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, чтобы 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, чтобы 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, чтобы 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 чтобы 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, чтобы 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, чтобы 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, чтобы 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{};

Чтобы можно было удалять строчку из середины таблицы, как и сказано в задании, я сделал вектор указателей.

В AddRow желательно было бы принимать аргументы конструктора CultureInfo (в виде TArgs&& … args), но из за разделения на .h и .cpp файлы шаблоны использовать не выйдет. А вручную указывать те же параметры что и у конструктора CultureInfo плохо, потому что это насильно привязывает реализацию DB к текущей реализации CultureInfo.

Для сортировки я использовал стандартную функцию и дополнительно оптимизировал, пропуская сортировку когда данные таблицы не изменились.

ЛР 5:

Задание

Создайте свой манипуляторы которые будут выполнять:

1. Вывод всех символов большими а чискл в OCT формате
2. Вывод всех символов маленькими и замено пробелов на #
3. Вывод всех чисел в HEX формате и не выводить пробелов

Комментарии к решению

1) Нет такого манипулятора, чтобы **все** символы были большими. uppercase влияет только на вывод 16-ичных чисел (0X123ABC vs 0x123abc), что никак не сочетается с oct (8-ичным) форматом.

Для демонстрации сделал переключение на hex формат после теста для oct формата.

2) Опять же, nouppercase влияет только на вывод в hex формате. И так же не манипулятора, заменяющего **все** пробелы. Есть манипулятор setfill, но как понятно из его названия – он меняет символы заливки, выполняемой оператором setw.

Для демонстрации добавил setw и снова hex.

3) Для “не выводить пробелов” не существует ничего и подобного. Существует ws, но это даже не манипулятор вывод – это манипулятор ввода. Кроме того, он включён по-умолчанию при запуске программы.

Для демонстрации сделал два отдельный манипулятора, один для вывода, один для ввода. При чём перед вызовом манипулятора ввода сделал ещё один ввод с noskipws, чтобы он небыл абсолютно бесполезным.

Таким образом задание к пятой ЛР, пожалуй, самое неадекватное из всех ЛР.

ЛР 6:

Задание

* Создать класс (из индивидуального задания) с заданными полями
* Создать конструктор без параметров (для начальной инициализации)
* Создать массив из 5 объектов заданного класса
* Написать функцию для изменения значений полей (с помощью ссылок)
* Написать функцию для форматированного вывода всех классов на экран (упорядоченных по первому полю)

Вариант:

|  |  |
| --- | --- |
| 10. | Автомастерская (марка автомобиля, стоимость ремонта, количество дней в ремонте, номерной знак) |

Комментарии к решению

Конструктор по-умолчанию я не сделал, потому что изначальную инициализацию смог обойти. Вместо этого я выделил неинициализированную память:

auto db = reinterpret\_cast<CarInRepair\*>(operator new[](N \* sizeof(CarInRepair)));

И затем использовал синтаксис new(адрес) CarInRepair(параметры) для инициализации с параметрами. Этот же способ используют типы вроде vector, выделяющие память заранее, но инициализирующие элемент только при добавлении.

5 элементов было маловато чтобы хорошо протестировать рандомный генератор, поэтому сделал 25.

Функции изменения полей нигде по заданию не используются, но всё равно сделал.

Для генерации элемента мне надо было две строки: марка и номерной знак машины. Для обоих я использовал разные алгоритмы:

Марки я брал из массива constexpr string\_view brands[]. Таким образом я удостоверился что они все будут упакованы на этапе компиляции, но в отличии от const char\* brands[] – ещё и обёрнуты в конкретный класс (и что важно в данном случае – поддерживающий operator<).

Для номерных знаков использовался тот же алгоритм, но вместо одной случайной строки выбирается 6 случайных символов (цифр или англ. букв). И перед этим я сделал выделение память методом .reserve, чтобы не пере-выделять память в процессе заполнения.

ЛР 7:

Задание

**ЛАБОРАТОРНАЯ РАБОТА №7А**

Записать в текстовый файл четное количество вещественных чисел.

Считать из файла вещественные числа.

Создать класс COMP. Создать массив из объектов класса COMP.

Составить из пар считанных чисел комплексные, распечатать их на экран и сохранить в новый файл. Использовать функции **read(), write().**

**ЛАБОРАТОРНАЯ РАБОТА №7Б**

Перегрузить оператор << для класса, который будет выводить строку без цифр и больших букв в текстовый файл. В классе – переменная string, КсП, show(), перегрузка <<.

