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: Суханов Егор Алексеевич
Группа: 80-206
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:
    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 выводит данные на экран и запишет в файл.

Программа должна обрабатывать следующие равносторонние фигуры:

1. Ромб;
2. Пятиугольник;
3. Шестиугольник.

Сформулируем требования:

1. Буфер.
    a. Ограниченный размер, задается параметром командной строки;
    b. Ввод фигур через стандартный поток ввода;
    c. При заполнении буфера запускается асинхронная обработка фигур. Основной поток джет завершения обработки. Буфер должен очистится;
    d. Обработка происходит с помощью обработчиков. Следует использовать паттерн “Наблюдатель” (Publish-Subscribe).
2. Обработчики.
    a. Вывод информации о фигурах в поток вывода;
    b. Вывод информации о фигурах в файл. (имя файла должно быть уникальным и создаваться обработчиком).

Добавление фигуры будет происходить с помощью фабричного метода (см прошлую лабораторную работу).

Идея паттерна “Наблюдатель“ заключается в отделении кода, который должен выполняться при каком-то событии экземпляра класса, от кода этого класса. Данный шаблон можно применить, например, для выполнения каких-то действий по истечению таймера или, например, для системы достижений в компьютерной игре.

2. Описание программы

Программа будет иметь одну команду - добавление фигуры. Обработка команд стандартная (как и в других лабораторных работах).

Классы фигур возьмем из прошлых лабораторных работ. Это ускорит разработку программы.

Основная задача сводится к двум вещам:

1. Буфер. В него и будут добавляться фигуры. При заполнении будет создаваться поток, в котором будут вызываться все наблюдатели. Основной поток будет ждать завершения выполнения обработки, затем буфер будет обнулен;
2. Система наблюдателей. Будет интегрирована в класс буфера. Для добавления нового наблюдателя нужно использовать функцию Attach(Функтор, аргумент которого принимает указатель на экземпляр буфера). Когда буфер будет переполнен, в новом потоке он вызовет каждого наблюдателя.

3.

3. Набор тестов

test_01.txt - Тестирование добавления фигур, Программа запускается с аргументом 3.
add r 0 0 1 1
add p 0 0 1 1
add h 0 0 1 1
Результаты выполнения:
Ромб, координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Шестиугольник, координаты: (1,1) (-0.366025,1.36603) (-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Пятиугольник, координаты: (1,1) (-0.64204,1.26007) (-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)

test_02.txt - Проверка работы очистки буфера (размер буфера 3).
add r 0 0 1 1
add r 0 0 1 1
add r 0 0 1 1
add p 0 0 1 1
add p 0 0 1 1
add p 0 0 1 1
Результаты выполнения:
Ромб, координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Ромб, координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Ромб, координаты: (1,1) (-1,1) (-1,-1) (1,-1)

Пятиугольник, координаты: (1,1) (-0.64204,1.26007) (-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Пятиугольник, координаты: (1,1) (-0.64204,1.26007) (-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Пятиугольник, координаты: (1,1) (-0.64204,1.26007) (-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)

