**Міністерство Освіти і Наука**

**ВСП Технічний Фаховий Коледж**

**Національного Університету «Львівська Політехніка»**

**Курсова робота**

**З об’єктно-орієнтованого програмування**

**Виконав:**

**Студент групи 32ПЗ**

**Боднарчук Віктор**

**Львів 2023**

**Зміст**

**Вступ – 3**

**Завдання – 3**

**Структура проекту – 4**

**UML Діаграма – 5**

**Опис Алгоритмів – 6**

**Тестування – 19**

**Висновок – 23**

**Список використаних джерел – 24**

**Додаток – 25**

**Вступ**

Об'єктно-орієнтоване програмування (ООП) є потужним інструментом, що дозволяє розробникам створювати модульні та ефективні програмні рішення. В мові програмування C++, яка володіє широкими можливостями, можна використовувати ООП-підходи для створення бібліотечних засобів, спрямованих на розв'язання задач лінійної алгебри.

Тема цієї курсової роботи - створення бібліотечних засобів для розв'язування

задач лінійної алгебри на мові програмування C++. Метою роботи є розробка

ефективних і зручних у використанні бібліотечних класів, що надають засоби для роботи з матрицями, векторами, розв'язування систем лінійних рівнянь та інших задач лінійної алгебри. Під час розробки бібліотеки ми використовуватимемо різні концепції ООП, такі як перевантаження операторів, дружні функції та шаблони. Ці засоби дозволяють створювати зрозумілі та зручні для використання інтерфейси, а також забезпечувати гнучкість та повторне використання коду. Використання перевантаження операторів дозволить нам надати зручний синтаксис для виконання арифметичних операцій над об'єктами бібліотеки, дружні функції допоможуть отримати доступ до приватних полів та методів класів, а використання шаблонів дозволить створити універсальні рішення, які працюватимуть з різними типами даних.

**Завдання**

Закріплення знань здобутих протягом семестру, набуття практичних навичок. Створення бібліотеки для розв’язування задач з лінійної алгебри: додавання та віднімання матриць, додавання та віднімання векторів, множення матриці на вектор, введення матриці та вектора із файлу, виведення матриці та вектора у файл, пошук оберненої матриці, розв’язування систем лінійних рівнянь методом Крамера, скалярне множення векторів, а також обчислення визначника матриці та обчислення довжини вектора

**Структура проекту**

У сучасному програмуванні розробка багатофайлових проектів є необхідною практикою, оскільки написання всього коду в одному файлі стає незручним, неефективним і приводить до відсутності модульності. Щоб забезпечити кращу організацію та структуру проекту, часто використовуються розділення коду на групи логічно пов'язаних файлів.

У контексті нашої курсової роботи, сама бібліотека для розв'язування задач лінійної алгебри буде написана в файлах з розширенням .h, що є стандартною практикою для заголовочних файлів. Це полегшує роботу програміста, оскільки він може просто підключити файл бібліотеки до своєї програми за допомогою директиви #include. В результаті, весь необхідний інтерфейс та оголошення класів, функцій та змінних знаходяться в заголовочному файлі, що дозволяє програмісту зосередитись на основній програмі та використовувати функціонал бібліотеки без необхідності детального знання її внутрішньої реалізації.

Основна програма, яка використовує функціонал бібліотеки, має розширення .cpp і містить фактичний код, що виконує роботу з об'єктами, викликає функції бібліотеки та обробляє результати. Це розділення на заголовочні файли та файли з розширенням .cpp дозволяє забезпечити кращу організацію проекту, покращити читабельність коду та сприяє його повторному використанню.

**Використані стандартні функції та методи С++**

Під час розробки цієї програми було використано кілька стандартних бібліотек, які є незамінними інструментами для різних аспектів програмування на мові C++.

iostream: Ця бібліотека дозволяє взаємодіяти зі стандартними потоками вводу та виводу (stdin, stdout). Вона надає класи і функції, які дозволяють зчитувати дані з клавіатури (cin) і виводити результати на екран (cout).

fstream: Ця бібліотека дозволяє працювати з файлами в C++. Вона надає класи і функції для зчитування даних з файлів (ifstream), запису даних у файли (ofstream) та одночасного введення-виведення (fstream). Використовуючи цю бібліотеку, можна виконувати операції зчитування та запису файлів

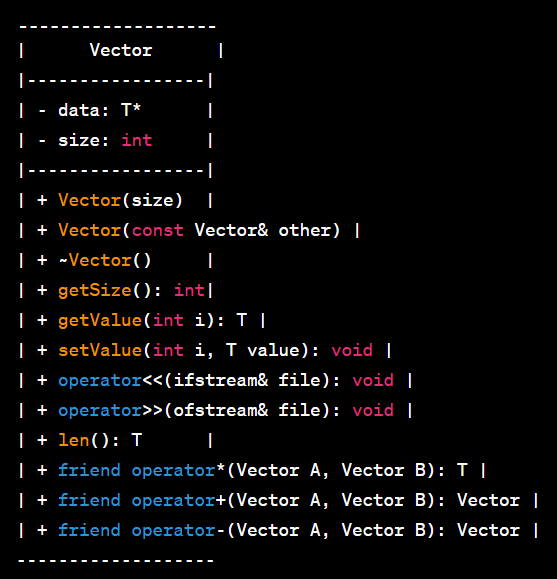
cmath: Ця бібліотека містить математичні функції та константи, які можна використовувати в програмах C++. Вона надає доступ до різних функцій, таких як обчислення кореня, тригонометричні операції, логарифми, експоненти та інші.

stdexcept: Ця бібліотека містить класи виключень, які дозволяють обробляти та керувати винятками в програмах C++. Вона надає різні класи виключень, які можна використовувати для виявлення та обробки помилок під час виконання програми.

