $a OK"
else
    echo "test $a NOT OK"
fi
let "a += 1"
rm tmp_test_file
rm Buffer_1

```

test_01.test

```

create trapeze 0 0 0 0 0 0 0 0
create rhombus 1 1 1 1 1 1 1 1
create pentagon 2 2 2 2 2 2 2 2 2 2
create trapeze 0 0 0 4 5 0 4 4
exit

```

test_01.result

Trapeze, p1: 0 0, p2: 0 0, p3: 0 0, p4: 0 0
Rhombus, p1: 1 1, p2: 1 1, p3: 1 1, p4: 1 1
Pentagon, p1: 2 2, p2: 2 2, p3: 2 2, p4: 2 2, p5: 2 2

test_01.buffer1.result

Trapeze, p1: 0 0, p2: 0 0, p3: 0 0, p4: 0 0
Rhombus, p1: 1 1, p2: 1 1, p3: 1 1, p4: 1 1
Pentagon, p1: 2 2, p2: 2 2, p3: 2 2, p4: 2 2, p5: 2 2

5. Результаты выполнения тестов

Все тесты завершились успешно. Программа работает так, как должна.

6. Листинг программы

main.cpp

```
#include <iostream>
#include <vector>
#include <deque>
#include <thread>
#include <sstream>
#include <memory>
#include "Processors/ConsoleProcessor.h"
#include "Processors/FileProcessor.h"
#include "Publisher.h"
#include "Figure.h"
#include "FigureFactory.h"
int main(int argc, char** argv) {
    const int buf_size = argc < 2 ? 10 : std::stoi(argv[1]);
    if (argc < 2) {
        std::cout << "Buffer size : " << buf_size << "\n";
    }
    std::shared_ptr<std::vector<std::shared_ptr<Figure>>> buffer =
std::make_shared<std::vector<std::shared_ptr<Figure>>>();
    Publisher pub;
    pub.AddWorker(std::make_shared<FileProcessor>());
    pub.AddWorker(std::make_shared<ConsoleProcessor>());
    std::thread thread(std::ref(pub));
    std::string command;
    while (true) {
        std::cin >> command;
        if (command == "create") {
            try {
```

```

        buffer->push_back(FigureFactory::CreateFigure(std::cin));
    } catch (std::exception& e) {
        std::cout << e.what() << "\n";
        continue;
    }
    if (buf_size == buffer->size()) {
        pub.SetBuffer(buffer);
        pub.Notify();
        buffer->clear();
    }
} else if (command == "exit") {
    pub.Finish();
    break;
} else {
    std::cout << "Unknown command\n";
    std::cin.ignore(32767, '\n');
}
}
thread.join();
return 0;
}

```

Figure.h

```

#pragma once
#include <numeric>
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
struct Point {
    double x = 0;
    double y = 0;
};
std::ostream& operator << (std::ostream& os, const Point& p);
std::istream& operator >> (std::istream& is, Point& p);
Point operator + (Point lhs, Point rhs);
Point operator - (Point lhs, Point rhs);
Point operator / (Point lhs, double a);
Point operator * (Point lhs, double a);
class Vector {
public:
    explicit Vector(double a, double b);
    explicit Vector(Point a, Point b);
    bool operator == (Vector rhs);
    Vector operator - ();
    friend double operator * (Vector lhs, Vector rhs);
    double length() const;
    double x;
    double y;
};
bool is_parallel(const Vector& lhs, const Vector& rhs);
bool is_perpendicular(const Vector& lhs, const Vector& rhs);

```

```

double point_and_line_distance(Point p1, Point p2, Point p3);
class Figure {
public:
    virtual void Print(std::ostream& os) const = 0;
    virtual ~Figure() = default;
};
std::ostream& operator << (std::ostream& os, const Figure& fig);

```

Figure.cpp

