Інститут спеціального зв'язку та захисту інформації

Національного технічного університету України

"Київський політехнічний інститут ім. Ігоря Сікорського"

Спеціальна кафедра №5

**ЛАБОРАТОРНА РОБОТА**

з навчальної дисципліни

**«Об’єктно-орієнтоване програмування»**

**Тема:** Дослідження механізмів реалізації успадкування в додатках, розроблених мовою об'єктно-орієнтованого програмування.

**Виконав:** курсант Жванський Роман

**Перевірив:** доцент Спеціальної кафедри №5  
Куліков В.М.

**Київ 2023**

**Варіант №3**

1. ***Дослідження засобів створення ієрархій класів в мові С++.***
   1. ***Завдання***

**Завдання 1**

Створити клас машина, який має марку (покажчик на рядок), кількість циліндрів, потужність. Визначити конструктори, деструктор і функцію друку. Створити public-похідний клас - вантажівки, який має вантажопідйомність кузова. Визначити конструктори за замовчуванням і з різною кількістю параметрів, деструктори, функцію друку. Визначити функції перепризначення марки і вантажопідйомності.

**Завдання 2**

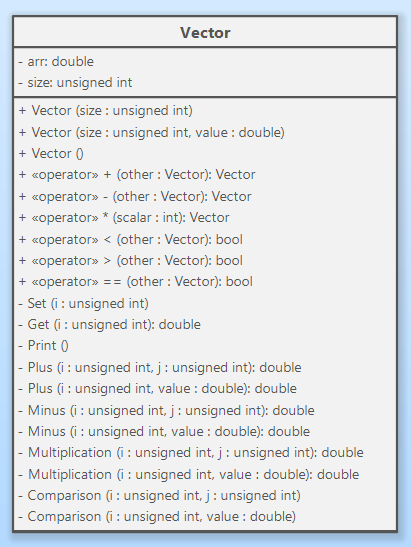
Створити клас двигун, який має потужність. Визначити конструктори і метод доступу Створити клас машина, що містить клас двигун. Додатково є марка (покажчик на рядок), ціна. Визначити конструктори і деструктор. Визначити public- похідний клас вантажівка, який має додатково вантажопідйомність. Визначити конструктори, деструктори і функцію друку.

**Завдання 3**

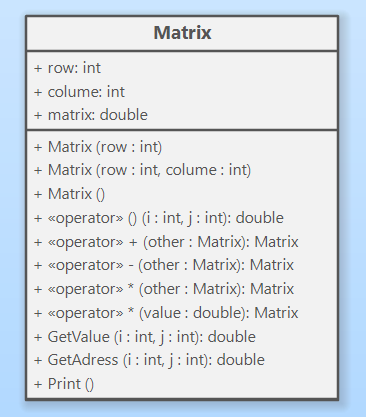
Створити ієрархію класів меблі і стіл. Перевизначити вивід в потік і ввід з потоку, конструктор копіювання, оператор присвоєння через відповідні функції базового класу.

* 1. **Об’єктні моделі мовою UML (діаграми)**

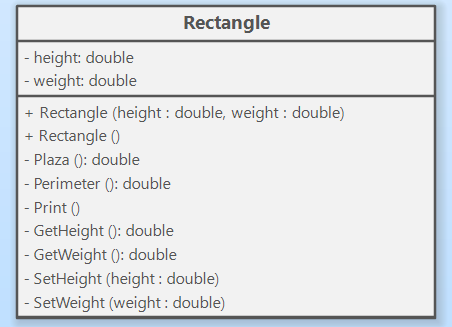
**Завдання 1**



**Завдання 2**



**Завдання 3**



* 1. ***Тексти програм(з коментарями, що пояснюють відповідність розроблених програм варіанту завдання)***

**Завдання 1**

#include <iostream>

#define default 10

class Vector

{

public:

Vector(); // конструктор за замовчуванням

Vector(unsigned int size); // конструктор з 1 параметром

Vector(unsigned int size, double value); // конструктор з 2 параметрами

~Vector(); // деструктор

void Set(unsigned int i); // для встановлення значення i-го елемента вектора

double Get(unsigned int i); // повертає i-тий елемент вектора

void Print(); // Друк

double Plus(unsigned int i, unsigned int j); // перший спосіб додавання

double Plus(unsigned int i, double value); // другий спосіб додавання

double Minus(unsigned int i, unsigned int j); // перший спосіб віднімання

double Minus(unsigned int i, double value); // другий спосіб віднімання

double Multiplication(unsigned int i, unsigned int j); // перший спосіб множення

double Multiplication(unsigned int i, double value); // другий спосіб множення

void Comparison(unsigned int i, unsigned int j); // перший спосіб порівняння

void Comparison(unsigned int i, double value); // другий спосіб порівняння

Vector operator+(Vector& other); // перегрузка оператора +

Vector operator-(Vector& other); // перегрузка оператора -

Vector operator\*(int& scalar); // перегрузка оператора \*

bool operator<(Vector& other); // перегрузка оператора <

bool operator>(Vector& other); // перегрузка оператора >

bool operator==(Vector& other); // перегрузка оператора ==

private:

double \*arr; // вектор

unsigned int size; // тип unsigned бо розмір не може бути від'ємним

};

Vector::Vector()

{

this->size = 1;

this->arr = new double[this->size];

arr[0] = 0;

}

Vector::Vector(unsigned int size)

{

this->size = size;

this->arr = new double[this->size];

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

{

arr[i] = i;

}

}

Vector::Vector(unsigned int size, double value)

{

this->size = size;

this->arr = new double[this->size];

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

{

arr[i] = value;

}

}

Vector::~Vector()

{

//delete[] arr;

}

void Vector::Set(unsigned int i)

{

if (i >= size)

{

std::cout <<"Erorr.Invalid value!\n";

return;

}

arr[i] = default;

}

double Vector::Get(unsigned int i)

{

if (i >= size)

{

std::cout << "Erorr.Invalid value!\n";

return -0.0;

}

return arr[i];

}

void Vector::Print()

{

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

{

std::cout <<"Element ["<<i <<"] : " << arr[i]<<"\n";

}

}

double Vector::Plus(unsigned int i, unsigned int j)

{

std::cout <<"Plus =" << arr[i] + arr[j];

return arr[i]+arr[j];

}

double Vector::Plus(unsigned int i, double value)

{

std::cout << "Plus =" << arr[i] + value;

return arr[i]+value;

}

double Vector::Minus(unsigned int i, unsigned int j)

{

return arr[i]-arr[j];

}

double Vector::Minus(unsigned int i, double value)

{

return arr[i] - value;

}

double Vector::Multiplication(unsigned int i, unsigned int j)

{

return arr[i]\*arr[j];

}

double Vector::Multiplication(unsigned int i, double value)

{

return arr[i] \* value;

}

void Vector::Comparison(unsigned int i, unsigned int j)

{

if (i >= size || j >= size || i==j)

{

std::cout << "Eror.Invalid value \n";

}

std::cout << "arr[i] is " << arr[i] << "arr[i] is " << arr[j] << "\n";

if (arr[i]==arr[j])

{

std::cout <<"Elements are equal \n";

}

else if (arr[i]>arr[j])

{

std::cout << "Elements arr[i] is more than arr[j]\n";

}

else

{

std::cout << "Elements arr[j] is more than arr[i]\n";

