**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: Ляшун Дмитрий Сергеевич

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:
7. Вывод информации о фигурах в буфере на экран;
8. 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.

**Вариант №24:**

|  |  |  |
| --- | --- | --- |
| **Фигура №1** | **Фигура №2** | **Фигура №3** |
| 8-угольник | Треугольник | Квадрат |

**2. Описание программы**

Для представления координат вершин фигур опишем шаблонную структуру **Point** с полями x и y.

Для реализации классов заданных фигур сперва опишем базовый шаблонный абстрактный класс **Figure** со следующими полями:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| square | T | Хранит площадь фигуры. |

Класс **Figure** будет иметь следующие методы:

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| Figure(int type) | Конструктор, создающий фигуру с типом type. |
| virtual void OutputInConsole() | Функция вывода фигуры в консоль, определяется только в классах-наследниках. |
| virtual void OutputInFile(FILE\* file) | Функция вывода фигуры в файл file, определяется только в классах-наследниках. |
| void SetSquare(T sq) | Изменяет значение площади фигуры на sq. |
| T GetSquare() | Возвращает значение площади фигуры. |

Классы **Triangle**, **Square** и **Octagon**, которые представляют треугольник, квадрат и 8-угольник соответственно, будут являться классами-наследниками от класса **Figure**. Данные фигуры будут представляться следующим образом:

1. Треугольник – через координаты своих вершин.
2. Квадрат – через координаты противоположных вершин.
3. Восьмиугольник – через координаты центра и вектора радиуса описанной окружности.

В этих классах не будут добавлены методы, а только переопределены виртуальные функции OutputInConsole() и OutputInFile(1) (перед выводом фигуры будет выводиться также тип фигуры) из класса-предка.

Для реализации шаблона Publish-Subscribe и обеспечения обмена сообщениями между главным потоком и потоком-обработчиком опишем шаблонный класс очереди сообщений **PubSubMQ** со следующими полями:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| messages | std::queue<T> | Хранит набор входящих сообщений в порядке их поступления. |
| mx | std::mutex | Мьютекс, обеспечивающий корректную одновременную работу с разными потоками, т.е. в любой момент времени осуществляет доступ к очереди сообщений только одному потоку. |

Методы класса **PubSubMQ**:

|  |  |
| --- | --- |
| **Метод** | **Назначение** |
| bool Empty() | Функция, проверяющая, пуста ли в данный момент очередь сообщений. |
| void Push(const T data) | Процедура вставки в очередь сообщений элемента data. |
| void Pop() | Процедура удаления элемента из начала очереди сообщений. |
| T Front() | Функция, возвращающая копию элемента из начала очереди. |

**3. Руководство по использованию программы**

**oop\_exercise\_08** – исполняемый файл программы.

**Входные данные**

При запуске программы в ее аргументах необходимо указать размер буфера с фигурами. При вводе самих фигур указывается их номер типа (1 – восьмиугольник, 2 – треугольник, 3 – квадрат), далее при вводе восьмиугольника – координаты центра и вектора описанной окружности, при вводе треугольника – координаты вершин, при вводе квадрата – координаты противоположных вершин. Для выхода из программы при вводе типа фигуры необходимо ввести 0.

**Выходные данные**

При переполнении буфера производится одновременный вывод фигур в консоль, а также в текстовый файл (при каждом переполнении буфера данные записываются в новый файл).

**Тестовые примеры**

dmitry@dmitry-VirtualBox:~/Work\_place/OOP\_labs/lab8$ ./main 3

Введите номер типа добавляемой фигуры (1 - Восьмиугольник, 2 - Треугольник, 3 - Квадрат, 0 - выход из программы).

> 1

Введите координаты центра восьмиугольника и вектора описанной окружности > 0 0 4 4

Фигура успешно добавлена.

> 2

Введите координаты вершин треугольника > 0 0 -2 0 -1 2

Фигура успешно добавлена.

> 3

Введите координаты противоположных вершин квадрата > 2 3 5 6

Фигура успешно добавлена.