```

#include "Figure.h"
Point operator + (Point lhs, Point rhs) {
    return {lhs.x + rhs.x, lhs.y + rhs.y};
}
Point operator - (Point lhs, Point rhs) {
    return {lhs.x - rhs.x, lhs.y - rhs.y};
}
Point operator / (Point lhs, double a) {
    return { lhs.x / a, lhs.y / a};
}
Point operator * (Point lhs, double a) {
    return {lhs.x * a, lhs.y * a};
}
bool operator < (Point lhs, Point rhs) {
    return (lhs.x * lhs.x + lhs.y * lhs.y) < (rhs.x * rhs.x + rhs.y *
lhs.y);
}
double operator * (Vector lhs, Vector rhs) {
    return lhs.x * rhs.x + lhs.y * rhs.y;
}
bool is_parallel(const Vector& lhs, const Vector& rhs) {
    return (lhs.x * rhs.y - lhs.y * rhs.x) == 0;
}
bool Vector::operator == (Vector rhs) {
    return
        std::abs(x - rhs.x) < std::numeric_limits<double>::epsilon() * 100
        && std::abs(y - rhs.y) < std::numeric_limits<double>::epsilon() *
100;
}
double Vector::length() const {
    return sqrt(x*x + y*y);
}
Vector::Vector(double a, double b)
: x(a), y(b) {
}
Vector::Vector(Point a, Point b)
: x(b.x - a.x), y(b.y - a.y){
}
Vector Vector::operator - () {
    return Vector(-x, -y);
}
bool is_perpendicular(const Vector& lhs, const Vector& rhs) {
    return (lhs * rhs) == 0;
}

```

```

}
double point_and_line_distance(Point p1, Point p2, Point p3) {
    double A = p2.y - p3.y;
    double B = p3.x - p2.x;
    double C = p2.x*p3.y - p3.x*p2.y;
    return (std::abs(A*p1.x + B*p1.y + C) / std::sqrt(A*A + B*B));
}
std::ostream& operator << (std::ostream& os, const Point& p) {
    return os << p.x << " " << p.y;
}
std::istream& operator >> (std::istream& is, Point& p) {
    return is >> p.x >> p.y;
}
std::ostream& operator << (std::ostream& os, const Figure& fig) {
    fig.Print(os);
    return os;
}

```

FigureFactory.h

```

#pragma once
#include "Figures/Pentagon.h"
#include "Figures/Trapeze.h"
#include <string>
#include "Figures/Rhombus.h"
#include <memory>
class FigureFactory {
public:
    static std::unique_ptr<Figure> CreateFigure(std::istream& is);
};

```

FigureFactory.cpp

```

#include "FigureFactory.h"
std::unique_ptr<Figure> FigureFactory::CreateFigure(std::istream& is) {
    std::string figure_type;
    is >> figure_type;
    for (char& c : figure_type) {
        c = std::tolower(c);
    }
    if (figure_type == "trapeze") {
        return std::make_unique<Trapeze>(is);
    } else if (figure_type == "pentagon") {
        return std::make_unique<Pentagon>(is);
    } else if (figure_type == "rhombus") {
        return std::make_unique<Rhombus>(is);
    } else {
        throw std::logic_error("Wrong type of figure");
    }
}

```


Processor.h

```
#pragma once
#include <vector>
#include <memory>
#include "Figure.h"
class Processor {
public:
    virtual void Process(const std::vector<std::shared_ptr<Figure>>& buf) =
0;
};
```

Publisher.h

```
#pragma once
#include <mutex>
#include <memory>
#include <condition_variable>
#include "Processor.h"
class Publisher {
public:
    void operator() ();
    void AddWorker(std::shared_ptr<Processor> worker);
    void SetBuffer(std::shared_ptr<std::vector<std::shared_ptr<Figure>>>
buffer);
    void Notify();
    void Finish();
private:
    std::mutex mutex_;
    std::condition_variable variable_;
    std::vector<std::shared_ptr<Processor>> workers_;
    std::shared_ptr<std::vector<std::shared_ptr<Figure>>> buffer_;
    bool finish_ = false;
};
```

Publisher.cpp

