**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: Хренов Геннадий

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. Программа должна: создавать классы, соответствующие введенным данных фигур; содержать внутренний буфер, в котором хранятся фигуры, размер буфера задается параметром командной строки, при накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться; содержать ровно два потока (thread). Один основной (main) и второй для обработчиков.

1. Описание программы

Программа позволяет создавать фигуры квадрат, прямоугольник, трапеция, и записывать их в буфер, размер которого подается при запуске. Когда буфер заполняется, запускаются два обработчика, один из которых печатает данные в стандартный поток, а другой - в указанный файл. Обработчики находятся в разных потоках и действуют асинхронно.

Основные команды:

1. add - Добавление фигуры. Далее вводится тип фигуры: S,R,T для квадрата, прямоугольника и трапеции соответственно. Далее вводятся 4 координаты.
2. exit - Выход из программы.

Вывод фигур на экран и запись в файл производится автоматически при заполнении буфера.

1. Набор testcases

задаем размер буфера = 2 при запуске:

а) add S 1 1 2 1 2 2 2 1 add T 1 1 2 2 3 2 4 1 - проверка правильности работы двух потоков

b) add S 1 1 2 2 3 3 4 4 - неверные координаты

c) не подаем на вход 2 аргумента

1. Результаты выполнения тестов.

a)add S 1 1 2 1 2 2 2 1

figure added

add T 1 1 2 2 3 2 4 1

figure added

ID: 1

Figure: Square

Coors:

A = (1.00, 1.00) B = (2.00, 1.00)

C = (2.00, 2.00) D = (2.00, 1.00)

ID: 2

Figure: Trape

Coors:

A = (1.00, 1.00) B = (2.00, 2.00)

C = (3.00, 2.00) D = (4.00, 1.00)

...All data processed successfully...

b) add S 1 1 2 2 3 3 4 4

it`s not a square

c) gennadii@lenovo-b560:~/workdir/oop/laba8$ ./oop\_exercise\_08

invalid input(enter exec file + buffer size)

1. Листинг программы

**lab8.cpp**

/\* Хренов Геннадий м8о - 207б

Создать приложение, которое будет считывать из стандартного ввода данные фигур,

выводить их характеристики на экран

и записывать в файл.

\*/

#include <iostream>

#include <future>

#include <chrono>

#include <map>

#include "figure.h"

#include <algorithm>

void help() {

std::cout << "you can use:\n";

std::cout << "'add' for adding square(S), rectangle(R) or trape(T)\n";

std::cout << "'exit' for exit\n";

}

void wait(std::future<int>& fut) {

std::chrono::milliseconds span (100);

int f = fut.get();

if (f) {

std::cout << "\n...All data processed successfully...\n\n";

}

}

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

if (argc < 2) {

fprintf(stderr, "invalid input(enter exec file + buffer size)\n");

exit(0);

}

using Point = Pair<double,double>;

int size = atoi(argv[1]);

std::vector<IFigure\*> buf;

Publisher pub(buf, size);

char sout[] = "stdout";

char fout[] = "output.txt";

Subscriber sub2(sout);

Subscriber sub1(fout);

pub.Subscribe(sub1);

pub.Subscribe(sub2);

int id = 1;

int oldid = 0;

Point a, b, c, d;

char ftype;

std::string str;

IFigure\* fig;

while (std::cin >> str) {

std::future<int> fut;

if (str == "add") {

std::cin >> ftype;

std::cin >> a >> b >> c >> d;

if (ftype == 'S') {

if(isSquare(a,b,c,d)) {

fig = new Square(id, a, b, c, d);

id++;

printf("figure added\n");

} else printf("it`s not a square\n");

} else if (ftype == 'R') {

if (isRectangle(a,b,c,d)){

fig = new Rectangle(id, a, b, c, d);

id++;

printf("figure added\n");

} else printf("it`s not a rectangle\n");

} else if (ftype == 'T') {

if (isTrape(a,b,c,d)) {

fig = new Trape(id, a, b, c, d);

id++;

printf("figure added\n");

} else printf("it`s not a trape\n");