}

}

void Vector::Comparison(unsigned int i, double value)

{

if (i >= size)

{

std::cout << "Eror.Invalid value \n";

}

std::cout << "arr[i] is " << arr[i] << "value is " << value<<"\n";

if (arr[i] == value)

{

std::cout << "Elements are equal \n";

}

else if (arr[i] > value)

{

std::cout << "Elements arr[i] is more than value\n";

}

else

{

std::cout << "Elements value is more than arr[i]\n";

}

}

Vector Vector::operator+(Vector& other)

{

Vector result(size, 0);

if (size == other.size) {

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

result.arr[i] = arr[i] + other.arr[i];

}

}

return result;

}

Vector Vector::operator-(Vector& other) {

Vector result(size, 0);

if (size == other.size) {

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

result.arr[i] = arr[i] - other.arr[i];

}

}

return result;

}

Vector Vector::operator\*(cost int& scalar) {

Vector result(size, 0);

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

result.arr[i] = arr[i] \* scalar;

}

return result;

}

bool Vector::operator<(Vector& other) {

if (size < other.size) {

return true;

}

else if (size > other.size) {

return false;

}

else {

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

if (arr[i] >= other.arr[i]) {

return false;

}

}

return true;

}

}

bool Vector::operator>(Vector& other) {

if (size > other.size) {

return true;

}

else if (size < other.size) {

return false;

}

else {

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

if (arr[i] <= other.arr[i]) {

return false;

}

}

return true;

}

}

bool Vector::operator==(Vector& other) {

if (size != other.size) {

std::cout << "Invalid size\n";

return false;

}

else {

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

if (arr[i] != other.arr[i]) {

std::cout << "Not equal \n";

return false;

}

}

return true;

}

}

void main()

{

Vector v3(3, 2.0);

Vector v2(3);

v2.Set(2);

std::cout << "v2[2] = " << v2.Get(2) << "\n";

std::cout << "v3: \n" ;

v3.Print();

std::cout << "v2: \n";

v2.Print();

std::cout << "v3 + v2: \n";

(v3+v2).Print();

std::cout << "v3 + v2: ";

v3.Plus(1, 2.0);

std::cout << "\n";

std::cout << "v2 < v3 : " << (v2 < v3) << "\n";

}

**Завдання 2**

#include <iostream>

class Matrix

{

public:

Matrix(); // конструктор за замовчуванням

Matrix(int row); // конструктор з 1 параметром

Matrix(int row, int colume); // конструктор з 2 параметрами

~Matrix(); //деструктор

double GetValue(int row, int colume); // отримання значення матриці

double \*GetAdress(int row, int colume); // отримання адреса

void Print(); // друк

double& operator() (int i, int j); //перегрузка оператора ()

Matrix operator+(Matrix& other); //перегрузка оператора +

Matrix operator-(Matrix& other); //перегрузка оператора -

Matrix operator\*(Matrix& other); //перегрузка оператора \*

Matrix operator\*(double value); //перегрузка оператора \* на число

int row, colume; // рядок, стовпець

double \*\*matrix; // матриця

};

Matrix::Matrix()

{

this->row = 1;

this->colume = 1;

this->matrix = new double\* [row];

for (int i = 0; i < row; i++)

this->matrix[i] = new double[colume];

for (int i = 0; i < row; i++)

for (int j = 0; j < colume; j++)

this->matrix[i][j] = 0;

}

Matrix::Matrix(int row)

{

this->row = row;

this->colume = 1;

this->matrix = new double\* [row];

for (int i = 0; i < row; i++)

this->matrix[i] = new double[colume];

for (int i = 0; i < row; i++)

for (int j = 0; j < colume; j++)

this->matrix[i][j] = 0;

}

Matrix::Matrix(int row, int colume)

{

this->row = row;

this->colume = colume;

this->matrix = new double\* [row];

for (int i = 0; i < row; i++)

this->matrix[i] = new double[colume];

for (int i = 0; i < row; i++)

for (int j = 0; j < colume; j++)

this->matrix[i][j] = 0;

}

Matrix::~Matrix()

{

}

double Matrix::GetValue(int i, int j)

{

return matrix[i][j];

}

double\* Matrix::GetAdress(int i, int j)

{

return &matrix[i][j];

}

void Matrix::Print()

{

std::cout << "Matrix: \n";

for (int i = 0; i < row; i++)

{

for (int j = 0; j < colume; j++)

{

std::cout <<matrix[i][j] << " ";

}

std::cout << " \n";

}

}

double& Matrix::operator() (int i, int j) { // метод отримання значення елемента

return matrix[i][j];

}

Matrix Matrix::operator+(Matrix& other)

{

Matrix result(row, colume);

for (int i = 0; i < row; i++)

{

for (int j = 0; j < colume; j++)

{

result(i, j) = matrix[i][j] + other.matrix[i][j];

}

}

result.Print();

return result;

}

Matrix Matrix::operator-(Matrix& other)

{

Matrix result(row, colume);

for (int i = 0; i < row; i++)

{

for (int j = 0; j < colume; j++)

{

result(i, j) = matrix[i][j] - other.matrix[i][j];

}

}

result.Print();

return result;

}

Matrix Matrix::operator\*(Matrix& other)

{

Matrix result(this->row, other.colume);

for (int i = 0; i < row; i++)

{

for (int j = 0; j < other.colume; j++)

{

double suma = 0.0;

for (int x = 0; x <colume; x++)

{

suma += matrix[i][x] \* other.matrix[x][j];

}

result(i, j) = suma;

}

}

result.Print();

return result;

}

Matrix Matrix::operator\*(double value)

{

Matrix result(this->row, this->colume);

for (int i = 0; i < row; i++)

{

for (int j = 0; j < colume; j++)

{

result(i,j) = value \* matrix[i][j];

}

}

result.Print();

return result;

}

int main()

{

Matrix x(2,2), y(2,2);

x(0, 0) = 1;

x(0, 1) = 2;

x(1, 0) = 3;

x(1, 1) = 4;

y(0, 0) = 1;

y(0, 1) = 2;

y(1, 0) = 3;

y(1, 1) = 4;

std::cout << "x ";

x.Print();

std::cout << "y ";

y.Print();

std::cout << "x \* 10 = ";

x \* 10.0;

std::cout << "x\*y = ";

x\* y;

}

**Завдання 3**

#include <iostream>

class Rectangle

{

public:

Rectangle(double height, double weight); // конструктор з 2 параметрами

~Rectangle(); // конструктор за замовчуванням

double Plaza(); // площа

double Perimeter(); // переметр

void Print(); // друк площи, периметра, ширини, висоти

double GetHeight(); // отримання висоти

double GetWeight(); // отримання ширини

void SetHeight(double height); // задати висоту

void SetWeight(double weight); // задати ширину

private:

double height, weight; // висота, ширина

};

Rectangle::Rectangle(double height, double weight)

{

this->height = height;

this->weight = weight;

}

Rectangle::~Rectangle()

{

}

double Rectangle::Plaza()

{

return height \* weight;

}

double Rectangle::Perimeter()

{

return 2\*(height+weight);

}

void Rectangle::Print()

{

std::cout << "Height - " << height << " Weight - " << weight << " Perimeter - " << Perimeter() << " Plaza - " << Plaza()<<"\n";

}

double Rectangle::GetHeight()

