**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: Юревич В.Ю.

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:

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.

Вариант 13.

Фигуры по варианту: ромб, пятиугольник, шестиугольник.

**Ссылка на репозиторий на GitHub:** https://github.com/vi-yurevich/oop\_exercise\_08

1. **Описание программы**

В программе реализован класс TFigure и его наследники - TRhombus, TPentagone, THexagone. В каждом классе есть конструкторы, определяющие координаты вершин для каждой фигуры, а также метод получения вектора координат, получения стороны фигуры и ее имени. Все эти методы реализованы аналогично 3-й лабораторной работе. Класс TFactory, реализован с помощью применения паттерна проектирования Singleton, позволяющего использовать единственный экземпляр класса для создания фигур. Отдельный класс TServer организует взаимодействие между "издателями" и "подписчиками" с помощью очереди. В связи с тем, что очередь используется двумя потоками, необходимо использовать mutex для защиты критической области. Класс TServer аналогично TFactory реализован на основе паттерна Singleton. “Подписчики” представляют собой лямбда-функции, хранящиеся с помощью std::function, а “публикации издателя” представлены очередью сообщений. В классе TServer реализован метод добавления в очередь сообщений массива фигур, а также метод выполнения лямбда-функций, в случае, когда имеются сообщения. Если очередь пуста, то управление передаётся основному потоку. В тех случаях, когда буфер на момент завершения программы имеет размер меньше заданного, он всё равно обрабатывается потоком. Каждый буфер сохраняется в уникальный файл, поэтому генерируется случайное имя файла, с помощью соответствующей лямбда-функции.

1. **Руководство по использованию программы**

Взаимодействие с пользователем происходит с помощью меню:

1 - Вывод меню

2 - Добавить ромб

3 - Добавить пятиугольник

4 - Добавить шестиугольник

5 - Вывод буфера

0 - Выход из программы

1. **Набор  тестов**

test\_01.txt:

2 //добавляем ромб

3

3

2

1

4 //добавляем шестиугольник

10

-1

2

4 //добавляем шестиугольник

60

61

22

3 //добавляем пятиугольник

-1

0

2

0 // выходим из программы, не заполнив буфер до конца

test\_02.txt:

./oop\_exercise\_08 0

1. **Результаты выполнения тестов**

test\_01.txt:

yurvch@yurvch-ASUS-EXPERTBOOK-P5440FA-P5440FA:~/MAI/OOP/exercise\_08$ ./oop\_exercise\_08 3

1 - Вывод меню

2 - Добавить ромб

3 - Добавить пятиугольник

4 - Добавить шестиугольник

5 - Вывод буфера

0 - Выход из программы

Выберете пункт меню: 2

Укажите координату Х центра фигуры: 3

Укажите координату Y центра фигуры: 3

Укажите длину вертикальной диагонали: 2

Укажите длину горизонтальной диагонали: 1

Выберете пункт меню: 4

Укажите координату Х центра фигуры: 10

Укажите координату Y центра фигуры: -1

Укажите длину стороны: 2

Выберете пункт меню: 4

Укажите координату Х центра фигуры: 60

Укажите координату Y центра фигуры: 61

Укажите длину стороны: 22

Ромб

(3 ; 4) (3.5 ; 3) (3 ; 2) (2.5 ; 3)

Шестиугольник

(9 ; 0.732051) (11 ; 0.732051) (12 ; -1) (11 ; -2.73205) (9 ; -2.73205) (8 ; -1)

Шестиугольник

(49 ; 80.0526) (71 ; 80.0526) (82 ; 61) (71 ; 41.9474) (49 ; 41.9474) (38 ; 61)

Буфер очищен и сохранен в файл 3mkkqgitohrzacaor.t

Выберете пункт меню: 3

Укажите координату Х центра фигуры: -1

Укажите координату Y центра фигуры: 0

Укажите длину стороны: 2

Выберете пункт меню: 0

Пятиугольник

(-1 ; 1.7013) (0.701302 ; 0) (0 ; -1.7013) (-2 ; -1.7013) (-2.7013 ; 0)