**ЛАБОРАТОРНАЯ РАБОТА №7В**

В бинарном файле из 10 чисел проверить, если 5 число – четное, заменить все числа до него числом 77, если нечетное – все числа после него числом 88.

**ЛАБОРАТОРНАЯ РАБОТА №7Г**

Общее задание: Используя наработки предыдущей лабораторной работы №1, создать **отдельный класс**, реализующий следующие функции:

* ввод исходных данных – массива объектов с клавиатуры (случайным образом или заданные явно в программе);
* вывод информации о массиве объектов класса на экран;
* сохранение одного объекта класса в файл (номер нужного объекта вводится с экрана). Функция должна иметь следующую сигнатуру:

*public void writeToFile(string filename, T data)* , где Т – тип объекта класса

* восстановление одного объекта класса из файла. Если в файле записано несколько объектов, то номер нужного объекта вводится с экрана (осуществить проверку на ввод неправильного номера). Функция должна иметь следующую сигнатуру:

*public T readFromFile (string filename)* , где Т – тип объекта класса

**или**

*public void readFromFile2 (string filename, T &data)*

* сохранение всего массива объектов в файл;
* чтение всего массива объектов класса из файла.

Разработать удобный и интуитивно понятный интерфейс для демонстрации работы всех реализованных функций!

Комментарии к решению

В задании 7А вместо неоднозначного COMP я сделал название класса Complex, полностью отражающее суть.

Вместо массива сделал вектор, чтобы программа работала не зависимо от размера входного файла.

Откуда взять и как применить функции read и write непонятно. Есть методы istream::read и ostream::write, но они тут не применимы, как минимум потому, что они используются для бинарного вывода. И при использовании этих методов не получится (или получится очень неэффективно) сделать проверки конца файла. Надо хотя бы использовать istream::readsome.  
В итоге сделал через operator<<, operator>>.

В задании 7Б вместо метода show я сделал ещё одну перегрузку operator<<. Для файлов вызывается более конкретная перегрузка, принимающая ofstream, а для остальных выходных потоков данных (в том числе cout) вызывается более общая перегрузка, принимающая ostream.

В задании 7В файл изменяется после каждого выполнения программы, поэтому в начале я сделал дополнительную функцию GenInp (Generate Input), чтобы не пересоздавать тестируемый файл для каждого тестового запуска.

В задании 7Г сказано иметь возможность сохранять (и восстанавливать) один элемент из файла. На сохранение наложил ограничение – добавлять элементы можно только в конец файла, а удалять можно только весь файл целиком. Это потому, первое поле элемента — это строка, а значит в файле все элементы могут занимать разный объём. То есть переместить последний элемент на место удалённого не выйдет, надо именно двигать все элементы файла.

Но с этим же связана другая проблема. Чтобы загружать элемент с любым индексом – надо помнить, где в файле начинается каждый из элементов. Для этого при открытии файла весь файл читается вхолостую, не сохраняя результат чтения, но сохраняя позицию в файле перед каждым элементом.

ЛР 8:

Задание

**ЛАБОРАТОРНАЯ РАБОТА №8А**

Создать **шаблон** класса для работы с числами (**целыми и вещественными**):

– К по У: – К с параметрами; – Ф. вывода;

– перегрузить операцию +.

**ЛАБОРАТОРНАЯ РАБОТА №8Б**

Создать класс-**шаблон для работы с массивом:**

ПОЛЯ – динамический массив, максимальный размер массива (size), индекс (0 < ind < size для работы не со всеми элементами массива);

ФУНКЦИИ

– конструктор по умолчанию (выделяет память под массив и инициализирует переменные size=100, ind=0);

– функция заполнения одного элемента массива (в main() реализовать в цикле заполнение массива до значения ind);

– функция подсчета суммы значений в массиве размером ind;

– функция подсчета среднеарифметического значений в массиве размером ind;

– функция печати массива размером ind;

– перегрузить операцию [ ] для подсчета максимального значения в массиве любого размера (<=ind);

– деструктор для удаления динамического массива.

Комментарии к решению

В задании 8А в качестве функции вывода я сделал оператор неявного преобразования:

operator T() { … }

Таким образом в cout можно отправлять экземпляры самого шаблонного класса, но только когда тип из шаблона имеет перегрузку оператора вывода.

В задании 8Б я в основном смотрел на реализацию класса vector. В частности, я выделил память под внутренний память не инициализируя её.

