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: Будникова Валерия Павловна
Группа: 80-207
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

ВАРИАНТ 31 Фигуры по варианту: треугольник, 6-угольник, 7-угольник.

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.

2. Описание программы

В программе реализован класс Figure и его наследники - Triangle, Hexagon, Octagon. В каждом классе есть конструкторы(определения координат вершин для каждой фигуры), также метод получения вектора координат, получения сторону фигуры и ее имени. Все эти методы реализованы аналогично 3-й лабораторной работе. Также реализован класс Factory, в котором есть методы создания фигур. По умолчанию размер буфера равен 1. Для хранения хранения лямбда функций я использовала std::function. Реализован шаблонный класс Server, который нужен для работы в Publish-Subscribe. В этом классе реализован шаблон Singleton, который позволяет использовать единственный экземпляр класса. Также в классе реализован метод добавления “подписчиков” класса - лямбда функций, которые добавляются в самом начале программы, вторая лямбда-функция захватывает переменную, которая определяет имя файла, куда будет записываться буффер, что обеспечивает уникальность файлов с буферами. В классе реализован метод добавления в очередь сообщений массива фигур. И метод исполнения лямбда-функций, в котором проверяется, пуста ли очередь, и, в случае, когда очередь не пуста, захватывается мьютекс выполняются лямбда-функции. Если очередь пуста, то управление передается основному потоку.

3. Руководство по использованию программы

Взаимодействие с пользователем происходит с помощью меню:

Введите:

1 - добавить треугольник

2 - добавить шестиугольник

3 - добавить восьмиугольник

4 - распечатать буфер

5 - завершить программу

4. Набор тестов и результаты работы программы

Описание: Программа запускается с аргументом 3, то есть задается буфер размера - три.

```
 Введите:
 1  - добавить треугольник
 2  - добавить шестиугольник
 3  - добавить восьмиугольник
 4  - распечатать буфер
 5  - завершить программу
Введите команду:1
Введите координаты верхней точки и длину стороны через пробел: 2 3 4
Введите команду:2
Введите координаты верхней точки и длину стороны через пробел: 3 4 5
```

```
Введите команду:3
Введите координаты верхней точки и длину стороны через пробел: 4 5 6
Triangle
 (2,3) (0,-0.464102) (4,-0.464102)
Hexagon
 (3,4) (-1.33013,1.5) (7.33013,1.5) (-1.33013,-3.5) (7.33013,-3.5) (3,-6)
Octagon
 (4,5) (9.54328,2.7039) (-1.54328,2.7039) (11.8394,-2.83938) (-3.83938,-2.83938)
(9.54328,-6.08655) (-1.54328,-6.08655) (4,-10.6788)

Буфер очищен и сохранен в файл 1.txt
Введите команду:4
Введите команду:1
Введите координаты верхней точки и длину стороны через пробел: 2 3 4
Введите команду:4
Triangle
 (2,3) (0,-0.464102) (4,-0.464102)
Введите команду:3
Введите координаты верхней точки и длину стороны через пробел: 4 5 6
Введите команду:3
Введите координаты верхней точки и длину стороны через пробел: 3 4 5
Triangle
 (2,3) (0,-0.464102) (4,-0.464102)
Octagon
 (4,5) (9.54328,2.7039) (-1.54328,2.7039) (11.8394,-2.83938) (-3.83938,-2.83938)
(9.54328,-6.08655) (-1.54328,-6.08655) (4,-10.6788)
Octagon
 (3,4) (7.6194,2.08658) (-1.6194,2.08658) (9.53281,-2.53281) (-3.53281,-2.53281)
(7.6194,-5.2388) (-1.6194,-5.2388) (3,-9.06563)

Буфер очищен и сохранен в файл 2.txt
Введите команду:4
Введите команду:5
```

6. Листинг программы

main.cpp

```
//Будникова Валерия Павловна М8О-207Б
//Вариант 31(Фигуры: треугольник, 6-угольник, 7-угольник)
//Задание:
//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.

```
#include <iostream>
#include <string>
#include <vector>
#include <functional>
#include <fstream>
#include <thread>
#include <unistd.h>
#include "figure.hpp"
#include "triangle.hpp"
#include "hexagon.hpp"
#include "octagon.hpp"
#include "Server.hpp"

using fig_type = std::vector<std::shared_ptr<Figure>>;

template<class A, class B, class C>
class Factory {
    public:
    Factory() {}
    ~Factory() {}
    std::shared_ptr<Figure> Triangle(double x1, double y1, int side) {
        return std::make_shared<A>(x1, y1, side);
    }
    std::shared_ptr<Figure> Hexagon(double x1, double y1, int side) {
        return std::make_shared<B>(x1, y1, side);
    }
    std::shared_ptr<Figure> Octagon(double x1, double y1, int side) {
        return std::make_shared<C>(x1, y1, side);
    }
};