```
#include "Publisher.h"
void Publisher::operator() () {
    while (true) {
        std::unique_lock<std::mutex> lock(mutex_);
        variable_.wait(lock, [&finish_ = this->finish_, &buffer_ = this-
>buffer_] () {return finish_ || buffer_!= nullptr;});
        if (finish_) {
            break;
        }
        for (std::shared_ptr<Processor>& ptr : workers_) {
            ptr->Process(*buffer_);
        }
        buffer_ = nullptr;
        variable_.notify_all();
    }
}
void Publisher::AddWorker(std::shared_ptr<Processor> worker) {
```

```

        workers_.push_back(std::move(worker));
    }
    void
    Publisher::SetBuffer(std::shared_ptr<std::vector<std::shared_ptr<Figure>>>
    buffer) {
        buffer_ = std::move(buffer);
    }
    void Publisher::Notify() {
        std::unique_lock<std::mutex> lock(mutex_);
        variable_.notify_all();
        variable_.wait(lock, [this] () { return buffer_ == nullptr;});
    }
    void Publisher::Finish() {
        finish_ = true;
        variable_.notify_all();
    }
}

```

ConsoleProcessor.h

```

#pragma once
#include "../Processor.h"
class ConsoleProcessor : public Processor {
public:
    void Process(const std::vector<std::shared_ptr<Figure>>& buf) override;
};

```

ConsoleProcessor.cpp

```

#include "ConsoleProcessor.h"
void ConsoleProcessor::Process(const std::vector<std::shared_ptr<Figure> >
&buf) {
    for (const std::shared_ptr<Figure>& ptr : buf) {
        ptr->Print(std::cout);
        std::cout << "\n";
    }
}

```

LineProcessor.h

```

#pragma once
#include <fstream>
#include "../Processor.h"
class FileProcessor : public Processor {
public:
    void Process(const std::vector<std::shared_ptr<Figure>>& buf) override;
private:
    unsigned counter_ = 1;
};

```

LineProcessor.cpp

```

#include "FileProcessor.h"
void FileProcessor::Process(const std::vector<std::shared_ptr<Figure>>
&buf) {
    std::ofstream fs("Buffer_" + std::to_string(counter_++), std::ios::out

```

```

| std::ios::trunc);
    if (!fs) {
        throw std::runtime_error("File wasn't open");
    }
    for (const std::shared_ptr<Figure>& ptr : buf) {
        ptr->Print(fs);
        fs << "\n";
    }
}

```

Trapeze.h

```

#pragma once
#include "../Figure.h"
#include <exception>
class Trapeze : public Figure {
public:
    Trapeze(std::istream& is);
    Trapeze(Point p1, Point p2, Point p3, Point p4);
    void Print(std::ostream& os) const override;
private:
    Point p1_, p2_, p3_, p4_;
};

```

Trapeze.cpp

```

#include "Trapeze.h"
Trapeze::Trapeze(Point p1, Point p2, Point p3, Point p4)
: p1_(p1), p2_(p2), p3_(p3), p4_(p4){
    Vector v1(p1_, p2_), v2(p3_, p4_);
    if (v1 = Vector(p1_, p2_), v2 = Vector(p3_, p4_), is_parallel(v1, v2))
    {
        if (v1 * v2 < 0) {
            std::swap(p3_, p4_);
        }
    } else if (v1 = Vector(p1_, p3_), v2 = Vector(p2_, p4_),
is_parallel(v1, v2)) {
        if (v1 * v2 < 0) {
            std::swap(p2_, p4_);
        }
        std::swap(p2_, p3_);
    } else if (v1 = Vector(p1_, p4_), v2 = Vector(p2_, p3_),
is_parallel(v1, v2)) {
        if (v1 * v2 < 0) {
            std::swap(p2_, p3_);
        }
        std::swap(p2_, p4_);
        std::swap(p3_, p4_);
    } else {
        throw std::logic_error("At least 2 sides of trapeze must be
parallel");
    }
}

```

