**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: Терво Александр Александрович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

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 данным 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 1. Квадрат, прямоугольник, треугольник.

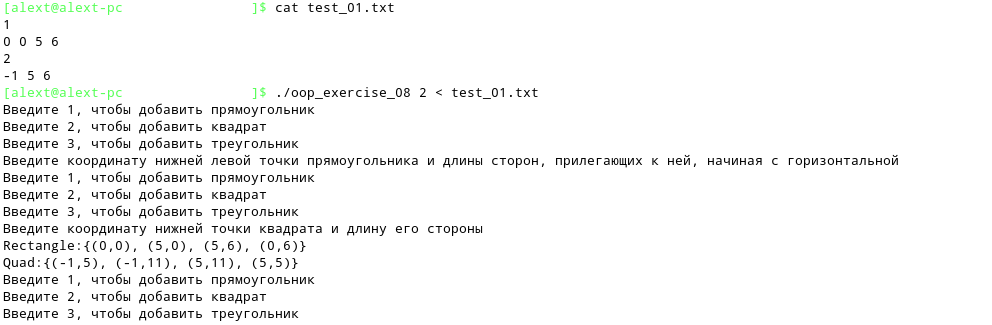
1. Описание программы

Используем классы фигур и TFactory из предыдущей лабораторной работы. Создадим отдельный класс TPubSubMQ, который будет моделировать очередь сообщений согласно шаблону «Publish and Subscribe». Так как очередь используется несколькими потоками, то нужно защищать все операции с очередью мьютексом std::mutex. Для асинхронной обработки испольуем std::thread, который будет обращаться к общей очереди сообщений. Поток обработки будет активно ждать сообщения, а основной поток будет ждать, пока сообщение не будет прочитано. Функции печати фигур на экран и в файл реализованы с помощью лямбда-функций и хранятся в в контейнере std::vector. В задании требуется сохранять каждый буфер фигур в уникальный файл, поэтому будем случайно генерировать название файла с помощью функции genFileName. Если в конце работы программы буфер имеет размер меньше заданного, то он всё равно обрабатывается потоком.

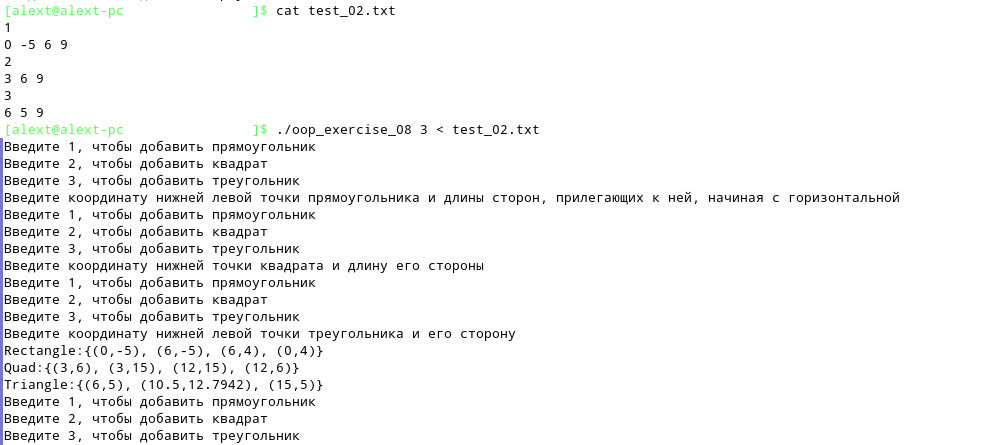
1. Набор тестов и результат их выполнения

Программа запускается с аргументом командной строки — размером буфера для асинхронной обработки. Программа обрабатывает строки до конца ввода. На каждой строке располагается тип фигуры и её данные.

Тест №1

Был создан файл с именем mvjbfxak.

Тест №2.

Был создан файл с именем adgf.

