Лабораторна робота №3.

Тема: *Перевантаження операцій та асоціативні класи.*

Методичні вказівки

1. Дружні функції та класи.

Іноді вимагаються виключення з правил доступу, коли деякої чи функції класу потрібно дозволити доступ до особистої частини об'єкта класу. Тоді у визначенні класу, до об'єктів якого дозволяється такий доступ, повинне бути оголошення функції чи іншого класу як `friend` (дружній). Оголошення дружньої функції це прототип функції з ключовим словом `friend`. Оголошення дружнього класу це прототип класу з ключовим словом `friend`, тоді всі функції класу дружнього класу мають доступ до поточного класу. Загальна схема оголошення така:

```
class   A{
        int     x;    // Особиста частина класу
        ...
friend  class   B;    // Клас B дружні A Функції класу B дружні A
                      // (мають доступ до приватної частини A)
friend  void  C::fun(A&);// Елемент-функція fun класу C має
                         // доступ до приватної частини A
friend  void    gfun(A&,int);// Функція gfun дружня класу A
        };
class   B
        {
public: int     fun1(A&);
        void    fun2(A&);
        };
class   C
        {
public: void    fun(A&);
        void    rfune(inr);
        };
```

Перевантаження бінарних і унарних операцій.

Наступним кроком у використанні класу як базового типу даних є перевизначення операцій мови, у яких один чи два операнди можуть бути об'єктами класу. Це досягається введенням функції-елемента спеціального виду, звертання до якої компілятор формує при трансляції такої операції. Природно, що така функція повинна мати результат, відмінний від void, якщо передбачається використання цієї операції усередині іншого виразу.

Перевизначення операцій здійснюється в рамках стандартного синтаксису мови С++, тобто позначення операцій і кількість операндів залишається незмінним. Можна описувати функції, що визначають значення наступних операцій:

```
+  -  *  /  %  ^  &  |  ~  !
=  <  >  += -= *= /= %= ^= &=
|= << >> >>= <<= == != <= >= &&
|| ++ -- [] () new delete
```

Останні чотири - це індексування, виклик функції, виділення вільної пам'яті і звільнення вільної пам'яті. Змінити пріоритети перерахованих операцій неможливо, як неможливо змінити і синтаксис виразів. Не можна, наприклад, визначити унарную операцію % або як бінарну !. Неможливо визначити нові лексичні символи операцій.

Для перевизначення операції використовується особлива форма функції-елемента з заголовком такого виду:

```
тип operator операція( список_параметров-операндів)
```

Ім'я функції-операції складається з ключового слова **operator** і символу даної **операції** в синтаксисі мови С++. Результат функції (тип) є результатом перевизначеної операції. Функції перевизначеної операції можуть бути функціями класу (для унарних – без параметрів, так як об'єкт класу є операндом, для бінарних – з одним параметром, який є другим операдом, а перший об'єкт класу) або дружніми функціями (відповідно – для унарних з одним параметром, для бінарних з двома). Виключенням є операції постфіксних ++ та --, в додається формальний параметр типу **int** , що відрізняти від префіксного.

Для розуміння необхідності та важливості перевантаження операцій в С++ розглянемо приклад обчислення векторів з аналітичної геометрії.

Нехай задано вектори : $\vec{a}(1,2,3)$, $\vec{b}(3,2,1)$ та $\vec{c}(1,2,1)$.

Необхідно обчислити :

$$\vec{r} = 5\vec{a} + 7\vec{b} - 3\vec{c}\,, \qquad \vec{q} = -2\vec{a} - \vec{b} + 7\vec{c} \text{ та } \vec{p} = (\vec{a}^T\vec{b})\vec{c} + \vec{r} \times \vec{q}\,,$$

де $\vec{a}^T\vec{b}$ - скалярний добуток векторів, $\vec{r} \times \vec{q}$ - векторний добуток векторів

Розглянемо вирішення цієї задачі з перевантаженням операцій.

```
#include<iostream>
#include<fstream>
using namespace std;
class Vector3
{
	double d[3];
public:
	Vector3() { d[0] = d[1] = d[2] = 0; }
	Vector3(Vector3& src);
	double  operator*(Vector3& b);
	Vector3 operator^(Vector3& b);
	Vector3 operator*(double b);
	Vector3 operator+(Vector3& b);
	Vector3 operator-(Vector3& b);
	friend istream& operator >> (istream& is, Vector3& a);
	friend ostream& operator<<(ostream& os, Vector3& a);
};
Vector3::Vector3(Vector3& src)
{
	for (int i = 0;i<3;i++) d[i] = src.d[i];
}
double  Vector3::operator*(Vector3& b)
{
	int i;
	double s = 0;
	for (i = 0;i<3;i++) s += d[i] * b.d[i];
	return s;
}
Vector3  Vector3:: operator^(Vector3& b)
{
	Vector3 temp;
	temp.d[0] = d[1] * b.d[2] - d[2] * b.d[1];
	temp.d[1] = d[2] * b.d[0] - d[0] * b.d[2];
	temp.d[0] = d[0] * b.d[1] - d[1] * b.d[0];
	return temp;
}
Vector3  Vector3:: operator*(double b)
{
	int i;
	Vector3 temp;
	for (i = 0;i<3;i++) temp.d[i] = d[i] * b;
	return temp;
}
Vector3  Vector3::operator+(Vector3& b)
```

```
{
	int i;
	Vector3 temp;
	for (i = 0;i<3;i++) temp.d[i] = d[i] + b.d[i];
	return temp;
}
Vector3 Vector3:: operator-(Vector3& b)
{
	int i;
	Vector3 temp;
	for (i = 0;i<3;i++) temp.d[i] = d[i] - b.d[i];
	return temp;
}
istream& operator >> (istream& is, Vector3& a)
{
	int i;
	for (i = 0;i<3;i++) is >> a.d[i];
	return is;
}
ostream& operator<<(ostream& os, Vector3& a)
{
	int i;
	for (i = 0;i<3;i++) os << "  " << a.d[i] << "  ";
	return os;
}
void main()
{
	Vector3 a, b, c, r, p, q;
	cout << "\nInput vector a \n"; cin >> a;
	cout << " Input vector b \n";  cin >> b;
	cout << " Input vector c \n";  cin >> c;
	r = a * 5 + b * 7 - c * 3;
	// r=5a+7b-3c
	cout << "\n r = " << r << endl;

	q = a*(-2) - b + c * 7;
	// q = -2a-b+7c
	cout << "\n q = " << q << endl;

	p = c*(a*b) - (r^q);
	// p = (a T b)*c - r x q
	cout << "\n p = " << p << endl;
}
```

Асоціативні масиви.