Но некоторые вещи сделал по-своему, к примеру capacity (size из задания, то есть текущий размер буфера) и count (ind из задания, то есть текущее кол-во элементов). Для них я создал по одному вложенному классу. У обоих перегружен оператор преобразования в size\_t (int размера машинного слова). А у capacity ещё перегружен оператор присвоения, позволяющий писать a.capacity=5, то есть устанавливать размер внутреннего буфера.

Для добавления элемента я сделал, как и у vector – функцию emplace, принимающую произвольное кол-во параметров, forward-ящую их в конструктор элемента и возвращающую итератор, указывающий на новый элемент.

Операция [], как у любой адекватной коллекции, возвращает элемент массива, а не максимум из элементов до него. Поэтому для предпоследнего пункта я сделал ещё один вложенный тип. Функция max() возвращает элемент этого типа, а в этом типа уже перегружен operator[].

ЛР 9:

Задание

**ЛАБОРАТОРНАЯ РАБОТА №9А**

Использовать класс-шаблон VECTOR.

Создать класс (две целые переменные и их сумма):

– К по У;

– К с параметрами, для нахождения суммы двух переменных;

– Ф. вывода суммы (get()).

Создать и сравнить 2 вектора на равенство и неравенство сумм ПОПАРНО. Для этого перегрузить операции ==, >, < (или одну из них, но с проверкой на все).

**ЛАБОРАТОРНАЯ РАБОТА №9Б**

Создать ассоциативный массив (телефонная книжка) на основе двух классов (NAME -> first and NUMBER -> second).

Перегрузить оператор < в классе NAME для упорядочивания по имени.

В каждом классе: ф. вывода get();

Организовать map<> из 3-5 элементов.

Напечатать весь список.

Осуществить поиск по ключу (по имени).

Напечатать всех, у кого фамилия начинается на определенную букву.

Удалить всех, у кого фамилия начинается на определенную букву.

**ЛАБОРАТОРНАЯ РАБОТА №9С**

Использование **ФУНКТОРА** и контейнера **map**.

Создать класс PERSON (name, age):

– КпоУ, КсП для name, КсП для age (отдельно для каждого);

– get\_name(), get\_age();

– перегрузить оператор < – Ф. СРАВНЕНИЯ для упорядочивания по name;

– перегрузить оператор () – ФУНКТОР для упорядочивания по age.

Создать класс STAFF для работы с классом PERSON.

В классе STAFF:

– ф. заполнения (insert());

– ф. печати всех полей из PERSON;

– ф. печати одного поля из PERSON (только фамилию или только возраст);

– ф. удаления по фамилии;

– ф. вставки;

– ф. поиска по каждому полю.

В классе STAFF создать два контейнера map (два объекта) для сортировки по имени (используя ф. сравнения) и для сортировки по возрасту (используя функтор).

Комментарии к решению

В задании 9А название коллекции я сделал vec, хотя бы было хотя бы не такое же как у стандартного типа.

Класс суммы (Sum) я сделал без конструктора по-умолчанию. Чтобы он был не нужен – я сделал так чтобы все элементы вектора можно было инициализировать callback ’ом, переданным в конструктор.

И вместо функции get в классе Sum я сделал operator() без параметров.

Далее, вместо всех операций (==, > и <) я перегрузил операцию <=>, выполняющую роль всех операций сравнения сразу. При чём и в Sum, и в vec:

В случае Sum сравнивается результат суммирования: 4+1 < 3+3.

А в случае вектора выполняется сравнение всех элементов с соответствующими элементами другого вектора. Если все соответствующие элементы равны, то и векторы равны. Иначе, если все больше или равны – один из векторов больше. Если часть элементов больше, а часть меньше – порядок определить нельзя (partial\_ordering::unordered).

Примеры вывода:  
1)

v1: 1+1=2 0+1=1

v2: 1+1=2 1+0=1

v1<=>v1: equivalent

v1<=>v2: equivalent

v2<=>v1: equivalent

Тут результаты сумм в соответствующих векторах равны. То, что слагаемые во второй паре разные не важно, потому что operator<=> суммы сравнивает только результат сложения.

2)

v1: 0+1=1 0+1=1

v2: 1+0=1 0+0=0

v1<=>v1: equivalent

v1<=>v2: greater

v2<=>v1: less

Первая пара сумм равна, а во второй паре элемент v1 больше, чем элемент v2, поэтому при сравнении v1 и v2 получаем ответ “больше”.