4. Листинг программы

Исходный код можно найти на github: https://github.com/Reterer/oop_exercise_08

main.cpp:

```
/*
  Лабораторная работа: 8
  Вариант: 21
  Группа: М8О-206Б-19
  Автор: Суханов Егор Алексеевич
```

```
  Создать приложение, которое будет считывать из стандартного ввода данные
фигур,
  согласно варианту задания, выводить их характеристики на экран и записывать
в файл.

  Сохранять введенные фигуры нужно в буфер. Размер которого задается
аргументом при запуске.
  При заполнении буфер, с помощью обработчиков выводит информацию о фигурах и
очищается.

  Фигуры:
      Ромб, 5-угольник, 6-уогльник
*/
#include <list>
#include <string>

#include <iostream>
#include <fstream>
#include <sstream>

#include <memory>
#include <functional>
#include <thread>

#include <iomanip>
#include <chrono>

#include "figure_factory.hpp"

class Buffer {
public:
      using shared figure ptr t = std::shared ptr<Figure>;
      using buffer t = std::list<shared figure ptr t>;
      using handler_t = std::function<void(const buffer_t&)>;
public:
      // Конструктор по умолчанию, максимальный размер равен 0.
      Buffer();
      // Конструктор, maxSize - максимальный размер буфера.
      Buffer(size_t maxSize);

      // Устанавилвает максимальный размер буфера.
      void SetMaxSize(size t size);
      // Возвращает максимальный размер буфера.
      size_t GetMaxSize();

      // Добавить фигуру figure в буффер.
      void Append(shared_figure_ptr_t figure);

      // Добавляет обработчик func.
      void Attach(handler_t handler);
private:
      // Выполняет всех наблюдателей в отдельном потоке.
      void Notify();

private:
      size t  maxSize;
      // Список фигур
```

```
	buffer_t  buffer;
	// Список обработчиков
	std::list<handler_t> _handlers;
};

Buffer::Buffer()
	: _maxSize(0), _buffer(), _handlers()
{}
Buffer::Buffer(size_t maxSize)
	: _maxSize(maxSize), _buffer(), _handlers()
{}

void Buffer::SetMaxSize(size_t size) {
	if (_buffer.size() >= size)
		Notify();

	_maxSize = size;
}

size_t Buffer::GetMaxSize() {
	return _maxSize;
}

void Buffer::Append(shared_figure_ptr_t figure) {
	if (_buffer.size() == _maxSize)
		Notify();
	 buffer.push_back(figure);
	if (_buffer.size() == _maxSize)
		Notify();
}
void Buffer::Attach(handler_t func) {
	_handlers.push_back(func);
}

void Buffer::Notify() {
	// Создаем поток и вызываем каждого обработчика.
	std::thread worker([this](){
		for (auto& handler : this-> handlers)
			handler(this->_buffer);
		});

	worker.join();
	// Обнуляем буфер.
	_buffer.clear();
}

void clear() {
	std::cin.clear();
	std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

void help() {
	std::cout <<
		"Использование: oop exercise_8 <max buffer size>\n"
		"Доступные команды:\n"
		"	help - выводит эту справку.\n"
		"	exit - выход;\n"
```

```
		"	add <r|p|h> <center> <vertex> - добавить фигуру
		(ромб|пяти|шестиугольник)\n"
		"			с координатами центра в <center> и первой
		вершиной в <vertex>;\n";
}

void add(std::shared_ptr<FigureMaker> figureMaker, Buffer& buffer) {
	try {
		if (Figure* figure = figureMaker->Make()) {
			buffer.Append(std::shared_ptr<Figure>(figure));
		}
		else {
			std::cout << "Фигура задана в неверном формате\n";
			clear();
		}
	}
	catch (std::invalid_argument& e) {
			std::cout << "Ошибка: " << e.what() << "\n";
	}

}

int main(int argc, char* argv[]) {
	setlocale(LC_ALL, "russian");

	Buffer buffer;

	// Обработчик, который выводит фигуры в поток вывода
	buffer.Attach([](const Buffer::buffer_t& buff) {
		for (auto& figure : buff)
			figure->Print(std::cout);
	});
	// Обработчик, который логирует фигуры в файл
	buffer.Attach([](const Buffer::buffer_t& buff) {
		auto timePoint =
		std::chrono::system_clock::to_time_t(std::chrono::system_clock::
		now());
		std::stringstream pathS;
		pathS << std::put_time(std::localtime(&timePoint), "%Y-%m-%d
		%H-%M-%S") << ".log";
		std::ofstream ofs(pathS.str(), std::fstream::out);
		if (!ofs) {
			std::cout << "Не могу открыть файл!\n";
			return;
		}
		for (auto& figure : buff)
			figure->Print(ofs);

		ofs.close();
	});

	if (argc != 2) {
		help();
		return 1;
	}

	{
		int64_t max_size;
```

```
            std::stringstream st(argv[1]);

            if (st >> max_size && max_size > 0 ) {
                  buffer.SetMaxSize(max_size);
            }
            else {
                  std::cout << "Ошибка ввода аргумента. Это должно быть
                  натуральное число!\n";
                  return 1;
            }
      }

      std::shared_ptr<FigureMaker> figureMaker(new
      FigureInputMaker(std::cin));

      std::string cmd;
      std::cout << '>';
      while (std::cin >> cmd) {
            // Обработка пользовательских команд
            if (cmd == "help") {
                  help();
            }
            else if (cmd == "exit") {
                  break;
            }
            else if (cmd == "add") {
                  add(figureMaker, buffer);
            }
            else {
                  std::cout << "Введена неизвестная команда\n";
                  clear();
            }
            std::cout << '>';
      }

      return 0;
}
```