Буфер очищен и сохранен в файл 3gqgfrxsbhzbgyvnm.t

yurvch@yurvch-ASUS-EXPERTBOOK-P5440FA-P5440FA:~/MAI/OOP/exercise\_08$ ls

3gqgfrxsbhzbgyvnm.t  Figure.cpp  main.cpp  oop\_exercise\_08  test\_1.txt

3mkkqgitohrzacaor.t  Figure.h    makefile  PubSub.h        test\_2.txt

yurvch@yurvch-ASUS-EXPERTBOOK-P5440FA-P5440FA:~/MAI/OOP/exercise\_08$ cat 3gqgfrxsbhzbgyvnm.t

Пятиугольник

(-1 ; 1.7013) (0.701302 ; 0) (0 ; -1.7013) (-2 ; -1.7013) (-2.7013 ; 0)

test\_02.txt:

yurvch@yurvch-ASUS-EXPERTBOOK-P5440FA-P5440FA:~/MAI/OOP/exercise\_08$ ./oop\_exercise\_08 0

Размер буфера должен быть больше нуля.

1. **Листинг программы**

**main.cpp:**

#include <iostream>

#include <string>

#include <limits>

#include <memory>

#include <fstream>

#include <sstream>

#include <vector>

#include <thread>

#include <chrono>

#include "Figure.h"

#include "PubSub.h"

class TFactory {

public:

   static TFactory& Object() {

       static TFactory ob;

       return ob;

   }

   std::shared\_ptr<TFigure> Rhomb(double x0, double y0, double h, double v) {

       return std::make\_shared<TRhombus>(x0, y0, h, v);

   }

   std::shared\_ptr<TFigure> Pent(double x0, double y0, double s) {

       return std::make\_shared<TPentagone>(x0, y0, s);

   }

   std::shared\_ptr<TFigure> Hex(double x0, double y0 , double s) {

       return std::make\_shared<THexagone>(x0, y0, s);

   }

private:

   TFactory() {}

   TFactory(const TFactory& ob) = delete;

   TFactory& operator=(const TFactory&) = delete;

};

void AddFigureToVec(char type, std::vector<std::shared\_ptr<TFigure>>& vec) {

   bool key = false;

   double x0, y0, side, vert, hor;

   do {

       std::cout << "Укажите координату Х центра фигуры: ";

       std::cin >> x0;

       if (std::cin.fail()) {

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

           std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

       }

       else {

           key = true;

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

       }

   } while (key != true);

   do {

       std::cout << "Укажите координату Y центра фигуры: ";

       std::cin >> y0;

       if (std::cin.fail()) {

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

           std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

       }

       else {

           key = false;

           std::cin.clear();

           std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

       }

   } while (key != false);

   switch(type) {

       case 'r': {

           do {

               std::cout << "Укажите длину вертикальной диагонали: ";

               std::cin >> vert;

               if (std::cin.fail()) {

                   std::cin.clear();

                   std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

                   std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

               }

               else if (vert < 0) {

                           std::cout << "Ошибка ввода. Длина не может быть отрицательной. Повторите ввод.\n";

                       }

               else {

                   key = true;

                   std::cin.clear();

                   std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

               }

           } while(key != true);

           do {

               std::cout << "Укажите длину горизонтальной диагонали: ";

               std::cin >> hor;

               if (std::cin.fail()) {

               std::cin.clear();

               std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

               std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

               }

               else if (hor < 0) {

                           std::cout << "Ошибка ввода. Длина не может быть отрицательной. Повторите ввод.\n";

                       }

               else {

                   key = false;

                   std::cin.clear();

                   std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

               }

           } while(key != false);

           vec.push\_back(TFactory::Object().Rhomb(x0, y0, vert, hor));

           break;

       }

       case 'p': {

           do {

               std::cout << "Укажите длину стороны: ";

               std::cin >> side;

               if (std::cin.fail()) {

                   std::cin.clear();

                   std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

                   std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

               }

               else if (side < 0) {

                           std::cout << "Ошибка ввода. Длина не может быть отрицательной. Повторите ввод.\n";