Асоціати́вний маси́в (англ. *associative array*) (або **словник**, **хеш**, в англійській літературі також застосовуються терміни *associative container*, *map*, *mapping*, *hash*, *dictionary*, *finite map*) тип даних, що дозволяє зберігати дані у вигляді набору пар *ключ—значення* та доступом до значень за їх ключем.

Реалізації асоціативних масивів зазвичай підтримують операції додавання пари, а також пошуку та видалення пари за ключем:

- **Вставити** (ключ, значення)
- **Шукати** (ключ)
- **Вилучити** (ключ).

Завдання до лабораторної роботи:

Розробити та реалізувати класи згідно варіантів. Передбачити введення початкових даних: з клавіатури, файлу та використовуючи датчик випадкових чисел. Написати тестові програми для кожної задачі.

У завданнях, за вибором студента, перевантажити достатньо тільки 30% операцій із всіх заданих операцій.

Завдання 1. Варіанти задач. Перевантаження операцій. Вектори.

Задача 1.1.

Створити тип даних - клас **VectorInt** (вектор цілих чисел), який має вказівник на **int**, число елементів **size** і змінну стану **codeError**. У класі визначити

- конструктор без параметрів(виділяє місце для одного елемента та інінціалізує його в нуль);
- конструктор з одним параметром - розмір вектора(виділяє місце та інінціалізує масив значенням нуль);
- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце (значення перший аргумент) та інінціалізує значенням другого аргументу) ;
- конструктор копіювання;
- деструктор звільняє пам'ять;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних **++** та **--**: одночасно збільшує (зменшує) значення елементів масиву на **1**;
  - унарної логічної **!** (заперечення): повертає значення **true**, якщо елементи якщо **size** не дорівнює – нулю, інакше **false**;
  - унарної побітової **~** (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
  - унарний арифметичний **-** (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
  - присвоєння **=**: копіює вектор(перевантажити через функцію класу);
  - арифметичних бінарні:
    - **+** додавання:
      - для двох векторів;
      - для вектора і скаляра типу **int**;
    - **-** (віднімання):
      - для двох векторів
      - для вектора і скаляра типу **int**;
    - *****(множення)
      - для двох векторів
      - для вектора і скаляра типу **int**;
    - **/** (ділення)
      - для вектора і скаляра типу **int**;
    - **%** (остача від ділення)
      - для вектора і скаляра типу **int**;
  - побітові бінарні

- `|` (побітове додавання)
  - для двох векторів
  - для вектора і скаляра типу `int`;
- `^` (побітове додавання за модулем 2)
  - для двох векторів
  - для вектора і скаляра типу `int`;
- `|` (побітове множення)
  - двох векторів
  - вектора і скаляра типу `int`;

- присвоєння з
  - `+=` додаванням:
    - для двох векторів
    - для вектора і скаляра типу `int`
  - `-=` (віднімання):
    - для двох векторів
    - для вектора і скаляра типу `int`;
  - `*=`(множення)
    - для двох векторів
    - для вектора і скаляра типу `int`;
  - `/=`(ділення)
    - для двох векторів
    - для вектора і скаляра типу `int`;
  - `%=` (остача від ділення)
    - для двох векторів
    - для вектора і скаляра типу `int`;
  - `|=` (побітове додавання)
    - для двох векторів
    - для вектора і скаляра типу `int`;
  - `^=`(побітове додавання за модулем 2)
    - для двох векторів
    - для вектора і скаляра типу `int`;
  - `|=` (побітове множення)
    - двох векторів
    - вектора і скаляра типу `int`;
- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  - введення `>>` (побітовий зсув право) ;
  - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом, якщо індекс невірний функція вказує на останній елемент масиву та встановлює код помилки у змінну `codeError`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:

- `>` (більше) для двох векторів;
- `>=` (більше рівне) для двох векторів;
- `<` (менше) для двох векторів;
- `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, коли не вистачає пам'яті, виходить за межі масиву. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.2.

Створити тип даних - клас вектор **`VectorFloat`** (вектор дійсних чисел), який має вказівник на **`float`**, число елементів **`size`** і змінну стану **`codeError`**. У класі визначити

- конструктор без параметрів(виділяє місце для одного елемента та інінціалізує його в нуль);
- конструктор з одним параметром - розмір вектора(виділяє місце та інінціалізує масив значенням нуль);
- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце (значення перший аргумент) та інінціалізує значенням другого аргументу).
- деструктор звільняє пам'ять;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1.0`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
  - присвоєння `=`: копіює вектор(перевантажити через функцію класу);
  - арифметичних бінарні:
    - `+` додавання:
      - для двох векторів;
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `float`
    - `-` (віднімання):
      - для двох векторів;
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `float`;
    - `*`(множення)
      - ~~для двох векторів~~
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `float`;
    - `/` (ділення)
      - ~~для двох векторів~~
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `float`;
    - `%` (остача від ділення для цілої части дійсного числа)
      - для двох векторів;
      - для вектора і скаляра типу `int`;

- для вектора і скаляра типу `float`;

- присвоєння з
  - `+=` додаванням:
    - для двох векторів;
    - для вектора і скаляра типу `int;`
    - для вектора і скаляра типу `float`;
  - `-=` (віднімання):
    - для двох векторів;
    - для вектора і скаляра типу `int;`
    - для вектора і скаляра типу `float`;
  - `*=`(множення)
    - для двох векторів
    - для вектора і скаляра типу `int`;
    - для вектора і скаляра типу `float`;
  - `/=`(ділення)
    - для вектора і скаляра типу `int`;
  - `%=` (остача від ділення для цілої части дійсного числа)
    - для вектора і скаляра типу `int`;
    - для вектора і скаляра типу `float`;
- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  - введення `>>` (побітовий зсув право) ;
  - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом, якщо індекс невірний функція вказує на останній елемент масиву та встановлює код помилки у змінну `codeError`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
  - `>` (більше) для двох векторів;
  - `>=` (більше рівне) для двох векторів;
  - `<` (менше) для двох векторів;
  - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, коли не вистачає пам'яті, виходить за межі масиву. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.3.

