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: Бирюков В. В.
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.
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 17: треугольник, квадрат, прямоугольник.

2. Описание программы

Классы “графических” примитивов Triangle, Square, Rectangle

наследуют абстрактый класс Figure, что позволяет единообразно их использовать и хранить в одной коллекции.

Ввод фигур осуществляется при помощи класса Factory.

Класс Processor осуществляет обработку буфера. Он содержит список обработчиков, которые явно добавляются методом add. Класс является функтором, что позволяет напрямую запускать его в поток. Главных поток уведомляет о начале обработки, вызывая метод publish, который помещает буфер в очередь. Затем главный поток ожидает, пока эта очередь не станет пустой, вызывая метод is_over. При этом, если эта очередь не пустая, в потоке обработки вызывается каждый обработчик от первого элемента очереди, а затем этот элемент удаляется из очереди.

Буфер представлен в виде std::vector.

Уникальные файлы для каждого буфера генерируются функцией get_filename, которая хранит количество уже обработанных буферов.

3. Набор тестов

Программа принимает на вход фигуры.

Для ввода фигур необходимо указать тип фигуры (`triangle, square, rectangle`) и данные о фигуре (координаты левой нижней вершины и длина стороны (сторон)) (фигуры правильные и одна из сторона считается параллельной оси Ox)

Тест 1:

```
$ ./oop_exercise_08 3
> square 0 0 1
> triangle 0 0 2
> rectangle 0 0 3 4
Processing...
Square:
(0, 0) (0, 1) (1, 1) (1, 0)
Triangle:
(0, 0) (1, 1.73205) (2, 0)
Rectangle:
(0, 0) (0, 4) (3, 4) (3, 0)
Processing is complete.
> triangle 1 1 1
> triangle 2 2 2
> triangle 3 3 3
Processing...
Triangle:
(1, 1) (1.5, 1.86603) (2, 1)
Triangle:
(2, 2) (3, 3.73205) (4, 2)
Triangle:
(3, 3) (4.5, 5.59808) (6, 3)
```

```
Processing is complete.
> square 5 5 1
> rectangle 5 5 1 2
> square 0 0 5
Processing...
Square:
(5, 5) (5, 6) (6, 6) (6, 5)
Rectangle:
(5, 5) (5, 7) (6, 7) (6, 5)
Square:
(0, 0) (0, 5) (5, 5) (5, 0)
Processing is complete.
> exit

// buffer0.txt
Square:
(0, 0) (0, 1) (1, 1) (1, 0)
Triangle:
(0, 0) (1, 1.73205) (2, 0)
Rectangle:
(0, 0) (0, 4) (3, 4) (3, 0)

// buffer1.txt
Triangle:
(1, 1) (1.5, 1.86603) (2, 1)
Triangle:
(2, 2) (3, 3.73205) (4, 2)
Triangle:
(3, 3) (4.5, 5.59808) (6, 3)

// buffer2.txt
Square:
(5, 5) (5, 6) (6, 6) (6, 5)
Rectangle:
(5, 5) (5, 7) (6, 7) (6, 5)
Square:
(0, 0) (0, 5) (5, 5) (5, 0)
```

Тест 2 (исключительные ситуации):

```
$ ./.oop_exercise_08
terminate called after throwing an instance of
'std::invalid_argument'
  what():  Buffer size expected
$ ./oop_exercise_08 0
terminate called after throwing an instance of
'std::invalid_argument'
  what():  Invalid buffer size
$ ./oop_exercise_08 1
> square 0 0 -1
Invalid square parameters
```

```
> cube
Unknown figure type
> exit
```