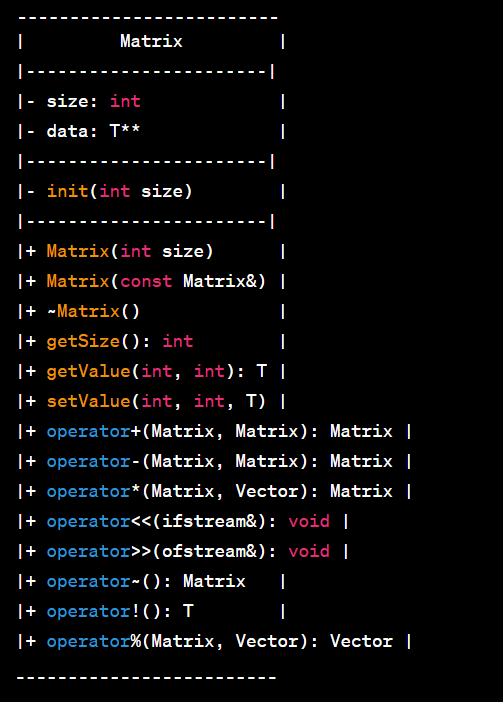
З використанням цих стандартних бібліотек, ми можемо ефективно взаємодіяти зі стандартними потоками, працювати з файлами, виконувати математичні обчислення та обробляти виключення в нашій програмі. Використання цих бібліотек дозволяє нам скоротити час розробки, полегшити процес програмування та забезпечити більшу надійність програми.

**UML Діаграма**

Клас Vector:



Клас Matrix:



Данні діаграми демонструють поля та методи класів. Тут ми можемо побачити конструктори, деструктори, методи, перевантаження.

**Опис Алгоритмів**

Основна програма

Так як основна програма лише демонструє можливості класу, та перевіряє правильність роботи користувач взаємодіяти з нею не може.

Клас Vector

Клас використовує 2 поля, які зписані в модифікаторі private:

private:

T\* data;

int size;

Шаблонний масив data використовується для зберігання вектора. А поле типу int size використовується для зберігання розміру вектора.

Даний клас використовує конструктор з параметрами, конструктор копіювання та деструктор, вони знаходяться в модифікаторі public:

//Конструктор

Vector(int size) {

this->size = size;

data = new T[size];

}

//Конструктор копіювання

Vector(const Vector& other) {

size = other.size;

data = new T[size];

for (size\_t i = 0; i < size; i++) {

data[i] = other.data[i];

}

}

//Деструктор

~Vector() {

delete[] data;

}

Конструктор з параметрами отримує розмір вектора та виділяє пам’ять.

Конструктор копіювання отримує екземпляр класу Vector, виділяє пам’ять та заповнює її значеннями з екземпляру класу.

Деструктор очищує масив.

//Отримання розміру вектора

int getSize() {

return size;

}

//Отримання значення вектора

T getValue(int i) {

return data[i];

}

//Запис значення в вектор

void setValue(int i, T value) {

data[i] = value;

}

Метод getSize повертає розмір вектора.

Метод getValue отримує номер елемента вектора і повертає значення

Метод setValue отримує номер елемента вектора та значення і записує данні в нього

//Перевантаження оператора << (Читання з файлу)

void operator<<(ifstream& file) {

for (size\_t i = 0; i < size; i++)

{

file >> data[i];

}

file.close();

}

//Перевантаження оператора >> (Запис в файл)

void operator>>(ofstream& file) {

for (size\_t i = 0; i < size; i++)

{

file << data[i] << "\t";

}

file << endl;

file.close();

}

Перевантажений оператор << отримує вказівник на об’єкт класу ifstream,   
зчитує вектор з файлу та закриває файл

Перевантажений оператор >> отримує вказівник на об’єкт класу ofstream, записує вектор в файл та закриває файл

T len() {

T result;

T promiz = 0.0;

for (size\_t i = 0; i < size; i++)

{

promiz += pow(data[i], 2);

}

result = pow(promiz, 1. / 2.);

return result;

}

Метод len() повертає значення довжини вектора

friend T operator\*(Vector A, Vector B) {

if (A.size != B.size) {

throw invalid\_argument("Vectors size`s are not equal");

}

T promiz = 0.0;

for (size\_t i = 0; i < A.size; i++)

{

promiz += A.data[i] \* B.data[i];

}

return promiz;

}

Перевантажений оператор \* (Скалярний добуток векторів) отримує 2 об’єкта класу Vector. Якщо розміри векторів не співпадають метод повертає виключення. Після виконання методу він повертає об’єкт класу Vector

friend Vector operator+(Vector A, Vector B) {

if (A.size != B.size) {

throw invalid\_argument("Vectors size`s are not equal");

}

Vector promiz(A.size);

for (size\_t i = 0; i < A.size; i++)

{

promiz.data[i] = A.data[i] + B.data[i];

}

return promiz;