{

return height;

}

double Rectangle::GetWeight()

{

return weight;

}

void Rectangle::SetHeight(double height)

{

this->height = height;

}

void Rectangle::SetWeight(double weight)

{

this->weight = weight;

}

int main()

{

Rectangle x(3,2);

x.Print();

x.SetHeight(4);

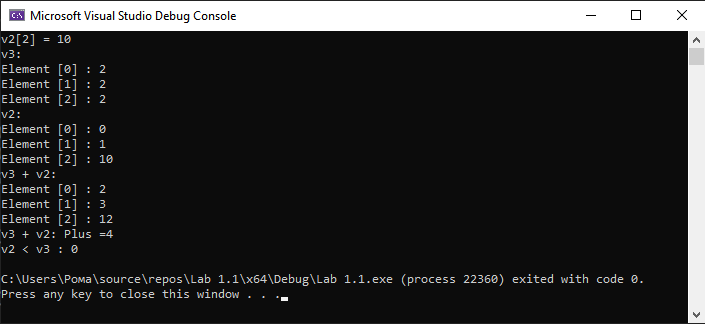
x.SetWeight(4);

x.Print();

}

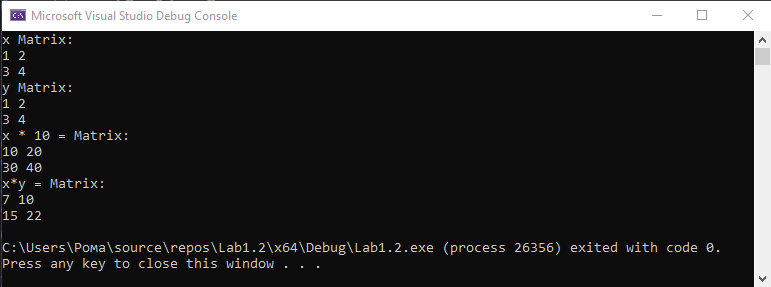
* 1. ***Висновки:***

**Завдання 1**



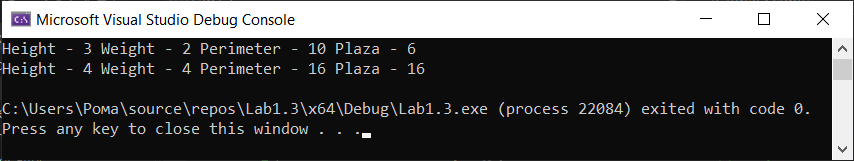
Клас працює вірно. Програма виводить вектор на екран, робить заміну 2 елемента на 10(за замовчуванням). Може робити арефметичні операції між двома векторами та їх порівняння. Також може проводити арифметичні дії з окремими елементи двох різних векторів.

**Завдання 2**



Клас працює вірно. Програма дозволяє виконувати арифметичні операції між двома матрицями. Також множити матрицю на задане число.

**Завдання 3**



Клас працює вірно. Програма може обчислювати периметр та площа за заданими величинами.

**IІ. Перетворення типів. Дружні функції. Конструктор копіювання.**

1. ***Завдання***

**Завдання 1**

Створити клас цілих чисел. Клас має конструктор за замовчуванням, конструктор, що перетворює int в об’єкт класу. Визначити оператор перетворення об’єкта типу цілих чисел в число типу int. Створити клас дійсних чисел та визначити взаємне перетворення з класом цілих чисел. Перевірити роботу цього класу.

**Завдання 2**

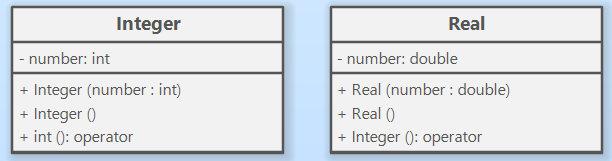
Створити клас цілих чисел Integer. Визначити перевантажену функцію, яка повертає максимальне з двох аргументів. Функція не є методом класу цілих чисел. Перевантажені функції мають аргументи типу int, double, Integer. Тіла перевантажених функцій повинні бути однаковими. Перевірити роботу цього класу.

**Завдання 3**

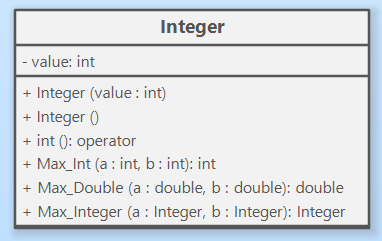
Створити два класи: дійсні (Float) і матриця (float \*\*). Визначити конструктори - за замовчуванням, з параметром, для класу матриця з двома параметрами, копіювання, деструктори. Визначити функцію множення матриці на дійсне (Float), як дружню. Перевірити роботу цього класу.

1. ***Об’єктні моделі***

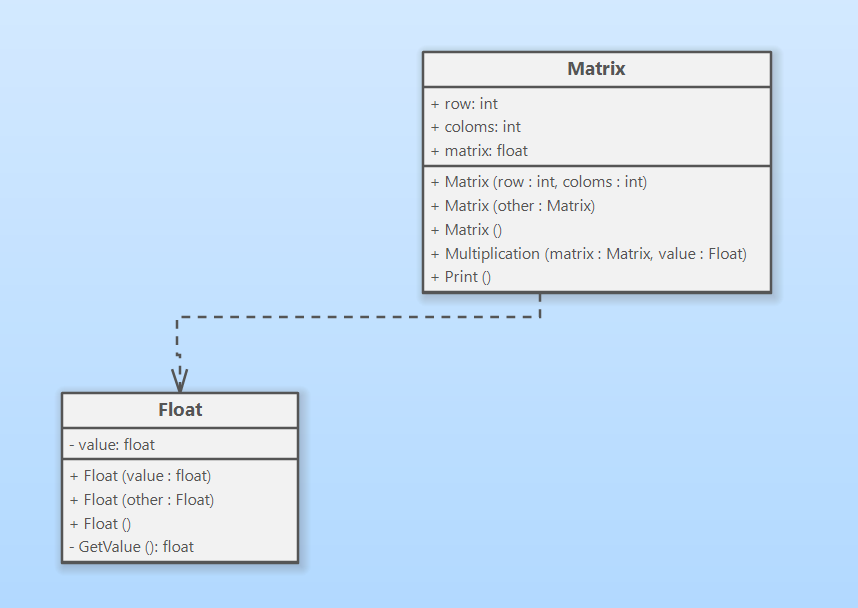
**Завдання 1**

****

**Завдання 2**

****

**Завдання 3**

****

1. ***Тексти програм***

**Завдання 1**

#include <iostream>

class Real;

class Integer

{

public:

Integer(); // конструктор за замовчування

Integer(int number); // конструктор з 1 параметром

~Integer();

operator int();

private:

int number;

};

Integer::Integer()

{

this->number = 0;

}

Integer::Integer(int number)

{

this->number = number;

}

Integer::~Integer()

{

}

Integer::operator int()

{

return number;

}

class Real

{

public:

Real();// конструктор за замовчування

Real(double number);// конструктор з 1 параметром

~Real();

operator Integer();

private:

double number;

};

Real::Real()

{

this->number = 0;

}

Real::Real(double number)

{

this->number = number;

}

Real::~Real()

{

}

Real::operator Integer()

{

return Integer((int)number);

}

int main()

{

int num = 10;