Створити тип даних - клас вектор `VectorDouble` (вектор дійсних чисел), який має вказівник на `double`, число елементів `size` і змінну стану `codeError`. У класі визначити

- конструктор без параметрів(виділяє місце для одного елемента та інінціалізує його в нуль);
- конструктор з одним параметром - розмір вектора(виділяє місце та інінціалізує масив значенням нуль);

- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце (значення перший аргумент) та інінціалізує значенням другого аргументу).
- деструктор звільняє пам'ять;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1.0`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
  - присвоєння `=`: копіює вектор(перевантажити через функцію класу);
  - арифметичних бінарні:
    - `+` додавання:
      - для двох векторів;
      - для вектора і скаляра типу `double`;
      - для вектора і скаляра типу `float`
    - `-` (віднімання):
      - для двох векторів;
      - для вектора і скаляра типу `double`;
      - для вектора і скаляра типу `float`;
    - `*`(множення)
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `double`;
      - для вектора і скаляра типу `float`;
    - `/` (ділення)
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `double`;
      - для вектора і скаляра типу `float`;
    - `%` (остача від ділення для цілої части дійсного числа)
      - для вектора і скаляра типу `int`;
  - присвоєння з
    - `+=` додаванням:
      - для двох векторів;
      - для вектора і скаляра типу `int;`
      - для вектора і скаляра типу `float`;
    - `-=` (віднімання):
      - для двох векторів;
      - для вектора і скаляра типу `int;`
      - для вектора і скаляра типу `float`;
    - `*=`(множення)
      - для двох векторів
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `float`;
    - `/=`(ділення)

        - для вектора і скаляра типу `int`;
    - `%=` (остача від ділення для цілої части дійсного числа)
        - для вектора і скаляра типу `int`;
        - для вектора і скаляра типу `float`;
- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
    - введення `>>` (побітовий зсув право) ;
    - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом, якщо індекс невірний функція вказує на останній елемент масиву та встановлює код помилки у змінну `codeError`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
    - `>` (більше) для двох векторів;
    - `>=` (більше рівне) для двох векторів;
    - `<` (менше) для двох векторів;
    - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, коли не вистачає пам'яті, виходить за межі масиву. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.4.

Створити тип даних - клас `VectorShort` (вектор цілих чисел), який має вказівник на `short`, число елементів `size` і змінну стану `codeError`. У класі визначити

- конструктор без параметрів(виділяє місце для одного елемента та інінціалізує його в нуль);
- конструктор з одним параметром - розмір вектора(виділяє місце та інінціалізує масив значенням нуль);
- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце (значення перший аргумент) та інінціалізує значенням другого аргументу) ;
- конструктор копіювання;
- деструктор звільняє пам'ять;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
    - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
    - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
    - унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
    - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
    - присвоєння `=`: копіює вектор(перевантажити через функцію класу);

- арифметичних бінарні:
  - `+` додавання:
    - для двох векторів;
    - для вектора і скаляра типу `short`;
  - `-` (віднімання):
    - для двох векторів;
    - для вектора і скаляра типу `short`;
  - `*`(множення)
    - для двох векторів
    - для вектора і скаляра типу `short`;
  - `/` (ділення)
    - для вектора і скаляра типу `short`;
  - `%` (остача від ділення)
    - для вектора і скаляра типу `short`;
- побітові бінарні
  - `|` (побітове додавання)
    - для двох векторів
    - для вектора і скаляра типу `short`
  - `^` (побітове додавання за модулем 2)
    - для двох векторів
    - для вектора і скаляра типу `short`;
  - `|` (побітове множення)
    - двох векторів
    - вектора і скаляра типу `short`;
- присвоєння з
  - `+=` додаванням:
    - для двох векторів
    - для вектора і скаляра типу `short`;
  - `-=` (віднімання):
    - для двох векторів
    - для вектора і скаляра типу `short`;
  - `*=`(множення)
    - для вектора і скаляра типу `short`;
  - `/=`(ділення)
    - для вектора і скаляра типу `short`;
  - `%=` (остача від ділення)
    - для вектора і скаляра типу `short`;
  - `|=` (побітове додавання)
    - для двох векторів
    - для вектора і скаляра типу `short`;
  - `^=`(побітове додавання за модулем 2)
    - для двох векторів
    - для вектора і скаляра типу `short`;
  - `|=` (побітове множення)
    - двох векторів;
    - вектора і скаляра типу `short`;

- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  - введення `>>` (побітовий зсув право) ;
  - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом, якщо індекс невірний функція вказує на останній елемент масиву та встановлює код помилки у змінну `codeError`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
  - `>` (більше) для двох векторів;
  - `>=` (більше рівне) для двох векторів;
  - `<` (менше) для двох векторів;
  - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, коли не вистачає пам'яті, виходить за межі масиву. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.5.

Створити тип даних - клас `VectorLong` (вектор цілих чисел), який має вказівник на `long`, число елементів `size` і змінну стану `codeError`. У класі визначити

- конструктор без параметрів(виділяє місце для одного елемента та інінціалізує його в нуль);
- конструктор з одним параметром - розмір вектора(виділяє місце та інінціалізує масив значенням нуль);
- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце (значення перший аргумент) та інінціалізує значенням другого аргументу) ;
- конструктор копіювання;
- деструктор звільняє пам'ять;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
  - унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
  - присвоєння `=`: копіює вектор(перевантажити через функцію класу);
  - арифметичних бінарні:
    - `+` додавання:
      - для двох векторів;
      - для вектора і скаляра типу `long`;
    - `-` (віднімання):

