**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 8**

Тема: Асинхронное программирование

Студент: Савров Никита

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

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. В программе должно быть ровно два потока (thread). Один основной (main) и второй для

обработчиков;

9. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен

быть реализован как отдельный подписчик.

10. 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.

1. Описание программы

Для реализации задачи реализуем класс figure от которого наследуем классы rectangle rhomb и trapeze. Для реализации паттерна реализуем класс event который выполняет нужное действие. Для распределения ивентов и выполнения их для массива фигур реализуем класс meneger.

1. Набор testcases

test\_01

1

0 0

10 10

2

20 20

3

0 0

0 10

10 10

20 0

2

15 15

1

10 10

20 20

2

30 30

test\_02

1

0 0

15 15

2

10 10

3

0 0

0 11

12 11

15 0

2

15 15

2

20 20

1

10 10

25 25

test\_03

1

0 0

10 10

2

10 10

2

20 20

2

30 30

1

20 20

40 40

1. Результаты выполнения тестов.

test\_01

1 - add rectangle

2 - add rhomb

3 - trapeze

1

0 0

10 10

2

20 20

3

0 0

0 10

10 10

20 0

2

15 15

1

10 10

20 20

2

30 30

input 2 coordinates for 2 vertex

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

1 - add rectangle

2 - add rhomb

3 - trapeze

input 2 coordinats for 4 vertex

0 0

0 10

10 10

10 0

0 0

11.4715 16.383

27.8546 27.8546

16.383 11.4715

0 0

0 10

10 10

20 0

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

1 - add rectangle

2 - add rhomb

3 - trapeze

input 2 coordinates for 2 vertex

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

0 0

9.13142 11.9003

21.0317 21.0317

11.9003 9.13142

10 10

10 20

20 20

20 10

0 0

15 25.9808

40.9808 40.9808

25.9808 15

1 - add rectangle

2 - add rhomb

3 - trapeze

test\_02

1 - add rectangle

2 - add rhomb

3 - trapeze

1

0 0

15 15

2

10 10

3

0 0

0 11

12 11

15 0

2

15 15

2

20 20

1

10 10

25 25

input 2 coordinates for 2 vertex

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

1 - add rectangle

2 - add rhomb

3 - trapeze

input 2 coordinats for 4 vertex

rectangle

0 0

0 15

15 15

15 0

rhomb

0 0

6.42788 7.66044

14.0883 14.0883

7.66044 6.42788

trapeze

0 0

0 11

12 11

15 0

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

1 - add rectangle

2 - add rhomb

3 - trapeze

input 2 coordinates for 2 vertex

rhomb

0 0

9.13142 11.9003

21.0317 21.0317

11.9003 9.13142

rhomb

0 0

11.4715 16.383

27.8546 27.8546

16.383 11.4715

rectangle

10 10

10 25

25 25

25 10

1 - add rectangle

2 - add rhomb

3 - trapeze

test\_03

1 - add rectangle

2 - add rhomb

3 - trapeze

1

0 0

10 10

2

10 10

2

20 20

2

30 30

1

20 20

40 40

input 2 coordinates for 2 vertex

rectangle

0 0

0 10

10 10

10 0

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

rhomb

0 0

6.42788 7.66044

14.0883 14.0883

7.66044 6.42788

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

rhomb

0 0

11.4715 16.383

27.8546 27.8546

16.383 11.4715

1 - add rectangle

2 - add rhomb

3 - trapeze

input angle and side

rhomb

0 0

15 25.9808

40.9808 40.9808

25.9808 15

1 - add rectangle

2 - add rhomb

3 - trapeze

input 2 coordinates for 2 vertex

rectangle

20 20

20 40

40 40

40 20

1 - add rectangle

2 - add rhomb

3 - trapeze

5.Листинг программы

rectangle.h

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <algorithm>

#include <utility>

#include <iostream>

#include <vector>

#include <cmath>

#include "figure.h"

class Rectangle : public Figure{

public:

std::pair <double, double> a,b,c,d;

int number\_vertex = 4;

std::string name;

Rectangle(){

a.first = 0;

a.second = 0;

b.first = 0;

b.second = 0;

c.first = 0;

c.second = 0;

d.first = 0;

d.second = 0;

}

void Set (std::vector <std::pair<double,double>> cord){

a.first = cord[0].first;

a.second = cord[0].second;

b.first = cord[1].first;

b.second = cord[1].second;

c.first = cord[2].first;

c.second = cord[2].second;

d.first = cord[3].first;

d.second = cord[3].second;

name = "rectangle";

}

std::string GetName(){

return name;