Восьмиугольник: ( 0, 0 ), ( 4, 4 ), S = 90.5097

Треугольник: ( 0, 0 ), ( -2, 0 ), ( -1, 2 ), S = 1.73205

Квадрат: ( 2, 3 ), ( 5, 6 ), S = 9

> 3

Введите координаты противоположных вершин квадрата > 0 0 1 1

Фигура успешно добавлена.

> 3

Введите координаты противоположных вершин квадрата > 0 0 2 2

Фигура успешно добавлена.

> 3

Введите координаты противоположных вершин квадрата > 0 0 3 3

Фигура успешно добавлена.

Квадрат: ( 0, 0 ), ( 1, 1 ), S = 1

Квадрат: ( 0, 0 ), ( 2, 2 ), S = 4

Квадрат: ( 0, 0 ), ( 3, 3 ), S = 9

> 0

dmitry@dmitry-VirtualBox:~/Work\_place/OOP\_labs/lab8$ cd files

dmitry@dmitry-VirtualBox:~/Work\_place/OOP\_labs/lab8/files$ cat 0.txt

Восьмиугольник: 0.000000 0.000000 4.000000 4.000000

Треугольник: 0.000000 0.000000 -2.000000 0.000000 -1.000000 2.000000

Квадрат: 2.000000 3.000000 5.000000 6.000000

dmitry@dmitry-VirtualBox:~/Work\_place/OOP\_labs/lab8/files$ cat 1.txt

Квадрат: 0.000000 0.000000 1.000000 1.000000

Квадрат: 0.000000 0.000000 2.000000 2.000000

Квадрат: 0.000000 0.000000 3.000000 3.000000

dmitry@dmitry-VirtualBox:~/Work\_place/OOP\_labs/lab8/files$

**4. Листинг программы**

Исходный код программы состоит из трех файлов: **figures.hpp** – содержит реализацию классов **Figure**, **Square**, **Triangle**, **Octagon**; **pubsub.hpp** – содержит реализацию класса **PubSubMQ**; **main.cpp** – основная программа. **CMakeLists.txt** – содержит код для утилиты cmake, которая проводит сборку программы из исходного кода.

**Содержимое файла figures.hpp**

/\* Ляшун Дмитрий Сергеевич, М8О-207Б-19 \*/

#pragma once

#include <iostream>

#include <cmath>

#include <vector>

template<class T>

struct Point {

T x;

T y;

};

template<class T>

std::ostream& operator<<(std::ostream& os, const Point<T>& p) {

return os << "( " << p.x << ", " << p.y << " )";

}

template<class T>

bool operator==(const Point<T> first, const Point<T> second) {

return (first.x == second.x && first.y == second.y);

}

template<class T>

class Figure {

public :

virtual void OutputInConsole() = 0;

virtual void OutputInFile(FILE\* file) = 0;

T GetSquare();

void SetSquare(T sq);

private :

T square;

};

template<class T>

T Figure<T>::GetSquare() {

return square;

}

template<class T>

void Figure<T>::SetSquare(T sq) {

square = sq;

}

template<class T>

class Square : public Figure<T> {

public :

Square(const Point<T>& point1, const Point<T>& point2) : point1(point1), point2(point2) {

T square = abs(point1.x - point2.x);

square = pow(square, 2);

this->SetSquare(square);

}

virtual void OutputInConsole() override;

virtual void OutputInFile(FILE\* file) override;

private:

Point<T> point1, point2;

};

template<class T>

class Triangle : public Figure<T> {

public :

Triangle(const Point<T> point1, const Point<T> point2, const Point<T> point3) : point1(point1), point2(point2), point3(point3) {

T side = sqrt(pow(point1.x - point2.x, 2) + pow(point1.y - point2.y, 2));

T square = sqrt(3) \* pow(side, 2) / 4;

this->SetSquare(square);

}

virtual void OutputInConsole() override;

virtual void OutputInFile(FILE\* file) override;

private:

Point<T> point1, point2, point3;

};

template<class T>

class Octagon : public Figure<T> {

public :

Octagon(const Point<T>& center, const T vec\_x, const T vec\_y) : center(center), vec\_x(vec\_x), vec\_y(vec\_y) {

T side = pow(vec\_x, 2) + pow(vec\_y, 2);

T square = 4.0 \* sin(acos(-1) / 4.0) \* side;

this->SetSquare(square);

}

virtual void OutputInConsole() override;

virtual void OutputInFile(FILE\* file) override;

private :

Point<T> center;

T vec\_x, vec\_y;

};

template<class T>

void Square<T>::OutputInFile(FILE\* file) {

fprintf(file, "Квадрат: ");

fprintf(file, "%lf ", point1.x);

fprintf(file, "%lf ", point1.y);

fprintf(file, "%lf ", point2.x);

fprintf(file, "%lf\n", point2.y);

}

template<class T>

void Triangle<T>::OutputInFile(FILE\* file) {

fprintf(file, "Треугольник: ");

fprintf(file, "%lf ", point1.x);

fprintf(file, "%lf ", point1.y);

fprintf(file, "%lf ", point2.x);

fprintf(file, "%lf ", point2.y);

fprintf(file, "%lf ", point3.x);

fprintf(file, "%lf\n", point3.y);

}

template<class T>

void Octagon<T>::OutputInFile(FILE\* file) {

fprintf(file, "Восьмиугольник: ");

fprintf(file, "%lf ", center.x);

fprintf(file, "%lf ", center.y);

fprintf(file, "%lf ", vec\_x);

fprintf(file, "%lf\n", vec\_y);

}

template<class T>

void Square<T>::OutputInConsole() {

std::cout << "Квадрат: ";

std::cout << point1 << ", " << point2 << ", S = " << Figure<T>::GetSquare() << std::endl;

}

template<class T>

void Triangle<T>::OutputInConsole() {

std::cout << "Треугольник: ";

std::cout << point1 << ", " << point2 << ", " << point3 << ", S = " << Figure<T>::GetSquare() << std::endl;

}

template<class T>

void Octagon<T>::OutputInConsole() {

std::cout << "Восьмиугольник: ";

std::cout << center << ", ( " << vec\_x << ", " << vec\_y << " )" << ", S = " << Figure<T>::GetSquare() << std::endl;

}

**Содержимое файла pubsub.hpp**

/\* Ляшун Дмитрий Сергеевич, М8О-207Б-19 \*/

#pragma once

#include <queue>

#include <mutex>

template<class T>

class PubSubMQ {

private:

std::queue<T> messages;

std::mutex mx;

public:

explicit PubSubMQ() noexcept : messages(), mx() {}

~PubSubMQ() {}

bool Empty() {

mx.lock();

bool ans = messages.empty();

mx.unlock();

return ans;

}

void Push(const T data) {

mx.lock();

messages.push(data);

mx.unlock();

}

void Pop() {

mx.lock();

messages.pop();

mx.unlock();

}

T Front() {

mx.lock();

T ans = messages.front();

mx.unlock();

return ans;

}

};

**Содержимое файла main.cpp**

/\* Ляшун Дмитрий Сергеевич, М8О-207Б-19 \*/

/\* Задание: 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).

Вариант №24: Фигуры: 8-угольник, треугольник, квадрат. \*/

#include "figures.hpp"

#include "pubsub.hpp"

#include <iostream>

#include <stdlib.h>

#include <thread>

#include <memory>

#include <functional>

using workType = double;

using pointerType = std::shared\_ptr<Figure<workType > >;

PubSubMQ<std::vector<pointerType> > messages;

FILE\* file;

int fileCount = 0;

const std::string FOLDER = "files/";

void ThreadFunc() {

using functionType = std::function<void(pointerType)>;

functionType OutputInConsole = [](pointerType fg) {

fg->OutputInConsole();