        - для двох векторів;
        - для вектора і скаляра типу `long`;
    - `*`(множення)
        - для двох векторів;
        - для вектора і скаляра типу `long`;
    - `/` (ділення)
        - для вектора і скаляра типу `long`;
    - `%` (остача від ділення)
        - для вектора і скаляра типу `long`;
- побітові бінарні
    - `|` (побітове додавання)
        - для двох векторів;
        - для вектора і скаляра типу `long`;
    - `^` (побітове додавання за модулем 2)
        - для двох векторів
        - для вектора і скаляра типу `long`;
    - `|` (побітове множення)
        - двох векторів;
        - вектора і скаляра типу `long`;
- присвоєння з
    - `+=` додаванням:
        - для двох векторів;
        - для вектора і скаляра типу `long`;
    - `-=` (віднімання):
        - для двох векторів;
        - для вектора і скаляра типу `long`;
    - `*=`(множення)
        - для вектора і скаляра типу `long`;
    - `/=`(ділення)
        - для вектора і скаляра типу `long`;
    - `%=` (остача від ділення)
        - для вектора і скаляра типу `long`;
    - `|=` (побітове додавання)
        - для двох векторів;
        - для вектора і скаляра типу `long`;
    - `^=`(побітове додавання за модулем 2)
        - для двох векторів;
        - для вектора і скаляра типу `long`;
    - `|=` (побітове множення)
        - двох векторів;
        - вектора і скаляра типу `long`;
- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
    - введення `>>` (побітовий зсув право) ;
    - введення `<<` (побітовий зсув ліво);

- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом, якщо індекс невірний функція вказує на останній елемент масиву та встановлює код помилки у змінну `codeError`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()`;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
  - `>` (більше) для двох векторів;
  - `>=` (більше рівне) для двох векторів;
  - `<` (менше) для двох векторів;
  - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, коли не вистачає пам'яті, виходить за межі масиву. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.6.

Створити тип даних - клас вектор `Vector3F`, який має поля `x`, `y` та `z` типу `float` і змінну стану `State` . У класі визначити

- конструктор без параметрів(інінціалізує поля в нуль);
- конструктор з одним параметром типу `float` (інінціалізує поля `x`, `y` та `z` значенням параметру);
- конструктор з одним параметром вказівник на тип (інінціалізує поля `x`, `y` та `z` значенням масиву за вказівником, якщо вказівник `NULL (nulptr)` то встановити код помилки);
- деструктор із виведенням інформації про стан вектора;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1.0`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `x`, `y` та `z` не дорівнюють – нулю, інакше `false`;
  - унарної побітової `~` (заперечення): повертає перпендикулярний вектор для заданого;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
  - присвоєння `=`: копіює вектор(перевантажити через функцію класу);
  - арифметичних бінарні:
    - `+` додавання:
      - для двох векторів;
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `float`
    - `-` (віднімання):
      - для двох векторів;
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `float`;
    - `*`(множення)

        - для двох векторів (скалярний добуток);
        - для вектора і скаляра типу `int`;
        - для вектора і скаляра типу `float`;
    - `/` (ділення)
        - для двох векторів (векторний добуток);
        - для вектора і скаляра типу `int`;
        - для вектора і скаляра типу `float`;
    - `%` (остача від ділення для цілої части дійсного числа)
        - для вектора і скаляра типу `int`;
        - для вектора і скаляра типу `float`;
- присвоєння з
    - `+=` додаванням:
        - для двох векторів;
        - для вектора і скаляра типу `int;`
        - для вектора і скаляра типу `float`;
    - `-=` (віднімання):
        - для двох векторів;
        - для вектора і скаляра типу `int;`
        - для вектора і скаляра типу `float`;
    - `*=`(множення)
        - ~~для двох векторів~~
        - для вектора і скаляра типу `int`;
        - для вектора і скаляра типу `float`;
    - `/=`(ділення)
        - для двох векторів (векторний добуток);
        - для вектора і скаляра типу `int`;
    - `%=` (остача від ділення для цілої части дійсного числа)
        - для вектора і скаляра типу `int`;
        - для вектора і скаляра типу `float`;
- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
    - введення `>>` (побітовий зсув право) ;
    - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом до `x`, `y` та `z`, якщо індекс невірний функція вказує на `z` та встановлює код помилки у змінну `State`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
    - `>` (більше) для двох векторів;
    - `>=` (більше рівне) для двох векторів;
    - `<` (менше) для двох векторів;
    - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, діленні на 0, при передачі NULL (nulptr) в конструкторі із вказівником. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.7.

Створити тип даних - клас вектор `Vector3D`, який має поля `x`, `y` та `z` типу `double` і змінну стану `State` . У класі визначити

- конструктор без параметрів(інінціалізує поля в нуль);
- конструктор з одним параметром типу `double` (інінціалізує поля `x`, `y` та `z` значенням параметру);
- конструктор з одним параметром вказівник на тип (інінціалізує поля `x`, `y` та `z` значенням масиву за вказівником, якщо вказівник `NULL (nulptr)` то встановити код помилки);
- деструктор із виведенням інформації про стан вектора;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1.0`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `x`, `y` та `z` не дорівнюють – нулю, інакше `false`;
  - унарної побітової `~` (заперечення): повертає перпендикулярний вектор для заданого;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
  - присвоєння `=`: копіює вектор(перевантажити через функцію класу);
  - арифметичних бінарні:
    - `+` додавання:
      - для двох векторів;
      - для вектора і скаляра типу `double`;
      - для вектора і скаляра типу `float`
    - `-` (віднімання):
      - для двох векторів;
      - для вектора і скаляра типу `double`;
      - для вектора і скаляра типу `float`;
    - `*`(множення)
      - для двох векторів (скалярний добуток);
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `float`;
    - `/` (ділення)
      - для двох векторів (векторний добуток);
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `double`;
    - `%` (остача від ділення для цілої части дійсного числа)
      - для вектора і скаляра типу `int`;
      - для вектора і скаляра типу `double`;
  - присвоєння з
    - `+=` додаванням:
      - для двох векторів;

- для вектора і скаляра типу `double;`
- для вектора і скаляра типу `float`;

- `-=` (віднімання):
  - для двох векторів;
  - для вектора і скаляра типу `double;`
  - для вектора і скаляра типу `float`;
- `*=`(множення)
  - для вектора і скаляра типу `double`;
  - для вектора і скаляра типу `float`;
- `/=`(ділення)
  - для двох векторів (векторний добуток);
  - для вектора і скаляра типу `int`;
- `%=` (остача від ділення для цілої части дійсного числа)
  - для вектора і скаляра типу `int`;

- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  - введення `>>` (побітовий зсув право) ;
  - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом до `x`, `y` та `z`, якщо індекс невірний функція вказує на `z` та встановлює код помилки у змінну `State`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
  - `>` (більше) для двох векторів;
  - `>=` (більше рівне) для двох векторів;
  - `<` (менше) для двох векторів;
  - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, діленні на 0, при передачі NULL (nulptr) в конструкторі із вказівником. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.8.