}

Перевантажений оператор + (Додавання векторів) отримує 2 об’єкта класу Vector. Якщо розміри векторів не співпадають метод повертає виключення. Після виконання методу він повертає об’єкт класу Vector

friend Vector operator-(Vector A, Vector B) {

if (A.size != B.size) {

throw invalid\_argument("Vectors size`s are not equal");

}

Vector promiz(A.size);

for (size\_t i = 0; i < A.size; i++)

{

promiz.data[i] = A.data[i] - B.data[i];

}

return promiz;

}

Перевантажений оператор – (Віднімання векторів) отримує 2 об’єкта класу Vector. Якщо розміри векторів не співпадають метод повертає виключення. Після виконання методу він повертає об’єкт класу Vector

**Клас Matrix**

Клас Matrix має два приватних поля, які називаються size та data. Також є конструктор і метод init, які також позначені модифікатором private. Ці методи викликаються з інших методів класу і не призначені для взаємодії зовнішніх об'єктів. Це зроблено з метою уникнення непередбачених наслідків, наприклад, некоректної ініціалізації даних.

private:

int size;

T\*\* data;

Matrix() {

};

void init(int size) {

this->size = size;

data = (T\*\*) new T \* [size];

for (size\_t i = 0; i < size; i++)

{

data[i] = (T\*) new T[size];

}

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

data[i][j] = 0;

}

}

}

Модифікатор public містить основні методи, оператори, конструктори та деструктор, які безпосередньо використовуються для взаємодії з об'єктом класу та його функціональністю.

public:

//Конструктор

Matrix(int size) {

this->size = size;

data = (T\*\*) new T \* [size];

for (size\_t i = 0; i < size; i++)

{

data[i] = (T\*) new T[size];

}

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

data[i][j] = 0;

}

}

}

Конструктор отримує параметр типу int, виділяє пам’ять та занулює поле data

//Конструктор копіювання

Matrix(const Matrix& other) {

if (other.data == nullptr) return;

size = other.size;

data = (T\*\*) new T \* [size];

for (size\_t i = 0; i < size; i++)

{

data[i] = (T\*)new T[size];

}

for (size\_t i = 0; i < size; i++)

{

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

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

}

}

}

Конструктор копіювання отримує об’єкт класу Matrix, виділяє пам’ять та копіює поля

//Деструктор

~Matrix() {

if (data != nullptr) {

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

delete[] data[i];

}

delete[] data;

}

}

Деструктор очищує динамічну пам’ять за допомогою оператора delete[]

//Метод для повернення значення розміру матриці

int getSize() {

return size;

}

Метод getSize повертає значення int розміру матриці

//Метод для повернення значення матриці

T getValue(int i, int j) {

if (i < 0 || j < 0) {

i = 0;

j = 0;

}

else if (i >= size) {

i = size - 1;

}

else if (j >= size) {

j = size - 1;

}

return data[i][j];

}

Метод getValue отримує 2 параметра типу int та повертає шаблонне значення матриці. Також є перевірка на некоректно введені данні.

//Метод для запису значення в матрицю

void setValue(int i, int j, T value) {

data[i][j] = value;

}

Метод setValue отримує 2 параметра типу int, та один шаблонний параметр та записує данні в матрицю

//Перевантаження оператора додавання

friend Matrix operator+(Matrix A, Matrix other) {

if (other.size != A.size) {

throw invalid\_argument("Matrix size`s are not equal");

}

Matrix result(A.size);

for (size\_t i = 0; i < A.size; i++)

{

for (size\_t j = 0; j < A.size; j++)

{

result.data[i][j] = A.data[i][j] + other.data[i][j];

}

}

return result;

}

Перевантажений оператор + (Додавання матриць) отримує 2 об’єкта класу Matrix. Якщо розміри матриць не співпадають метод повертає виключення. Метод виконує обчислення та повертає об’єкт класу Matrix

//Перевантаження оператора віднімання

friend Matrix operator-(Matrix A, Matrix other) {

if (other.size != A.size) {

throw invalid\_argument("Matrix size`s are not equal");

}

Matrix result(A.size);

for (size\_t i = 0; i < A.size; i++)

{

for (size\_t j = 0; j < A.size; j++)

{

result.data[i][j] = A.data[i][j] - other.data[i][j];

}

}

return result;

}

Перевантажений оператор - (Віднімання матриць) отримує 2 об’єкта класу Matrix. Якщо розміри матриць не співпадають метод повертає виключення. Метод виконує обчислення та повертає об’єкт класу Matrix

//Перевантаження оператора множення(Множення матриці на вектор)

template <typename T>

friend Matrix<T> operator\*(Matrix<T> A, Vector<T> other) {

if (other.getSize() != A.size) {

throw std::invalid\_argument("Matrix and Vector sizes are not equal");

}

int size = A.size;

Matrix<T> result(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

result.data[i][j] = A.data[i][j] \* other.getValue(j);

}

}

return result;

}

Перевантажений оператор \* (Множення матриці на вектор) отримує 2 об’єкта класу Matrix та класу Vector. Якщо розміри матриці та вектора не співпадають метод повертає виключення. Метод виконує обчислення та повертає об’єкт класу Matrix

//Перевантаження оператора << (Читання з файлу)

void operator<<(ifstream& file) {

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

file >> data[i][j];

}

}

file.close();

}

//Перевантаження оператора >> (Запис в файл)

void operator>>(ofstream& file) {

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

file << data[i][j] << "\t";