                       }

               else {

                   key = true;

                   std::cin.clear();

                   std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

               }

           } while(key != true);

           vec.push\_back(TFactory::Object().Pent(x0, y0, side));

           break;

       }

       case 'h': {

           do {

               std::cout << "Укажите длину стороны: ";

               std::cin >> side;

               if (std::cin.fail()) {

                   std::cin.clear();

                   std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

                   std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

               }

               else if (side < 0) {

                           std::cout << "Ошибка ввода. Длина не может быть отрицательной. Повторите ввод.\n";

                       }

               else {

                   key = true;

                   std::cin.clear();

                   std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

               }

           } while(key != true);

           vec.push\_back(TFactory::Object().Hex(x0, y0, side));

           break;

       }

   }

}

std::ostream& operator<<(std::ostream &os, std::vector<std::shared\_ptr<TFigure>> fig) {

   for (auto f: fig) os << std::endl << f->Name() << std::endl << f->Coordinate() << std::endl;

   return os;

}

using fig\_t = std::vector<std::shared\_ptr<TFigure>>;

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

   unsigned long buffSize;

   int menu;

   bool key;

   fig\_t vec;

   if (argc < 2) {

       std::cout << "Необходимо в качестве параметра указать размер буфера." << std::endl;

       return(1);

   }

   else buffSize = atoi(argv[1]);

   std::cout << buffSize << std::endl;

   if (buffSize < 1) {

       std::cout << "Размер буфера должен быть больше нуля." << std::endl;

       return(1);

   }

   std::srand(time(NULL));

   TServer<fig\_t>::Get().AddSub([] (fig\_t &figure) {

       std::cout << figure << std::endl;

       std::cout.flush();

   });

   TServer<fig\_t>::Get().AddSub([&buffSize] (fig\_t &figure) {

       std::ofstream f;

       std::string name = std::to\_string(buffSize);

       for (short i = 0; i < 16; ++i) {

           name.push\_back('a' + std::rand() % 26);

       }

       name.append(".t");

       f.open(name.c\_str());

       f << figure;

       f.close();

       std::cout << "Буфер очищен и сохранен в файл " << name << std::endl;

       std::cout.flush();

   });

   std::thread thread([]() {

       TServer<fig\_t>::Get().Launch();

   });

   menu = 1;

   key = false;

   while(menu != 0) {

       switch(menu) {

           case 1:

               std::cout << "1 - Вывод меню\n2 - Добавить ромб\n3 - Добавить пятиугольник\n";

               std::cout << "4 - Добавить шестиугольник\n5 - Вывод буфера\n0 - Выход из программы\n";

               break;

           case 2:

               AddFigureToVec('r', vec);

               break;

           case 3:

               AddFigureToVec('p', vec);

               break;

           case 4:

               AddFigureToVec('h', vec);

               break;

           case 5:

               std::cout << vec;

               break;

       }

       if (vec.size() == buffSize) {

               TServer<fig\_t>::Get().MakePub(vec);

               vec.clear();

               std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

           }

       do {

           std::cout << "Выберете пункт меню: ";

           std::cin >> menu;

           if (std::cin.fail()) {

               std::cin.clear();

               std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

               std::cout << "Ошибка ввода. Повторите ввод." << std::endl;

           }

           else if ((menu < 0) || (menu > 11)) {

               std::cout << "Указанный пункт отсутствует в меню. Попробуйте ещё раз." << std::endl;

           }

           else {

               key = true;

               std::cin.clear();

               std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

           }

       } while(key != true);

       key = false;

   }

   TServer<fig\_t>::Get().MakePub(vec);

   vec.clear();

   TServer<fig\_t>::Get().MakePub(vec);

   thread.join();

   return(0);

}

**PubSub.h:**

#include <iostream>

#include <queue>

#include <vector>

#include <functional>

#include <mutex>

#include <thread>

template <class M>

class TServer {

   private:

       TServer(){};

       //TServer(const TServer& ob) = delete;