}

if(oldid < id) {

fut = std::async(&Publisher::Add,&pub,fig);

wait(fut);

}

oldid = id;

} else if (str == "help") {

help();

} else if (str == "exit") {

exit(0);

}

}

return 0;

}

**figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

#include <utility>

#include <vector>

#include <iostream>

#include <fstream>

#include <memory>

#include <cmath>

#include <iomanip>

#include <chrono>

#include <thread>

#include <string.h>

using namespace std::chrono\_literals;

enum FigureType {

TRAPE,

SQUARE,

RECTANGLE

};

template <class A, class B>

class Pair : public std::pair<A, B> {

public:

Pair() : std::pair<A, B>() {

this->first = 0;

this->second = 0;

}

Pair(A firstI, B secondI) : std::pair<A, B>() {

this->first = firstI;

this->second = secondI;

}

friend std::ostream& operator<< (std::ostream &out, const Pair<A,B> &point) {

out << std::fixed << std::setprecision(2) << '(' << point.first << ", " << point.second << ')';

return out;

}

friend std::istream& operator>> (std::istream &in, Pair<A,B> &point) {

in >> point.first;

in >> point.second;

return in;

}

};

using Point = Pair<double,double>;

template <class T>

double distance(Pair<T,T> a, Pair<T,T> b) {

return sqrt((a.first - b.first) \* (a.first - b.first) + (a.second - b.second) \* (a.second - b.second));

}

class IFigure {

public:

virtual std::ostream& Print(std::ostream& out) const = 0;

virtual int Id() const = 0;

virtual ~IFigure() = default;

Point a, b, c, d;

};

class Square : public IFigure {

public:

Square(): Id\_(0), a(0,0), b(0,0), c(0,0), d(0,0)

{

}

Square(int id, Point A, Point B, Point C, Point D):

Id\_(id), a(std::move(A)), b(std::move(B)), c(std::move(C)), d(std::move(D))

{

}

int Id() const override {

return Id\_;

}

auto centr() {

Point cnt;

cnt.first = (a.first + b.first + c.first + d.first) / 4;

cnt.second = (a.second + b.second + c.second + d.second) /4;

return cnt;

}

double area () {

return distance(a, b) \* distance(b,c);

}

std::ostream& Print(std::ostream& out) const override {

out << "ID: " << Id\_ << "\n";

out << "Figure: Square" << "\n";

out << "Coors:\n";

out << "A = " << a << "\tB = " << b

<< "\nC = " << c << "\tD = " << d << '\n';

return out;

}

private:

int Id\_;

Point a, b, c, d;

};

class Rectangle : public IFigure {

public:

Rectangle(): Id\_(0), a(0,0), b(0,0), c(0,0), d(0,0)

{

}

Rectangle(int id, Point A, Point B, Point C, Point D):

Id\_(id), a(std::move(A)), b(std::move(B)), c(std::move(C)), d(std::move(D))

{

}

int Id() const override {

return Id\_;

}

auto centr() {

Point cnt;

cnt.first = (a.first + b.first + c.first + d.first) / 4;

cnt.second = (a.second + b.second + c.second + d.second) /4;

return cnt;

}

double area () {

return distance(a, b) \* distance(b,c);

}

std::ostream& Print(std::ostream& out) const override {

out << "ID: " << Id\_ << "\n";

out << "Figure: Rectangle" << "\n";

out << "Coors:\n";

out << "A = " << a << "\tB = " << b

<< "\nC = " << c << "\tD = " << d << '\n';

return out;

}

private:

int Id\_;

Point a, b, c, d;

};

class Trape : public IFigure {

public:

Trape(): Id\_(0), a(0,0), b(0,0), c(0,0), d(0,0)

{

}

Trape(int id, Point A, Point B, Point C, Point D):

Id\_(id), a(std::move(A)), b(std::move(B)), c(std::move(C)), d(std::move(D))

{

}

int Id() const override {

return Id\_;

}

auto centr() {

Point cnt;

cnt.first = (a.first + b.first + c.first + d.first) / 4;

cnt.second = (a.second + b.second + c.second + d.second) /4;

return cnt;