}

file << endl;

}

}

Перевантажений оператор << отримує вказівник на об’єкт класу ifstream,   
зчитує матрицю з файлу

Перевантажений оператор >> отримує вказівник на об’єкт класу ofstream, записує матрицю в файл

//Перевантаження оператора ~ (Пошук оберненої матриці)

Matrix<T> operator~() const {

if (size == 0) {

throw std::invalid\_argument("Size must be more than 0");

}

// Створення розширеної матриці з оберненими та початковими елементами

Matrix<T> extendedMatrix(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

extendedMatrix.setValue(i, j, data[i][j]);

extendedMatrix.setValue(i, j + size, (i == j) ? 1.0 : 0.0);

}

}

// Виконання методу Гаусса-Жордана для знаходження оберненої матриці

for (size\_t i = 0; i < size; i++) {

// Ділення поточного рядка на ведучий елемент

T pivot = extendedMatrix.getValue(i, i);

for (size\_t j = 0; j < 2 \* size; j++) {

extendedMatrix.setValue(i, j, extendedMatrix.getValue(i, j) / pivot);

}

// Віднімання від інших рядків поточного рядка, помноженого на відповідний елемент

for (size\_t j = 0; j < size; j++) {

if (j != i) {

T factor = extendedMatrix.getValue(j, i);

for (size\_t k = 0; k < 2 \* size; k++) {

extendedMatrix.setValue(j, k, extendedMatrix.getValue(j, k) - factor \* extendedMatrix.getValue(i, j));

}

}

}

}

// Створення та заповнення оберненої матриці

Matrix<T> inverseMatrix(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

inverseMatrix.setValue(i, j, extendedMatrix.getValue(i, j + size));

}

}

return inverseMatrix;

}

Перевантажений оператор ~ (Пошук оберненої матриці)

Оператор ~ виконує наступні дії:

1. Перевіряє, чи розмір матриці (size) є більшим за 0. Якщо ні, викидається виключення invalid\_argument з повідомленням "Size must be more than 0".
2. Створює розширену матрицю extendedMatrix розміром size x (2\*size), яка включає початкову матрицю та одиничну матрицю.
3. Виконує метод Гаусса-Жордана для знаходження оберненої матриці. Цей метод включає кроки ділення поточного рядка на ведучий елемент та віднімання від інших рядків поточного рядка, помноженого на відповідний елемент.
4. Створює обернену матрицю inverseMatrix розміром size x size та заповнює її значеннями з розширеної матриці extendedMatrix.
5. Отриману обернену матрицю inverseMatrix повертається як результат перевантаження оператора

T operator!() {

int size = getSize();

T determinant = 1;

if (size == 1) {

determinant = data[0][0];

}

else {

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

Matrix<T> submatrix(size - 1);

int iTemp = 0;

for (int i = 1; i < size; i++) {

int jTemp = 0;

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

if (j == k) {

continue;

}

submatrix.setValue(iTemp, jTemp, data[i][j]);

jTemp++;

}

iTemp++;

}

T subDeterminant = !submatrix;

determinant += ((k % 2 == 0) ? 1 : -1) \* data[0][k] \* subDeterminant;

}

}

return determinant;

}

Перевантажений оператор! (Обчислення визначника матриці)

Оператор ! виконує наступні дії:

1. Отримує розмір матриці (size) за допомогою методу getSize().
2. Ініціалізує змінну determinant значенням 1, що буде використовуватись для обчислення визначника.
3. Перевіряє, чи розмір матриці дорівнює 1. Якщо так, присвоює значення визначника елементу data[0][0].
4. Якщо розмір матриці більший за 1, виконує цикл для кожного стовпця матриці:
   * Створює підматрицю submatrix розміром (size-1) x (size-1), яка видаляє поточний стовпець та перший рядок.
   * Заповнює підматрицю submatrix значеннями зі зсувом на один рядок вгору та на один стовпець вліво від відповідних елементів матриці.
   * Рекурсивно обчислює визначник підматриці submatrix за допомогою оператора ! та зберігає його у змінну subDeterminant.
   * Додає до determinant вираз ((k % 2 == 0) ? 1 : -1) \* data[0][k] \* subDeterminant, де k - номер поточного стовпця. Цей вираз враховує альтернативне додавання або віднімання добутку значення елементу data[0][k] на визначник підматриці submatrix, в залежності від номера стовпця k.

//Перевантаження оператора % (Метод Крамера)

friend Vector<T> operator%(Matrix<T> A, Vector<T>other) {

if (other.getSize() != A.size) {

throw std::invalid\_argument("Matrix and Vector sizes must be equal");

}

Matrix<T>\* mass = new Matrix<T>[A.size];// Створення масиву матриць

for (size\_t i = 0; i < A.size; i++)

{

mass[i].init(A.size);

}

T base\_det = !A; // Визначник матриці

for (size\_t i = 0; i < A.size; i++) {

for (size\_t j = 0; j < A.size; j++) {

for (size\_t k = 0; k < A.size; k++) {

mass[i].setValue(j, k, A.data[j][k]); // Заповнення кожної матриці значеннями

}

}

}

for (size\_t i = 0; i < A.size; i++) {

for (size\_t j = 0; j < A.size; j++) {

mass[i].data[j][i] = other.getValue(j); // Заповнення певного стовпця кожної матриці значеннями вектора