figure_factory.hpp:

```
#pragma once
#include "figure.hpp"

class FigureMaker {
public:
      virtual Figure* Make() = 0;
};

class FigureInputMaker: public FigureMaker {
public:
      FigureInputMaker(std::istream& in);
      Figure* Make() override;
private:
      std::istream& in;
};
```

```
class FigureLoadMaker : public FigureMaker {
public:
	FigureLoadMaker(std::ifstream& ifs);
	Figure* Make() override;
private:
	std::istream& ifs;
};
```

figure_factory.cpp:

```
#include <string>
#include "figure_factory.hpp"

FigureInputMaker::FigureInputMaker(std::istream& in) : in{in} {}

Figure* FigureInputMaker::Make() {
	std::string figureType;
	Point center, vertex;
	if(!(in >> figureType >> center >> vertex))
		return nullptr;


	if(figureType == "r")
		return new Rhombus(center, vertex);
	if(figureType == "p")
		return new Pentagon(center, vertex);
	if(figureType == "h")
		return new Hexagon(center, vertex);

	return nullptr;
}

FigureLoadMaker::FigureLoadMaker(std::ifstream& ifs) : ifs{ ifs } {}
Figure* FigureLoadMaker::Make() {
	std::string figureType;
	Point center, vertex;

	{
		char type;
		bool success = true;
		success &= static_cast<bool>(ifs.read(&type, sizeof(type)));
		success &=
static_cast<bool>(ifs.read(reinterpret_cast<char*>(&center),
sizeof(center)));
		success &=
static_cast<bool>(ifs.read(reinterpret_cast<char*>(&vertex),
sizeof(vertex)));

		if (!success)
			return nullptr;

		figureType = type;
	}

	if (figureType == "r")
		return new Rhombus(center, vertex);
```

```
	if (figureType == "p")
		return new Pentagon(center, vertex);
	if (figureType == "h")
		return new Hexagon(center, vertex);

	return nullptr;
}
```

figure.hpp:

```
#pragma once
#include <fstream>
#include "point.hpp"

class Figure {
public:
	Figure(Point center, Point vertex);
	Point GetCenter();
	virtual void Print(std::ostream& out) = 0;

protected:
	Point center;
	Point vertex;
};

class Rhombus : public Figure {
public:
	Rhombus(Point center, Point vertex);
	virtual void Print(std::ostream& out) override;
};

class Pentagon : public Figure {
public:
	Pentagon(Point center, Point vertex);
	virtual void Print(std::ostream& out) override;

};

class Hexagon : public Figure {
public:
	Hexagon(Point center, Point vertex);
	virtual void Print(std::ostream& out) override;
};
```

