Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Отчёт по Лабораторной работе №8
“Асинхронное программирование“
по курсу “Объектно-Объективное Программирование“
III Семестр

Студент:	Катермин В.С.
Группа:	М8О-208Б-18
Преподаватель:	Журавлёв А.А.
Оценка:	
Дата:	**28.12.19**

1. **Тема**: Асинхронное программирование в C++.
2. **Код программы:**

figure.h

```
#ifndef D_FIGURE_H
#define D_FIGURE_H

#include<iostream>
#include<fstream>

struct figure {
   virtual void read(std::istream& is) = 0;
   virtual void print(std::ostream& os) const = 0;
   virtual ~figure() = default;
};

struct point {
   int x, y;
};

#endif //D_FIGURE_H
```

square.h

```
#ifndef D_SQUARE_H
#define D_SQUARE_H

#include "figure.h"

#include<memory>
#include<array>

struct square : figure{
private:
   std::array<point, 4> vertices;
public:
   void read(std::istream& is) override;
   void print(std::ostream& os) const override;
};
#endif //D_SQUARE_H
```

square.cpp

```
#include "square.h"

#include<iostream>
#include<fstream>
#include<cassert>

void square::read(std::istream& is) {
      for (int i = 0; i < 4; i++) {
            is >> vertices[i].x >> vertices[i].y;
      }
       assert(((vertices[1].x - vertices[0].x)*(vertices[3].x - vertices[0].x))+((vertices[1].y -
vertices[0].y)*(vertices[3].y - vertices[0].y)) == 0);
       assert(((vertices[2].x - vertices[1].x)*(vertices[0].x - vertices[1].x))+((vertices[2].y -
vertices[1].y)*(vertices[0].y - vertices[1].y)) == 0);
       assert(((vertices[3].x - vertices[2].x)*(vertices[1].x - vertices[2].x))+((vertices[3].y -
vertices[2].y)*(vertices[1].y - vertices[2].y)) == 0);
       assert((vertices[1].x - vertices[0].x) == (vertices[0].y - vertices[3].y));
       assert((vertices[2].x - vertices[1].x) == (vertices[1].y - vertices[0].y));
       assert((vertices[3].x - vertices[2].x) == (vertices[2].y - vertices[1].y));
}
```

```
void square::print(std::ostream& os) const {
   os << "Square:" << std::endl;
      for (int i = 0; i < 4; i++) {
            os << vertices[i].x << ' ' << vertices[i].y << std::endl;
      }
}
```

rectangle.h

```
#ifndef D_RECTANGLE_H
#define D_RECTANGLE_H

#include "figure.h"

#include<memory>
#include<array>

struct rectangle : figure{
private:
   std::array<point, 4> vertices;
public:
   void read(std::istream& is) override;
   void print(std::ostream& os) const override;
};
#endif //D_RECTANGLE_H
```

rectangle.cpp

```
#include "rectangle.h"

#include<iostream>
#include<fstream>
#include<cassert>

void rectangle::read(std::istream& is) {
      for (int i = 0; i < 4; i++) {
            is >> vertices[i].x >> vertices[i].y;
      }
       assert(((vertices[1].x - vertices[0].x)*(vertices[3].x - vertices[0].x))+((vertices[1].y -
vertices[0].y)*(vertices[3].y - vertices[0].y)) == 0);
       assert(((vertices[2].x - vertices[1].x)*(vertices[0].x - vertices[1].x))+((vertices[2].y -
vertices[1].y)*(vertices[0].y - vertices[1].y)) == 0);
       assert(((vertices[3].x - vertices[2].x)*(vertices[1].x - vertices[2].x))+((vertices[3].y -
vertices[2].y)*(vertices[1].y - vertices[2].y)) == 0);
}
void rectangle::print(std::ostream& os) const {
   os << "Rectangle:" << std::endl;
      for (int i = 0; i < 4; i++) {
            os << vertices[i].x << ' ' << vertices[i].y << std::endl;
      }
}
```

trapezoid.h

```
#ifndef D_TRAPEZOID_H
#define D_TRAPEZOID_H

#include "figure.h"

#include<memory>
#include<array>

struct trapezoid : figure{
private:
```

```
    std::array<point, 4> vertices;
public:
    void read(std::istream& is) override;
    void print(std::ostream& os) const override;
};
#endif //D_TRAPEZOID_H
```

trapezoid.cpp

```
#include "trapezoid.h"

#include<iostream>
#include<fstream>
#include<cassert>

void trapezoid::read(std::istream& is) {
        for (int i = 0; i < 4; i++) {
                is >> vertices[i].x >> vertices[i].y;
        }
         assert(((vertices[1].x - vertices[0].x)*(vertices[3].y - vertices[2].y)) == ((vertices[3].x -
vertices[2].x)*(vertices[1].y - vertices[0].y)));
}
void trapezoid::print(std::ostream& os) const {
    os << "Trapezoid:" << std::endl;
        for (int i = 0; i < 4; i++) {
                os << vertices[i].x << ' ' << vertices[i].y << std::endl;
        }
}
```