}

}

T\* dets = new T[A.size]; // Створення масиву для визначників

for (size\_t i = 0; i < A.size; i++) {

dets[i] = !mass[i]; // Заповнення масиву визначників

}

Vector<T> result(A.size); // Створення вектора результату

for (size\_t i = 0; i < A.size; i++) {

result.setValue(i, (dets[i] / base\_det)); // Обчислення кожного елемента вектора результату

}

delete[] mass; // Звільнення пам'яті для масиву матриць

delete[] dets; // Звільнення пам'яті для масиву визначників

return result;

}

Перевантаження оператора% (Система лінійних рівнянь методом Крамера)

Оператор % виконує наступні дії:

1. Перевіряє, чи розміри матриці A та вектора other співпадають. Якщо ні, викидає виключення з повідомленням про неправильні розміри.
2. Створює масив матриць mass розміром A.size для зберігання окремих матриць, які виникають при застосуванні методу Крамера.
3. Обчислює базовий визначник матриці A шляхом виклику оператора !.
4. Заповнює кожну матрицю масиву mass значеннями елементів матриці A.
5. Замінює стовпці кожної матриці масиву mass на вектор other.
6. Створює масив dets для зберігання визначників кожної матриці масиву mass.
7. Обчислює визначник кожної матриці масиву mass шляхом виклику оператора ! та зберігає їх у масиві dets.
8. Створює вектор result розміром A.size для зберігання результату.
9. Обчислює кожен елемент вектора result, який є часткою відповідного елемента масиву dets на базовий визначник base\_det.
10. Звільняє пам'ять, виділену для масиву mass та масиву dets.
11. Повертає вектор result як результат виконання оператора %.

**Тестування**

Створюємо 2 об’єкта класу Matrix та заповнюємо їх значеннями випадкових величин

int size = 3;

Matrix<int>\* A = new Matrix<int>(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

A->setValue(i, j, rand()%100);

}

}

cout << "Перша матриця: " << endl;

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

cout << A->getValue(i, j) << " ";

}

cout << endl;

}

Matrix<int>\* B = new Matrix<int>(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

B->setValue(i, j, rand() % 100);

}

}

cout << "Друга матриця: " << endl;

for (size\_t i = 0; i < size; i++) {

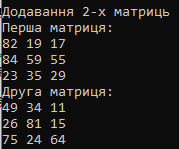
for (size\_t j = 0; j < size; j++) {

cout << B->getValue(i, j) << " ";

}

cout << endl;

}



Виконуємо додавання та віднімання матриць та виводимо результат

Matrix<int>\* C = new Matrix<int>(size);

\*C = \*A + \*B;

cout << "Результат додавання матриць" << endl;

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

cout << C->getValue(i, j) << " ";

}

cout << endl;

}

cout << "Віднімання матриць: " << endl;

\*C = \*A - \*B;

for (size\_t i = 0; i < size; i++) {

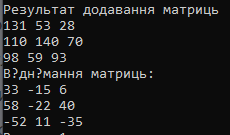
for (size\_t j = 0; j < size; j++) {

cout << C->getValue(i, j) << " ";

}

cout << endl;

}



Створюємо 2 об’єкта класу Vector та заповнюємо випадковими величинами

Vector <int> \*A1 = new Vector<int>(size);

Vector <int>\* B1 = new Vector<int>(size);

for (size\_t i = 0; i < size; i++)

{

A1->setValue(i, rand() % 100);

B1->setValue(i, rand() % 100);

}

cout << "Вектор 1" << endl;

for (size\_t i = 0; i < size; i++)

{

cout << A1->getValue(i) << " ";

}

cout << endl;

cout << "Вектор 2" << endl;

for (size\_t i = 0; i < size; i++)

{

cout << B1->getValue(i) << " ";

}

cout << endl;



Виконуємо додавання та віднімання векторів

cout << "Додавання векторів" << endl;

Vector <int>\* C1 = new Vector<int>(size);

\*C1 = \*A1 + \*B1;

for (size\_t i = 0; i < size; i++)

{

cout << C1->getValue(i) << " ";

}

cout << endl;

cout << "Віднімання векторів" << endl;

\*C1 = \*A1 - \*B1;

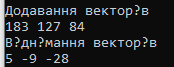
for (size\_t i = 0; i < size; i++)

{

cout << C1->getValue(i) << " ";

}

cout << endl;



Виконуємо множення матриці на вектор

\*C = \*A \* \*A1;

cout << "Множення першої матриці на вектор 1" << endl;

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

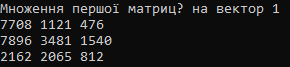
{

cout << C->getValue(i,j) << " ";

}

cout << endl;

}



Пошук оберненої матриці:

\*C = ~\*A;

cout << endl << "Пошук оберненої першої матриці" << endl;

for (size\_t i = 0; i < size; i++) {

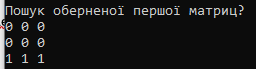
for (size\_t j = 0; j < size; j++) {

cout << C->getValue(i, j) << " ";

}

cout << endl;

}



Розв’язування системи лінійних рівнянь методом Крамера:

cout << "Розв'язування систем лінійних рівнянь методом Крамера:" << endl;

int x = 1;

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

cout << A->getValue(i, j) << " \* x" << x << " + ";

x++;

}

x = 1;

cout << " = " << A1->getValue(i) << endl;

