**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: Бутырев Даниил Вячеславович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

[https://github.com/SteepturN/oop\_exercise\_0](https://github.com/SteepturN/oop_exercise_01)8

1. Постановка задачи

Лабораторная работа № 08

Тема: Асинхронное программирование

Цель:

Знакомство с асинхронным программированием;

Получение практических навыков в параллельной обработке данных;

Получение практических навыков в синхронизации потоков;

Порядок выполнения работы

1. Ознакомиться с теоретическим материалом.

2. Получить у преподавателя вариант задания.

3. Реализовать задание своего варианта в соответствии с поставленными требованиями.

4. Подготовить тестовые наборы данных.

5. Создать репозиторий на GitHub.

6. Отправить файлы лабораторной работы в репозиторий.

7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

Требования к программе

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться oop\_exercise\_08 (в случае использования Windows oop\_exercise\_08.exe)

Необходимо зарегистрироваться на GitHub (если студент уже имеет регистрацию на GitHub то можно использовать ее) и создать репозитарий для задания лабораторной работы.

Преподавателю необходимо предъявить ссылку на публичный репозиторий на Github. Имя репозитория должно быть https://github.com/login/oop\_exercise\_08

Где login – логин, выбранный студентом для своего репозитория на Github.

Репозиторий должен содержать файлы:

· main.cpp //файл с заданием работы

· CMakeLists.txt // файл с конфигураций CMake

· report.doc // отчет о лабораторной работе

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.

Отчет

1. Код программы на языке C++.

2. Ссылка на репозиторий на GitHub.

3. Набор testcases.

4. Результаты выполнения тестов.

5. Объяснение результатов работы программы.

16 variant

octahedron triangle square.

1. Описание программы

1) В main происходит обработка пользовательского ввода

2) там же есть лямбда функции, которые вызываются в дркгом потоке при переполнении буфера

1. Набор тестов

add o 1 1 3 6.283 0

add o 0 0 4 3.1415 0

add s 0 0 1 0 1 1 0

add s 3 3 3 2 2 2 0

add s 1 1 2 1 2 2 0

add t 0 0 3 0 0

add t 1 1 4 0 0

add t 0 0 3 3.1415 0

add o 0 0 3 0 0

1. Результаты выполнения тестов

You can use

--add figure (square, triangle, octahendron): add \*s || t || o\*

\*(point) 3 times for square, for octahedron and triangle center position,

radius, starting angle\* \*position to put in\*

--help for this list of commands

--exit

Octahedron

sides' length:

side 0: 3.06147

side 1: 3.06147

side 2: 3.06147

side 3: 3.06147

side 4: 3.06147

side 5: 3.06147

side 6: 3.06147

side 7: 3.06147

coordinates:

(-4, 0.000370614)

(-2.82869, -2.82817)

(-0.000370614, -4)

(2.82817, -2.82869)

(4, -0.000370614)

(2.82869, 2.82817)

(0.000370614, 4)

(-2.82817, 2.82869)

area = 45.2548

Octahedron

sides' length:

side 0: 2.2961

side 1: 2.2961

side 2: 2.2961

side 3: 2.2961

side 4: 2.2961

side 5: 2.2961

side 6: 2.2961

side 7: 2.2961

coordinates:

(4, 0.999444)

(3.12171, 3.12093)

(1.00056, 4)

(-1.12093, 3.12171)

(-2, 1.00056)

(-1.12171, -1.12093)

(0.999444, -2)

(3.12093, -1.12171)

area = 25.4558

square

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

coordinates:

(3, 3)

(3, 2)

(2, 2)

(2, 3)

area = 1

square

sides' length:

side 0: 1

side 1: 1

side 2: 1

side 3: 1

coordinates:

(0, 0)

(1, 0)

(1, 1)

(0, 1)

area = 1

You can use

--add figure (square, triangle, octahendron): add \*s || t || o\*

\*(point) 3 times for square, for octahedron and triangle center position,

radius, starting angle\* \*position to put in\*

--help for this list of commands

--exit

triangle

sides' length:

side 0: 6.9282

side 1: 6.9282

side 2: 6.9282

coordinates:

(5, 1)

(-1, 4.4641)

(-1, -2.4641)

area = 20.7846

triangle

sides' length:

side 0: 5.19615

side 1: 5.19615

side 2: 5.19615

coordinates:

(3, 0)

(-1.5, 2.59808)

(-1.5, -2.59808)

area = 11.6913

Появившиеся файлы в директории:

figuredpVdu3

figureS70dWb

figureiCNmwA

1. Листинг программы

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include "Figure.hpp"

template <typename T>

Figure<T>::Figure(const Figure<T>& obj)

: verteces(obj.verteces) {}

template <typename T>

Figure<T>::Figure()

: verteces(std::vector<std::pair<T, T>>()) {}

template <typename T>

Figure<T>& Figure<T>::operator=(const Figure<T>& obj) noexcept {

for(std::size\_t i = 0; i < size(); ++i) {

verteces[i] = obj.verteces[i];

}

return \*this;

}

template <typename T>

bool Figure<T>::operator==(const Figure<T>& obj) const noexcept{

bool result = true;

for(std::size\_t i = 0; i < size(); ++i) {

if(verteces[i] != obj.verteces[i]) {

result = false;

break;

}

}

return result;

}

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Figure<T>& obj) {

cout << obj.type() << '\n';

cout << "sides' length:\n";

for(std::size\_t i = 0; i < (&obj)->size(); ++i) {

auto v1 = obj.verteces[i];

auto v2 = obj.verteces[(i + 1) % (&obj)->size()];

cout << "side " << i << ": " << sqrt(pow(v1.first - v2.first, 2) +

pow(v1.second - v2.second, 2)) << '\n';

}

cout << "coordinates:\n";

for(std::size\_t i = 0; i < obj.size(); ++i)

cout << '(' << obj.verteces[i].first << ", "

<< obj.verteces[i].second << ")\n";

cout << "area = " << obj.area() <<'\n';

return cout;

}

template <typename T>

double distance(std::pair<T, T> o1, std::pair<T, T> o2) {

return sqrt(pow(o1.first - o2.first, 2)+pow(o1.second - o2.second, 2));

}

//«std::istream& operator>><int>(std::istream&, Figure<int>&)»

// template <typename T>

// std::istream& operator>>(std::istream& cin, Figure<T>& t) {

// enum {

// FIRST\_AXIS = 0,

// SECOND\_AXIS = 1,

// };

// char ch(' ');

// Figure<T> copy = t;

// for(int i = 0, cur\_axis = 0; i < ( t.size() - 1 ) \* 2;

// ++i, cur\_axis = (cur\_axis + 1)%2) {

// while((ch == '\t') || (ch == ' ') || (ch == '\n')) {

// cin >> ch;

// if(cin.eof()) {

// t = copy;

// return cin;

// }

// }

// cin.unget();

// ch = ' ';

// if(cur\_axis == FIRST\_AXIS)

// cin >> t.verteces[i/2].first;

// else //(cur\_axis == SECOND\_AXIS)

// cin >> t.verteces[i/2].second;

// if(cin.fail()) {

// t = copy;

// return cin;

// }

// }

// if((distance(t.verteces[0], t.verteces[1]) !=

// distance(t.verteces[1], t.verteces[2])) ||

// (distance(t.verteces[0], t.verteces[2])

// != distance(t.verteces[0], t.verteces[1])\*sqrt(2))){

// t = copy;

// cin.setstate(std::ios\_base::failbit);

// return cin;

// }

// t.verteces[3].first = t.verteces[0].first - t.verteces[1].first

// + t.verteces[2].first;

// t.verteces[3].second = t.verteces[0].second - t.verteces[1].second

// + t.verteces[2].second;

// return cin;

// }

// template <typename T>

// void make\_2\_more\_verteces(Figure<T>& t, double side\_length) {

// double \_distance = distance(t.verteces[0], t.verteces[2]);

// std::pair<double, double> center(

// ( t.verteces[0].first + t.verteces[2].first) / 2.0,

// ( t.verteces[0].second + t.verteces[2].second ) / 2.0);