Integer a = num;

int num2 = (int)a;

Real a2(3.14);

Integer a3 = a2;

std::cout << "a = " << a << "\n";

std::cout << "num2 = " << num2 << "\n";

std::cout << "a3 = " << a3 << "\n";

}

**Завдання 2**

#include <iostream>

class Integer

{

public:

Integer(int value);// конструктор з 1 параметром

~Integer(); // конструктор за замовчування

operator int();

friend int Max\_Int(int a, int b); // дружня функція для виведення макс числа типу int

friend double Max\_Double(double a , double b);// дружня функція для виведення макс числа типу double

friend Integer Max\_Integer(Integer& a, Integer &b);// дружня функція для виведення макс числа типу Integer

private:

int value;

};

Integer::Integer(int value)

{

this->value = value;

}

Integer::~Integer()

{

}

Integer::operator int()

{

return value;

}

int Max\_Int(int a, int b)

{

return (a > b) ? a : b;

}

double Max\_Double(double a, double b)

{

return (a > b) ? a : b;

}

Integer Max\_Integer(Integer& a, Integer& b)

{

return (a.value > b.value) ? a : b;

}

int main()

{

Integer a(3), b(1);

std::cout << "Max Integer - ";

std::cout << Max\_Integer(a, b) << "\n";

int x = 2, y = 5;

std::cout << "Max Int - ";

std::cout << Max\_Int(x, y) << "\n";

double c = 3.12, d = 41.23;

std::cout << "Max double - ";

std::cout << Max\_Double(c, d) << "\n";

}

**Завдання 3**

#include <iostream>

class Float

{

public:

Float(); // конструктор за замовчування

Float(float value); // конструктор з 1 параметром

Float(Float & other);// конструктор копіювання

~Float();

float GetValue(); // отримання значення

private:

float value;

};

Float::Float()

{

this->value = 0.0;

}

Float::Float(float value)

{

this->value = value;

}

Float::Float(Float& other)

{

this->value = other.value;

}

Float::~Float()

{

}

float Float::GetValue()

{

return value;

}

class Matrix

{

public:

Matrix(); // конструктор за замовчування

Matrix(int row, int coloms); // конструктор з 1 параметром

Matrix(Matrix & other); // конструктор копіювання

~Matrix();

friend void Multiplication(Matrix & matrix, Float & value); // функція множення

void Print(); // друк

int row, coloms;

float\*\* matrix;

};

Matrix::Matrix()

{

this->row = 1;

this->coloms = 1;

this->matrix = new float\* [row];

for (int i = 0; i < row; i++)

this->matrix[i] = new float[coloms];

for (int i = 0; i < row; i++)

for (int j = 0; j < coloms; j++)

this->matrix[i][j] = 0;

}

Matrix::Matrix(int row, int coloms)

{

this->row = row;

this->coloms = coloms;

this->matrix = new float\* [row];

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

this->matrix[i] = new float[coloms];

for (int j = 0; j < coloms; j++)

this->matrix[i][j] = 0;

}

}

Matrix::Matrix(Matrix& other)

{

this->row = other.row;

this->coloms = other.coloms;

matrix = new float\* [other.row];

for (int i = 0; i < other.row; i++) {

matrix[i] = new float[other.coloms];

for (int j = 0; j < other.coloms; j++) {

matrix[i][j] = other.matrix[i][j];

}

}

}

Matrix::~Matrix()

{

for (int i = 0; i < row; i++)

{

delete[] matrix[i];

}

delete[] matrix;

}

void Matrix::Print()

{

std::cout << "Matrix:" << std::endl;

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

for (int j = 0; j < coloms; ++j) {

std::cout << matrix[i][j] << " ";

}

std::cout <<"\n";

}

}

void Multiplication(Matrix& matrix, Float& value)

{

for (int i = 0; i < matrix.row; ++i)

for (int j = 0; j < matrix.coloms; ++j)

matrix.matrix[i][j] \*= value.GetValue();

}

int main()

{

Matrix x(2, 2);

x.matrix[0][0] = 1;

x.matrix[0][1] = 2;

x.matrix[1][0] = 3;

x.matrix[1][1] = 4;

x.Print();

Float z(3);

std::cout << "matrix \* 3 ";

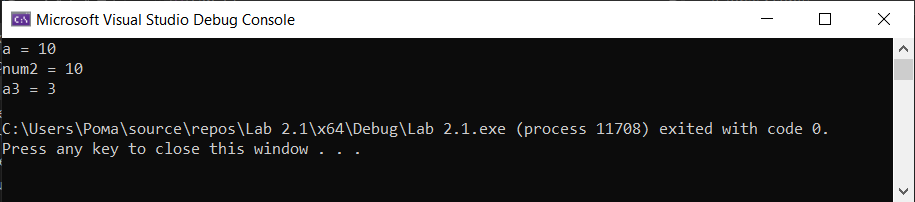
Multiplication(x, z);

x.Print();

}

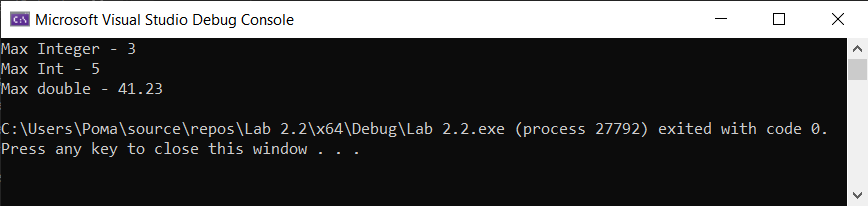
1. ***Висновок***

**Завдання 1**



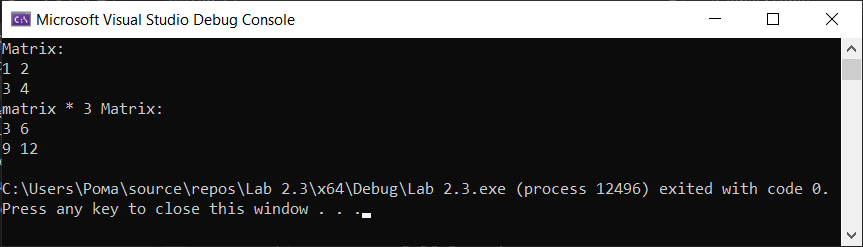
Клас працює вірно. Програма може перетворювати об’єкти типу цілих чисел число типу int. Може перетворювати об’єкт типу int в об’єкт класу(Integer). Також має взаємне перетворення цілих чисел з класом дійсних(Real).

**Завдання 2**



Клас працює вірно. Програма може визначати максимальне значення двох аргументів різних типів: int, double, Integer(наш клас).

**Завдання 3**



Клас працює вірно. Програма може множити матрицю на дійсне число.

**ІІІ.** **Перевантаження операцій.**

1. ***Завдання***

**Завдання 1**

Створити клас дійсних чисел. Визначити оператор ++ (інкремент), як функцію-метод і –– (декремент), як дружню функцію. Перевірити роботу цього класу.

**Завдання 2**

Створити клас цілих чисел. Визначити оператор – (мінус), як функцію-метод і + (плюс), як дружню функцію. Перевірити роботу цього класу.