}

Vector <int>\* kramer = new Vector<int>(size);

\*kramer = \*A % \*A1;

x = 1;

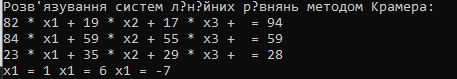
for (size\_t i = 0; i < size; i++)

{

cout << "x" << x << " = " << kramer->getValue(i) << " ";

}

cout << endl;



Скалярний добуток векторів та пошук довжин векторів:

int scalar;

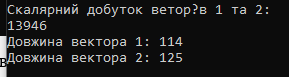
scalar = \*A1 \* \*B1;

cout << "Скалярний добуток веторів 1 та 2: " << endl;

cout << scalar << endl;

cout << "Довжина вектора 1: " << A1->len() << endl;

cout << "Довжина вектора 2: " << B1->len() << endl;



Після закінчення всіх обчислень обов’язково очищаємо динамічну пам’ять:

// Звільняємо пам'ять

delete A;

delete B;

delete C;

delete A1;

delete B1;  
 delete kramer;

**Висновок**

У даній курсовій роботі було розглянуто питання написання бібліотечних засобів для розв'язування задач лінійної алгебри з використанням об'єктно-орієнтованого підходу. Головною метою роботи було створення класових типів чисельної квадратної матриці та одновимірного динамічного масиву з використанням шаблонів для конкретизації типу елементів.

У роботі було розроблено конструктори, деструктори та конструктор копіювання для коректного створення та знищення об'єктів динамічного типу. Також було реалізовано перевантажені оператори для виконання операцій з матрицями та векторами, такі як складання, віднімання, множення, введення та виведення у файл, пошук оберненої матриці, розв'язування систем лінійних рівнянь методом Крамера та скалярне множення векторів. Крім того, були реалізовані компонентні функції для обчислення визначника матриці та довжини вектору.

Окремо була розроблена програма для тестування розроблених програмних засобів, яка демонструє їх функціональні можливості. Тестування дозволило перевірити коректність реалізованих функцій та визначити їх ефективність у різних сценаріях використання.

У результаті даної курсової роботи було успішно реалізовано бібліотечні засоби для розв'язування задач лінійної алгебри, що дозволяють зручно та ефективно виконувати операції з матрицями та векторами. Розроблені програмні засоби можуть бути використані у різних областях, де виникає потреба у роботі з лінійною алгеброю, і сприяють підвищенню продуктивності та зручності роботи з цими об'єктами.

Курсова робота має велике значення для студента 3 курсу коледжу, оскільки вона дозволяє отримати практичні навички в розробці бібліотек та застосуванні об'єктно-орієнтованого підходу до програмування. Результати роботи можуть бути використані для подальшого удосконалення програмних засобів та розширення їх функціональності.

**Список використаних джерел**

iostream, fstream https://en.cppreference.com/w/cpp/io

cmath https://en.cppreference.com/w/cpp/numeric/math

except <https://en.cppreference.com/w/cpp/error>

Метод Крамера <https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%9A%D1%80%D0%B0%D0%BC%D0%B5%D1%80%D0%B0>

Додаток:

kursova.h:

#pragma once

#include <fstream>

#include <stdexcept>

#include <cmath>

using namespace std;

template <class T>

class Vector {

private:

T\* data;

int size;

public:

//Конструктор

Vector(int size) {

this->size = size;

data = new T[size];

}

//Конструктор копіювання

Vector(const Vector& other) {

size = other.size;

data = new T[size];

for (size\_t i = 0; i < size; i++) {

data[i] = other.data[i];

}

}

//Деструктор

~Vector() {

delete[] data;

}

//Отримання розміру вектора

int getSize() {

return size;

}

//Отримання значення вектора

T getValue(int i) {

return data[i];

}

//Запис значення в вектор

void setValue(int i, T value) {

data[i] = value;

}

//Перевантаження оператора << (Читання з файлу)

void operator<<(ifstream& file) {

for (size\_t i = 0; i < size; i++)

{

file >> data[i];

}

file.close();

}

//Перевантаження оператора >> (Запис в файл)

void operator>>(ofstream& file) {

for (size\_t i = 0; i < size; i++)

{

file << data[i] << "\t";

}

file << endl;

file.close();

}

T len() {

T result;

T promiz = 0.0;

for (size\_t i = 0; i < size; i++)

{

promiz += pow(data[i], 2);

}

result = pow(promiz, 1. / 2.);

return result;

}

friend T operator\*(Vector A, Vector B) {

if (A.size != B.size) {

throw invalid\_argument("Vectors size`s are not equal");

}

T promiz = 0.0;

for (size\_t i = 0; i < A.size; i++)

{

promiz += A.data[i] \* B.data[i];

}

return promiz;

}

friend Vector operator+(Vector A, Vector B) {

if (A.size != B.size) {

throw invalid\_argument("Vectors size`s are not equal");

}

Vector promiz(A.size);

for (size\_t i = 0; i < A.size; i++)

{

promiz.data[i] = A.data[i] + B.data[i];

}

return promiz;

}

friend Vector operator-(Vector A, Vector B) {

if (A.size != B.size) {

throw invalid\_argument("Vectors size`s are not equal");

}

Vector promiz(A.size);

for (size\_t i = 0; i < A.size; i++)

{

promiz.data[i] = A.data[i] - B.data[i];

}

return promiz;