// double alpha = std::atan(( t.verteces[2].second - t.verteces[0].second ) /

// ( t.verteces[2].first - t.verteces[0].first ));

// double side = sqrt(pow( side\_length, 2 )

// - pow( \_distance / 2.0 , 2 ));

// std::cout << center.first - side \* sin(alpha) << ' ' << center.second + side \* cos(alpha) << '+' <<

// center.first + side \* sin(alpha) << ' ' <<

// center.second - side \* cos(alpha);

// t.verteces[1] = std::make\_pair<T, T>(

// static\_cast<T>( center.first - side \* sin(alpha) ),

// static\_cast<T>( center.second + side \* cos(alpha) ));

// t.verteces[3] = std::make\_pair<T, T>(

// static\_cast<T>(center.first + side \* sin(alpha)),

// static\_cast<T>(center.second - side \* cos(alpha)));

// }

// //std::istream& operator>><int>(std::istream&, Figure<int>&)

// template <typename T>

// std::istream& operator>> (std::istream& cin, Figure<T>& t) {

// enum {

// FIRST\_AXIS = 0,

// SECOND\_AXIS = 1,

// };

// char ch(' ');

// Figure<T> copy = t;

// for(int i = 0, cur\_axis = 0; i < ( NUM\_OF\_VERTECES / 2 ) \* 2;

// ++i, cur\_axis = (cur\_axis + 1)%2) {

// while((ch == '\t') || (ch == ' ') || (ch == '\n')) {

// cin >> ch;

// if(cin.eof()) {

// t = copy;

// return cin;

// }

// }

// cin.unget();

// ch = ' ';

// if(cur\_axis == FIRST\_AXIS)

// cin >> t.verteces[ i / 2 ].first; //0 & 2 points

// else //(cur\_axis == SECOND\_AXIS)

// cin >> t.verteces[ i / 2 ].second;

// if(cin.fail()) {

// t = copy;

// return cin;

// }

// }

// t.verteces[2] = t.verteces[1];

// double side\_length;

// cin >> side\_length;

// if(cin.fail() ||

// distance(t.verteces[0], t.verteces[1]) > 2 \* side\_length) {

// t = copy;

// return cin;

// }

// make\_2\_more\_verteces(t, side\_length);

// return cin;

// }

template <typename T>

typename Figure<T>::vertex\_type Figure<T>::center() const noexcept{

typename Figure<T>::vertex\_type \_center(0, 0);

for(std::size\_t i = 0; i < size(); ++i) {

\_center.first+=verteces[i].first;

\_center.second+=verteces[i].second;

}

\_center.first/=static\_cast<double>(size());

\_center.second/=static\_cast<double>(size());

return \_center;

}

template <typename T> void Figure<T>::coordinates() const noexcept {

for(std::size\_t i = 0; i < size(); ++i)

std::cout << '(' << verteces[i].first << ", "

<< verteces[i].second << ')';

}

template <typename T> double Figure<T>::area() const noexcept {

double area = 0;

for(std::size\_t i = 0; i < size(); ++i) {

area +=

verteces[ i ].first \* verteces[( i + 1 ) % size()].second

- verteces[( i + 1 ) % size()].first \* verteces[ i ].second;

}

area /= 2.0;

return area < 0 ? -area : area;

}

#include "Figure.tpp"

#pragma once

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include <string>

template <typename T>

class Figure;

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Figure<T>& t);

// template <typename T>

// std::istream& operator>>(std::istream& cin, Figure<T>& t);

template <typename T>

class Figure {

public:

using size\_type = std::size\_t;

using vertex\_type = std::pair<T, T>;

std::vector<vertex\_type> verteces;

Figure(const Figure<T>& t);

Figure();

virtual ~Figure() = default;

inline virtual size\_type size() const noexcept = 0;//{return 0;}

inline virtual std::string type() const noexcept = 0;//{std::cout << "figure";}

vertex\_type center() const noexcept;

void coordinates() const noexcept;

double area() const noexcept;

Figure<T>& operator=(const Figure<T>& t) noexcept;

bool operator==(const Figure<T>& t) const noexcept;

friend std::ostream& operator<< <>(std::ostream& cout,

const Figure<T>& t);