```

void Trapeze::Print(std::ostream& os) const {
    os << "Trapeze, p1: " << p1_ << ", p2: " << p2_ << ", p3: " << p3_ <<
    ", p4: " << p4_;
}
Trapeze::Trapeze(std::istream &is) {
    Point p1,p2,p3,p4;
    is >> p1 >> p2 >> p3 >> p4;
    *this = Trapeze(p1,p2,p3,p4);
}

```

Rhombus.h

```

#pragma once
#include "../Figure.h"
class Rhombus : public Figure {
public:
    Rhombus(std::istream& is);
    Rhombus(Point p1_, Point p2_, Point p3_, Point p4_);
    void Print(std::ostream& os) const override;
private:
    Point p1_, p2_, p3_, p4_;
};

```

Rhombus.cpp

```

#include "Rhombus.h"
Rhombus::Rhombus(Point p1, Point p2, Point p3, Point p4)
: p1_(p1), p2_(p2), p3_(p3), p4_(p4) {
    if (Vector(p1_, p2_).length() == Vector(p1_, p4_).length()
        && Vector(p3_, p4_).length() == Vector(p2_, p3_).length()
        && Vector(p1_, p2_).length() == Vector(p2_, p3_).length()) {
    } else if (Vector(p1_, p4_).length() == Vector(p1_, p3_).length()
        && Vector(p2_, p3_).length() == Vector(p2_, p4_).length()
        && Vector(p1_, p4_).length() == Vector(p2_, p4_).length()) {
        std::swap(p2_, p3_);
    } else if (Vector(p1_, p3_).length() == Vector(p1_, p2_).length()
        && Vector(p2_, p4_).length() == Vector(p3_, p4_).length()
        && Vector(p1_, p2_).length() == Vector(p2_, p4_).length()) {
        std::swap(p3_, p4_);
    } else {
        throw std::logic_error("This is not rhombus, sides arent equal");
    }
}
void Rhombus::Print(std::ostream& os) const {
    os << "Rhombus, p1: " << p1_ << ", p2: " << p2_ << ", p3: " << p3_ <<
    ", p4: " << p4_;
}
Rhombus::Rhombus(std::istream &is) {
    Point p1,p2,p3,p4;
    is >> p1 >> p2 >> p3 >> p4;
    *this = Rhombus(p1,p2,p3,p4);
}

```

Pentagon.h

```
#pragma once
#include "../Figure.h"
class Pentagon : public Figure {
public:
    Pentagon(std::istream& is);
    explicit Pentagon(const Point& p1, const Point& p2, const Point& p3,
const Point& p4, const Point& p5);
    void Print(std::ostream& os) const override;
private:
    Point p1_, p2_, p3_, p4_, p5_;
};
```

Pentagon.cpp

```
#include "Pentagon.h"
Pentagon::Pentagon(const Point& p1, const Point& p2, const Point& p3,
const Point& p4, const Point& p5)
    : p1_(p1), p2_(p2), p3_(p3), p4_(p4), p5_(p5) {}
void Pentagon::Print(std::ostream& os) const {
    os << "Pentagon, p1: " << p1_ << ", p2: " << p2_ << ", p3: " << p3_ <<
", p4: " << p4_ << ", p5: " << p5_;
}
Pentagon::Pentagon(std::istream &is) {
    Point p1, p2, p3, p4, p5;
    is >> p1 >> p2 >> p3 >> p4 >> p5;
    *this = Pentagon(p1,p2,p3,p4,p5);
}
```

7. Вывод

Выполняя данную работу, я узнал о возможностях асинхронного программирования в C++. Этот язык предоставляет возможность запускать функции асинхронно, создавать новые потоки, а так же синхронизировать их с помощью примитивов синхронизации (mutex, condition_variable) и оберток для них(unique_lock, lock_guard). Кроме того, я познакомился с библиотекой boost, в частности с ее фреймворком для юнит тестов.