void Menu(){
        std::cout << "Введите:\n 1    - добавить треугольник\n 2    - добавить шестиугольник\n 3  - добавить восьмиугольник\n";
    std::cout << " 4  - распечатать буфер \n 5  - завершить программу\n";
}

void Print(){
    std::cout << "Введите координаты верхней точки и длину стороны через пробел: ";
}


int main(int argc, char * argv[]) {
    int file = 1;
```

```
    Server<fig_type>::get().register_subscriber([] (fig_type &fig) {
        std::cout << fig << std::endl;
        std::cout.flush();
    });
    Server<fig_type>::get().register_subscriber([&file] (fig_type &fig) {
        std::ofstream f;
        std::string name = std::to_string(file);
        name.append(".txt");
        f.open(name.c_str());
        f << fig;
        f.close();
        ++file;
         std::cout << "Буфер очищен и сохранен в файл " << (file - 1) << ".txt" <<
std::endl;
        std::cout.flush();
    });
    std::thread thread([]() {
        Server<fig_type>::get().run();
    });
    int size_buf = 1;

    if (argc == 2) size_buf = atoi(argv[1]);
    double x1, y1;
    int side, m;
    fig_type fig;
    Factory<Triangle, Hexagon, Octagon> addfigure;
    Menu();
    std::cout << "Введите команду:";
    while (std::cin >> m && m < 5 && m > 0) {
        switch (m) {
            case 1: {
                Print();
                std::cin >> x1 >> y1 >> side;
                fig.push_back(addfigure.Triangle(x1, y1, side));
                break;
            }
            case 2: {
                Print();
                std::cin >> x1 >> y1 >> side;
                fig.push_back(addfigure.Hexagon(x1, y1, side));
                break;
            }
            case 3: {
                Print();
                std::cin >> x1 >> y1 >> side;
                fig.push_back(addfigure.Octagon(x1, y1, side));
                break;
            }
            case 4: {
                std::cout << fig;
                break;
            }
            default:
                break;
        }
```

```
        if (fig.size() == size_buf) {
            Server<fig_type>::get().publish(fig);
            fig.clear();
            sleep(1);
        }
        std::cout << "Введите команду:";
    }
    fig.clear();
    Server<fig_type>::get().publish(fig);
    thread.join();
}
```

Server.cpp

```
#pragma once
#include <iostream>
#include <queue>
#include <vector>
#include <functional>
#include <mutex>
#include <thread>

template <class T>
class Server {
    private:
        Server(){};
        std::vector<std::function<void(T &)>> subscribers;
        std::queue<T> message_queue;
        std::mutex mutex;
    public:
        using subscriber_type = std::function<void(T &)>;
        static Server &get() {
            static Server instance;
            return instance;
        }
        void register_subscriber(const subscriber_type &sub) {
            std::lock_guard<std::mutex> lock(mutex);
            subscribers.push_back(sub);
        }
        void publish(const T &msg) {
            std::lock_guard<std::mutex> lock(mutex);
            message_queue.push(msg);
        }
        void run() {
            while (!false) {
                if (!message_queue.empty()) {
                    std::lock_guard<std::mutex> lock(mutex);
                    T fig = message_queue.front();
                    if (fig.empty()) {
                        break;
                    }
                    message_queue.pop();
                    for (auto sub : subscribers)
```

```
                        try {
                            sub(fig);
                        } catch (std::exception &ex) {
                             std::cout << "Exception in subscriber:" << ex.what() <<
std::endl;
                        }
                } else {
                    std::this_thread::yield();
                }
            }
        }
};
```

figure.hpp

```
//Будникова Валерия Павловна М8О-207Б
//Вариант 31(Фигуры: треугольник, 6-угольник, 7-угольник)
#pragma once
#include <iostream>
#include <vector>
#include <string>

struct Pair {
    double x;
    double y;
    Pair(): x(0), y(0) {}
    Pair(double a, double b): x(a), y(b) {}
};

class Figure {
    protected:
        std::vector<Pair> points;
    public:
        Figure() {}
        Figure(double x1, double y1, int c) {
            points.emplace_back(Pair(x1,y1));
        }
        virtual std::string Name() {
            return "Point";
        }
        virtual int Get() {
            return 0;
        }

        virtual std::vector<Pair> Coord() {
            return points;
        }
};

std::ostream& operator<<(std::ostream &os, Pair p) {
    os << '(' << p.x << ',' << p.y << ')';
    return os;
}

std::ostream& operator<<(std::ostream &os, std::vector<Pair> pair) {
    for (auto p: pair) os << " " << p;
```