};

functionType OutputInFile = [](pointerType fg) {

fg->OutputInFile(file);

};

std::vector<functionType> funcs({OutputInConsole, OutputInFile});

bool awake = true;

while (awake) {

if (!messages.Empty()) {

std::vector<pointerType> mes = messages.Front();

if (mes.empty()) {

awake = false;

break;

}

file= fopen((FOLDER+std::to\_string(fileCount)+std::string(".txt")).c\_str(), "w");

if (file == NULL) {

std::cout << "Ошибка создания файла!" << std::endl;

return;

}

++fileCount;

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

for (auto func : funcs) {

func(mes[i]);

}

}

fclose(file);

messages.Pop();

}

}

}

const int OCTAGON\_ID = 1;

const int TRIANGLE\_ID = 2;

const int SQUARE\_ID = 3;

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

if (argc != 2) {

std::cout << "Ошибка! При запуске программы необходимо указать размер буфера!" << std::endl;

return -1;

}

size\_t sizeBuffer;

try {

sizeBuffer = stoi(std::string(argv[1]));

}

catch (std::exception& e) {

std::cout << e.what() << std::endl;

return -1;

}

std::thread thr(ThreadFunc);

std::vector<pointerType> figures;

unsigned int type;

std::cout << "Введите номер типа добавляемой фигуры (1 - Восьмиугольник, 2 - Треугольник, 3 - Квадрат, 0 - выход из программы)." << std::endl;

while (true) {

std::cout << "> ";

std::cin >> type;

if (type == 0) {

break;

}

if (type == SQUARE\_ID) {

std::vector<workType> data(4);

std::cout << "Введите координаты противоположных вершин квадрата > ";

for (int i = 0; i < 4; ++i) {

std::cin >> data[i];

}

std::shared\_ptr<Square<workType> > sq(new Square<workType>({data[0], data[1]}, {data[2], data[3]}));

figures.push\_back(sq);

}

else if (type == TRIANGLE\_ID) {

std::cout << "Введите координаты вершин треугольника > ";

std::vector<workType> data(6);

for (int i = 0; i < 6; ++i) {

std::cin >> data[i];

}

std::shared\_ptr<Triangle<workType> > tr(new Triangle<workType>({data[0], data[1]}, {data[2], data[3]}, {data[4], data[5]}));

figures.push\_back(tr);

}

else if (type == OCTAGON\_ID) {

std::cout << "Введите координаты центра восьмиугольника и вектора описанной окружности > ";

std::vector<workType> data(4);

for (int i = 0; i < 4; ++i) {

std::cin >> data[i];

}

std::shared\_ptr<Octagon<workType> > oc(new Octagon<workType>({data[0], data[1]}, data[2], data[3]));

figures.push\_back(oc);

}

else {

std::cout << "Ошибка! Неверный ввод!" << std::endl;

continue;

}

std::cout << "Фигура успешно добавлена." << std::endl;

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

messages.Push(figures);

while (!messages.Empty()) {}

figures.clear();

}

}

messages.Push(figures);

figures.clear();

messages.Push(figures);

figures.clear();

thr.join();

return 0;

}

**Содержимое файла CMakeLists.txt**

cmake\_minimum\_required(VERSION 2.8) # Указание необходимой версии CMake

set(SOURCES main.cpp) # присвоение переменной SOURCES значения main.cpp

add\_executable(oop\_exercise\_08 ${SOURCES}) # Компиляция исполняемого файла oop\_exercise\_08

**5. Выводы**

В результате выполнения лабораторной работы я познакомился с основами асинхронного программирования в С++, получил практические навыки в параллельной обработке данных, а также в синхронизации потоков за счет использования технологии мьютекса.

**Список литературы**

1. Справочник по языку C++ [Электронный ресурс]. URL: [http://www.cplusplus.com/](about:blank) (дата обращения: 21.12.2020).

2. Archived Stack Overflow Documentation [Электронный ресурс]. URL: <https://www.riptutorial.com/> (дата обращения 21.12.2020).