}

double area () {

double h = sqrt(pow(distance(a,b), 2) - pow((distance(a,d) - distance(b,c))/2,2));

return (distance(a, b) + distance(c,d)) / 2 \* h;

}

std::ostream& Print(std::ostream& out) const override {

out << "ID: " << Id\_ << "\n";

out << "Figure: Trape" << "\n";

out << "Coors:\n";

out << "A = " << a << "\tB = " << b

<< "\nC = " << c << "\tD = " << d << '\n';

return out;

}

private:

int Id\_;

Point a, b, c, d;

};

bool isSquare(Point a, Point b, Point c, Point d)

{

if (distance(a, b) == distance(c,d)) {

if (distance(b,c) == distance(a,d)) {

if(distance(a,b) == distance(b,c)) {

if ((int)(pow(distance(a,b),2) + pow(distance(b,c), 2)) == int(pow(distance(a,c),2))) {

return true;

}

}

}

}

return false;

}

bool isRectangle(Point a, Point b, Point c, Point d)

{

if (distance(a, b) == distance(c,d)) {

if (distance(b,c) == distance(a,d)) {

return true;

}

}

return false;

}

bool isTrape(Point a, Point b, Point c, Point d)

{

if (distance(a, b) == distance(c,d)) {

if (distance(b,c) != distance(a,d)) {

return true;

}

}

return false;

}

class Subscriber {

char\* out\_file;

public:

Subscriber() = default;

Subscriber(char\* out\_file) : out\_file(out\_file) {

if (strcmp(out\_file, "stdout") != 0) {

std::ofstream fl (out\_file, std::ofstream::out);

fl.close();

}

}

int Process(std::vector<IFigure\*>& figs, int size) {

if (strcmp(out\_file, "stdout") != 0) {

std::ofstream fl (out\_file, std::ofstream::app);

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

figs[i]->Print(fl);

}

fl.close();

} else {

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

figs[i]->Print(std::cout);

}

}

return true;

}

};

class Publisher {

int size;

int max\_size;

std::vector<IFigure\*>& figs;

std::vector<Subscriber> subs;

public:

Publisher(std::vector<IFigure\*>& figs, int max\_size):

size(0), max\_size(max\_size), figs(figs), subs() {}

int Add(IFigure\* fig) {

figs.push\_back(fig);

size++;

if (size == max\_size) {

for (Subscriber& sub : subs) {

sub.Process(figs, max\_size);

}

this->size = 0;

std::this\_thread::sleep\_for(1000ms);

for (int i = figs.size() - 1; i >= 0; i--) {

delete figs[i];

figs.erase(figs.begin() + i);

}

return 1;

}

return 0;

}

void Subscribe(Subscriber& sub) {

subs.push\_back(sub);

}

};

#endif

6. Выводы

Асинхронные вычисления - форма, при которой допускается продолжение работы до окончания передачи данных. Асинхронные события происходят независимо от основного потока исполнения. Во многих случаях асинхронность дает дает хороший выигрыш по времени, так как многие тяжеловесные вызовы можно выносить в отдельные потоки и продолжать вычисления. Однако появляется дополнительная ответственность, так как нужно следить за выполнением уже каждого процесса.

СПИСОК ЛИТЕРАТУРЫ

1. Асинхронность в С++ [Электронный ресурс]. URL:<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=2ahUKEwibzoaIjMzmAhWzwMQBHQIoBZwQFjABegQIBBAB&url=https%3A%2F%2Fotus.ru%2Fnest%2Fpost%2F337%2F&usg=AOvVaw1DAXgMK0jgQ2FyE-xkjB8j>

(дата обращения: 30.11.2019).

2. Асинхронное программирование [Электронный ресурс]. URL:

<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&cad=rja&uact=8&ved=2ahUKEwiSt6bKiczmAhWlwsQBHVxlAsYQFjAHegQIBRAB&url=https%3A%2F%2Fproglib.io%2Fp%2Fasynchrony%2F&usg=AOvVaw3dtozhGdW9hvtf3I-cIYV4>

(дата обращения: 30.11.2019).