// friend std::istream& operator>> <>(std::istream& cin,

// Figure<T>& t);

};

//#include "../template/Figure.tpp"

#pragma once

template class Figure<double>;

//template std::istream& operator>> <double>(std::istream&, Figure<double>&);

template std::ostream& operator<< <double>(std::ostream&, const Figure<double>&);

//template double distance<double>(std::pair<double,double>, std::pair<double,double>);

//template void make\_2\_more\_verteces(Figure<double>& t, double side\_length);

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include "../Figure.hpp"

#include "Octahedron.h"

template <typename T>

Octahedron<T>::Octahedron()

: Figure<T>() {

Figure<T>::verteces.resize(size());

}

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Octahedron<T>& obj) {

cout << "Octahedron\nsides' length:\n";

for(std::size\_t i = 0; i < obj.size(); ++i) {

auto v1 = obj.verteces[i];

auto v2 = obj.verteces[(i + 1) % obj.size()];

cout << "side " << i << ": " << sqrt(pow(v1.first - v2.first, 2) +

pow(v1.second - v2.second, 2)) << '\n';

}

cout << "coordinates: ";

obj.coordinates();

return cout;

}

template <typename T>

double distance(std::pair<T, T> o1, std::pair<T, T> o2) {

return sqrt(pow(o1.first - o2.first, 2)+pow(o1.second - o2.second, 2));

}

template <typename T>

std::istream& operator>>(std::istream& cin, Octahedron<T>& t) {

const double octahedron\_central\_angle = M\_PI/4.0;

Octahedron<T> copy = t;

std::pair<double, double> center;

double radius;

double fi;

cin >> center.first;

if(cin.fail()) {

t = copy;

return cin;

}

cin >> center.second;

if(cin.fail()) {

t = copy;

return cin;

}

cin >> radius;

if(cin.fail()) {

t = copy;

return cin;

}

cin >> fi;

if(cin.fail()) {

t = copy;

return cin;

}

for(double i = 0; i < t.size(); ++i) {

t.verteces[i].first = center.first +

radius \* cos( fi + i \* octahedron\_central\_angle );

t.verteces[i].second = center.second +

radius \* sin( fi + i \* octahedron\_central\_angle );

}

return cin;

}

#include "Octahedron.tpp"

#pragma once

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include <string>

#include "../Figure.hpp"

#define NUM\_OF\_VERTECES\_O 8

template <typename T>

class Octahedron;

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Octahedron<T>& t);

template <typename T>

std::istream& operator>>(std::istream& cin, Octahedron<T>& t);

template <typename T>

class Octahedron : public Figure<T> {

public:

Octahedron();

inline typename Figure<T>::size\_type size() const noexcept override {

return NUM\_OF\_VERTECES\_O;

}

inline std::string type() const noexcept override {

return "Octahedron";

}

friend std::ostream& operator<< <>(std::ostream& cout, const Octahedron<T>& t);

friend std::istream& operator>> <>(std::istream& cin, Octahedron<T>& t);

};

#pragma once

template class Octahedron<double>;

template std::istream& operator>> <double>(std::istream&, Octahedron<double>&);

template std::ostream& operator<< <double>(std::ostream&, const Octahedron<double>&);

template double distance<double>(std::pair<double,double>, std::pair<double,double>);

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include "../Figure.hpp"

#include "Square.h"

template <typename T>

Square<T>::Square()

: Figure<T>() {

Figure<T>::verteces.resize(size());

}

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Square<T>& obj) {

cout << "square\nsides' length:\n";

for(std::size\_t i = 0; i < obj.size(); ++i) {

auto v1 = obj.verteces[i];

auto v2 = obj.verteces[(i + 1) % obj.size()];

cout << "side " << i << ": " << sqrt(pow(v1.first - v2.first, 2) +

pow(v1.second - v2.second, 2)) << '\n';

}

cout << "coordinates: ";

obj.coordinates();

return cout;

}

template <typename T>

double distance(std::pair<T, T> o1, std::pair<T, T> o2) {

return sqrt(pow(o1.first - o2.first, 2)+pow(o1.second - o2.second, 2));

