**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: Воронов Кирилл Михайлович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

**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. 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:
   1. Вывод информации о фигурах в буфере на экран;
   2. 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.

*Вариант 27 прямоугольник, трапеция ромб*

**2. Описание программы**

Создадим абстрактный класс Figure, чтобы наследовать от него классы других фигур (Rectangle, Romb, Trapeze) и иметь возможность хранить их вместе. Также реализуем класс Factory, который будет читать и создавать фигуры. Для работы с потоками будем использовать mutex и thread. Также реализуем класс Developer для очереди сообщений с двумя параметрами: самим сообщением и мьютексом. После заполнения вектора фигур поток в main зайдёт в цикл ожидания, пока второй поток будет выводить фигуры на экран и в файл. Генерация имён происходит с помощью функции mrandom.

**3. Набор тестов и результаты их выполнения**

*Тест 1*

kirill@kirill-VirtualBox:~/Рабочий стол/OOP/lab8$ ./oop\_exercise\_08 2

Введите 1, чтобы добавить прямоугольник

Введите 2, чтобы добавить ромб

Введите 3, чтобы добавить трапецию

1

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

0 0 5 6

Введите 1, чтобы добавить прямоугольник

Введите 2, чтобы добавить ромб

Введите 3, чтобы добавить трапецию

2

Введите координату нижней точки ромба и длину его диагоналей, начиная с горизонтальной

-1 5 6 9

------------------

Rectangle

(0,6) (5,6)

(0,0) (5,0)

------------------

Romb

(-1,14)

(-4,9.5) (2,9.5)

(-1,5)

Введите 1, чтобы добавить прямоугольник

Введите 2, чтобы добавить ромб

Введите 3, чтобы добавить трапецию

Создан файл с именем qr

*Тест 2*

kirill@kirill-VirtualBox:~/Рабочий стол/OOP/lab8$ ./oop\_exercise\_08 3

Введите 1, чтобы добавить прямоугольник

Введите 2, чтобы добавить ромб

Введите 3, чтобы добавить трапецию

1

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

0 -5 6 9

Введите 1, чтобы добавить прямоугольник

Введите 2, чтобы добавить ромб

Введите 3, чтобы добавить трапецию

2

Введите координату нижней точки ромба и длину его диагоналей, начиная с горизонтальной

3 6 9 9

Введите 1, чтобы добавить прямоугольник

Введите 2, чтобы добавить ромб

Введите 3, чтобы добавить трапецию

3

Введите координату нижней левой точки трапеции, наибольшее основание, наименьшее основание и длину высоты (трапеция равнобедренная)

6 5 9 2 3

------------------

Rectangle

(0,4) (6,4)

(0,-5) (6,-5)

------------------

Romb

(3,15)

(-1.5,10.5) (7.5,10.5)

(3,6)

------------------

Trapeze

(9.5,8) (11.5,8)

(6,5) (6,14)

Введите 1, чтобы добавить прямоугольник

Введите 2, чтобы добавить ромб

Введите 3, чтобы добавить трапецию

Создан файл с именем wame

**4. Листинг программы**

main.cpp

/\*

\* Воронов К.М. М8О-207Б-19

\* 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 производится в отдельном потоке;

\* 6. Реализовать два обработчика, которые должны обрабатывать

\* данные буфера:

\* 1. Вывод информации о фигурах в буфере на экран;

\* 2. 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.

Вариант 27 Прямоугольник ромб трапеция

\*/

#include<iostream>

#include <string>

#include <memory>

#include <vector>

#include <thread>

#include <functional>

#include"figure.hpp"

#include"publish\_subscribe.hpp"

using namespace std;

Developer < shared\_ptr<Figure> > d;

FILE \*file = NULL;

string mrandom() {

unsigned int size = rand() % 10;

string s = "";

char c;

for (int i = 0; i < size; ++i) {

c = rand() % 26 + 'a';

s += c;

}

return s;

}

void threadFunction() {

function<void(shared\_ptr<Figure> fig)> Print\_fig = [](shared\_ptr<Figure> fig) {

fig->Print();

};

function<void(shared\_ptr<Figure> fig)> Write\_fig = [](shared\_ptr<Figure> fig) {

fig->Write(file);

};

vector<function<void(shared\_ptr<Figure> fig)>> functions({Print\_fig, Write\_fig});

bool a = true;

while(a) {

if (!d.Empty()) {

if (d.Front() == NULL) {

break;

}

file = fopen((mrandom()).c\_str(), "wb");

while (!d.Empty()) {

shared\_ptr<Figure> f\_3 = d.Front();

d.Pop();

for (auto func : functions) {

func(f\_3);

}

}

fclose(file);

}

}

}

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

srand(time(NULL));

unsigned long long size = stoi(string(argv[1]));

if (size <= 0) {

printf("Неверный размер буффера\n");

return -1;