Створити тип даних - клас `VectorUShort` (вектор без знакових цілих чисел), який має вказівник на `unsigned short`, число елементів `num` і змінну стану `State`. У класі визначити

- конструктор без параметрів(виділяє місце для одного елемента та інінціалізує його в нуль);
- конструктор з одним параметром - розмір вектора(виділяє місце та інінціалізує масив значенням нуль);
- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце (значення перший аргумент) та інінціалізує значенням другого аргументу) ;
- конструктор копіювання;
- деструктор звільняє пам'ять;

- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `num` не дорівнює – нулю, інакше `false`;
  - унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
  - присвоєння `=`: копіює вектор(перевантажити через функцію класу);
  - арифметичних бінарні:
    - `+` додавання:
      - для двох векторів
      - для вектора і скаляра типу `unsigned short`;
    - `-` (віднімання):
      - для двох векторів
      - для вектора і скаляра типу `unsigned short`;
    - `*`(множення)
      - для двох векторів
      - для вектора і скаляра типу `unsigned short`;
    - `/` (ділення)
      - для вектора і скаляра типу `unsigned short`;
    - `%` (остача від ділення)
      - для вектора і скаляра типу `unsigned short`;
  - побітові бінарні
    - `|` (побітове додавання)
      - для двох векторів
      - для вектора і скаляра типу `unsigned short`;
    - `^` (побітове додавання за модулем 2)
      - для двох векторів
      - для вектора і скаляра типу `unsigned short`;
    - `|` (побітове множення)
      - двох векторів
      - вектора і скаляра типу `unsigned short`;
  - присвоєння з
    - `+=` додаванням:
      - для двох векторів;
      - для вектора і скаляра типу `unsigned short`;
    - `-=` (віднімання):
      - для двох векторів;
      - для вектора і скаляра типу `unsigned short`;
    - `*=`(множення)
      - для двох векторів;
      - для вектора і скаляра типу `unsigned short`;
    - `/=`(ділення)
      - для двох векторів;

        - для вектора і скаляра типу `unsigned short`;
    - `%=` (остача від ділення)
        - для двох векторів;
        - для вектора і скаляра типу `unsigned short`;
    - `|=` (побітове додавання)
        - для двох векторів
        - для вектора і скаляра типу `unsigned short`;
    - `^=`(побітове додавання за модулем 2)
        - для двох векторів
        - для вектора і скаляра типу `unsigned short`;
    - `|=` (побітове множення)
        - двох векторів
        - вектора і скаляра типу `unsigned short`;
- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
    - введення `>>` (побітовий зсув право) ;
    - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом, якщо невірний вказує на останній елемент масиву та встановлює код помилки у змінну `State`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
    - `>` (більше) для двох векторів;
    - `>=` (більше рівне) для двох векторів;
    - `<` (менше) для двох векторів;
    - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, коли не вистачає пам'яті, виходить за межі масиву. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.9.

Створити тип даних - клас `VectorUInt` (вектор без знакових цілих чисел), який має вказівник на `unsigned int`, число елементів `size` і змінну стану `codeError`. У класі визначити

- конструктор без параметрів(виділяє місце для одного елемента та інінціалізує його в нуль);
- конструктор з одним параметром - розмір вектора(виділяє місце та інінціалізує масив значенням нуль);
- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце (значення перший аргумент) та інінціалізує значенням другого аргументу) ;
- конструктор копіювання;
- деструктор звільняє пам'ять;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):

- унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
- унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
- унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
- унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
- присвоєння `=`: копіює вектор(перевантажити через функцію класу);
- арифметичних бінарні:
  - `+` додавання:
    - для двох векторів
    - для вектора і скаляра типу `unsigned int`;
  - `-` (віднімання):
    - для двох векторів
    - для вектора і скаляра типу `unsigned int`;
  - `*`(множення)
    - для двох векторів
    - для вектора і скаляра типу `unsigned int`;
  - `/` (ділення)
    - для вектора і скаляра типу `unsigned int`;
  - `%` (остача від ділення)
    - для вектора і скаляра типу `unsigned int`;
- побітові бінарні
  - `|` (побітове додавання)
    - для двох векторів
    - для вектора і скаляра типу `unsigned int`;
  - `^` (побітове додавання за модулем 2)
    - для двох векторів
    - для вектора і скаляра типу `unsigned int`;
  - `|` (побітове множення)
    - двох векторів
    - вектора і скаляра типу `unsigned int`;
- присвоєння з
  - `+=` додаванням:
    - для двох векторів;
    - для вектора і скаляра типу `unsigned int`;
  - `-=` (віднімання):
    - для двох векторів;
    - для вектора і скаляра типу `unsigned int`;
  - `*=`(множення)
    - для двох векторів;
    - для вектора і скаляра типу `unsigned int`;
  - `/=`(ділення)
    - для двох векторів;
    - для вектора і скаляра типу `unsigned int`;

- `%=` (остача від ділення)
  - для двох векторів;
  - для вектора і скаляра типу `unsigned int`;
- `|=` (побітове додавання)
  - для двох векторів
  - для вектора і скаляра типу `unsigned int`;
- `^=`(побітове додавання за модулем 2)
  - для двох векторів
  - для вектора і скаляра типу `unsigned int`;
- `|=` (побітове множення)
  - двох векторів
  - вектора і скаляра типу `unsigned int`;

- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  - введення `>>` (побітовий зсув право) ;
  - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом, якщо невірний вказує на останній елемент масиву та встановлює код помилки у змінну `codeError`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
  - `>` (більше) для двох векторів;
  - `>=` (більше рівне) для двох векторів;
  - `<` (менше) для двох векторів;
  - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, коли не вистачає пам'яті, виходить за межі масиву. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 1.10.