}

template <typename T>

std::istream& operator>>(std::istream& cin, Square<T>& t) {

enum {

FIRST\_AXIS = 0,

SECOND\_AXIS = 1,

};

char ch(' ');

Square<T> copy = t;

for(std::size\_t i = 0, cur\_axis = 0; i < ( t.size() - 1 ) \* 2;

++i, cur\_axis = ( cur\_axis + 1 ) % 2) {

while((ch == '\t') || (ch == ' ') || (ch == '\n')) {

cin >> ch;

if(cin.eof()) {

t = copy;

return cin;

}

}

cin.unget();

ch = ' ';

if(cur\_axis == FIRST\_AXIS)

cin >> t.verteces[i/2].first;

else //(cur\_axis == SECOND\_AXIS)

cin >> t.verteces[i/2].second;

if(cin.fail()) {

t = copy;

return cin;

}

}

if((distance(t.verteces[0], t.verteces[1]) !=

distance(t.verteces[1], t.verteces[2])) ||

(distance(t.verteces[0], t.verteces[2])

!= distance(t.verteces[0], t.verteces[1])\*sqrt(2))){

t = copy;

cin.setstate(std::ios\_base::failbit);

return cin;

}

t.verteces[3].first = t.verteces[0].first - t.verteces[1].first

+ t.verteces[2].first;

t.verteces[3].second = t.verteces[0].second - t.verteces[1].second

+ t.verteces[2].second;

return cin;

}

#include "Square.tpp"

#pragma once

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include "../Figure.hpp"

#include <string>

#define NUM\_OF\_VERTECES\_S 4

template <typename T>

class Square;

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Square<T>& t);

template <typename T>

std::istream& operator>>(std::istream& cin, Square<T>& t);

template <typename T>

class Square : public Figure<T> {

public:

Square();

inline typename Figure<T>::size\_type size() const noexcept override {

return NUM\_OF\_VERTECES\_S;

}

inline std::string type() const noexcept override {

return "square";

}

friend std::ostream& operator<< <>(std::ostream& cout, const Square<T>& t);

friend std::istream& operator>> <>(std::istream& cin, Square<T>& t);

};

#pragma once

template class Square<double>;

template std::istream& operator>> <double>(std::istream&, Square<double>&);

template std::ostream& operator<< <double>(std::ostream&, const Square<double>&);

template double distance<double>(std::pair<double,double>, std::pair<double,double>);

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include "../Figure.hpp"

#include "Triangle.h"

template <typename T>

Triangle<T>::Triangle()

: Figure<T>() {

Figure<T>::verteces.resize(size());

}

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Triangle<T>& obj) {

cout << "Triangle\nsides' length:\n";

for(std::size\_t i = 0; i < obj.size(); ++i) {

auto v1 = obj.verteces[i];

auto v2 = obj.verteces[(i + 1) % obj.size()];

cout << "side " << i << ": " << sqrt(pow(v1.first - v2.first, 2) +

pow(v1.second - v2.second, 2)) << '\n';

}

cout << "coordinates: ";

obj.coordinates();

return cout;

}

template <typename T>

double distance(std::pair<T, T> o1, std::pair<T, T> o2) {

return sqrt(pow(o1.first - o2.first, 2)+pow(o1.second - o2.second, 2));

}

template <typename T>

std::istream& operator>>(std::istream& cin, Triangle<T>& t) {

const double triangle\_central\_angle = 2 \* M\_PI/ 3.0;

Triangle<T> copy = t;

std::pair<double, double> center;

double radius;

double fi;

cin >> center.first;

if(cin.fail()) {

t = copy;

return cin;

}

cin >> center.second;

if(cin.fail()) {

t = copy;

return cin;

}

cin >> radius;

if(cin.fail()) {

t = copy;

return cin;

}

cin >> fi;

if(cin.fail()) {

t = copy;

return cin;

}

for(double i = 0; i < t.size(); ++i) {

t.verteces[i].first = center.first +

radius \* cos( fi + i \* triangle\_central\_angle );

t.verteces[i].second = center.second +

radius \* sin( fi + i \* triangle\_central\_angle );