4. Листинг программы

```cpp
// figure.hpp
// Абстрактный класс фигуры.

#pragma once

#include <fstream>
#include <iostream>

class Figure {
public:
    virtual void print(std::ostream& = std::cout) = 0;
    virtual ~Figure() = default;
};

// triangle.hpp
// Треугольник. Хранит данные как координаты левой вершины и длину стороны.

#pragma once

#include <cmath>
#include <utility>
#include <fstream>
#include <iostream>
#include <stdexcept>

#include "figure.hpp"

template <typename T>
class Triangle : public Figure {
public:
    std::pair<T,T> x;
    T a;

    Triangle() = default;
    Triangle(T x1, T x2, T a) : x(x1,x2), a(a) {
       if (a <= 0) {
             throw std::invalid_argument("Invalid triangle parameters");
       }
    }
    ~Triangle() = default;

    void print(std::ostream& os = std::cout) override {
       os << "Triangle:\n"
             "(" << x.first << ", " << x.second << ") " <<
             "(" << x.first + 1.0 / 2 * a << ", " <<
             x.second + sqrt(3) / 2 * a << ") " <<
             "(" << x.first + a << ", " << x.second << ")" << std::endl;
    }


    template <class U>
    friend std::istream& operator>>(std::istream&, Triangle<U>&);
};
```

```cpp
template <class T>
std::istream& operator>>(std::istream& is, Triangle<T>& tr) {
    is >> tr.x.first >> tr.x.second >> tr.a;
    if (tr.a <= 0) {
       throw std::invalid_argument("Invalid triangle parameters");
    }
    return is;
}

// square.hpp
// Квадрат. Хранит данные как координаты левой нижней вершины и длину
стороны.

#pragma once

#include <utility>
#include <fstream>
#include <iostream>
#include <stdexcept>

#include "figure.hpp"

template <typename T>
class Square : public Figure {
public:
    std::pair<T,T> x;
    T a;

    Square() = default;
    Square(T x1, T x2, T a) : x(x1,x2), a(a) {
       if (a <= 0) {
             throw std::invalid_argument("Invalid square parameters");
       }
    }
    ~Square() = default;

    void print(std::ostream& os = std::cout) override {
       os << "Square:\n" <<
             "(" << x.first << ", " << x.second << ") " <<
             "(" << x.first << ", " << x.second + a << ") " <<
             "(" << x.first + a << ", " << x.second + a << ") " <<
             "(" << x.first + a << ", " << x.second << ")" << std::endl;
    }


    template <class U>
    friend std::istream& operator>>(std::istream&, Square<U>&);
};

template <class T>
std::istream& operator>>(std::istream& is, Square<T>& sq) {
    is >> sq.x.first >> sq.x.second >> sq.a;
    if (sq.a <= 0) {
       throw std::invalid_argument("Invalid square parameters");
    }
    return is;
}
```

```cpp
// rectangle.hpp
// Прямоугольник. Хранит данные как координаты левой нижней вершины и длины сторон.

#pragma once

#include <utility>
#include <fstream>
#include <iostream>
#include <stdexcept>

#include "figure.hpp"

template <typename T>
class Rectangle : public Figure {
public:
    std::pair<T,T> x;
    T a;
    T b;

    Rectangle() = default;
    Rectangle(T x1, T x2, T a, T b) : x(x1,x2), a(a), b(b) {
       if (a <= 0 || b <= 0) {
             throw std::invalid_argument("Invalid rectangle parameters");
       }
    }
    ~Rectangle() = default;

    void print(std::ostream& os = std::cout) override {
       os << "Rectangle:\n"
             "(" << x.first << ", " << x.second << ") " <<
             "(" << x.first << ", " << x.second + b << ") " <<
             "(" << x.first + a << ", " << x.second + b << ") " <<
             "(" << x.first + a << ", " << x.second << ")" << std::endl;
    }

    template <class U>
    friend std::istream& operator>>(std::istream&, Rectangle<U>&);
};

template <class T>
std::istream& operator>>(std::istream& is, Rectangle<T>& rect) {
    is >> rect.x.first >> rect.x.second >> rect.a >> rect.b;
    if (rect.a <= 0 || rect.b <= 0) {
       throw std::invalid_argument("Invalid rectangle parameters");
    }
    return is;
}

// factory.hpp
// Класс, создающий фигуры.

#pragma once

#include <memory>
#include <fstream>
#include <iostream>
#include <stdexcept>

#include "figure.hpp"
#include "triangle.hpp"
```

```cpp
#include "square.hpp"
#include "rectangle.hpp"

template <class T>
class Factory {
public:
    std::shared_ptr<Figure> create(std::string type, std::istream& is =
      std::cin) const {
       std::shared_ptr<Figure> figure;
       if (type == "triangle") {
             Triangle<T>* tr = new Triangle<T>;
             is >> *tr;
             figure =
                  std::shared_ptr<Figure>(reinterpret_cast<Figure*>(tr));