factory.h

```
#ifndef D_FACTORY_H
#define D_FACTORY_H

#include<iostream>
#include<memory>

#include"figure.h"
#include"square.h"
#include"rectangle.h"
#include"trapezoid.h"

struct factory {
public:
        virtual std::unique_ptr<figure> build(std::istream& is) = 0;
        virtual ~factory() = default;
};

struct square_factory : factory {
        std::unique_ptr<figure> build(std::istream& is) override {
                std::unique_ptr<square> t_sqr;
                t_sqr = std::make_unique<square>();
                t_sqr->read(is);
                return std::move(t_sqr);
        }
};

struct rectangle_factory : factory {
        std::unique_ptr<figure> build(std::istream& is) override {
                std::unique_ptr<rectangle> t_rect;
                t_rect = std::make_unique<rectangle>();
                t_rect->read(is);
```

```
            return std::move(t_rect);
        }
};

struct trapezoid_factory : factory {
        std::unique_ptr<figure> build(std::istream& is) override {
            std::unique_ptr<trapezoid> t_trpz;
            t_trpz = std::make_unique<trapezoid>();
            t_trpz->read(is);
            return std::move(t_trpz);
        }
};

#endif //D_FACTORY_H
```

handler.h

```
#ifndef D_HANDLER_H
#define D_HANDLER_H

#include<vector>
#include<string>
#include<fstream>
#include"figure.h"

struct handler {
        virtual void execute(std::vector<std::unique_ptr<figure>>& figures) = 0;
        virtual ~handler() = default;
};

struct file_handler : handler {
        void execute(std::vector<std::unique_ptr<figure>>& figures) override {
            static int count_file = 0;
            std::string filename = "";
            ++count_file;

            filename = "file_" + std::to_string(count_file) + ".txt";
            std::ofstream file(filename);
            for (int i = 0; i < figures.size(); ++i) {
                        figures[i]->print(file);
            }
        }
};
struct console_handler : handler {
        void execute(std::vector<std::unique_ptr<figure>>& figures) override {
            for (int i = 0; i < figures.size(); ++i) {
                        figures[i]->print(std::cout);
            }
        }
};

#endif //D_HANDLER_H
```

main.cpp

```
#include <iostream>
#include <memory>
#include <vector>
#include <thread>
#include <mutex>
#include <future>
#include <condition_variable>
```

```
#include "figure.h"
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"
#include "factory.h"
#include "handler.h"

enum Commands{
   cmd_quit,
   cmd_sqr,
   cmd_rect,
   cmd_trpz,
};

void handle(std::vector<std::unique_ptr<figure>>& figures, int buffer_size, std::condition_variable& cv_mtx1,
std::condition_variable& cv_mtx2, std::mutex& mtx, bool& stop_thrd) {
	std::unique_lock<std::mutex> lock(mtx);
	cv_mtx2.notify_all();
	std::vector<std::unique_ptr<handler>> handlers;

	handlers.push_back(std::make_unique<file_handler>());
	handlers.push_back(std::make_unique<console_handler>());
	while (!(stop_thrd)) {
		cv_mtx1.wait(lock);
		//std::cout << figures.size() << std::endl;
		if (figures.size() != 0) {
			for (int i = 0; i < handlers.size(); ++i) {
				handlers[i]->execute(figures);
			}
		}
		figures.clear();
		cv_mtx2.notify_all();
	}
	return;
}
int main(int argc, char* argv[]) {
	if (argc != 2)
		return 1;
	std::condition_variable cv_mtx1;
	std::condition_variable cv_mtx2;
	std::vector<std::unique_ptr<figure>> figures;
	std::unique_ptr<factory> my_factory;
	std::mutex mtx;
	std::unique_lock<std::mutex> lock(mtx);
	int buffer_size, command;
	buffer_size = std::stoi(argv[1]);
	bool stop_thrd = false;
	std::thread handler(handle, std::ref(figures), buffer_size, std::ref(cv_mtx1), std::ref(cv_mtx2), ref(mtx),
std::ref(stop_thrd));
	cv_mtx2.wait(lock);
	while (true) {
		for (int i = 0; i < buffer_size; ++i) {
			std::cout << "1 - Square" << std::endl;
			std::cout << "2 - Rectangle" << std::endl;
			std::cout << "3 - Trapezoid" << std::endl;
			std::cin >> command;
			switch (command) {
			case cmd_sqr :
				my_factory = std::make_unique<square_factory>();
				figures.push_back(my_factory->build(std::cin));
				break;
			case cmd_rect :
				my_factory = std::make_unique<rectangle_factory>();
				figures.push_back(my_factory->build(std::cin));
```

```
                        break;
                case cmd_trpz :
                        my_factory = std::make_unique<trapezoid_factory>();
                        figures.push_back(my_factory->build(std::cin));
                        break;
                }
        }
        cv_mtx1.notify_all();
        cv_mtx2.wait(lock);
        std::cout << "The buffer is filled" << std::endl;
        std::cout << "Continue? 'y' - Yes 'n' - No" << std::endl;
        char answer;
        std::cin >> answer;
        if (answer != 'y')
                break;
    }
    stop_thrd = true;
    cv_mtx1.notify_all();
    lock.unlock();
    handler.join();
    return 0;
}
```