}

thread thread\_2(threadFunction);

vector< shared\_ptr<Figure> > figures;

unsigned long long id;

printf("Введите 1, чтобы добавить прямоугольник\n");

printf("Введите 2, чтобы добавить ромб\n");

printf("Введите 3, чтобы добавить трапецию\n");

while (cin >> id) {

if (!cin || id <= 0 || id > 3) {

printf("Вы ввели неверную команду\n");

thread\_2.join();

return -1;

}

if (id == Rec\_Id) {

figures.push\_back(Factory<double, Rectangle<double>>:: Create());

} else {

if (id == Romb\_Id) {

figures.push\_back(Factory<double, Romb<double>>:: Create());

} else {

if (id == Tr\_Id) {

figures.push\_back(Factory<double, Trapeze<double>>:: Create());

}

}

}

if (figures.size() == size) {

for (int i = 0; i < figures.size(); ++i) {

d.Add(figures[i]);

}

while (!d.Empty()) {}

figures.clear();

}

printf("Введите 1, чтобы добавить прямоугольник\n");

printf("Введите 2, чтобы добавить ромб\n");

printf("Введите 3, чтобы добавить трапецию\n");

}

for (int i = 0; i < figures.size(); ++i) {

d.Add(figures[i]);

}

figures.clear();

shared\_ptr<Figure> null = NULL;

d.Add(null);

thread\_2.join();

}

figure.hpp

#ifndef FIGURE\_HPP

#define FIGURE\_HPP

#include <iostream>

#include <memory>

using namespace std;

class Figure {

public:

virtual void Print() = 0;

virtual void Write(FILE\* out) = 0;

virtual ~Figure() {}

};

const unsigned long long Rec\_Id = 1;

const unsigned long long Romb\_Id = 2;

const unsigned long long Tr\_Id = 3;

template <typename T>

class Rectangle: public Figure{

public:

using type = T;

pair<T,T> t;

T a;

T b;

Rectangle(){};

Rectangle(T x,T y,T a1, T b1): t(x,y), a(a1), b(b1){};

void Print() override {

pair<double,double> t2(t.first + a, t.second );

pair<double,double> t3(t.first + a, t.second + b);

pair<double,double> t4(t.first, t.second + b);

cout << "------------------" << endl;

cout << "Rectangle" << endl;

cout << "(" << t4.first << "," << t4.second << ") "

<< "(" << t3.first << "," << t3.second << ")"

<< endl << endl << "(" << t.first << ","

<< t.second << ") " << "(" << t2.first << ","

<< t2.second << ")" << endl;

}

void Write(FILE\* out) override {

fwrite(&Rec\_Id, sizeof(unsigned long long), 1, out);

fwrite(&t.first, sizeof(T), 1, out);

fwrite(&t.second, sizeof(T), 1, out);

fwrite(&a, sizeof(T), 1, out);

fwrite(&b, sizeof(T), 1, out);

}

};

template <typename T>

class Romb: public Figure{

public:

using type = T;

pair<T,T> t;

T a;

T b;

Romb(T x,T y,T a1, T b1): t(x,y), a(a1), b(b1){};

void Print() override {

pair<T,T> t2(t.first,t.second+b);

pair<T,T> t3(t.first + a/2,t.second+b/2);

pair<T,T> t4(t.first - a/2,t.second+b/2);

cout << "------------------" << endl;

cout << "Romb" << endl;

cout << " (" << t2.first << "," << t2.second << ") " << endl

<< endl << "(" << t4.first << "," << t4.second << ") " << "("

<< t3.first << "," << t3.second << ")" << endl << endl << " ("

<< t.first << "," << t.second << ")" << endl;

}

void Write(FILE\* out) override {

fwrite(&Romb\_Id, sizeof(unsigned long long), 1, out);

fwrite(&t.first, sizeof(T), 1, out);

fwrite(&t.second, sizeof(T), 1, out);

fwrite(&a, sizeof(T), 1, out);

fwrite(&b, sizeof(T), 1, out);

}

};

template <typename T>

class Trapeze: public Figure{

public:

using type = T;

pair<T,T> t;

T a;

T b;

T c;

Trapeze(T x,T y,T a1, T b1, T s1): t(x,y), a(a1), b(b1), c(s1){};

void Print() override {

pair<T,T> t2(t.first,t.second + a);

pair<T,T> t3(t.first + (a - b)/2, t.second + c);

pair<T,T> t4(t2.first + a - (a - b)/2, t.second + c);

cout << "------------------" << endl;

cout << "Trapeze" << endl;

cout << " (" << t3.first << "," << t3.second << ") ("

<< t4.first << "," << t4.second << ")" << endl << endl

<< "(" << t.first << "," << t.second << ") ("

<< t2.first << "," << t2.second << ")" << endl;