Створити тип даних - клас `VectorULong` (вектор без знакових цілих чисел), який має вказівник на `unsigned long`, число елементів `size` і змінну стану `State`. У класі визначити

- конструктор без параметрів(виділяє місце для одного елемента та інінціалізує його в нуль);
- конструктор з одним параметром - розмір вектора(виділяє місце та інінціалізує масив значенням нуль);
- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце (значення перший аргумент) та інінціалізує значенням другого аргументу) ;
- конструктор копіювання;
- деструктор звільняє пам'ять;
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):

- унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
- унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
- унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
- унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу вектор з протилежним знаком;
- присвоєння `=`: копіює вектор(перевантажити через функцію класу);
- арифметичних бінарні:
  - `+` додавання:
    - для двох векторів
    - для вектора і скаляра типу `unsigned long`;
  - `-` (віднімання):
    - для двох векторів
    - для вектора і скаляра типу `unsigned long`;
  - `*`(множення)
    - для двох векторів
    - для вектора і скаляра типу `unsigned long`;
  - `/` (ділення)
    - для вектора і скаляра типу `unsigned long`;
  - `%` (остача від ділення)
    - для вектора і скаляра типу `unsigned int`;
- побітові бінарні
  - `|` (побітове додавання)
    - для двох векторів
    - для вектора і скаляра типу `unsigned long`;
  - `^` (побітове додавання за модулем 2)
    - для двох векторів
    - для вектора і скаляра типу `unsigned long`;
  - `|` (побітове множення)
    - двох векторів
    - вектора і скаляра типу `unsigned long`;
- присвоєння з
  - `+=` додаванням:
    - для двох векторів;
    - для вектора і скаляра типу `unsigned long`;
  - `-=` (віднімання):
    - для двох векторів;
    - для вектора і скаляра типу `unsigned long`;
  - `*=`(множення)
    - для двох векторів;
    - для вектора і скаляра типу `unsigned long`;
  - `/=`(ділення)
    - для двох векторів;
    - для вектора і скаляра типу `unsigned long`;

- `%=` (остача від ділення)
  - для двох векторів;
  - для вектора і скаляра типу `unsigned long`;
- `|=` (побітове додавання)
  - для двох векторів
  - для вектора і скаляра типу `unsigned long`;
- `^=`(побітове додавання за модулем 2)
  - для двох векторів
  - для вектора і скаляра типу `unsigned long`;
- `|=` (побітове множення)
  - двох векторів
  - вектора і скаляра типу `unsigned long`;

- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  - введення `>>` (побітовий зсув право) ;
  - введення `<<` (побітовий зсув ліво);
- рівності `==` та нерівності `!=` , функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- операцію індексації `[]` – функцію, яка звертається до елементу масиву за індексом, якщо невірний вказує на останній елемент масиву та встановлює код помилки у змінну `State`;
- розподілу пам'яті `new` та `delete`;
- виклику функції `()` ;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом), які повертають `true` або `false`:
  - `>` (більше) для двох векторів;
  - `>=` (більше рівне) для двох векторів;
  - `<` (менше) для двох векторів;
  - `<=`(менше рівне) для двох векторів.

У змінну стани встановлювати код помилки, коли не вистачає пам'яті, виходить за межі масиву. Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Завдання 2. Варіанти задач. Перевантаження операцій. Матриці.

Задача 2.1. Створити клас `MatrixInt` (матриця цілих чисел), використовуючи клас VectorInt. Розробити такі елементи класу:

- Поля (захищені):
  - `VectorInt` * **IntArray**; // масив
  - `unsigned int` **n,size**; // розміри матриці
  - `unsigned char` **codeError**; // код помилки
  - `static int` **num_matrix**; // кількість матриць (загально доступнк)
- У класі визначити:
  - конструктори:

- конструктор без параметрів(**IntArray** = `nullptr`, **n** =**size** = **codeError** = `0`);
- конструктор з одним параметрів типу `unsigned int` **`isize`**(створює одиничну матрицю **`n=isize`** на **`size=isize`**);
- конструктор із двома параметрами - розміри матриці (виділяє місце та ініціалізує значенням нуль);
- конструктор із трьома параметрами - розміри матриці та значення ініціалізації;
- копіювання;

• функції доступу до полів : **n, size** та **codeError.**
• деструктор (деструктор звільняє пам'ять);
• перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):

- унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
- унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше `false`;
- унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу матриця;
- унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
- присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
- арифметичних бінарні:
  a. `+` додавання:
     i. для двох матриць
     ii. для матриці та скаляра типу `int`
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `int`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `VectorInt` ,
     iii. для матриці та скаляра типу `int`;
  d. `/`(ділення)
     i. для матриці та скаляра типу `int`;
  e. `%`(остача від ділення)
     i. для матриці та скаляра типу `int`;
  f. `+=` присвоєння з додаванням:
     i. для двох матриць
     ii. для матриці та скаляра типу `int`
  g. `-=` присвоєння з відніманням:
     i. для двох матриць
     ii. для матриці та скаляра типу `int`;
  h. `*=` присвоєння з множенням
     i. для двох матриць,
     ii. для матриці та вектора `VectorInt`,
     iii. для матриці та скаляра типу `int`;
  i. `/=` присвоєння з діленням

i. для матриці та скаляра типу `int`;

j. `%=` присвоєння з остачею від ділення

i. для матриці та скаляра типу `int`;

– побітові бінарні операції

a. `|` (побітове додавання)

i. для двох матриць

ii. для матриці та скаляра типу `int`;

b. `^` (побітове додавання за модулем 2)

i. для двох матриць

ii. для матриці та скаляра типу `int`;

c. `&` (побітове множення)

i. двох векторів

ii. вектора і скаляра типу `int`;

d. `>>=` (присвоєння з побітовим зсувом в право)

i. для двох матриць

ii. для матриці та скаляра типу `int`;

e. `<<=` (присвоєння з побітовим зсувом в ліво)

i. для двох матриць

ii. для матриці та скаляра типу `int`;

f. `|=` (присвоєння з побітовим додаванням)

i. для двох матриць

ii. для матриці та скаляра типу `int`;

g. `^=` (присвоєння з побітовим додаванням за модулем 2)

i. для двох матриць

ii. для матриці та скаляра типу `int`;

h. `&=` (присвоєння з побітове множення)

i. двох векторів

ii. вектора і скаляра типу `int`;

– операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;

– порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)

a. `>` (більше) для двох матриць;

b. `>=` (більше рівне) для двох матриць;

c. `<` (менше) для двох матриць;

d. `<=` (менше рівне) для двох матриць.