}

return cin;

}

#include "Triangle.tpp"

#pragma once

#include <vector>

#include <utility>

#include <iostream>

#include <cmath>

#include <string>

#include "../Figure.hpp"

#define NUM\_OF\_VERTECES\_T 3

template <typename T>

class Triangle;

template <typename T>

std::ostream& operator<<(std::ostream& cout, const Triangle<T>& t);

template <typename T>

std::istream& operator>>(std::istream& cin, Triangle<T>& t);

template <typename T>

class Triangle : public Figure<T> {

public:

Triangle();

inline typename Figure<T>::size\_type size() const noexcept override {

return NUM\_OF\_VERTECES\_T;

}

inline std::string type() const noexcept override {

return "triangle";

}

friend std::ostream& operator<< <>(std::ostream& cout, const Triangle<T>& t);

friend std::istream& operator>> <>(std::istream& cin, Triangle<T>& t);

};

#pragma once

template class Triangle<double>;

template std::istream& operator>> <double>(std::istream&, Triangle<double>&);

template std::ostream& operator<< <double>(std::ostream&, const Triangle<double>&);

template double distance<double>(std::pair<double,double>, std::pair<double,double>);

#include <iostream>

#include <string>

#include <cstdio>

#include <set>

#include "Read\_input.hpp"

#include "../Figure/Square/Square.h"

#include "../Figure/Triangle/Triangle.h"

#include "../Figure/Octahedron/Octahedron.h"

read\_return\_t get\_command(std::set<std::string>& valid\_commands, char\* command)

{

char ch(' ');

while((ch == '\t') || (ch == ' ')) {

std::cin >> ch;

if(std::cin.eof()) exit(0);

if(ch == '\n') ch = ' ';

}

std::cin.unget();

std::cin >> command;

if(std::cin.fail()) {

std::cin.clear();

return INVALID\_INPUT;

}

if(valid\_commands.count(command)) return VALID\_INPUT;

else return INVALID\_INPUT;

}

template <typename T>

read\_return\_t get\_value(T& d)

{

char ch(' ');

while((ch == '\t') || (ch == ' ')) {

std::cin >> ch;

if(std::cin.eof()) exit(0);

if(ch == '\n') ch = ' ';

}

std::cin.unget();

std::cin >> d;

if(std::cin.fail()) {

std::cin.clear();

return INVALID\_INPUT;

}

return VALID\_INPUT;

}

template <>

read\_return\_t get\_value (unsigned int& d) {

char ch(' ');

while((ch == '\t') || (ch == ' ')) {

std::cin >> ch;

if(std::cin.eof()) exit(0);

if(ch == '\n') ch = ' ';

}

std::cin.unget();

if(ch == '-') return INVALID\_INPUT;

std::cin >> d;

if(std::cin.fail()) {

std::cin.clear();

return INVALID\_INPUT;

}

return VALID\_INPUT;

}

template <>

read\_return\_t get\_value (unsigned long long& d) {

char ch(' ');

while((ch == '\t') || (ch == ' ')) {

std::cin >> ch;

if(std::cin.eof()) exit(0);

if(ch == '\n') ch = ' ';

}

std::cin.unget();

if(ch == '-') return INVALID\_INPUT;

std::cin >> d;

if(std::cin.fail()) {

std::cin.clear();

return INVALID\_INPUT;

}

return VALID\_INPUT;

}

#include "Read\_input.tpp"

#ifndef READ\_INPUT\_H\_

#define READ\_INPUT\_H\_

#include <set>

#include <string>

typedef enum {

END\_OF\_FILE,

VALID\_INPUT,

INVALID\_INPUT,

END\_OF\_LINE,

} read\_return\_t;

read\_return\_t get\_command(std::set<std::string>&, char\*);

template <typename T>

read\_return\_t get\_value(T&);

template <>

read\_return\_t get\_value (unsigned int&);

template <>

read\_return\_t get\_value (unsigned long long&);

#endif // READ\_INPUT\_H\_

#pragma once

template read\_return\_t get\_value (Square<double>& d);

template read\_return\_t get\_value (Triangle<double>& d);

template read\_return\_t get\_value (Octahedron<double>& d);

template read\_return\_t get\_value (int& d);

template read\_return\_t get\_value (std::size\_t& d);