}

void Write(FILE\* out) override {

fwrite(&Tr\_Id, sizeof(unsigned long long), 1, out);

fwrite(&t.first, sizeof(T), 1, out);

fwrite(&t.second, sizeof(T), 1, out);

fwrite(&a, sizeof(T), 1, out);

fwrite(&b, sizeof(T), 1, out);

fwrite(&c, sizeof(T), 1, out);

}

};

template<class T, class FIGURE>

class Factory;

template<class T>

class Factory<T, Rectangle<T> > {

public:

static shared\_ptr<Figure> Create() {

pair <T,T> c;

T a,b;

printf("Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной\n");

cin >> c.first >> c.second >> a >> b;

if((!cin)||(a<=0)||(b<=0)){

printf("Введены неверные данные, взяты значения по умолчанию: 0 0 1 1\n");

c.first = 0;

c.second = 0;

a = 1;

b = 1;

}

Rectangle<T> \* rect = new Rectangle<T>(c.first,c.second, a, b);

return shared\_ptr<Figure>(rect);

}

static shared\_ptr<Figure> Read(FILE\* in) {

pair <T,T> c;

T a,b;

fread(&c.first, sizeof(T), 1, in);

fread(&c.second, sizeof(T), 1, in);

fread(&a, sizeof(T), 1, in);

fread(&b, sizeof(T), 1, in);

Rectangle<T> \* rect = new Rectangle<T>(c.first,c.second, a, b);

return shared\_ptr<Figure>(rect);

}

};

template<class T>

class Factory<T, Romb<T> > {

public:

static shared\_ptr<Figure> Create() {

pair <T,T> c;

T a,b;

printf("Введите координату нижней точки ромба и длину его диагоналей, начиная с горизонтальной\n");

cin >> c.first >> c.second >> a >> b;

if((!cin)||(a<=0)||(b<=0)){

printf("Введены неверные данные, взяты значения по умолчанию: 0 0 1 1\n");

c.first = 0;

c.second = 0;

a = 1;

b = 1;

}

Romb<T> \* romb = new Romb<T>(c.first,c.second, a, b);

return shared\_ptr<Figure>(romb);

}

static shared\_ptr<Figure> Read(FILE\* in) {

pair <T,T> c;

T a,b;

fread(&c.first, sizeof(T), 1, in);

fread(&c.second, sizeof(T), 1, in);

fread(&a, sizeof(T), 1, in);

fread(&b, sizeof(T), 1, in);

Romb<T> \* romb = new Romb<T>(c.first, c.second, a, b);

return shared\_ptr<Figure>(romb);

}

};

template<class T>

class Factory<T, Trapeze<T> > {

public:

static shared\_ptr<Figure> Create() {

pair <T,T> c;

T a,b,d;

printf("Введите координату нижней левой точки трапеции, наибольшее основание, наименьшее основание и длину высоты (трапеция равнобедренная)\n");

cin >> c.first >> c.second >> a >> b >> d;

if((a<=0)||(b<=0)||(d<=0)){

printf("Введены неверные данные, взяты значения по умолчанию 0 0 5 2 4\n");

c.first = 0;

c.second = 0;

a = 10;

b = 5;

d = 5;

}

Trapeze<T> \* tr = new Trapeze<T>(c.first,c.second, a, b, d);

return shared\_ptr<Figure>(tr);

}

static shared\_ptr<Figure> Read(FILE\* in) {

pair <T,T> c;

T a,b,d;

fread(&c.first, sizeof(T), 1, in);

fread(&c.second, sizeof(T), 1, in);

fread(&a, sizeof(T), 1, in);

fread(&b, sizeof(T), 1, in);

fread(&d, sizeof(T), 1, in);

Trapeze<T> \* tr = new Trapeze<T>(c.first,c.second, a, b, d);

return shared\_ptr<Figure>(tr);

}

};

#endif

publish\_subscribe.hpp

#ifndef PUBLISH\_SUBSCRIBE\_HPP

#define PUBLISH\_SUBSCRIBE\_HPP

#include <mutex>

#include <queue>

using namespace std;

template<class T>

class Developer {

public:

queue<T> messages;

mutex m;

void Add(T & mes) {

m.lock();

messages.push(mes);

m.unlock();

}

bool Empty() {

m.lock();

bool a = messages.empty();

m.unlock();

return a;

}

T Front() {

m.lock();

T a = messages.front();

m.unlock();

return a;

}

void Pop() {

m.lock();

messages.pop();

m.unlock();

}

};

#endif

**5. Выводы**

В ходе выполнения лабораторной работы я познакомился со стандартными средствами асинхронного программирования на C++, узнал, что STL предоставляет объектно-ориентированную оболочку над системными вызовами POSIX, реализовал программу по асинхронной обработке данных.

**Список литературы**

Мануал std::mutex [Электронный ресурс]

URL: https://en.cppreference.com/w/cpp/thread/mutex (дата обращения 25.12.2020)

Мануал std::thread [Электронный ресурс]

URL: <https://en.cppreference.com/w/cpp/thread/thread> (дата обращения 25.12.2020)