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();

}

ЛР2: Перегрузка

Задание

Лабораторная работа **№ 2** из **методички**

**ЛАБОРАТОРНАЯ РАБОТА №2А**

*Без проверок (из ЛР 2 из методички), просто попробовать перегрузку*

Класс (по варианту) с перегруженными операциями ==, =, + как ф.-члены класса.

**ЛАБОРАТОРНАЯ РАБОТА №2Б**

*Без проверок (из ЛР 2 из методички), просто попробовать перегрузку*

Класс (по варианту) с перегруженными операциями ==, + как дружественные ф.

– **Перегрузить [ ]**, чтобы посчитать длину переменной типа char\*.

– **Перегрузить ( )**, чтобы проинициализировать переменные класса.

Комментарии к решению

Для заданий я сделал классы A и B, содержащие соответствующие методы, а также класс base, содержащий общие поля, конструкторы и вывод.

Тип культуры CultureType я объявил, как enum class, чтоб уменьшить вероятность непредвиденных ошибок.

При реализации оператором сложения я решил, что складывать можно только области с одинаковым типом культуры, при этом площади областей складываются, а эффективность результата равняется уравновешенному среднему обоих сложенных областей.

Весь код

#include <iostream>

#include <string>

using namespace std;

enum class CultureType { зерновые, бобовые };

class base {

protected:

string name{};

CultureType type{};

int area\_in\_use{};

float efficiency{};

public:

base() {

cout << "%constructor%" << endl;

cout << "Enter name: ";

cin >> this->name;

cout << "Enter culture type:" << endl;

cout << "\t1 = cereal" << endl;

cout << "\t2 = bean" << endl;

int i;

cin >> i;

switch (i)

{

case 1:

this->type = CultureType::зерновые;

break;

case 2:

this->type = CultureType::бобовые;

break;

default:

throw i;

}

cout << "Enter area in use: ";

cin >> this->area\_in\_use;

cout << "Enter efficiency: ";

cin >> this->efficiency;

}

base(string name, CultureType type, int area\_in\_use, float efficiency)

: name{ name }

, type{ type }

, area\_in\_use{ area\_in\_use }

, efficiency{ efficiency }

{}

friend ostream& operator<<(ostream& out, const base& a) {

out << "%contents%" << endl;

out << "name: " << a.name << endl;

static const char\* type\_name\_table[] = { "cereal", "bean" };

out << "type: " << type\_name\_table[(int)a.type] << endl;

out << "area\_in\_use: " << a.area\_in\_use << endl;

out << "efficiency: " << a.efficiency << endl;

return out;

}

};

class A : public base {

public:

A() : base{} {};

A(string name, CultureType type, int area\_in\_use, float efficiency)

: base{ name, type, area\_in\_use, efficiency }

{};

bool operator==(const A& other) const {

return

name == other.name &&

type == other.type &&

area\_in\_use == other.area\_in\_use &&

efficiency == other.efficiency;

}

void operator=(const A& other) {

name = other.name;

type = other.type;

area\_in\_use = other.area\_in\_use;

efficiency = other.efficiency;

}

A operator+(const A& other) const {

if (type != other.type) throw 0;

return {

name + '+' + other.name,

type,

area\_in\_use + other.area\_in\_use,

(efficiency \* area\_in\_use + other.efficiency \* other.area\_in\_use) / (area\_in\_use + other.area\_in\_use),

};

}

};

class B : public base {

public:

B() : base{} {};

B(string name, CultureType type, int area\_in\_use, float efficiency)

: base{ name, type, area\_in\_use, efficiency }

{};

friend bool operator==(const B& o1, const B& o2) {

return

o1.name == o2.name &&

o1.type == o2.type &&

o1.area\_in\_use == o2.area\_in\_use &&

o1.efficiency == o2.efficiency;

}

friend B operator+(const B& o1, const B& o2) {

if (o1.type != o2.type) throw 0;

B res;

return {

o1.name + '+' + o2.name,

o1.type,

o1.area\_in\_use + o2.area\_in\_use,

(o1.efficiency \* o1.area\_in\_use + o2.efficiency \* o2.area\_in\_use) / (o1.area\_in\_use + o2.area\_in\_use),

};

}

void operator()(string name, CultureType type, int area\_in\_use, float efficiency) {

this->name = name;

this->type = type;

this->area\_in\_use = area\_in\_use;

this->efficiency = efficiency;