       //TServer& operator=(const TServer&) = delete;

       std::vector<std::function<void(M&)>> sub;

       std::queue<M> messageQue;

       std::mutex mutex;

   public:

       using sub\_t = std::function<void(M&)>;

       static TServer& Get() {

           static TServer ob;

           return ob;

       }

       void AddSub(const sub\_t &s) {

           std::lock\_guard<std::mutex> lock(mutex);

           sub.push\_back(s);

       }

       void MakePub(const M &m) {

           std::lock\_guard<std::mutex> lock(mutex);

           messageQue.push(m);

       }

       void Launch() {

           while (!false) {

               if (!messageQue.empty()) {

                   std::lock\_guard<std::mutex> lock(mutex);

                   M figure = messageQue.front();

                   if (figure.empty()) {

                       break;

                   }

                   messageQue.pop();

                   for (auto s : sub) s(figure);

               } else {

                   std::this\_thread::yield();

               }

           }

       }

};

**Figure.h:**

#include <iostream>

#include <queue>

#include <vector>

#include <functional>

#include <mutex>

#include <thread>

template <class M>

class TServer {

   private:

       TServer(){};

       //TServer(const TServer& ob) = delete;

       //TServer& operator=(const TServer&) = delete;

       std::vector<std::function<void(M&)>> sub;

       std::queue<M> messageQue;

       std::mutex mutex;

   public:

       using sub\_t = std::function<void(M&)>;

       static TServer& Get() {

           static TServer ob;

           return ob;

       }

       void AddSub(const sub\_t &s) {

           std::lock\_guard<std::mutex> lock(mutex);

           sub.push\_back(s);

       }

       void MakePub(const M &m) {

           std::lock\_guard<std::mutex> lock(mutex);

           messageQue.push(m);

       }

       void Launch() {

           while (!false) {

               if (!messageQue.empty()) {

                   std::lock\_guard<std::mutex> lock(mutex);

                   M figure = messageQue.front();

                   if (figure.empty()) {

                       break;

                   }

                   messageQue.pop();

                   for (auto s : sub) s(figure);

               } else {

                   std::this\_thread::yield();

               }

           }

       }

};

#endif //OOP\_EXERSICE\_8\_PUBSUB\_H

Figure.h:

#include <iostream>

#include <vector>

#include <string>

struct Pair {

   double x;

   double y;

   Pair();

   Pair(double a, double b);

   friend std::ostream& operator<<(std::ostream& os, const Pair& p);

};

std::ostream& operator<<(std::ostream& os, const Pair& p);

std::ostream& operator<<(std::ostream& os, const std::vector<Pair>& v);

class TFigure {

   protected:

       std::vector<Pair> points;

       Pair center;

   public:

       TFigure();

       TFigure(double x0, double y0);

       virtual std::string Name() = 0;

       virtual double Area() = 0;

       virtual Pair Center() = 0;

       virtual std::vector<Pair> Coordinate() = 0;

       virtual Pair Side() = 0;

};

class TRhombus : public TFigure {

   private:

       double verticalDiag;

       double horisontalDiag;

   public:

       TRhombus();

       TRhombus(double x0, double y0, double v, double h);

       std::string Name() override;

       double Area() override;

       Pair Center () override;

       std::vector<Pair> Coordinate() override;

       virtual Pair Side() override;

};

class TPentagone : public TFigure {

   private:

       double side;

   public:

       TPentagone();

       TPentagone(double x0, double y0, double s);

       std::string Name() override;

       double Area() override;

       Pair Center() override;

       std::vector<Pair> Coordinate() override;

       virtual Pair Side() override;

};

class THexagone : public TFigure {

   private:

       double side;

   public:

       THexagone();

       THexagone(double x0, double y0, double s);

       std::string Name() override;

       double Area() override;

       Pair Center() override;

       std::vector<Pair> Coordinate() override;

       virtual Pair Side() override;

};

**Figure.cpp:**

#include "Figure.h"

#include <cmath>

#include <vector>

Pair::Pair() : x(0.0), y(0.0) {}

Pair::Pair(double a, double b) : x(a), y(b) {}

std::ostream& operator<<(std::ostream& os, const Pair& p){

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

   return(os);