figure.cpp:

```
#include <stdexcept>
#include <iostream>
#include <cmath>
#include "figure.hpp"

const double PI = 3.141592653589793;
```

```cpp
Figure::Figure(Point center, Point vertex)
{
	if (center == vertex)
		throw std::invalid_argument("center can be eq vertex");

	this->center = center;
	this->vertex = vertex;
}

Point Figure::GetCenter()
{
	return this->center;
}

template <int VERTEX_COUNT>
void _printCords(std::ostream& out, const Point center, const Point vertex)
{
	const double ANGLE = 2 * PI / VERTEX_COUNT; // угол между двумя соседними вершинами и центром
	out << "координаты: " << vertex;

	Point vec = vertex - center;
	for (int i = 2; i <= VERTEX_COUNT; ++i)
	{
		vec = vec.rotate(ANGLE);
		out << ' ' << center + vec;
	}
	out << std::endl;
}

template <int VERTEX_COUNT>
double _calcArea(const Point center, const Point vertex)
{
	Point vecRadius = vertex - center; // Вектор-радиус описанной окружности
	double sqRadius = vecRadius * vecRadius;	// Квадрат радуиса описанной окружности
	return VERTEX_COUNT / 2. * sqRadius * std::sin(2 * PI / VERTEX_COUNT);
}

Rhombus::Rhombus(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Rhombus::Print(std::ostream& out)
{
	out << "Ромб, ";
	_printCords<4>(out, center, vertex);
}

Pentagon::Pentagon(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Pentagon::Print(std::ostream& out)
{
	out << "Пятиугольник, ";
	_printCords<5>(out, center, vertex);
```

```
}

Hexagon::Hexagon(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Hexagon::Print(std::ostream& out)
{
	out << "Шестиугольник, ";
	_printCords<6>(out, center, vertex);
}
```

point.hpp:

```
#pragma once
#include <iostream>

struct Point {
	double x;
	double y;

	Point rotate(double radians);

	friend bool operator== (const Point a, const Point b);
	friend bool operator!= (const Point a, const Point b);

	friend Point operator+ (const Point a, const Point b);
	friend Point operator- (const Point a, const Point b);

	friend double operator* (const Point a, const Point b);
	friend Point operator* (const Point a, double b);

	friend std::ostream& operator<< (std::ostream& out, const Point& p);
	friend std::istream& operator>> (std::istream& in, Point& p);
};
```

point.cpp:

```
#include "point.hpp"

bool operator== (const Point a, const Point b)
{
	return a.x == b.x && a.y == b.y;
}
bool operator!= (const Point a, const Point b)
{
	return a.x != b.x || a.y != b.y;
}

Point operator+ (const Point a, const Point b)
{
	return { a.x + b.x, a.y + b.y };
}

Point operator- (const Point a, const Point b)
```

```
{
	return { a.x - b.x, a.y - b.y };
}

double operator*(const Point a, const Point b)
{
	return a.x * b.x + a.y * b.y;
}

Point operator*(const Point a, double b)
{
	return Point{ a.x * b, a.y * b };
}

std::istream& operator>>(std::istream& in, Point& p)
{
	double x, y;
	in >> x >> y;
	p = Point{ x, y };
	return in;
}

std::ostream& operator<< (std::ostream& out, const Point& p)
{
	out << '(' << p.x << ',' << p.y << ')';
	return out;
}

Point Point::rotate(double radians) {
	Point rotated;
	rotated.x = x * std::cos(radians) - y * std::sin(radians);
	rotated.y = x * std::sin(radians) + y * std::cos(radians);
	return rotated;
}
```

5. Выводы

Выполняя данную лабораторную работу я познакомился с потоками в c++, а так же паттерном “Наблюдатель”.

Потоки, в отличии от процессов, имеют общую память. Это значительно упрощает взаимодействие потоков с друг другом. Однако, с потоками надо работать очень осторожно. Например потоки, могут впасть в состояние взаимной блокировки. Или, например, один поток может обновлять какие-то данные, а другой поток в то же самое время может обратиться к ним, или, что хуже, что-то записать. Это называется гонка данных, в результате которой данные станут некорректными.

С помощью потоков можно ускорить выполнение какой-то задачи или, например, выполнить определенную задачу, при этом не останавливая выполнение остальной программы. Потоки используются почти во всех повседневных приложениях.

Шаблон “Наблюдатель” полезен, когда нужно выполнить какие-то действия в определенных условиях. Например, обновление GUI. Или, например, система достижения в видео играх.

6. Список литературы

1. Страуструп, Бьёрн. Язык программирования C++. Краткий курс, 2-е изд. : Пер. с англ. - СПб.: ООО "Диалектика", 2019. - 320 с.: ил. - Парал. тит. англ.;
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного проектирования. - СПб.: Питер, 2021. - 448 с.: ил. - (Серия “Библиотека программиста”).