#include <string>

template read\_return\_t get\_value (std::string& d);

#include <iostream>

#include "Read\_input/Read\_input.hpp"

#include "Figure/Figure.hpp"

#include "Figure/Triangle/Triangle.h"

#include "Figure/Square/Square.h"

#include "Figure/Octahedron/Octahedron.h"

#include <cstdio>

#include <set>

#include <string>

#include <algorithm>

#include <list>

#include <thread>

#include <functional>

#include <fstream>

#include <unistd.h>

#include <stdlib.h>

#include <sstream>

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

std::size\_t max\_size;

if(argc != 2) {

std::cout << "not enough arguments" << std::endl;

return -1;

}

else {

max\_size = std::stoi(argv[1]);

}

std::list<std::shared\_ptr<Figure<double>>> buff;

std::vector<std::function<void()>> managers;

managers.push\_back([&buff](){ //std::cout

for(auto figure : buff) {

std::cout << \*figure << '\n';

}

});

managers.push\_back([&buff](){

std::stringstream ss;

for(auto figure : buff) {

ss << \*figure << '\n';

}

char filename[] = "./figureXXXXXX";

int fd = mkstemp(filename);

close(fd);

std::fstream file(filename, std::ios\_base::out | std::ios\_base::trunc);

file << ss.str();

file.close();

});

std::string help\_message = "You can use\n\

--add figure (square, triangle, octahendron): add \*s || t || o\*\n\

\*(point) 3 times for square, for octahedron and triangle center\

position,\n radius, starting angle\* \*position to put in\*\n \

--help for this list of commands\n\

--exit\n";

char ch(' ');

char command[20];

std::set<std::string> valid\_commands = {"add", "exit", "help"};

std::cout << help\_message;

do {

while(get\_command(valid\_commands, command) != VALID\_INPUT) {

do ch=getchar();

while((ch != EOF) && (ch != '\n'));

std::cout << "wrong input" << std::endl;

if(ch == EOF) return 0;

}

std::string&& command\_string = static\_cast<std::string>(command);

if(command\_string == "help") {

std::cout << help\_message;

} else if (command\_string == "add") {

if(buff.size() == max\_size) {

std::thread th([&managers, &buff](){

for(auto func : managers) {

func();

}

buff.clear();

});

th.join();

}

Figure<double>\* f;

std::set<std::string> valid\_figures{"t", "o", "s"};

char figure\_type[3];

if(get\_command(valid\_figures, figure\_type) != VALID\_INPUT) {

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

continue;

}

std::string figure\_type\_str =

static\_cast<std::string>(figure\_type);

if(figure\_type\_str == "t") {

f = new Triangle<double>;

if(get\_value<Triangle<double>>

(\*dynamic\_cast<Triangle<double>\*>(f)) != VALID\_INPUT) {

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

delete f;

continue;

}

} else if(figure\_type\_str == "o") {

f = new Octahedron<double>;

if(get\_value<Octahedron<double>>

(\*dynamic\_cast<Octahedron<double>\*>(f)) != VALID\_INPUT) {

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

delete f;

continue;

}

} else if(figure\_type\_str == "s") {

f = new Square<double>;

if(get\_value<Square<double>>

(\*dynamic\_cast<Square<double>\*>(f)) != VALID\_INPUT) {

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

delete f;

}

}

std::size\_t index;

if(get\_value<std::size\_t>(index) != VALID\_INPUT || index > buff.size()) {

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

delete f;

} else {

auto i = buff.begin();

for(int j = index; j > 0; --j, ++i);

buff.insert(i, std::shared\_ptr<Figure<double>>(f));

}

} else if(command\_string == "exit") return 0;

do ch = getchar(); while((ch != '\n') && (ch != EOF));

if(ch == EOF) return 0;

} while(true);

return 0;

}

Вывод: можно использовать лямбда функции для асинхронного программирования

ЛИТЕРАТУРА

1. Б. Страуструп "Язык программирования С++", четвёртое издание, 2013.