}

};

int main()

{

{

cout << "a1: ";

A a1{};

auto a2 = a1;

cout << "a1 == a2(copy): " << (a1 == a2 ? "true" : "false") << endl;

cout << "a2: ";

a2 = A{};

cout << "a1 == a2: " << (a1 == a2 ? "true" : "false") << endl;

cout << "a1 + a2: " << a1 + a2;

}

cout << endl << endl << endl;

{

cout << "b1: ";

B b1{};

cout << "b2: ";

B b2{};

cout << "b1 == b2: " << (b1 == b2 ? "true" : "false") << endl;

b2("name2", CultureType::зерновые, 3, 4.5);

cout << "b1 + b2: " << b1 + b2;

}

}

ЛР 3:

Задание

**ЛАБОРАТОРНАЯ РАБОТА №3А** *Наследование без передачи параметров*

Создать класс SOBAKA (вес и возраст – инициализация и печать этих параметров).

ПК от него SPANIEL (цвет – инициализация и печать).

В каждом классе – К и Д (просто печать, что это К и Д соответствующего класса),

– функция *golos* – печать "ГАВ" или "гав" в зависимости от веса (>= или < 10 кг).

**ЛАБОРАТОРНАЯ РАБОТА №3Б** *Наследование с передачей параметров через конструктор*

Создать класс BOOK (название книги, автор, количество страниц),

ПК – LIBRARY (номер каталога, метка о наличии (есть или нет)).

В каждом классе – К и функция вывода переменных.

Осуществить передачу параметров в конструктор для инициализации переменных БК через ПК.

Создать массив из 3-х книг и проверить:

– есть ли книга вообще (по названию);

– есть ли в библиотеке или "на руках" (по метке о наличии);

– смоделировать ситуацию, когда сначала книга была, а потом ее взяли почитать (т.е. изменилась метка о наличии).

**ЛАБОРАТОРНАЯ РАБОТА №3В**

Сгенерировать 10 случайных чисел.

БК – переменная x, ф. инициализации, ф. вывода.

ПК1 от БК – числа (из х), которые при делении на 7 дают в остатке 3 (т.е. в переменную класса сохранить остаток при делении на 7);

ПК2 от ПК1– числа (из х), которые при делении на 5 дают в остатке 2 (т.е. в переменную класса сохранить остаток при делении на 5);

В ПК2 распечатать числа (из х), которые удовлетворяют первому И второму условиям.

**ЛАБОРАТОРНАЯ РАБОТА №3Г**

Создать класс БК FIGURE, который описывает плоскую фигур, для вычисления площади которой достаточно 2-х измерений.

В этом БК создать **чистую виртуальную функцию** show() для печати значения площади и функцию get() для инициализации переменных.

Создать 3 ПК (в каждом определена своя функция show() для печати значения площади).

ПК1 – площадь треугольника;  
 ПК2 – площадь прямоугольника;  
 ПК3 – площадь круга (использовать **аргументы по умолчанию**).

Работать через объекты ПК1, ПК2 и ПК3 И через указатель на БК.

**ЛАБОРАТОРНАЯ РАБОТА №3Д**

Создать иерархическую структуру студент, преподаватель, зав.кафедры.

Реализовать механизм наследования и создать **класс**, позволяющий вывести имена студентов, которые имеют хотя бы один долг и преподавателей, младше 50.

Разработать систему классов:

• Абстрактный класс **Persona** –

поля – фио, возраст;

виртуальные функции: print(), who(). ask(), которые будут реализованы в классах наследниках.

• Наследники класса **Persona**

Класс Prepod –

поля – дисциплина, которую преподает; степень (ктн – 1, дтн – 2, нет – 0);

функции: who() возвращает 1, ask() возвращает степень,

print() печатает всю информацию о преподавателе.

Класс Student –

поля – оценки (массив из 5 оценок);

функции: who() возвращает 3, ask() возвращает количество двоек,

print() печатает всю информацию о студенте.

• Наследник класса **Prepod**

Класс Zav\_kaf –

поля – должность;

функции: who() возвращает 2, ask() возвращает возраст,

print() печатает всю информацию о зав.кафедрою.

Для работы с данными классами создать класс **VUZ**.

Организовать заполнение динамического массива Shtat персонами разных типов (два преподавателя, три студента, один зав.кафедрой). Распечатать преподавателей младше 50 лет, вывести имена студентов, которые имеют хотя бы один долг.