**Завдання 3**

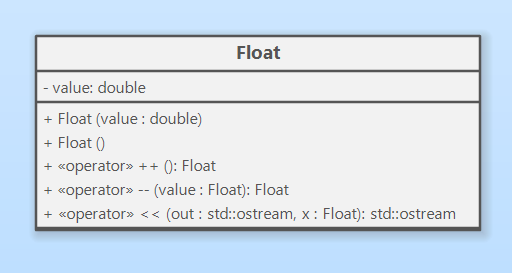
Створити клас вектор, що містить покажчик на float\*, розмірність вектора. Клас має конструктори за замовчуванням, конструктор з одним і двома параметрами, конструктор копіювання і деструктор. Визначити оператор +, -, \* , - як дружні функції, =, +=, -=, \*=,[] - як функції-методи. Визначити оператори =, +, %, \*, +=, -=, \*= з цілим числом, оператори ++ (інкремент) і – (декремент). Визначити оператори =, +, /, \*, +=, -=, \*= з дійсним числом. Визначити функцію друку. Перевірити роботу цього класу.

**Завдання 4**

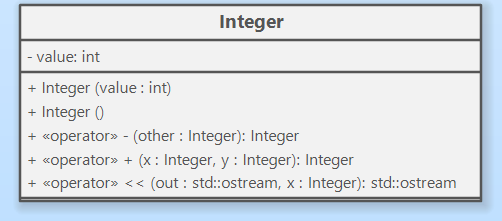
Створити клас матриця, що містить покажчик на float\*, число рядків і стовпців. Визначити конструктори за замовчуванням, конструктор з одним і з двома параметрами, конструктор копіювання, деструктор. Визначити оператори =, +, -, +=, -=, \*, \*= з об’єктами цього класу, з float і з вектором, визначеним в завданні 3. Визначити оператор []. Перевірити роботу цього класу.

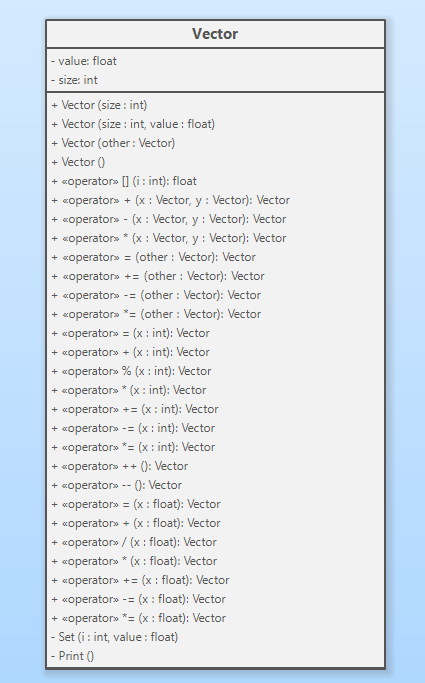
1. ***Об’єктні моделі***

**Завдання 1**

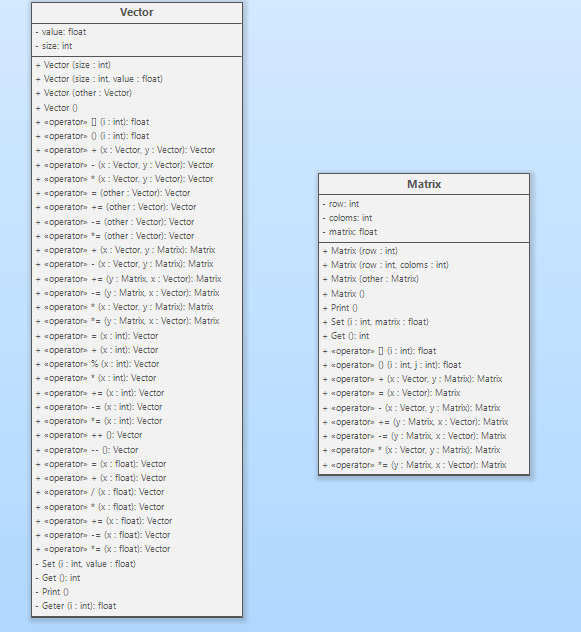
****

**Завдання 2**

****

**Завдання 3  
**

**Завдання 4**

****

1. ***Тексти програм***

**Завдання 1**

#include <iostream>

class Float

{

public:

Float(); // конструктор за замовчування

Float(double value);// конструктор з 1 параметром

~Float();

Float& operator++();// перегрузка оператора ++

friend Float& operator--(Float& value); // перегрузка оператора --

friend std::ostream& operator<<(std::ostream& out, Float& x);// перегрузка оператора <<

private:

double value;

};

Float::Float()

{

this->value = 0.0;

}

Float::Float(double value)

{

this->value = value;

}

Float::~Float()

{

}

Float& Float::operator++()

{

++value;

return \*this;

}

Float& operator--(Float& value)

{

--value.value;

return value;

}

std::ostream& operator<<(std::ostream& out, Float& x)

{

out << x.value;

return out;

}

int main()

{

std::cout << "x = " <<x<< "\n";

std::cout << "--x" << "\n";

--x;

std::cout << "x = " << x << "\n";

std::cout << "++x" << "\n";

++x;

std::cout << "x = " << x << "\n";

}

**Завдання 2**

#include <iostream>

class Integer

{

public:

Integer(); // конструктор за замовчуванням

Integer(int value); //конструктор з 1 параметром

~Integer();

Integer operator-(Integer& other);// перегрузка оператора -

friend Integer operator+( Integer& x, Integer& y);// перегрузка оператора +

friend std::ostream& operator<<(std::ostream& out, Integer& x); // перегрузка оператора <<

private:

int value;

};

Integer::Integer()

{

this->value = 1;

}

Integer::Integer(int value)

{

this->value = value;

}

Integer::~Integer()

{

}

Integer Integer::operator-(Integer& other)

{

return Integer(this->value - other.value);

}

Integer operator+( Integer& x, Integer& y)

{

return Integer(x.value + y.value);

}

std::ostream& operator<<(std::ostream& out, Integer& x)

{

out << x.value;

return out;

}

int main()

{

Integer x(4), y(24);

std::cout << "y = " << y << "\n";

std::cout << "x = " << x << "\n";

x = x - y;

std::cout << "x-y = " << x << "\n";

std::cout << "\n";

std::cout << "x = " << x << "\n";

std::cout << "y = " << y << "\n";

x = x + y;

std::cout << "x+y = " << x << "\n";

}

**Завдання 3**

#include <iostream>

#include <cmath>

class Vector

{

public:

Vector(); // конструктор за замовчуванням

Vector(int size); // конструктор з 1 параметром

Vector(int size, float\* value); // конструктор з 2 параметрами

Vector(Vector& other); // конструктор копіювання

~Vector();

void Set(int i, float value); // задає i-тому елементу значення

void Print(); //друк

float operator[](int i); // перегрузка оператора []

friend Vector operator+(Vector& x, Vector& y);// перегрузка оператора +

friend Vector operator-(Vector& x, Vector& y); // перегрузка оператора -

friend Vector operator\*(Vector& x, Vector& y); // перегрузка оператора \*

Vector& operator=(Vector& other);// перегрузка оператора =

Vector& operator+=(Vector& other);// перегрузка оператора +=

Vector& operator-=(Vector& other);// перегрузка оператора -=

Vector& operator\*=(Vector& other);// перегрузка оператора \*=

//Перегрузка для цілих чисел

Vector& operator=(int x);

Vector& operator+(int x);