3)

v1: 0+0=0 1+0=1

v2: 1+1=2 0+0=0

v1<=>v1: equivalent

v1<=>v2: unordered

v2<=>v1: unordered

В первой паре больше элемент вектора v2, а во второй – элемент v1, поэтому эти вектора не сравнимы.

В задании 9Б для ввода в первую очередь я реализовал hungry\_get\_line, поглощающее и потока ввода не только вводимую строку, но и пробельный символ из-за которой она закончилась. Эта функция позволяет вводить пустую строку (и проверять это), но при этом не зацикливаться, забирая одну и ту же пустую строку из ввода.

Для вывода вместо get я сделал operator<< в обоих классах.

В задании 9C (которое 9В вообще то) сказано сделать конструкторы, инициализирующие только одно из полей (имя или возраст) класса Person. Чтобы явным образом хранить отсутствие значения я использовал optional.

map не использует operator() для сравнений элементов. Оба сравнения должны быть реализованы как operator<. Поэтому я сделал второй оператор сравнения в структуре-функторе Person::AgeComparer.

ЛР 10:

Задание

ЛАБОРАТОРНАЯ РАБОТА № 10 Тестирование

Сделать 3 функции-шаблоны (шаблон для целых, м.б. вещественных, символьных) и 3 теста.

Функции:

а) min() – нахождения минимального;

б) sort() – сортировка массива по возрастанию;

в) sum() – нахождения в массиве суммы положительных.

Тесты:

а) для ф. min() – проверить результаты сравнения полож., отриц., и смешанных:

– с помощью ф. min() найти минимум двух чисел (символов);

– сравнить результаты (assert()).

б) для ф. sort():

– создать массив1 неотсортированный;

– создать массив2 ожидаемый отсортированный;

– отсортировать массив1, используя ф. sort();

– в цикле поэлементно сравнить результаты (assert()).

в) для ф. sum():

– создать массив;

– с помощью sum() найти сумму положительных;

– сравнить результаты ожидаемый и полученный (assert()).

Смоделировать ситуации, когда assert() выдает ошибки:

– в ф. min() поменять знак в сравнении (знак > на знак <);

– в ф. sort()поменять знак при сортировке (т.е. будет сортировка по убыванию);

– в ф. sum() убрать условие нахождения положительных.

Комментарии к решению

Функцию min я назвал min\_f в коде, чтобы выбор перегрузок не находил min стандартной библиотеки, ту, что объявлена с #define.

И для неё я реализовал две шаблонные перегрузки, для типов у которых размер меньше или равен битности процессора и для бо́льших типов. Это сделано для того, чтобы не создавать лишнюю ссылку (размер которой равен битности системы) когда копия значения будет весить не больше. Для того чтобы проверить что обе работают я тестировал с типом int (потому что его размер 32 бита, а битность системы может быть 32 или 64) и string, потому что его размер гораздо больше.

Функцию sort реализовал обычной пузырьковой сортировкой. Но объявление сделал необычным:

template <ranges::random\_access\_range TRange>

void sort(TRange& range)

random\_access\_range это концепт – фича из современного C++. Указывая его вместо typename в объявлении шаблона, я ограничиваю шаблон так, чтобы в него можно было передать только коллекции с итератором, который можно будет двигать взад-вперёд. Таким образом мой sort работает со всеми стандартными коллекциями вроде vector, при этом принимая только один параметр (вместо begin и end как в случае обычного sort). Но, так же, он спокойно работает со статическим массивом – что делает его вызов из тестов предельно простым:

int x[]{ 5, 4, 3, 2, 1 };

sort(x);

TRange вычисляется компилятором как int[5], поэтому функция std::end работает, возвращая range+5 в качестве итератора конца.

Само название sum не говорит, что на коллекцию будет накладываться ограничение. Поэтому я сделал ему параметр, позволяющий указывать условие:

class TPred = decltype([](ranges::range\_value\_t<TRange>) { return true; }),

У этого параметра есть значение по-умолчанию, которое выглядит так:

TPred pred = {}. Это работает, потому что тип лямбды, не захватывающей переменные, имеет конструктор по-умолчанию.

Но как видно из предыдущего кода – этот параметр, по умолчанию, использует все элемент коллекции. Условие из задания я задал в самих тестах, перед этим ещё протестировав случай со стандартным условием:

int x[]{ -2,-1,0,+1,+2 };

assert(("sum1: ", sum(x) == 0));

assert(("sum2: ", sum(x, [](auto x) {return x > 0; }) == 3));