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: Ивенкова Любовь Васильевна
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

Содержание

1. Постановка задачи

Вариант: 1
Задача:

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

Программа принимает в себя данные из консоли и из файла, при перенаправлении потока ввода вывода, и выполняет заданные действия. Также при запуске программы через терминал можно указать желаемое размер будущего буфера. При запуске появляется меню выбора операций, после выбора которой вводятся данные. В список операций входит *добавление фигуры* и *выход из программы*.

Программа состоит из четырех файлов: main.cpp, handler.h, factory.h и figures.h:

- figures.h - описание классов фигур. В начале файла прописаны “id” фигур, которые будут использоваться при записи и считывании документов. Класс Figure является виртуальным родительским. От него наследуются 3 шаблонных класса фигур: класс треугольника, класс прямоугольника и класс квадрата. Публичные поля содержат функцию вывода координат фигур и сохранение фигуры в файл.
- factory.h - содержит класс Factory, который создает графические примитивы фигур с помощью функции CreateFigure().

- handler.h - содержит класс Handler, который осуществляет работу обработчиков.
  Handler() - конструктор, который создает второй поток и запоминаем размер буфера.
  void Functions() - добавляет функции в вектор обработчиков.
  void Push() - добавление фигуры в буфер, проверка размера буфера. В зависимости от размера, выбирается дальнейшее действие.
  static void Printing() - выводит содержимое буфера на экран и в файл.

- main.cpp - главный файл. В нем создается объект класса Handler, задается размер буфера (размер по умолчанию - 2), а также задаются будущие функции обработчиков. С помощью switch считывается выбор пользователя, исходя из которого выбираются дальнейшие действия.

  void menu() - вывод меню.

Переменные классов

class Figure

нет переменных класса

class Triangle

using coords = std::pair<T, T>;
coords a, b, c - координаты треугольника;
size_t side - сторона правильного треугольника;

class Rectangle

using coords = std::pair<T, T>;
coords a, b, c, d - координаты прямоугольника;
size_t side, height - высота и ширина прямоугольника;

class Square

using coords = std::pair<T, T>;
coords a, b, c, d - координаты квадраты;
size_t side - сторона квадрата;

class Factory

нет переменных

class Handler

using TFig = std::shared_ptr<Figure>;
std::mutex mutex - базовый элемент синхронизации;
std::thread thread - новый поток для обработчиков;
std::condition_variable cv - условная переменная;
std::list<TFig> figures - буфер фигур;
std::vector<std::function<void(std::list<TFig>&)>> handlers - вектор обработчиков;
size_t max = 0 - размер буфера;
bool running - состояние работы потока.

3. Набор тестов

Таблица 1. Тест 1

1 sq 5	(при запуске программы ничего не указали, поэтому размер буфера по умолчанию стал 2); - добавление квадрата со стороной 5;

1 rec 7 8 1 rec 5 8 1 tr 8 0	- добавление прямоугольника со сторонами 7 и 8; - в буфере максимальное количество фигур (2), значит произойдет вывод буфера на экран и в файл; - добавление прямоугольника со стороной 5 и 8; - добавление треугольника со стороной 8; - в буфере максимальное количество фигур (2), значит произойдет вывод буфера на экран и в файл; - выход из программы

Таблица 2. Тест 2

	(при запуске программы указали размер буфера 3)
1 s 10	- добавление квадрата со стороной 10;
1 rectangle 8 9	- добавление прямоугольника со сторонами 8 и 9;
1 tr 8 1 tr 3 1 s 9 1 tr 8 0	- добавление треугольника со стороной 8; - в буфере максимальное количество фигур (3), значит произойдет вывод буфера на экран и в файл; - добавление треугольника со стороной 3; - добавление квадрата со стороной 9; - добавление треугольника со стороной 8; - в буфере максимальное количество фигур (3), значит произойдет вывод буфера на экран и в файл; - выход из программы;

4. Результат выполнения тестов

Тест 1:

```
Enter 0-1 to:
1 - add figure
0 - exit
1
Enter name of figure: sq
Enter side: 5
Done!
What's next?
1
Enter name of figure: rec
Enter side: 7 8
Square: (0, 0), (5, 0), (5, 5), (0, 5)
Rectangle: (0, 0), (7, 0), (7, 8), (0, 8)
Done!
```

What's next?
1
Enter name of figure: rec
Enter side: 5 8
Done!
What's next?
1
Enter name of figure: tr
Enter side: 8
Rectangle: (0, 0), (5, 0), (5, 8), (0, 8)
Triangle: (0, 0), (8, 0), (4, 6.9282)
Done!
What's next?
0
The program is closed, goodbye!