       }

       else if (type == "square") {
             Square<T>* sq = new Square<T>;
             is >> *sq;
             figure =
                  std::shared_ptr<Figure>(reinterpret_cast<Figure*>(sq));
       }

       else if (type == "rectangle") {
             Rectangle<T>* rect = new Rectangle<T>;
             is >> *rect;
             figure =
                  std::shared_ptr<Figure>(reinterpret_cast<Figure*>(rect));
       }

       else {
             throw std::runtime_error("Unknown figure type");
       }
       return figure;
    }

    Factory() = default;
    ~Factory() = default;
};

// processor.hpp
// Класс обработчиков

#pragma once

#include <vector>
#include <queue>
#include <list>
#include <functional>
#include <memory>
#include <mutex>
#include <stdexcept>

#include "figure.hpp"

using buffer_type = std::vector<std::shared_ptr<Figure>>;
using processor_type = std::function<void(const buffer_type&)>;

class Processor {
private:
    bool work;
```

```
    std::mutex mutex;
    std::queue<buffer_type> message_queue;
    std::list<processor_type> processors;

public:
    Processor() : work(false), mutex(), message_queue(), processors() {}

    void add(const processor_type &pr) {
       processors.push_back(pr);
    }

    void operator() () {
       work = true;

       while (work) {
             if (!message_queue.empty()) {
                   mutex.lock();
                   for (auto &processor: processors) {
                         try {
                               processor(message_queue.front());
                         } catch (const std::exception& ex) {
                               std::cerr << "Error in processor " <<
                                           ex.what() << std::endl;
                         }
                   }
                   message_queue.pop();
                   mutex.unlock();
             } else {
                   std::this_thread::yield();
             }
       }
    }

    void publish(buffer_type& buffer) {
       mutex.lock();
       message_queue.push(buffer);
       mutex.unlock();
    }

    void stop() {
       work = false;
    }

    bool is_over() {
       return message_queue.empty();
    }

};

// main.cpp
#include <stdexcept>
#include <memory>
#include <vector>
#include <string>
#include <iostream>
#include <thread>
#include <mutex>

#include "figure.hpp"
#include "factory.hpp"
#include "processor.hpp"
```

```cpp
using coord_type = int;

std::string get_filename() {
    static int count{0};
    return "buffer" + std::to_string(count++) + ".txt";
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
       throw std::invalid_argument("Buffer size expected");
    }
    unsigned long long buffer_size = std::strtoull(argv[1], nullptr, 10);
    if (buffer_size == 0) {
       throw std::invalid_argument("Invalid buffer size");
    }

    Factory<int> factory;

    buffer_type buffer;
    buffer.reserve(buffer_size);

    Processor processor;
    processor.add([](const buffer_type& buffer) {
       for (const std::shared_ptr<Figure>& figure: buffer) {
             figure->print();
       }
    });
    processor.add([](const buffer_type& buffer) {
       std::ofstream file(get_filename());
       if (file.fail()) {
             throw std::runtime_error("Error opening file");
       }
       for (const std::shared_ptr<Figure>& figure: buffer) {
             figure->print(file);
       }
       file.close();
    });

    std::thread processing_thread(std::ref(processor));

    std::string figure_type;
    std::cout << "> ";
    while (std::cin >> figure_type && figure_type != "exit") {
       try {
             buffer.push_back(factory.create(figure_type));

             if (buffer.size() == buffer_size) {
                   std::cout << "Processing..." << std::endl;
                   processor.publish(buffer);
                   while (!processor.is_over());
                   std::cout << "Processing is complete." << std::endl;

                   buffer.clear();
             }
       } catch (const std::exception& ex) {
             std::cerr << ex.what() << std::endl;
       }
       std::cout << "> ";
    }
```

```
    processor.stop();
    processing_thread.join();
}
```

5. Выводы

В ходе лабораторной работы я познакомился с мультипрограммированием, а также с реализовал многопоточную обработку данных, которую можно легко превратить в асинхронную.

Литература

1. Справочник по языку C++ [Электронный ресурс]. URL: https://ru.cppreference.com (дата обращения: 17.12.20).