}

};

template <class T>

class Matrix {

private:

int size;

T\*\* data;

Matrix() {

};

void init(int size) {

this->size = size;

data = (T\*\*) new T \* [size];

for (size\_t i = 0; i < size; i++)

{

data[i] = (T\*) new T[size];

}

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

data[i][j] = 0;

}

}

}

public:

//Конструктор

Matrix(int size) {

this->size = size;

data = (T\*\*) new T \* [size];

for (size\_t i = 0; i < size; i++)

{

data[i] = (T\*) new T[size];

}

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

data[i][j] = 0;

}

}

}

//Конструктор копіювання

Matrix(const Matrix& other) {

if (other.data == nullptr) return;

size = other.size;

data = (T\*\*) new T \* [size];

for (size\_t i = 0; i < size; i++)

{

data[i] = (T\*)new T[size];

}

for (size\_t i = 0; i < size; i++)

{

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

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

}

}

}

//Деструктор

~Matrix() {

if (data != nullptr) {

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

delete[] data[i];

}

delete[] data;

}

}

//Метод для повернення значення розміру матриці

int getSize() {

return size;

}

//Метод для повернення значення матриці

T getValue(int i, int j) {

if (i < 0 || j < 0) {

i = 0;

j = 0;

}

else if (i >= size) {

i = size - 1;

}

else if (j >= size) {

j = size - 1;

}

return data[i][j];

}

//Метод для запису значення в матрицю

void setValue(int i, int j, T value) {

data[i][j] = value;

}

//Перевантаження оператора додавання

friend Matrix operator+(Matrix A, Matrix other) {

if (other.size != A.size) {

throw invalid\_argument("Matrix size`s are not equal");

}

Matrix result(A.size);

for (size\_t i = 0; i < A.size; i++)

{

for (size\_t j = 0; j < A.size; j++)

{

result.data[i][j] = A.data[i][j] + other.data[i][j];

}

}

return result;

}

//Перевантаження оператора віднімання

friend Matrix operator-(Matrix A, Matrix other) {

if (other.size != A.size) {

throw invalid\_argument("Matrix size`s are not equal");

}

Matrix result(A.size);

for (size\_t i = 0; i < A.size; i++)

{

for (size\_t j = 0; j < A.size; j++)

{

result.data[i][j] = A.data[i][j] - other.data[i][j];

}

}

return result;

}

//Перевантаження оператора множення(Множення матриці на вектор)

template <typename T>

friend Matrix<T> operator\*(Matrix<T> A, Vector<T> other) {

if (other.getSize() != A.size) {

throw std::invalid\_argument("Matrix and Vector sizes are not equal");

}

int size = A.size;

Matrix<T> result(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

result.data[i][j] = A.data[i][j] \* other.getValue(j);

}

}

return result;

}

//Перевантаження оператора << (Читання з файлу)

void operator<<(ifstream& file) {

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

file >> data[i][j];

}

}

file.close();

}

//Перевантаження оператора >> (Запис в файл)

void operator>>(ofstream& file) {

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

file << data[i][j] << "\t";

}

file << endl;

}

}

//Перевантаження оператора ~ (Пошук оберненої матриці)

Matrix<T> operator~() const {

if (size == 0) {

throw std::invalid\_argument("Size must be more than 0");

}

// Створення розширеної матриці з оберненими та початковими елементами

Matrix<T> extendedMatrix(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

extendedMatrix.setValue(i, j, data[i][j]);

extendedMatrix.setValue(i, j + size, (i == j) ? 1.0 : 0.0);

}

}

// Виконання методу Гаусса-Жордана для знаходження оберненої матриці

for (size\_t i = 0; i < size; i++) {

// Ділення поточного рядка на ведучий елемент

T pivot = extendedMatrix.getValue(i, i);

for (size\_t j = 0; j < 2 \* size; j++) {

extendedMatrix.setValue(i, j, extendedMatrix.getValue(i, j) / pivot);

}

// Віднімання від інших рядків поточного рядка, помноженого на відповідний елемент

for (size\_t j = 0; j < size; j++) {

if (j != i) {

T factor = extendedMatrix.getValue(j, i);

for (size\_t k = 0; k < 2 \* size; k++) {

extendedMatrix.setValue(j, k, extendedMatrix.getValue(j, k) - factor \* extendedMatrix.getValue(i, j));

}

}

}

}

// Створення та заповнення оберненої матриці

Matrix<T> inverseMatrix(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

inverseMatrix.setValue(i, j, extendedMatrix.getValue(i, j + size));

}

}

return inverseMatrix;

}

T operator!() {

int size = getSize();

T determinant = 1;

if (size == 1) {

determinant = data[0][0];

}

else {

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

Matrix<T> submatrix(size - 1);

int iTemp = 0;

for (int i = 1; i < size; i++) {

int jTemp = 0;

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

if (j == k) {

continue;

}

submatrix.setValue(iTemp, jTemp, data[i][j]);

jTemp++;

}

iTemp++;

}

T subDeterminant = !submatrix;

determinant += ((k % 2 == 0) ? 1 : -1) \* data[0][k] \* subDeterminant;

}

}

return determinant;

}

//Перевантаження оператора % (Метод Крамера)

friend Vector<T> operator%(Matrix<T> A, Vector<T>other) {

if (other.getSize() != A.size) {

throw std::invalid\_argument("Matrix and Vector sizes must be equal");