Vector& operator%(int x);

Vector& operator\*(int x);

Vector& operator+=(int x);

Vector& operator-=(int x);

Vector& operator\*=(int x);

Vector& operator++();

Vector& operator--();

//Перегрузка для дійсних чисел

Vector& operator=(float x);

Vector& operator+(float x);

Vector& operator/(float x);

Vector& operator\*(float x);

Vector& operator+=(float x);

Vector& operator-=(float x);

Vector& operator\*=(float x);

private:

float\* value;

int size;

};

Vector::Vector()

{

this->size = 0;

this->value = 0;

}

Vector::Vector(int size)

{

this->size = size;

this->value = new float[size];

}

Vector::Vector(int size, float\* value)

{

this->size = size;

this->value = new float[size];

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

this->value[i] =value[i];

}

}

Vector::Vector(Vector& other)

{

this->size = other.size;

this->value = other.value;

}

Vector::~Vector()

{

}

void Vector::Set(int i, float value)

{

this->value[i] = value;

}

void Vector::Print()

{

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

{

std::cout <<"[" << i << "]" << " elemet - " << value[i] << "\n";

}

}

float Vector::operator[](int i)

{

return value[i];

}

Vector operator+(Vector& x, Vector& y)

{

Vector result(x.size);

for (int i = 0; i < x.size; i++)

{

result.Set(i, x.value[i] + y.value[i]);

}

return result;

}

Vector operator-(Vector& x, Vector& y)

{

Vector result(x.size);

for (int i = 0; i < x.size; i++)

{

result.Set(i, x.value[i] - y.value[i]);

}

return result;

}

Vector operator\*(Vector& x, Vector& y)

{

Vector result(x.size);

for (int i = 0; i < x.size; i++)

{

result.Set(i, x.value[i] \* y.value[i]);

}

return result;

}

Vector& Vector::operator=(Vector& other)

{

if (this != &other) {

delete[] value;

size = other.size;

value = new float[size];

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

value[i] = other.value[i];

}

}

return \*this;

}

Vector& Vector::operator+=(Vector& other)

{

if (size != other.size) {

std::cout<<"Erorr.Invalid size";

}

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

value[i] += other.value[i];

}

return \*this;

}

Vector& Vector::operator-=(Vector& other)

{

if (size != other.size) {

std::cout << "Erorr.Invalid size";

}

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

value[i] -= other.value[i];

}

return \*this;

}

Vector& Vector::operator\*=(Vector& other)

{

if (size != other.size) {

std::cout << "Erorr.Invalid size";

}

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

value[i] \*= other.value[i];

}

return \*this;

}

Vector& Vector::operator=(int x)

{

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

value[i] = x;

}

return \*this;

}

Vector& Vector::operator+(int x)

{

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

value[i] += x;

}

return \*this;

}

Vector& Vector::operator%(int x)

{

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

value[i] = fmod(value[i], x);

}

return \*this;

}

Vector& Vector::operator\*(int x)

{

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

value[i] \*= x;

}

return \*this;

}

Vector& Vector::operator+=(int x)

{

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

value[i] += x;

}

return \*this;

}

Vector& Vector::operator-=(int x)

{

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

value[i] -= x;

}

return \*this;

}

Vector& Vector::operator\*=(int x)

{

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

value[i] \*= x;

}

return \*this;

}

Vector& Vector::operator++()

{

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

value[i] ++;

}

return \*this;

}

Vector& Vector::operator--()

{

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

value[i]--;

}

return \*this;

}

Vector& Vector::operator=(float x)

{

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

value[i] = x;

}

return \*this;

}

Vector& Vector::operator+(float x)

{

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

value[i] += x;

}

return \*this;

}

Vector& Vector::operator/(float x)

{

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

value[i] /= x;

}

return \*this;

}

Vector& Vector::operator\*(float x)

{

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

value[i] \*= x;

}

return \*this;

}

Vector& Vector::operator+=(float x)

{

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

value[i] += x;

}

return \*this;

}

Vector& Vector::operator-=(float x)

{

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

value[i] -= x;

}

return \*this;

}

Vector& Vector::operator\*=(float x)

{

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

value[i] \*= x;

}

return \*this;

}

int main()

{

float a[] = { 1,2,3 };

float b[] = { 3,2,1 };

Vector x(3,a), c(3,b);

std::cout << "Vector x: \n";

x.Print();

std::cout << "Vector c: \n";

c.Print();

std::cout << "Vector x\*c: \n";

x \*= c;

x.Print();

std::cout << "Vector x+3: \n";

x += 3;

x.Print();

std::cout << "Vector ++x: \n";

++x;

x.Print();

std::cout << "Vector --x: \n";

--x;

x.Print();

std::cout << "Vector x\* 2.5: \n";

x \* 2.5f;

x.Print();

}

**Завдання 4**

#include <iostream>

#include <cmath>

class Matrix;

class Vector

{

public:

Vector(); // конструктор за замовчуванням

Vector(int size); // конструктор з 1 параметром

Vector(int size, float\* value); // конструктор з 2 параметрами

Vector(Vector& other); // конструктор копіювання

~Vector();

void Set(int i, float value); // задає i-тому елементу значення

int Get(); // отримання size

void Print(); //друк

float Geter(int i); // отримання i-того значення вектора

float operator[](int i); // перегрузка оператора []

float& operator() (int i);// перегрузка оператора ()

friend Vector operator+(Vector& x, Vector& y);// перегрузка оператора +

friend Vector operator-(Vector& x, Vector& y);// перегрузка оператора -

friend Vector operator\*(Vector& x, Vector& y);// перегрузка оператора \*

Vector& operator=(Vector& other); // перегрузка оператора =

Vector& operator+=(Vector& other); // перегрузка оператора +=

Vector& operator-=(Vector& other); // перегрузка оператора -=

Vector& operator\*=(Vector& other); // перегрузка оператора \*=

friend Matrix operator+(Vector& x, Matrix& y); // перегрузка оператора +

friend Matrix operator-(Vector& x, Matrix& y); // перегрузка оператора -

friend Matrix operator+=(Matrix& y, Vector& x);// перегрузка оператора +=

friend Matrix operator-=(Matrix& y, Vector& x);// перегрузка оператора -=

friend Matrix operator\*(Vector& x, Matrix& y); // перегрузка оператора \*

friend Matrix operator\*=(Matrix& y, Vector& x);// перегрузка оператора \*=

// перегрузка операцій з цілими числами

Vector& operator=(int x);

Vector& operator+(int x);

Vector& operator%(int x);

Vector& operator\*(int x);

Vector& operator+=(int x);

Vector& operator-=(int x);

Vector& operator\*=(int x);

Vector& operator++();

Vector& operator--();

// перегрузка операцій з дійсними числами

Vector& operator=(float x);

Vector& operator+(float x);

Vector& operator/(float x);

Vector& operator\*(float x);

Vector& operator+=(float x);

Vector& operator-=(float x);

Vector& operator\*=(float x);

private:

float\* value;

int size;

};

Vector::Vector()

{

this->size = 0;

this->value = 0;

}

Vector::Vector(int size)

{

this->size = size;

this->value = new float[size];

}

Vector::Vector(int size, float\* value)

{

this->size = size;

this->value = new float[size];

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

this->value[i] = value[i];

}

}

Vector::Vector(Vector& other)