Makefile

```
cmake_minimum_required(VERSION 3.10)
project(lab8)

set(CMAKE_CXX_STANDARD 17)

find_package(Threads REQUIRED)
add_executable(lab8
  ./main.cpp
  ./square.cpp
  ./rectangle.cpp
  ./trapezoid.cpp)

target_link_libraries(lab8 Threads::Threads)
```

3. **Ссылка на репозиторий:**
   https://github.com/GitGood2000/oop_exercise_08

4. **Результаты выполнения тестов (testcases в этом случае неудобно использовать):**

Тест 1:

```
user@PSB133S01ZFH:~/3sem_projects/oop_exercise_08/build/$ ./lab8 1
1 - Square
2 - Rectangle
3 - Trapezoid
1
-1 1 0 2 1 1 0 0
Square:
-1 1
0 2
1 1
0 0
Continue? 'y' - Yes 'n' - No
y
1 - Square
2 - Rectangle
3 - Trapezoid
2
```

```
-1 1 1 3 2 2 0 0
Rectangle:
-1 1
1 3
2 2
0 0
Continue? 'y' - Yes 'n' - No
y
1 - Square
2 - Rectangle
3 - Trapezoid
3
-1 1 0 2 2 2 0 0
Trapezoid:
-1 1
0 2
2 2
0 0
Continue? 'y' - Yes 'n' - No
n
user@PSB133S01ZFH:~/3sem_projects/oop_exercise_08/build/$ ls
CMakeCache.txt  Makefile             file_1.txt  file_3.txt  lab8
CMakeFiles      cmake_install.cmake  file_2.txt
user@PSB133S01ZFH:/mnt/c/Users/User/Desktop/oop_exercise_08/build$ cat  file_1.txt
Square:
-1 1
0 2
1 1
0 0
user@PSB133S01ZFH:/mnt/c/Users/User/Desktop/oop_exercise_08/build$ cat file_2.txt
Rectangle:
-1 1
1 3
2 2
0 0
user@PSB133S01ZFH:/mnt/c/Users/User/Desktop/oop_exercise_08/build$ cat file_3.txt
Trapezoid:
-1 1
0 2
2 2
0 0
```

Тест 2:

```
user@PSB133S01ZFH:~/3sem_projects/oop_exercise_08/build/$ ./lab8 3
1 - Square
2 - Rectangle
3 - Trapezoid
1
-1 1 0 2 1 1 0 0
1 - Square
2 - Rectangle
3 - Trapezoid
2
-1 1 1 3 2 2 0 0
1 - Square
2 - Rectangle
3 - Trapezoid
3
-1 1 0 2 2 2 0 0
Square:
-1 1
0 2
1 1
0 0
```

```
Rectangle:
-1 1
1 3
2 2
0 0
Trapezoid:
-1 1
0 2
2 2
0 0
Continue? 'y' - Yes 'n' - No
n
user@PSB133S01ZFH:~/3sem_projects/oop_exercise_08/build/$ ls
CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake  file_1.txt  lab8
user@PSB133S01ZFH:~/3sem_projects/oop_exercise_08/build/$ cat file_1.txt
Square:
-1 1
0 2
1 1
0 0
Rectangle:
-1 1
1 3
2 2
0 0
Trapezoid:
-1 1
0 2
2 2
0 0
```

5. **Объяснение результатов работы программы:**
   1) При запуске программы мы вводим размер буфера, в который будем добавлять наши элементы.
   2) Выбираем фигуру, координаты которой будем вводить:
      A) 1 — Квадрат;
      B) 2 — Прямоугольник;
      C) 3 — Трапеция.
   3) Продолжаем вводить, пока буфер не будет заполнен. Когда буфер заполняется, то отправляет вектор с указателями на фигуры на обработку во второй поток. Этот поток создается при запуске основной программы и ожидает заполнения буфера. Перед тем как приступить к обработке второй поток ожидает освобождения мьютекса. О событии сообщает cv_mtx1.
   4) После обработки пользователю предлагается продолжить работу или завершить. При завершении работы обработка завершается, вектор очищается, мьютекс отпускается и работа программы прекращается.

6. **Вывод:** 1) Ознакомились с мьютексами в C++ и усвоили навык работы с ними; 2) Асинхронное программирование позволяет продуктивнее задействовать временной ресурс, так как одновременно на выполнение можно подать два потока, независящих друг от друга. Для стыковки результатов работы возможно использование мьютекса и условных переменных.