Предисловие

ЛР1: Классы

Задание

**ЛАБОРАТОРНАЯ РАБОТА №1А**

Создать **класс Point** (2 переменные х и у). Реализовать функции класса:

– Конструктор по умолчанию;

– Конструктор с параметром;

– Конструктор копирования;

– функция инициализации переменных х и у (set());

– функция, которая возвращает сумму и произведение переменных (передача параметров через ссылки);

– вывод результатов.

**ЛАБОРАТОРНАЯ РАБОТА №1Б**

Создать **класс** **Figure** для вычисления площади фигур (круга и треугольника). В **конструкторе** использовать аргумент по умолчанию.

**ЛАБОРАТОРНАЯ РАБОТА №1В**

Создать **класс** **Complex**, представляющий комплексное число (вещественная (r) и мнимая (m) часть r + mі = 5 + 4і). Реализовать функции класса:

– инициализации r и m;

– добавления к r и m целых чисел;

– сложить два комплексных числа (второе комплексное число передать как аргумент функции);

– вывод комплексного числа.

**ЛАБОРАТОРНАЯ РАБОТА №1Г**

Разработать классы DOG и MASTER

1. DOG (Кличка, вес, возраст). MASTER (ФИО; ID клуба – **константа**; Собаки – **возможно несколько**!; count – **статический** счетчик создаваемых членов клуба).

Также реализовать следующие пункты.

1. Наличие множества конструкторов для инициализации объектов класса.
2. Реализовать инкапсуляцию (методы set/get) для изменения сведений о собаках и их владельцах.

**Для тестирования программы создать двух владельцев собак. У первого – две собаки, у второго – одна. Для каждого вывести** ФИО, ID клуба и информацию по всем собакам. А также – количество членов клуба. **В интерфейсе обеспечить возможность добавление новых собак и владельцев.**

Комментарии к решению

Все задания я сложил в один файл, при этом разделяя его на регионы A, B, C и D. Каждый из регионов имеет одноимённую функцию без параметров и возвращаемого значения, выполняющую тестировании соответствующего варианта.

В варианте A я разделил тестирование на две части:

1. Тестирование конструкторов (конструктор по умолчанию, конструктор с параметрами и копирующий конструктор);
2. Тестирование функции sum\_mlt, которая находит сумму и произведение элементов точки.

void A() {

cout << "Enter point coords: ";

double x, y;

cin >> x >> y;

Point p{ x,y };

cout << "empty=" << Point{} << ' '

<< "original=" << p << ' '

<< "copy=" << Point{ p } << endl;

double sum, mlt;

p.sum\_mlt(sum, mlt);

cout << "sum=" << sum << " mlt=" << mlt << endl;

}

В варианта B мне было сказано использовать аргумент по умолчанию, чтоб считать площадь круг или треугольник. Я сделал треугольник – фигурой с двумя параметрами: Основанием и высотой. При этом значение по умолчанию для высоты – NAN, что значит, что основание воспринимается как диаметр круга.

При этом в тестировании вводятся общие значения основания и высоты, но основание используется два раза, для круга и для треугольника:

void B() {

cout << "Enter w,h: ";

double w, h;

cin >> w >> h;

cout << "Area of the circle with diameter " << w << " = " << Figure{ w }() << endl;

cout << "Area of the triangle with base " << w << " and height " << h << " = " << Figure{ w, h }() << endl;

}

В варианта C я сделал элементы типа double, а не int, потому что второе не очень странно в случае комплексных чисел.

Так же я добавил функции AddR и AddI, по заданию, но не добавлял их в тесты, потому что добавление целых по 1 компоненте так же бессмысленно. Правильно добавлять новый объект типа Complex, то есть, к примеру c + Complex{0, 2}, когда надо добавить 2 к мнимой части числа c.

void C() {

cout << "Enter real and imaginary parts of complex c: ";

double r, i;

cin >> r >> i;

Complex c{r, i};

cout << "Empty=" << Complex{} << "Reconstructed c: " << c << endl;

cout << "c+c = " << c + c << endl;

}

В варианта D я не выполнил условие что у какого-то из классов должно быть множество конструкторов. Вместо этого у класса Dog есть конструктор по умолчанию, совмещающий конструктор собаки из всех данных, а также конструктор новорождённого щенка.

Далее, в класс Master я добавил функции begin() и end(), для перечисления собак определённого владельца.