1. Листинг программы

**figure.hpp:**

#ifndef FIGURE\_HPP

#define FIGURE\_HPP

#include <iostream>

#include <memory>

#include <cmath>

using namespace std;

class Figure {

public:

virtual void Print() = 0;

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

virtual ~Figure() {}

};

const unsigned long long ID\_RECTANGLE = 1;

const unsigned long long ID\_QUAD = 2;

const unsigned long long ID\_TRIANGLE = 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 << "Rectangle:" << "{("

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

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

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

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

<< "}" << endl;

}

void Write(FILE\* out) override {

fwrite(&ID\_RECTANGLE, 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 Quad: public Figure{

public:

using type = T;

pair<T,T> t;

T a;

Quad(T x,T y,T a1): t(x,y), a(a1){};

void Print() override {

//cout << "PRINT IS HERE\n";

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

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

pair<T,T> t4(t.first + a, t.second);

cout << "Quad:" << "{("

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

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

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

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

<< "}" << endl;

//cout << "WRITE WORKED SUCCESSFULLY!\n";

}

void Write(FILE\* out) override {

//cout << "WRITE IS HERE\n";

fwrite(&ID\_QUAD, 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);

//cout << "WRITE WORKED SUCCESSFULLY!\n";

}

};

template <typename T>

class Triangle: public Figure{

public:

using type = T;

pair<T,T> t;

T a;

Triangle(T x,T y,T a1): t(x,y), a(a1){};

void Print() override {

pair<T,T> t2(t.first + a \* 0.5, t.second + a \* (std::pow(3, 0.5) / 2));

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

cout << "Triangle:" << "{("

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

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

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

<< "}" << endl;

}

void Write(FILE\* out) override {

fwrite(&ID\_TRIANGLE, 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);

}

};

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, Quad<T> > {

public:

static shared\_ptr<Figure> Create() {

pair <T,T> c;

T a;

printf("Введите координату нижней точки квадрата и длину его стороны\n");

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

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

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

c.first = 0;

c.second = 0;

a = 1;

}

Quad<T> \* quad = new Quad<T>(c.first,c.second, a);

return shared\_ptr<Figure>(quad);

}

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);

Quad<T> \* quad = new Quad<T>(c.first, c.second, a);

return shared\_ptr<Figure>(quad);

}

};

template<class T>

class Factory<T, Triangle<T> > {

public:

static shared\_ptr<Figure> Create() {

pair <T,T> c;

T a;

printf("Введите координату нижней левой точки треугольника и его сторону\n");

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

if(a<=0){

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

c.first = 0;

c.second = 0;

a = 10;

}

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

return shared\_ptr<Figure>(tr);

}

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

pair <T,T> c;

T a;

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

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

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

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

return shared\_ptr<Figure>(tr);

}

};

#endif

**pub\_sub.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

**main.cpp:**

#include<iostream>

#include <string>

#include <memory>

#include <vector>

#include <thread>

#include <functional>

#include"figure.hpp"

#include"pub\_sub.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) {

//cout << "THREAD 2: came to PRINT\n";

//cout << "THREAD 2: figure a is ";

fig->Print();

//cout << "THREAD 2: PRINT overcome\n";

};

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

//cout << "THREAD 2: came to WRITE\n";

fig->Write(file);

//cout << "THREAD 2: WRITE overcome\n";

};

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();

//cout << "THREAD 2: came to lambdas\n";

for (auto func : functions) {

func(f\_3);

}

//cout << "THREAD 2: lambdas overcome\n";

d.Pop();

}

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 квадрат\n");

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

while (cin >> id) {

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

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

thread\_2.join();

return -1;

}

if (id == ID\_RECTANGLE) {

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

} else {

if (id == ID\_QUAD) {

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

} else {

if (id == ID\_TRIANGLE) {

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

}

}

}

//cout << "THREAD 1: CAME TO BUFF LIMIT\n";

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

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

d.Add(figures[i]);

}

//cout << "THREAD 1: throwed all figures to queue, waiting...\n";

while (!d.Empty()) {}

//cout << "THREAD 1: waiting done, clearing...\n";

figures.clear();

//cout << "THREAD 1: clear success!\n";

}

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

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

printf("Введите 3, чтобы добавить треугольник\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();

}

1. 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.

1. Список литературы

1. Мануал std::mutex [Электронный ресурс]

URL: https://en.cppreference.com/w/cpp/thread/mutex

2. Мануал std::thread [Электронный ресурс]

URL: <https://en.cppreference.com/w/cpp/thread/thread>