{

this->size = other.size;

this->value = other.value;

}

Vector::~Vector()

{

}

void Vector::Set(int i, float value)

{

this->value[i] = value;

}

int Vector::Get()

{

return size;

}

void Vector::Print()

{

std::cout << "Our vector x: \n";

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

{

std::cout << i << " elemet - " << value[i] << "\n";

}

}

float Vector::Geter(int i)

{

return value[i];

}

float Vector::operator[](int i)

{

return value[i];

}

float& Vector::operator()(int i)

{

return value[i];

}

Vector operator+(Vector& x, Vector& y)

{

Vector result(x.size);

for (int i = 0; i < x.size; i++)

{

result.Set(i, x.value[i] + y.value[i]);

}

return result;

}

Vector operator-(Vector& x, Vector& y)

{

Vector result(x.size);

for (int i = 0; i < x.size; i++)

{

result.Set(i, x.value[i] - y.value[i]);

}

return result;

}

Vector operator\*(Vector& x, Vector& y)

{

Vector result(x.size);

for (int i = 0; i < x.size; i++)

{

result.Set(i, x.value[i] \* y.value[i]);

}

return result;

}

Vector& Vector::operator=(Vector& other)

{

if (this != &other) {

delete[] value;

size = other.size;

value = new float[size];

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

value[i] = other.value[i];

}

}

return \*this;

}

Vector& Vector::operator+=(Vector& other)

{

if (size != other.size) {

std::cout << "Erorr.Invalid size";

}

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

value[i] += other.value[i];

}

return \*this;

}

Vector& Vector::operator-=(Vector& other)

{

if (size != other.size) {

std::cout << "Erorr.Invalid size";

}

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

value[i] -= other.value[i];

}

return \*this;

}

Vector& Vector::operator\*=(Vector& other)

{

if (size != other.size) {

std::cout << "Erorr.Invalid size";

}

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

value[i] \*= other.value[i];

}

return \*this;

}

Vector& Vector::operator=(int x)

{

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

value[i] = x;

}

return \*this;

}

Vector& Vector::operator+(int x)

{

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

value[i] += x;

}

return \*this;

}

Vector& Vector::operator%(int x)

{

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

value[i] = fmod(value[i], x);

}

return \*this;

}

Vector& Vector::operator\*(int x)

{

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

value[i] \*= x;

}

return \*this;

}

Vector& Vector::operator+=(int x)

{

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

value[i] += x;

}

return \*this;

}

Vector& Vector::operator-=(int x)

{

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

value[i] -= x;

}

return \*this;

}

Vector& Vector::operator\*=(int x)

{

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

value[i] \*= x;

}

return \*this;

}

Vector& Vector::operator++()

{

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

value[i] ++;

}

return \*this;

}

Vector& Vector::operator--()

{

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

value[i]--;

}

return \*this;

}

Vector& Vector::operator=(float x)

{

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

value[i] = x;

}

return \*this;

}

Vector& Vector::operator+(float x)

{

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

value[i] += x;

}

return \*this;

}

Vector& Vector::operator/(float x)

{

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

value[i] /= x;

}

return \*this;

}

Vector& Vector::operator\*(float x)

{

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

value[i] \*= x;

}

return \*this;

}

Vector& Vector::operator+=(float x)

{

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

value[i] += x;

}

return \*this;

}

Vector& Vector::operator-=(float x)

{

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

value[i] -= x;

}

return \*this;

}

Vector& Vector::operator\*=(float x)

{

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

value[i] \*= x;

}

return \*this;

}

class Matrix

{

public:

Matrix(); // конструктор за замовчуванням

Matrix(int row); // конструктор з 1 параметром

Matrix(int row, int coloms);// конструктор з 2 параметром

Matrix(Matrix& other); // конструктор копіювання

~Matrix();

void Print(); // друк

void Set(int i, float matrix); //задати i-тому елементу значення

int Get(); //отримуємо row\*coloms(тобто розмір)

float operator[](int i); // перегрузка оператора []

float& operator() (int i, int j); // перегрузка оператора ()

friend Matrix operator+(Vector& x, Matrix& y); // перегрузка оператора +

Matrix& operator=(Vector& x); // перегрузка оператора =

friend Matrix operator-(Vector& x, Matrix& y); // перегрузка оператора -

friend Matrix operator+=(Matrix& y, Vector& x);// перегрузка оператора +=

friend Matrix operator-=(Matrix& y, Vector& x);// перегрузка оператора -=

friend Matrix operator\*(Vector& x, Matrix& y); // перегрузка оператора \*

friend Matrix operator\*=(Matrix& y, Vector& x);// перегрузка оператора \*=

private:

int row, coloms;

float\* matrix;

};

Matrix::Matrix()

{

this->row = 1;

this->coloms = 1;

this->matrix = new float [row];

this->matrix[0] = 0;

}

Matrix::Matrix(int row)

{

this->row = row;

this->coloms = 1;

this->matrix = new float[row];

}

Matrix::Matrix(int row, int coloms)

{

this->row = row;

this->coloms = coloms;

this->matrix = new float[row \* coloms];

}

Matrix::Matrix(Matrix& other)

{

this->row = other.row;

this->coloms = other.coloms;

this->matrix = new float[other.row \* other.coloms];

for (int i = 0; i < row \* coloms; i++) {

matrix[i] = other.matrix[i];

}

}

Matrix::~Matrix()

{

delete[] matrix;

}

void Matrix::Print()

{

std::cout << "Our vector y: \n";

for (int i = 0; i < Get(); i++)

{

std::cout << i << " elemet - " << matrix[i] << "\n";

}

}

void Matrix::Set(int i, float matrix)

{

this->matrix[i] = matrix;

}

int Matrix::Get()

{

return row\*coloms;

}

float Matrix::operator[](int i)

{

return matrix[i];

}

float& Matrix::operator() (int i, int j) { // метод отримання значення елемента

return matrix[i\*coloms+j];

}

Matrix& Matrix::operator=(Vector& x)

{

Matrix result(x.Get());

for (int i = 0; i < x.Get(); i++)

{

result.Set(i, x.Geter(i));

std::cout << "Result - " << result.matrix[i] << "\n";

}

return result;

}

Matrix operator+(Vector& x, Matrix& y)

{

Matrix result(y.Get());

for (int i = 0; i < y.Get(); i++)

{

result.Set(i, x.value[i] + y.matrix[i]);

std::cout << "Result - " << result.matrix[i] << "\n";

}

return result;

}

Matrix operator-(Vector& x, Matrix& y)

{

Matrix result(y.Get());

for (int i = 0; i < y.Get(); i++)

{

result.Set(i, x.value[i] - y.matrix[i]);

std::cout << "Result - " << result.matrix[i] << "\n";

}

return result;

}

Matrix operator+=(Matrix& y, Vector& x)

{

if (y.Get() !=x.Get() ) {

std::cout << "Erorr.Invalid size";

}

for (int i = 0; i < y.Get(); i++) {

y.matrix[i] += x.value[i];

}

return y;

}

Matrix operator-=(Matrix& y, Vector& x)

{

if (y.Get() != x.Get()) {

std::cout << "Erorr.Invalid size";

}

for (int i = 0; i < y.Get(); i++) {

y.matrix[i] -= x.value[i];

}

return y;

}

Matrix operator\*(Vector& x, Matrix& y)