}

std::vector <std::pair <double, double>> GetCoord(){

std::vector <std::pair <double, double>> v(4);

v[0].first = a.first;

v[0].second = a.second;

v[1].first = b.first;

v[1].second = b.second;

v[2].first = c.first;

v[2].second = c.second;

v[3].first = d.first;

v[3].second = d.second;

return v;

}

friend std::istream& operator>>(std::istream& in, Rectangle &obj){

std::vector <std::pair <double,double>> v(4);

in >> v[0].first >> v[0].second >> v[2].first >> v[2].second;

v[1].first = v[0].first;

v[1].second = v[2].second;

v[3].first = v[2].first;

v[3].second = v[0].second;

obj.Set(v);

return in;

}

friend std::ostream& operator<<(std::ostream& out, Rectangle &obj){

out << obj.a.first << " " << obj.a.second << '\n';

out << obj.b.first << " " << obj.b.second << '\n';

out << obj.c.first << " " << obj.c.second << '\n';

out << obj.d.first << " " << obj.d.second << '\n';

return out;

}

};

trapeze.h

#ifndef TRAPEZE\_H

#define TRAPEZE\_H

#include <algorithm>

#include <utility>

#include <iostream>

#include <vector>

#include <cmath>

#include "figure.h"

class Trapeze : public Figure{

public:

std::pair <double, double> a,b,c,d;

int number\_vertex = 4;

std::string name;

Trapeze(){

a.first = 0;

a.second = 0;

b.first = 0;

b.second = 0;

c.first = 0;

c.second = 0;

d.first = 0;

d.second = 0;

}

void Set (std::vector <std::pair<double,double>> cord){

a.first = cord[0].first;

a.second = cord[0].second;

b.first = cord[1].first;

b.second = cord[1].second;

c.first = cord[2].first;

c.second = cord[2].second;

d.first = cord[3].first;

d.second = cord[3].second;

name = "trapeze";

}

std::string GetName(){

return name;

}

std::vector <std::pair <double, double>> GetCoord(){

std::vector <std::pair <double, double>> v(4);

v[0].first = a.first;

v[0].second = a.second;

v[1].first = b.first;

v[1].second = b.second;

v[2].first = c.first;

v[2].second = c.second;

v[3].first = d.first;

v[3].second = d.second;

return v;

}

friend std::istream& operator>>(std::istream& in, Trapeze &obj){

std::vector <std::pair <double,double>> v(4);

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

in >> v[i].first >> v[i].second;

}

obj.Set(v);

return in;

}

friend std::ostream& operator<<(std::ostream& out, Trapeze &obj){

out << obj.a.first << " " << obj.a.second << '\n';

out << obj.b.first << " " << obj.b.second << '\n';

out << obj.c.first << " " << obj.c.second << '\n';

out << obj.d.first << " " << obj.d.second << '\n';

return out;

}

};

#endif

rhomb.h

#ifndef RHOMB\_H

#define RHOMB\_H

#include <algorithm>

#include <utility>

#include <iostream>

#include <vector>

#include <cmath>

#include "figure.h"

class Rhomb : public Figure{

public:

std::pair <double, double> a,b,c,d;

int number\_vertex = 4;

std::string name;

Rhomb(){

a.first = 0;

a.second = 0;

b.first = 0;

b.second = 0;

c.first = 0;

c.second = 0;

d.first = 0;

d.second = 0;

}

void Set (std::vector <std::pair<double,double>> cord){

a.first = cord[0].first;

a.second = cord[0].second;

b.first = cord[1].first;

b.second = cord[1].second;

c.first = cord[2].first;

c.second = cord[2].second;

d.first = cord[3].first;

d.second = cord[3].second;

name = "rhomb";

}

std::string GetName(){

return name;

}

std::vector <std::pair <double, double>> GetCoord(){

std::vector <std::pair <double, double>> v(4);

v[0].first = a.first;

v[0].second = a.second;

v[1].first = b.first;

v[1].second = b.second;

v[2].first = c.first;

v[2].second = c.second;

v[3].first = d.first;

v[3].second = d.second;

return v;

}

friend std::istream& operator>>(std::istream& in, Rhomb &obj){

std::vector <std::pair <double,double>> v(4);

int angle;

double side, pi = M\_PI;

std::cin >> angle >> side;

v[0].first = 0;

v[0].second = 0;

if (angle <= 90){

double alpha = (90 - angle) / 2;

v[3].second = sin(alpha \* pi / 180) \* side;

v[3].first = cos(alpha \* pi / 180) \* side;

v[1].first = sin(alpha \* pi / 180) \* side;

v[1].second = cos(alpha \* pi / 180) \* side;

v[2].second = sin(alpha \* pi / 180) \* side + v[1].second;

v[2].first = cos(alpha \* pi / 180) \* side + v[1].first;