```
    return os;
}

std::ostream& operator<<(std::ostream &os, std::vector<std::shared_ptr<Figure>>
fig) {
    for (auto f: fig) os << f->Name() << std::endl << f->Coord() << std::endl;
    return os;
}
```

triangle.hpp

```
//Будникова Валерия Павловна М8О-207Б
//Вариант 31(Фигуры: треугольник, 6-угольник, 7-угольник)
#pragma once
#include <cmath>
#include "figure.hpp"

class Triangle: public Figure {
    private:
        int side;
    public:
        Triangle() : Figure(), side(0) {}

        Triangle(double x1, double y1, int c): side(c) {
            points.emplace_back(Pair(x1, y1));
            points.emplace_back(Pair(x1 - (double)side / 2, y1 - (double)side *
sqrt(3) / 2));
            points.emplace_back(Pair(x1 + (double)side / 2, y1 - (double)side *
sqrt(3) / 2));
        }

        std::string Name() override {
            return "Triangle";
        }

        int Get() override {
            return side;
        }

        std::vector<Pair> Coord() override {
            return points;
        }
};
```

hexagon.hpp

```
//Будникова Валерия Павловна М8О-207Б

//Вариант 31(Фигуры: треугольник, 6-угольник, 7-угольник)

#pragma once

#include <cmath>

#include "figure.hpp"


class Hexagon: public Figure {

    private:

        int side;

    public:

        Hexagon() : Figure(), side(0) {}
```

```
        Hexagon(double x1, double y1, int c): side(c) {
            points.push_back(Pair(x1, y1));
            points.push_back(Pair(x1 - (double)c * sqrt(3) / 2, y1 - (double) c /
2));
            points.push_back(Pair(x1 + (double)c * sqrt(3) / 2, y1 - (double) c /
2));
            points.push_back(Pair(x1 - (double)c * sqrt(3) / 2, y1 - (double) c / 2
- c));
            points.push_back(Pair(x1 + (double)c * sqrt(3) / 2, y1 - (double) c / 2
- c));
            points.push_back(Pair(x1, y1 - 2 * c));
        }

        std::string Name() override {
            return "Hexagon";
        }
        int Get() override {
            return side;
        }

        std::vector<Pair> Coord() override {
            return points;
        }
};
```

octagon.hpp

```
//Будникова Валерия Павловна М8О-207Б
//Вариант 31(Фигуры: треугольник, 6-угольник, 7-угольник)
#pragma once
#include <cmath>
#include "figure.hpp"

class Octagon: public Figure {
    private:
        int side;
    public:
        Octagon() : Figure(), side(0) {}
        Octagon(double x1, double y1, int c): side(c) {
            points.push_back(Pair(x1, y1));
            points.push_back(Pair(x1 + (double)c * cos(M_PI / 8), y1 -  (double)c *
sin(M_PI / 8)));
            points.push_back(Pair(x1 - (double)c * cos(M_PI / 8), y1 -  (double)c *
sin(M_PI / 8)));
            points.push_back(Pair(x1 + (double)c * sqrt(1 / sqrt(2) + 1), y1 -
(double)c * sqrt(1 / sqrt(2) + 1)));
```

```
            points.push_back(Pair(x1 - (double)c * sqrt(1 / sqrt(2) + 1), y1 -
(double)c * sqrt(1 / sqrt(2) + 1)));
            points.push_back(Pair(x1 + (double)c * cos(M_PI / 8), y1 - 2 *
(double)c * cos(M_PI / 8)));
            points.push_back(Pair(x1 - (double)c * cos(M_PI / 8), y1 - 2 *
(double)c * cos(M_PI / 8)));
            points.push_back(Pair(x1, y1 - 2 * (double)c * sqrt(1 / sqrt(2) + 1)));
        }
        std::string Name() override {
            return "Octagon";
        }

        int Get() override {
            return side;
        }

        std::vector<Pair> Coord() override {
            return points;
        }
};
```

makefile

```
CC=g++
CFLAGS=-std=c++17
OUTPUT=lab8

all:
	$(CC) $(CFLAGS) main.cpp -o $(OUTPUT)
run:
	./$(OUTPUT) 3
```

7. Выводы

При выполнении лабораторной работы я научилась работе с потоками. Также я повторила лямбда-функции и научилась работать с std::function. Реализовала класс Server и научилась работать с мьютексами в Си++ .

Список литературы

1. Презентация “Введение в мультипрограммирование ЛЕКЦИЯ 13” - Дзюба Д.В. (дата обращения: 16.12.20).
2. cppreference.com [Электронный ресурс]. URL: https://en.cppreference.com/w/ (дата обращения: 16.12.20).