{

Matrix result(y.Get());

for (int i = 0; i < y.Get(); i++)

{

result.Set(i, x.value[i] \* y.matrix[i]);

std::cout << "Result - " << result.matrix[i] << "\n";

}

return result;

}

Matrix operator\*=(Matrix& y, Vector& x)

{

if (y.Get() != x.Get()) {

std::cout << "Erorr.Invalid size";

}

for (int i = 0; i < y.Get(); i++) {

y.matrix[i] \*= x.value[i];

}

return y;

}

int main()

{

Matrix y(2, 2);

y(0, 0) = 1;

y(0, 1) = 2;

y(1, 0) = 3;

y(1, 1) = 4;

y.Print();

Vector x(4);

x(0) = 1;

x(1) = 2;

x(2) = 3;

x(3) = 4;

x.Print();

std::cout << "y\*x: ";

y\*=x;

y.Print();

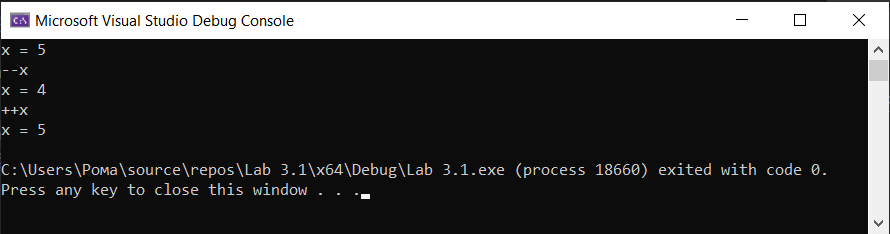
std::cout << "y=x: \n";

y=x;

}

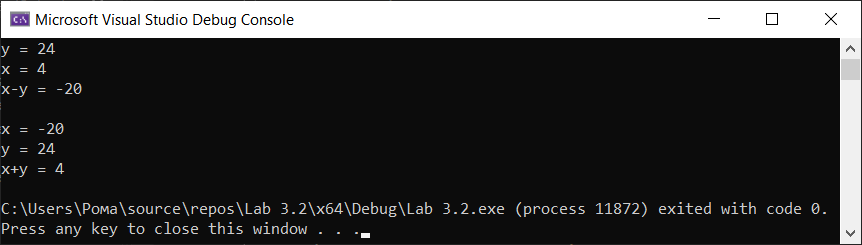
1. ***Висновок***

**Завдання 1**



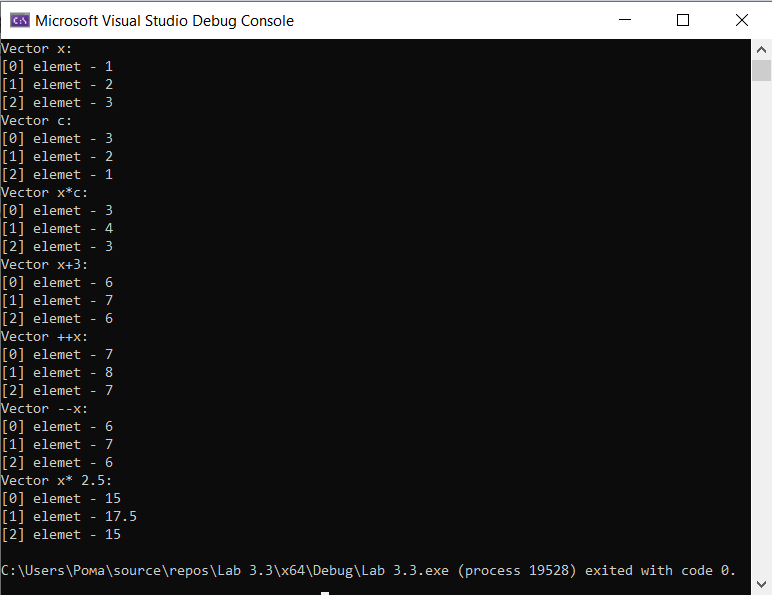
Клас працює вірно. Програма може виконувати унарні операції(інкремент і декремент).

**Завдання 2**



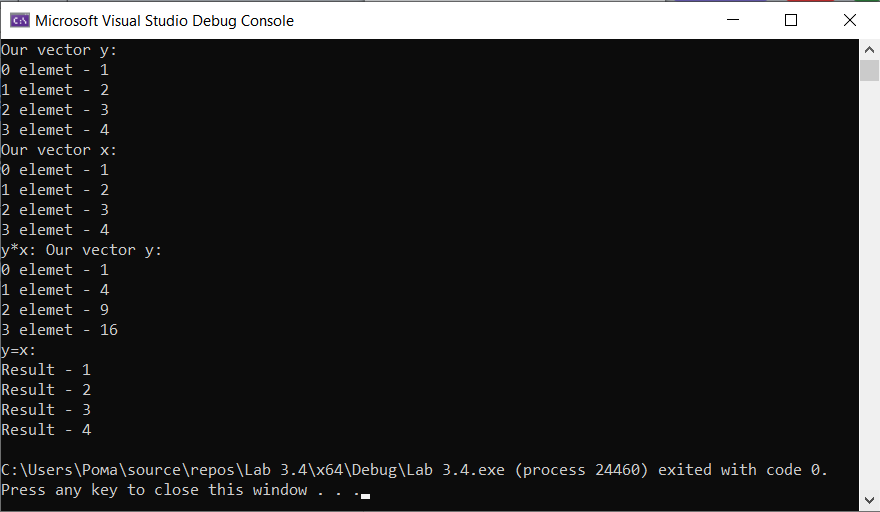
Клас працює вірно. Програма може виконувати бінарні операції(плюс, мінус).

**Завдання 3**



Клас працює вірно. Програма може виконувати арефметичні операції з цілими та дійсними числами вектора. Також має унарні операції(інкремент і декремент). Дозволяє множити вектор на дійсне число.

**Завдання 4**



Класи працюють вірно. Програма має такий самий функціонал як в завданні 3, але добавлені арифметичні операції між класом Vector і класом Matrix.

1. ***Контрольні запитання***
2. Клас - абстрактний тип даних, що визначається користувачем і являє собою модель реального об’єкта у вигляді даних та функцій для роботи з ними.
3. Конструктор класу – це спеціальний метод-функція класу. Конструктор викликається при створенні об’єкту класу. Як правило, конструктор використовується для виділення пам’яті для об’єкту класу або для початкової ініціалізації внутрішніх даних класу.
4. Деструктор – це спеціальний метод, що викликається при видаленні об’єкту. Як правило, деструктор використовується для звільнення пам’яті, динамічно виділеної під внутрішні дані класу.
5. Перевантаження функції – це оголошення функції з тим же іменем декілька разів. Таким чином, в деякій області видимості ім’я “перевантажена” функція оголошується декілька разів. Щоб компілятор міг відрізняти “перевантажені” функції, ці функції повинні відрізнятися між собою списком вхідних параметрів.
6. Дружня функція — це функція, яка має доступ до закритих членів класу, наче вона сама є членом цього класу.
7. Перевантаження оператору – спосіб оголошення та реалізації оператору таким чином, що він обробляє об’єкти конкретних класів або виконує деякі інші дії. При перевантаженні оператору в класі викликається відповідна операторна функція, яка виконує дії, що стосуються даного класу.