} else if (angle < 180) {

double alpha = (270 - angle) / 2;

v[3].second = cos(alpha \* pi / 180) \* side;

v[3].first = sin(alpha \* pi / 180) \* side;

v[1].first = cos(alpha \* pi / 180) \* side;

v[1].second = sin(alpha \* pi / 180) \* side;

double d = sqrt(2 \* side \* side - 2 \* side \* side \* (cos((180 - alpha) \* pi / 180)));

v[2].first = cos(45 \* pi / 180) \* side;

v[2].second = cos(45 \* pi / 180) \* side;

} else {

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

v[i].first = 0;

v[i].second = 0;

}

}

obj.Set(v);

return in;

}

friend std::ostream& operator<<(std::ostream& out, Rhomb &obj){

out << obj.a.first << " " << obj.a.second << '\n';

out << obj.b.first << " " << obj.b.second << '\n';

out << obj.c.first << " " << obj.c.second << '\n';

out << obj.d.first << " " << obj.d.second << '\n';

return out;

}

};

#endif

figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

virtual void Set (std::vector <std::pair<double,double>> cord) = 0;

virtual std::string GetName() = 0;

virtual std::vector<std::pair<double, double>> GetCoord() = 0;

};

#endif

input.h

#ifndef INPUT\_H

#define INPUT\_H

bool IsCorrect(std::string check){

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

if (check[i] < '0' || check[i] > '9'){

return false;

}

}

return true;

}

void AddRectangle(std::vector <Figure\*> &buffer){

std::cout << "input 2 coordinates for 2 vertex\n";

std::vector<std::pair<double, double>> tmp(4);

std::string tmp1;

std::cin >> tmp1;

if (!IsCorrect(tmp1)){

std::cout << "not correct input\n";

return;

}

tmp[0].first = atof(tmp1.c\_str());

std::cin >> tmp1;

if (!IsCorrect(tmp1)){

std::cout << "not correct input\n";

return;

}

tmp[0].second = atof(tmp1.c\_str());

std::cin >> tmp1;

if (!IsCorrect(tmp1)){

std::cout << "not correct input\n";

return;

}

tmp[2].first = atof(tmp1.c\_str());

std::cin >> tmp1;

if (!IsCorrect(tmp1)){

std::cout << "not correct input\n";

return;

}

tmp[2].second = atof(tmp1.c\_str());

tmp[1].first = tmp[0].first;

tmp[1].second = tmp[2].second;

tmp[3].first = tmp[2].first;

tmp[3].second = tmp[0].second;

Rectangle temp;

temp.Set(tmp);

Figure \*fig1 = new Rectangle(temp);

buffer.push\_back(fig1);

}

void AddTrapeze(std::vector <Figure\*> &buffer){

std::cout << "input 2 coordinats for 4 vertex\n";

std::vector <std::pair<double, double>> tmp(4);

std::string tmp1,tmp2;

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

std::cin >> tmp1 >> tmp2;

if (!IsCorrect(tmp1) || !IsCorrect(tmp2)){

std::cout << "not correct input\n";

return;

}

tmp[i].first = atof(tmp1.c\_str());

tmp[i].second = atof(tmp2.c\_str());

}

Trapeze temp;

temp.Set(tmp);

Figure \*fig1 = new Trapeze(temp);

Figure \*fig2 = new Trapeze(temp);

buffer.push\_back(fig1);

}

void AddRhomb(std::vector <Figure\*> &buffer){

std::cout << "input angle and side\n";

std::vector <std::pair<double, double>> v(4);

std::string tmp1,tmp2;

double pi = M\_PI, angle,side;

std::cin >> tmp1 >> tmp2;

if (!IsCorrect(tmp1) || !IsCorrect(tmp2)){

std::cout << "not correct input\n";

return;

}

angle = atof(tmp1.c\_str());

side = atof(tmp2.c\_str());

v[0].first = 0;

v[0].second = 0;

if (angle <= 90 && side > 0){

double alpha = (90 - angle) / 2;

v[3].second = sin(alpha \* pi / 180) \* side;

v[3].first = cos(alpha \* pi / 180) \* side;

v[1].first = sin(alpha \* pi / 180) \* side;

v[1].second = cos(alpha \* pi / 180) \* side;

v[2].second = sin(alpha \* pi / 180) \* side + v[1].second;

v[2].first = cos(alpha \* pi / 180) \* side + v[1].first;

} else if (angle < 180 && side > 0) {

double alpha = (270 - angle) / 2;

v[3].second = cos(alpha \* pi / 180) \* side;

v[3].first = sin(alpha \* pi / 180) \* side;

v[1].first = cos(alpha \* pi / 180) \* side;

v[1].second = sin(alpha \* pi / 180) \* side;

double d = sqrt(2 \* side \* side - 2 \* side \* side \* (cos((180 - alpha) \* pi / 180)));

v[2].first = cos(45 \* pi / 180) \* side;

v[2].second = cos(45 \* pi / 180) \* side;