И сделал перегрузку operator<<, НЕ являющуюся friend функцией, чтоб была причина использовать get функции обоих классов.

void D() {

Master m1{ "m.First" };

m1.add\_dog({ "dog1", 12 });

m1.add\_dog({ "dog2", 15, 3 });

cout << m1;

Master m2{ "m.Second" };

m2.add\_dog({ "dog1", 13 });

cout << m2;

}

Весь код

#include <iostream>

#include <numbers>

#include <vector>

using namespace std;

#pragma region A

class Point {

double x, y;

public:

Point()

: x{}, y{}

{};

Point(double x, double y)

: x{ x }, y{ y }

{};

Point(const Point& p)

: x{ p.x }, y{ p.y }

{};

void set(double x, double y) {

this->x = x;

this->y = y;

}

void sum\_mlt(double& sum, double& mlt) {

sum = x + y;

mlt = x \* y;

}

friend ostream& operator<<(ostream& otp, const Point& p) {

return otp << "Point[" << p.x << ", " << p.y << "]";

}

};

void A() {

cout << "Enter point coords: ";

double x, y;

cin >> x >> y;

Point p{ x,y };

cout << "Empty=" << Point{} << ' '

<< "Reconstructed=" << p << ' '

<< "Copy=" << Point{ p } << endl;

double sum, mlt;

p.sum\_mlt(sum, mlt);

cout << "sum=" << sum << " mlt=" << mlt << endl;

}

#pragma endregion

#pragma region B

class Figure {

double w, h;

public:

Figure(double w, double h = NAN)

: w{ w }, h{ h }

{}

double operator()() {

return isnan(h) ?

std::numbers::pi\_v<double>\*w\*w/4 :

w \* h / 2;

}

};

void B() {

cout << "Enter w,h: ";

double w, h;

cin >> w >> h;

cout << "Area of the circle with diameter " << w << " = " << Figure{ w }() << endl;

cout << "Area of the triangle with base " << w << " and height " << h << " = " << Figure{ w, h }() << endl;

}

#pragma endregion

#pragma region C

class Complex {

double r, i;

public:

Complex()

: r{}, i{}

{}

Complex(double r, double i)

: r{ r }, i{ i }

{}

void AddR(int r) {

this->r += r;

}

void AddI(int i) {

this->i += i;

}

friend Complex operator+(const Complex& c1, const Complex& c2) {

return { c1.r + c2.r, c1.i + c2.i };

}

friend ostream& operator<<(ostream& otp, const Complex& c) {

return otp << c.r << "+i\*" << c.i;

}

};

void C() {

cout << "Enter real and imaginary parts of complex c: ";

double r, i;

cin >> r >> i;

Complex c{r, i};

cout << "Empty=" << Complex{} << "Reconstructed c: " << c << endl;

cout << "c+c = " << c + c << endl;

}

#pragma endregion

#pragma region D

class Dog {

string name;

double weight, age;

public:

Dog(string name, double weight, double age = 0)

: name{ name }

, weight{ weight }, age{ age }

{}

auto get\_name() const {

return name;

}

auto get\_weight() const {

return weight;

}

auto get\_age() const {

return age;

}

void update\_name(string name) {

this->name = name;

}

void update\_info(double weight, double age) {

this->weight = weight;

this->age = age;

}

};

class Master {

string name;

static int last\_id;

const int id = ++last\_id;

vector<Dog> dogs{};

public:

Master(string name)

: name{ name }

{}

void update\_name(string name) {

this->name = name;

}

template <class T = Dog>

void add\_dog(T&& dog) {

dogs.push\_back(forward<T>(dog));

}

auto get\_name() const {

return name;

}

auto get\_id() const {

return id;

}

auto begin() const {

return dogs.begin();

}

auto end() const {

return dogs.end();

}

};

int Master::last\_id = 0;

ostream& operator<<(ostream& otp, const Master& m) {

otp << "Master " << m.get\_name() << "#" << m.get\_id() << " has dogs:" << endl;

for (auto& d : m) {

cout << "\tdog \"" << d.get\_name() << "\""

<< " weighs " << d.get\_weight()

<< " at the age of " << d.get\_age()

<< endl;

}

return otp;

}

void D() {

Master m1{ "m.First" };

m1.add\_dog({ "dog1", 12 });

m1.add\_dog({ "dog2", 15, 3 });

cout << m1;

Master m2{ "m.Second" };

m2.add\_dog({ "dog1", 13 });

cout << m2;

}

#pragma endregion

int main()

{

A();

B();

C();

D();

}