}

std::ostream& operator<<(std::ostream& os, const std::vector<Pair>& v) {

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

       os << v[i] << " ";

   }

   return(os);

}

TFigure::TFigure(double x0, double y0) {

   center.x = x0;

   center.y = y0;

}

TFigure::TFigure() : TFigure (0.0, 0.0) {}

TRhombus::TRhombus(double x0, double y0, double v, double h) {

   center.x = x0;

   center.y = y0;

   verticalDiag = v;

   horisontalDiag = h;

   points.push\_back(Pair(x0, y0 + v/2.0));

   points.push\_back(Pair(x0 + h/2.0, y0));

   points.push\_back(Pair(x0, y0 - v/2.0));

   points.push\_back(Pair(x0 - h/2.0, y0));

}

TRhombus::TRhombus() : TFigure() {

   verticalDiag = 0.0;

   horisontalDiag = 0.0;

}

std::string TRhombus::Name() {

   return("Ромб");

}

double TRhombus::Area() {

   return (verticalDiag \* horisontalDiag / 2.0);

}

Pair TRhombus::Center() {

   return(center);

}

std::vector<Pair> TRhombus::Coordinate() {

   return(points);

}

Pair TRhombus::Side() {

   return(Pair{verticalDiag, horisontalDiag});

}

TPentagone::TPentagone(double x0, double y0, double s) {

   center.x = x0;

   center.y = y0;

   side = s;

   double r = s \* sqrt(2.0) / sqrt(5.0 - sqrt(5.0));

   points.push\_back(Pair(x0, y0 + r));

   points.push\_back(Pair(x0 + r, y0));

   points.push\_back(Pair(x0 + s / 2.0, y0 - r));

   points.push\_back(Pair(x0 - s / 2.0, y0 - r));

   points.push\_back(Pair(x0 - r, y0));

}

TPentagone::TPentagone() : TFigure() {

   side = 0.0;

}

std::string TPentagone::Name() {

   return("Пятиугольник");

}

double TPentagone::Area() {

   double a = pow(side, 2.0) \* sqrt(5.0) \* sqrt(5.0 + 2.0 \* sqrt(5.0)) / 4.0;

   return(a);

}

Pair TPentagone::Center() {

   return(center);

}

std::vector<Pair> TPentagone::Coordinate() {

   return(points);

}

Pair TPentagone::Side() {

   return(Pair{side, 0.0});

}

THexagone::THexagone(double x0, double y0, double s) {

   center.x = x0;

   center.y = y0;

   side = s;

   points.push\_back(Pair(x0 - s / 2.0, y0 + sqrt(3.0) \* s / 2.0));

   points.push\_back(Pair(x0 + s / 2.0, y0 + sqrt(3.0) \* s / 2.0));

   points.push\_back(Pair(x0 + s, y0));

   points.push\_back(Pair(x0 + s / 2.0, y0 - sqrt(3.0) \* s / 2.0));

   points.push\_back(Pair(x0 - s / 2.0, y0 - sqrt(3.0) \* s / 2.0));

   points.push\_back(Pair(x0 - s, y0));

}

THexagone::THexagone() : TFigure() {

   side = 0.0;

}

std::string THexagone::Name() {

   return("Шестиугольник");

}

double THexagone::Area() {

   double a = 3.0 \* sqrt(3.0) \* pow(side, 2.0) / 2.0;

   return(a);

}

Pair THexagone::Center() {

   return(center);

}

std::vector<Pair> THexagone::Coordinate() {

   return(points);

}

Pair THexagone::Side() {

   return(Pair{side, 0.0});

}

1. **Выводы**

    В процессе выполнения данной работы получен опыт реализации паттерна Publisher-Subscriber, опыт работы с несколькими потоками С++. Попрактиковался в создании асинхронного кода

**Список литературы**

1. [cppreference.com](https://en.cppreference.com/) [Электронный ресурс]. URL: <https://en.cppreference.com/w/> (дата обращения: 23.12.20).