Комментарии к решению

В задании 3А я заменил ломанные SOBAKA и golos, на полностью английские Dog и Sound().

Для печати параметров собак я сделал два operator<<, такие чтоб печать Spaniel вызывала печать Dog части себя, для вывода параметров, наследованных от Dog:

friend ostream& operator<<(ostream& otp, const Dog& d) {

return otp << "Dog[weight=" << d.weight << "; age=" << d.age << "]";

}

И затем:

friend ostream& operator<<(ostream& otp, const Spaniel& d) {

return otp << "Spaniel[color=" << d.color << "; " << static\_cast<const Dog&>(d) << "]";

}

Таким образом я не дублирую код и заодно показываю, что Spaniel содержит в себе всё содержимое Dog.

Так же, для тестирования, сделал функцию:

void Test(const Spaniel& d) {

cout << d << endl;

d.Sound();

}

Вызываемую как:

Test(Spaniel{ 15, 2, "red" });

Test(Spaniel{ 5, 1, "brown" });

В задании 3Б я передал rvalue-ссылку на Book в конструктор LibraryRecord, чтоб было чем инициализировать объект базового класса.

LibraryRecord(Book&& book, bool in\_library)

: Book{ book }

, in\_library{ in\_library }

{

static int last\_catalog\_id = 0;

catalog\_id = last\_catalog\_id++;

}

Таким образом объект этого типа инициализируется двумя вложенными парами фигурных скобок:

{ {"book1","author1",200}, true }

И в то же время избегается дублирование кода – перечисление всех полей Book в параметрах конструктора LibraryRecord.

В задании 3В в классах я сделал только расчёт остатка от деления и условие, а весь ввод/вывод сделал в main.

Так же, я решил не генерировать случайные числа для демонстрации, а сначала распечатать первые несколько подходящих чисел и затем вводить числа с клавиатуры и сразу проверять их.

В задании 3Г я опять переименовал названия с кривых русских, записанных английскими буквами, на полностью английские:  
FIGURE на Shape, get которое “для инициализации” на InitFrom (и сделал перегрузку operator>>, использующую его) и show на GetArea.

Далее, всем производным классы я сделал final, чтоб они были концами цепочек наследования.

Для вызова методов каждого класса я сделал шаблонную функцию:

template <class TShape>

void Test(TShape&& s) {

cout << "Enter data for " << typeid(TShape).name() << ": ";

cin >> s;

cout << "Area is " << s.GetArea() << endl;

}

Она использует typeid чтобы вывести, какой тип сейчас тестируется, operator>> вызывающий InitFrom для инициализации фигуры и затем вызывает GetArea напрямую, чтоб получить и вывести площадь.

В задании 3Д я сделал дополнительную функцию PrintCore в классах Person и Teacher, вызываемые из функций show, чтобы минимизировать кол-во дублей кода.

ЛР 4:

Задание

|  |  |  |  |
| --- | --- | --- | --- |
| Сельскохозяйственные культуры | | | |
| Наименование | Тип | Посевная площадь (га) | Урожайность (ц/га) |
| Соя | Б | 13000 | 45 |
| Чумиза | З | 8000 | 17 |
| Рис | З | 25650 | 24 |
| Примечание: З - зерновые, Б - бобовые | | | |

Написать электронную таблицу для сбора информации из вашего варианта задания.

* для хранения информации не использовать статические переменные, а динамически добавлять их с помощью операторов ***new*** и ***delete***.
* все объекты и структуры также должны создаваться динамически.
* необходимо создать методы для добавления новой записи в конец таблицы и для удаления последней записи из таблицы (методы ***add\_record***, ***delete\_record***).
* необходимо предусмотреть возможность сортировки таблицы по первому полю

Комментарии к решению

Для начала я взял класс base из ЛР 2, но значительно переделал его.

Кроме всего прочего, я занёс CultureType во внутрь класса-записи:

enum class CultureType { cereal, bean } type{};

ЛР 5:

Задание

Комментарии к решению

ЛР 6:

Задание

Комментарии к решению

Весь код

ЛР 7:

Задание

Комментарии к решению

Весь код

ЛР 8:

Задание

Комментарии к решению

Весь код

ЛР 9:

Задание

Комментарии к решению

Весь код

ЛР 10:

Задание

Комментарии к решению

Весь код