Тест 2:

Enter 0-1 to:
1 - add figure
0 - exit
1
Enter name of figure: s
Enter side: 10
Done!
What's next?
1
Enter name of figure: rectangle
Enter side: 8 9
Done!
What's next?
1
Enter name of figure: tr
Enter side: 8
Square: (0, 0), (10, 0), (10, 10), (0, 10)
Rectangle: (0, 0), (8, 0), (8, 9), (0, 9)
Triangle: (0, 0), (8, 0), (4, 6.9282)
Done!
What's next?
1
Enter name of figure: tr
Enter side: 3
Done!
What's next?
1
Enter name of figure: s
Enter side: 9
Done!

What's next?
1
Enter name of figure: tr
Enter side: 8
Triangle: (0, 0), (3, 0), (1.5, 2.59808)
Square: (0, 0), (9, 0), (9, 9), (0, 9)
Triangle: (0, 0), (8, 0), (4, 6.9282)
Done!
What's next?
0
The program is closed, goodbye!

5. Листинг программы

main.cpp

```
/* Ивенкова Любовь Васильевна, М8О-208Б-19

   https://github.com/Li-Iven/OOP/tree/main/oop_exercise_08 */

#pragma once
#include <condition_variable>
#include <mutex>
#include <shared_mutex>
#include <thread>
#include <list>
#include <functional>
#include <atomic>
#include <vector>
#include <iostream>

class Handler {
    using TFig = std::shared_ptr<Figure>;
    std::mutex mutex;
    std::thread thread;
    std::condition_variable cv;
    std::list<TFig> figures;
    std::vector<std::function<void(std::list<TFig>&)>> handlers;
    size_t max = 0;

public:
    bool running;

    Handler(size_t size) {
        this->max = size;
        running = true;
        thread = std::thread(Printing, this);
    };

    ~Handler() {
        running = false;
        cv.notify_one();
        thread.join();
    }

    void Functions(std::function<void(const std::list<TFig>&)>&& func) {
        handlers.push_back(func);
    }
```

```
    void Push(TFig el) {
        std::unique_lock<std::mutex> lk(mutex);
        figures.push_back(el);
        if (Full()) {
            cv.notify_one();
            cv.wait(lk, [this]() {
                return figures.empty();
                });
        }
    }

    bool Full() {
        return figures.size() == max;
    }

    static void Printing(Handler* t) {
        while (t->running) {
            std::unique_lock<std::mutex> lock(t->mutex);
            t->cv.wait(lock, [t]() {
                return t->Full() || !t->running;
                });
            for (auto& item : t->handlers) {
                item(t->figures);
            }

            t->figures.clear();
            lock.unlock();
            t->cv.notify_one();
        }
    }

};
```

factory.h

```
#pragma once
#include<memory>
#include"figures.h"

template<typename T>
class Factory {
public:
	static std::shared_ptr<Figure> CreateFigure(char* type) {
		std::shared_ptr<Figure> fig;
		std::cout << "Enter side: ";
		T side;
		if (type[0] == 's') {
			std::cin >> side;
			fig = std::make_shared<Square<T>>(side);
		}
		if (type[0] == 't') {
			std::cin >> side;
			fig = std::make_shared<Triangle<T>>(side);
		}
		if (type[0] == 'r') {
			T height;
			std::cin >> side >> height;
			fig = std::make_shared<Rectangle<T>>(side, height);
```

```
                }
                return fig;
        }
};
```