} else {

std::cout << "wrong input\n";

return;

}

Rhomb temp;

temp.Set(v);

Figure \*fig1 = new Rhomb(temp);

Figure \*fig2 = new Rhomb(temp);

buffer.push\_back(fig1);

}

#endif

meneger.h

#ifndef MENEGER\_H

#define MENEGER\_H

#include "event.h"

#include <mutex>

class Meneger { //meneger message

std::queue<Event> events;

std::string file\_names; //file for save

char name\_buff;

public:

Meneger(){

file\_names = "buffer";

name\_buff = 'a';

}

void AddEvent(Event ev){

events.push(ev);

}

void DoEvent(std::vector <Figure\*> v){

while(!events.empty()){ //do event save or print

Event ev = events.front();

events.pop();

if (ev.IsFile()){

std::string file\_name = file\_names;

file\_name.push\_back(name\_buff);

for (auto fig : v){

ev.Do(fig, file\_name);

}

name\_buff = name\_buff + 1;

} else {

for (auto fig : v){

ev.Do(fig,file\_names);

}

}

}

}

};

#endif

event.h

#ifndef EVENT\_H

#define EVENT\_H

#include <algorithm>

#include <utility>

#include <iostream>

#include <vector>

#include <stack>

#include <string>

#include <cmath>

#include "figure.h"

#include <fstream>

#include "trapeze.h"

#include "rectangle.h"

#include "rhomb.h"

#include <sstream>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <queue>

class Event {

std::string type\_event; //print or log

bool is\_file;

std::string file\_name; //file for save

public:

Event(std::string type){

type\_event = type;

if (type\_event == "print"){

is\_file = false;

file\_name = "NaN";

} else {

is\_file = true;

}

}

void Do(Figure \*fig, std::string file){

if (is\_file){ //print or save

file\_name = file;

std::ofstream oss;

oss.open(file\_name, std::ios::app);

if (!oss.is\_open()){

std::cout << "cant open file\n";

return;

}

std::vector <std::pair< double, double>> v(4);

v = fig->GetCoord();

oss << fig->GetName() << '\n';

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

oss << v[i].first << " " << v[i].second << '\n';

}

} else {

fig->Print();

}

}

bool IsFile(){

return is\_file;

}

};

#endif

main.cpp

// Лабораторная работа №8 Савров Никита М80-207Б-18

// 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. В программе должно быть ровно два потока (thread). Один основной (main) и второй для

// обработчиков;

// 9. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен

// быть реализован как отдельный подписчик.

// 10. 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.

#include "meneger.h"

#include <thread>

#include "input.h"

void Subscribe(std::vector <Figure\*> &buffer, Meneger &man){ //method for do event/subscribe

Event ev1("print");

Event ev2("log");

man.AddEvent(ev1);

man.AddEvent(ev2);

man.DoEvent(buffer);

}

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

int buff\_size;

if (argc < 2){

buff\_size = 3;

} else

buff\_size = std::atoi(argv[1]); //argument cmd

std::vector <Figure\*> buffer;

std::string input;

Meneger man;

std::cout << "1 - add rectangle\n2 - add rhomb\n3 - trapeze\n";

while(std::cin >> input){

if (input == "1"){

AddRectangle(buffer);

} else if (input == "2"){

AddRhomb(buffer);

} else if (input == "3"){

AddTrapeze(buffer);

} else {

std::cout << "check menu\n";

}

if (buffer.size() == buff\_size){ //if buffer full do event

std::thread thread(Subscribe, std::ref(buffer), std::ref(man));

thread.join();

std::cout.flush();

buffer.clear();

buffer.resize(0);

}

std::cout << "1 - add rectangle\n2 - add rhomb\n3 - trapeze\n";

}

}

https://github.com/trol53/oop\_exercise\_01/tree/master/oop\_exercise\_08

6.Вывод

Данная программа может быть полезна для работы с геометрическими фигурами. Так-же она позволяет ближе познакомиться с многопоточным и ассинхронным программированием.Благодаря парадигме ООП мы можем представлять пользовательские типы данных в виде объектов что значительно упрощает программирование. Так-же эта работа показывает практическое применение наследования классов и потоков.

Литература

1.Онлайн библиотека c++ cppreference[электронный ресурс] URL:<https://ru.cppreference.com/w/cpp/thread/thread>

(дата обращения 20.12.2019)

2.Информационный портал хабр [электронный ресурс] URL: <https://habr.com/ru/post/452974/>

(дата обращения 19.12.2019)