}

Matrix<T>\* mass = new Matrix<T>[A.size];// Створення масиву матриць

for (size\_t i = 0; i < A.size; i++)

{

mass[i].init(A.size);

}

T base\_det = !A; // Визначник матриці

for (size\_t i = 0; i < A.size; i++) {

for (size\_t j = 0; j < A.size; j++) {

for (size\_t k = 0; k < A.size; k++) {

mass[i].setValue(j, k, A.data[j][k]); // Заповнення кожної матриці значеннями

}

}

}

for (size\_t i = 0; i < A.size; i++) {

for (size\_t j = 0; j < A.size; j++) {

mass[i].data[j][i] = other.getValue(j); // Заповнення певного стовпця кожної матриці значеннями вектора

}

}

T\* dets = new T[A.size]; // Створення масиву для визначників

for (size\_t i = 0; i < A.size; i++) {

dets[i] = !mass[i]; // Заповнення масиву визначників

}

Vector<T> result(A.size); // Створення вектора результату

for (size\_t i = 0; i < A.size; i++) {

result.setValue(i, (dets[i] / base\_det)); // Обчислення кожного елемента вектора результату

}

delete[] mass; // Звільнення пам'яті для масиву матриць

delete[] dets; // Звільнення пам'яті для масиву визначників

return result;

}

};

Kursova.cpp:

#include <iostream>

#include "kursova.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "UKR");

srand(time(NULL));

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* Міністерство освіти і науки \*" << endl;

cout << "\* ВСП-Технічний фаховий коледж НУ Львівська Політехніка \*" << endl;

cout << "\* \*" << endl;

cout << "\* Відділення інформаційних технологій\*" << endl;

cout << "\* та комп'ютерної техніки\*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* Курсова робота \*" << endl;

cout << "\* з об'єктно-орієнтованого програмування \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* Виконав: \*" << endl;

cout << "\* студент групи 32ПЗ \*" << endl;

cout << "\* Боднарчук Віктор Романович \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* Львів 2023 \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "Додавання 2-х матриць" << endl;

int size = 3;

Matrix<int>\* A = new Matrix<int>(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

A->setValue(i, j, rand()%100);

}

}

cout << "Перша матриця: " << endl;

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

cout << A->getValue(i, j) << " ";

}

cout << endl;

}

Matrix<int>\* B = new Matrix<int>(size);

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

B->setValue(i, j, rand() % 100);

}

}

cout << "Друга матриця: " << endl;

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

cout << B->getValue(i, j) << " ";

}

cout << endl;

}

Matrix<int>\* C = new Matrix<int>(size);

\*C = \*A + \*B;

cout << "Результат додавання матриць" << endl;

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

cout << C->getValue(i, j) << " ";

}

cout << endl;

}

cout << "Віднімання матриць: " << endl;

\*C = \*A - \*B;

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

cout << C->getValue(i, j) << " ";

}

cout << endl;

}

Vector <int> \*A1 = new Vector<int>(size);

Vector <int>\* B1 = new Vector<int>(size);

for (size\_t i = 0; i < size; i++)

{

A1->setValue(i, rand() % 100);

B1->setValue(i, rand() % 100);

}

cout << "Вектор 1" << endl;

for (size\_t i = 0; i < size; i++)

{

cout << A1->getValue(i) << " ";

}

cout << endl;

cout << "Вектор 2" << endl;

for (size\_t i = 0; i < size; i++)

{

cout << B1->getValue(i) << " ";

}

cout << endl;

cout << "Додавання векторів" << endl;

Vector <int>\* C1 = new Vector<int>(size);

\*C1 = \*A1 + \*B1;

for (size\_t i = 0; i < size; i++)

{

cout << C1->getValue(i) << " ";

}

cout << endl;

cout << "Віднімання векторів" << endl;

\*C1 = \*A1 - \*B1;

for (size\_t i = 0; i < size; i++)

{

cout << C1->getValue(i) << " ";

}

cout << endl;

\*C = \*A \* \*A1;

cout << "Множення першої матриці на вектор 1" << endl;

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

cout << C->getValue(i,j) << " ";

}

cout << endl;

}

\*C = ~\*A;

cout << endl << "Пошук оберненої першої матриці" << endl;

for (size\_t i = 0; i < size; i++) {

for (size\_t j = 0; j < size; j++) {

cout << C->getValue(i, j) << " ";

}

cout << endl;

}

cout << "Розв'язування систем лінійних рівнянь методом Крамера:" << endl;

int x = 1;

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size; j++)

{

cout << A->getValue(i, j) << " \* x" << x << " + ";

x++;

}

x = 1;

cout << " = " << A1->getValue(i) << endl;

}

Vector <int>\* kramer = new Vector<int>(size);

\*kramer = \*A % \*A1;

x = 1;

for (size\_t i = 0; i < size; i++)

{

cout << "x" << x << " = " << kramer->getValue(i) << " ";

}

cout << endl;

int scalar;

scalar = \*A1 \* \*B1;

cout << "Скалярний добуток веторів 1 та 2: " << endl;

cout << scalar << endl;

cout << "Довжина вектора 1: " << A1->len() << endl;

cout << "Довжина вектора 2: " << B1->len() << endl;

// Звільняємо пам'ять

delete A;

delete B;

delete C;

delete A1;

delete B1;

delete kramer;

//delete kramer;

return 0;

}