figures.h

```
#pragma once
#include<iostream>
#include <fstream>

class Figure {
public:
        virtual void Print() = 0;
        virtual void Write(std::ostream& file) = 0;
        virtual ~Figure() = default;
};

template<typename T>
class Square : public Figure {
private:
        using coords = std::pair<T, T>;
        coords a, b, c, d;
        size_t side;
public:
        Square(T s) :side(s) {
                b.first = c.second = c.first = d.second = s;
                a.first = a.second = b.second = d.first = 0;
        }
        Square() {}
        ~Square() {
                side = 0;
        }

        void Print() override {
                std::cout << "Square: ";
                std::cout << "(" << a.first << ", " << a.second << "), ";
                std::cout << "(" << b.first << ", " << b.second << "), ";
                std::cout << "(" << c.first << ", " << c.second << "), ";
                std::cout << "(" << d.first << ", " << d.second << ")" << std::endl;
        }

        void Write(std::ostream& file) override {
                file << "Square ";
                file << "{";
                file << "(" << a.first << ", " << a.second << "), ";
                file << " (" << b.first << ", " << b.second << "), ";
                file << " (" << c.first << ", " << c.second << "), ";
                file << " (" << d.first << ", " << d.second << ")";
                file << "}\n";
        }
};

template<typename T>
class Triangle : public Figure {
private:
        using coords = std::pair<T, T>;
        coords a, b, c;
        size_t side;

public:
```

```
	Triangle(size_t s) : side(s) {
		a.first = a.second = b.second = 0;
		b.first = s;
		c.second = s * sqrt(3) / 2;
		c.first = (double)s / 2;
	}
	Triangle() {}
	~Triangle() {
		side = 0;
	}

	void Print() override {
		std::cout << "Triangle: ";
		std::cout << "(" << a.first << ", " << a.second << "), ";
		std::cout << "(" << b.first << ", " << b.second << "), ";
		std::cout << "(" << c.first << ", " << c.second << ")" << std::endl;
	}

	void Write(std::ostream& file) override {
		file << "Triangle ";
		file << "{";
		file << "(" << a.first << ", " << a.second << "), ";
		file << " (" << b.first << ", " << b.second << "), ";
		file << " (" << c.first << ", " << c.second << ")";
		file << "}\n";
	}
};

template<typename T>
class Rectangle : public Figure {
private:
	using coords = std::pair<T, T>;
	coords a, b, c, d;
	size_t side, height;
public:
	Rectangle(T s, T h) :side(s), height(h) {
		a.first = a.second = b.second = d.first = 0;
		b.first = c.first = s;
		c.second = d.second = h;
	}

	Rectangle() {}

	~Rectangle() {
		side = 0;
		height = 0;
	}

	void Print() override {
		std::cout << "Rectangle: ";
		std::cout << "(" << a.first << ", " << a.second << "), ";
		std::cout << "(" << b.first << ", " << b.second << "), ";
		std::cout << "(" << c.first << ", " << c.second << "), ";
		std::cout << "(" << d.first << ", " << d.second << ")" << std::endl;
	}

	void Write(std::ostream& file) override {
		file << "Rectangle ";
		file << "{";
		file << "(" << a.first << ", " << a.second << "), ";
		file << " (" << b.first << ", " << b.second << "), ";
		file << " (" << c.first << ", " << c.second << "), ";
```

```
            file << " (" << d.first << ", " << d.second << ")";
            file << "}\n";
        }
};
```

handler.h

```
#pragma once
#include <condition_variable>
#include <mutex>
#include <shared_mutex>
#include <thread>
#include <list>
#include <functional>
#include <atomic>
#include <vector>
#include <iostream>

class Handler {
    using TFig = std::shared_ptr<Figure>;
    std::mutex mutex;
    std::thread thread;
    std::condition_variable cv;
    std::list<TFig> figures;
    std::vector<std::function<void(std::list<TFig>&)>> handlers;
    size_t max = 0;

public:
    bool running;

    Handler(size_t size) {
        this->max = size;
        running = true;
        thread = std::thread(Printing, this);
    };

    ~Handler() {
        running = false;
        cv.notify_one();
        thread.join();
    }

    void Functions(std::function<void(const std::list<TFig>&)>&& func) {
        handlers.push_back(func);
    }

    void Push(TFig el) {
        std::unique_lock<std::mutex> lk(mutex);
        figures.push_back(el);
        if (Full()) {
            cv.notify_one();
            cv.wait(lk, [this]() {
                return figures.empty();
                });
        }
    }

    bool Full() {
        return figures.size() == max;
    }
```

```
    static void Printing(Handler* t) {
        while (t->running) {
            std::unique_lock<std::mutex> lock(t->mutex);
            t->cv.wait(lock, [t]() {
                return t->Full() || !t->running;
                });
            for (auto& item : t->handlers) {
                item(t->figures);
            }

            t->figures.clear();
            lock.unlock();
            t->cv.notify_one();
        }
    }

};
```

6. Вывод

Благодаря данной лабораторной работе я научилась работать с потоками. Смогла реализовать поток обработчика, буфер обработчиков и организовать правильную работу двух потоков.

Список используемых источников

1. std::condition_variable [Электронный ресурс]. URL: https://en.cppreference.com/w/cpp/thread/condition_variable (Дата обращения: 19.04.2021).
2. Добро пожаловать в параллельный мир. Часть 1: Мир многопоточный [Электронный ресурс]. URL: http://scrutator.me/post/2012/04/04/parallel-world-p1.aspx (Дата обращения: 19.04.2021).
3. Потоки [Электронный ресурс]. URL: https://habr.com/ru/post/279653/ (Дата обращения: 19.04.2021).
4. std::mutex [Электронный ресурс]. https://en.cppreference.com/w/cpp/thread/mutex (Дата обращения: 19.04.2021).
5. Лямбда-выражения (анонимные функции) в C++ [Электронный ресурс]. https://ravesli.com/lyambda-vyrazheniya-anonimnye-funktsii-v-s/#toc-3 (Дата обращения: 27.12.2020).
6. Немного о многопоточном программировании [Электронный ресурс]. https://habr.com/ru/post/150801/ (Дата обращения: 19.04.2021).