– операцію індексації `[]` – функцію, яка звертається до елементу `VectorInt`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError**.

– розподілу пам'яті `new` та `delete`;

– виклику функції `()`;

– побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)

a. введення `>>` (побітовий зсув право) ;

b. введення `<<` (побітовий зсув ліво);

- Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.
- У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці.

Задача 2.2. Створити клас `MatrixFload` (матриця цілих чисел), використовуючи клас `VectorFload`. Розробити такі елементи класу:

- Поля (захищені):
  - `VectorFload` * **FloadArray**; // масив
  - `int` **n,size**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_matrix**; // кількість матриць (загально доступних)
  - У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:
- конструктори:
  - конструктор без параметрів(**FloadArray** = `nullptr`, **n =m = codeError** = `0`);
  - конструктор з одним параметрів типу `int size`(створює одиничну матрицю `n=size` на `size`);
  - конструктор із двома параметрами - розміри матриці (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
  - копіювання;
- функції доступу до полів : **n, size**та **codeError.**
- деструктор (деструктор звільняє пам'ять);
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше `false`;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
  - присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
  - арифметичних бінарні:
    a. `+` додавання:
       i. для двох матриць;
       ii. для матриці та скаляра типу `int;`
       iii. для матриці та скаляра типу `float`;
    b. `-` (віднімання):
       i. для двох матриць
       ii. для матриці та скаляра типу `int`;
       iii. для матриці та скаляра типу `float`;
       iv. для матриці та скаляра типу `double`;
    c. `*`(множення)
       i. для двох матриць,
       ii. для матриці та вектора `VectorFload`,
       iii. для матриці та скаляра типу `float`;
       iv. для матриці та скаляра типу `double`;
    d. `/`(ділення)
       i. для матриці та скаляра типу `int`;
       ii. для матриці та скаляра типу `float`;

iii. для матриці та скаляра типу `double`;

e. `%`(остача від ділення)
   i. ~~для двох матриць~~
   ii. для матриці та скаляра типу `int`;
   iii. для матриці та скаляра типу `float`;

f. `+=` присвоєння з додаванням:
   i. для двох матриць
   ii. для матриці та скаляра типу `float`;
   iii. для матриці та скаляра типу `double`;

g. `-=` присвоєння з відніманням:
   i. для двох матриць
   ii. для матриці та скаляра типу `float`;
   iii. для матриці та скаляра типу `double`;

h. `*=` присвоєння з множенням
   i. для двох матриць,
   ii. для матриці та вектора `VectorFload`,
   iii. для матриці та скаляра типу `float`;
   iv. для матриці та скаляра типу `double`;

i. `/=` присвоєння з діленням
   i. ~~для двох матриць~~
   ii. для матриці та скаляра типу `int`;
   iii. для матриці та скаляра типу `float`;

j. `%=` присвоєння з остачею від ділення
   i. ~~для двох матриць~~
   ii. для матриці та скаляра типу `int`;
   iii. для матриці та скаляра типу `float`;

– операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
   a. `>` (більше) для двох матриць;
   b. `>=` (більше рівне) для двох матриць;
   c. `<` (менше) для двох матриць;
   d. `<=` (менше рівне) для двох матриць.
– операцію індексації `[]` – функцію, яка звертається до елементу `VectorFload`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError.**
– розподілу пам'яті `new` та `delete`;
– виклику функції `()`;
– побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
   a. введення `>>` (побітовий зсув право) ;
   b. введення `<<` (побітовий зсув ліво);

• Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 2.3. Створити клас `MatrixDouble` (матриця цілих чисел), використовуючи клас `VectorDouble`. Розробити такі елементи класу:

• Поля (захищені):

- `VectorDouble` * **DoubleArray**; // масив
- `int` **n,size**; // розміри матриці
- `int` **codeError**; // код помилки
- `static int` **num_matrix**; // кількість матриць (загально доступних)
- У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:

• конструктори:
- конструктор без параметрів(**DoubleArray** = `nullptr`, **n** =**m** = **codeError** = `0`);
- конструктор з одним параметрів типу `int` **`size`**(створює одиничну матрицю **`n=size`** на **`size`**);
- конструктор із двома параметрами - розміри матриці (виділяє місце та ініціалізує значенням нуль);
- конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- копіювання;

• функції доступу до полів : **n, size**та **codeError.**
• деструктор (деструктор звільняє пам'ять);
• перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
- унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
- унарної логічної `!` (заперечення): повертає значення **`true`**, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше **`false`**;
- унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
- присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
- арифметичних бінарні:
  a. `+` додавання:
     i. для двох матриць;
     ii. для матриці та скаляра типу `int;`
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `int`;
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `VectorDouble`,
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  d. `/`(ділення)
     i. для матриці та скаляра типу `int`;
     ii. для матриці та скаляра типу `float`;
  e. `%`(остача від ділення)
     i. для матриці та скаляра типу `int`;

ii. для матриці та скаляра типу `double`;
iii. для матриці та скаляра типу `float`;

f. `+=` присвоєння з додаванням:
i. для двох матриць
ii. для матриці та скаляра типу `double`;
iii. для матриці та скаляра типу `float`;

g. `-=` присвоєння з відніманням:
i. для двох матриць;
ii. для матриці та скаляра типу `double`;
iii. для матриці та скаляра типу `float`;

h. `*=` присвоєння з множенням
i. для двох матриць,
ii. для матриці та вектора `VectorDouble`;
iii. для матриці та скаляра типу `double`;
iv. для матриці та скаляра типу `float`;

i. `/=` присвоєння з діленням
i. для матриці та скаляра типу `int`;
ii. для матриці та скаляра типу `double`;
iii. для матриці та скаляра типу `float`;

j. `%=` присвоєння з остачею від ділення
i. для матриці та скаляра типу `int`;
ii. для матриці та скаляра типу `double`;
iii. для матриці та скаляра типу `float`;

– операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
a. `>` (більше) для двох матриць;
b. `>=` (більше рівне) для двох матриць;
c. `<` (менше) для двох матриць;
d. `<=` (менше рівне) для двох матриць.
– операцію індексації `[]` – функцію, яка звертається до елементу `VectorDouble`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError.**
– розподілу пам'яті `new` та `delete`;
– виклику функції `()`;
– побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
a. введення `>>` (побітовий зсув право) ;
b. введення `<<` (побітовий зсув ліво);

• Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 2.4. Створити клас `Matrix3F` (матриця цілих чисел), використовуючи клас `Vector3F`. Розробити такі елементи класу:

• Поля (захищені):
– `Vector3F` * **PointArray**; // масив
– `int` **n**; // розмір матриці (к-ть векторів)
– `int` **codeError**; // код помилки

- `static int` **num_matrix**; // кількість матриць (загально доступних)
- У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:

• конструктори:
- конструктор без параметрів(**PointArray** = `nullptr`, **n** = **codeError** = `0`);
- конструктор з одним параметрів типу `int` **`size`**(створює матрицю **`n=size`** на `3`, інінціалізує поля в нуль);
- конструктор із двома параметрами - розмір матриці (виділяє місце та ініціалізує значенням другого параметру);
- конструктор із чотирма параметрами - розмір матриці, значення ініціалізації полів вектора;
- копіювання;

• функції доступу до полів : **n** та **codeError.**
• деструктор (деструктор звільняє пам'ять);
• перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
- унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
- унарної логічної `!` (заперечення): повертає значення **`true`**, якщо елементи якщо **n** не дорівнює – нулю, інакше **`false`**;
- унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
- присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
- арифметичних бінарні:
  a. `+` додавання:
     i. для двох матриць;
     ii. для матриці та скаляра типу `int;`
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `int`;
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `Vector3F`;
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  d. `/`(ділення)
     i. векторне множення елементів масивів **PointArray;**
     ii. для матриці та скаляра типу `int`;
     iii. для матриці та скаляра типу `float`;
  e. `%`(остача від ділення)
     ~~i. для двох матриць~~
     ii. для матриці та скаляра типу `int`;
     iii. для матриці та скаляра типу `double`;

iv. для матриці та скаляра типу `float`;

f. `+=` присвоєння з додаванням:
   i. для двох матриць
   ii. для матриці та скаляра типу `double`;
   iii. для матриці та скаляра типу `float`;

g. `-=` присвоєння з відніманням:
   i. для двох матриць;
   ii. для матриці та скаляра типу `double`;
   iii. для матриці та скаляра типу `float`;

h. `*=` присвоєння з множенням
   i. для двох матриць,
   ii. для матриці та вектора `Vector3F`;
   iii. для матриці та скаляра типу `double`;
   iv. для матриці та скаляра типу `float`;

i. `/=` присвоєння з діленням
   i. Присвоення з векторним множення елементів масивів **PointArray** для двох матриць;
   ii. для матриці та скаляра типу `int`;
   iii. для матриці та скаляра типу `double`;
   iv. для матриці та скаляра типу `float`;

j. `%=` присвоєння з остачею від ділення
   i. для матриці та скаляра типу `int`;
   ii. для матриці та скаляра типу `double`;
   iii. для матриці та скаляра типу `float`;

- операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;
- порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
  a. `>` (більше) для двох матриць;
  b. `>=` (більше рівне) для двох матриць;
  c. `<` (менше) для двох матриць;
  d. `<=` (менше рівне) для двох матриць.
- операцію індексації `[]` – функцію, яка звертається до елементу `Vector3F`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError.**
- розподілу пам'яті `new` та `delete`;
- виклику функції `()`;
- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  a. введення `>>` (побітовий зсув право) ;
  b. введення `<<` (побітовий зсув ліво);

• Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 2.5. Створити клас `Matrix3D` (матриця цілих чисел), використовуючи клас `Vector3D`. Розробити такі елементи класу:

• Поля (захищені):
- `Vector3D` * **PointArray**; // масив
- `int` **n**; // розмір матриці (к-ть векторів)

- `int` **codeError**; // код помилки
- `static int` **num_matrix**; // кількість матриць (загально доступних)
- У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:

• конструктори:
- конструктор без параметрів(**PointArray** = `nullptr`, **n** = **codeError** = `0`);
- конструктор з одним параметрів типу `int` **`size`**(створює матрицю **`n=size`** на `3,` інінціалізує поля в нуль);
- конструктор із двома параметрами - розмір матриці (виділяє місце та ініціалізує значенням другого параметру);
- конструктор із чотирма параметрами - розмір матриці, значення ініціалізації полів вектора;
- копіювання;

• функції доступу до полів : **n** та **codeError.**
• деструктор (деструктор звільняє пам'ять);
• перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
- унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
- унарної логічної `!` (заперечення): повертає значення **`true`**, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше **`false`**;
- унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
- присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
- арифметичних бінарні:
  a. `+` додавання:
     i. для двох матриць;
     ii. для матриці та скаляра типу `int;`
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `int`;
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `Vector3D`,
     iii. для матриці та скаляра типу `double`;
     iv. для матриці та скаляра типу `float`;
  d. `/`(ділення)
     i. векторне множення елементів масивів **PointArray;**
     ii. для матриці та скаляра типу `int`;
     iii. для матриці та скаляра типу `float`;
  e. `%`(остача від ділення)
     ~~i. для двох матриць~~
     ii. для матриці та скаляра типу `int`;

iii. для матриці та скаляра типу `double`;
iv. для матриці та скаляра типу `float`;

f. `+=` присвоєння з додаванням:
i. для двох матриць
ii. для матриці та скаляра типу `double`;
iii. для матриці та скаляра типу `float`;

g. `-=` присвоєння з відніманням:
i. для двох матриць;
ii. для матриці та скаляра типу `double`;
iii. для матриці та скаляра типу `float`;

h. `*=` присвоєння з множенням
i. для двох матриць,
ii. для матриці та вектора `Vector3D`;
iii. для матриці та скаляра типу `double`;
iv. для матриці та скаляра типу `float`;

i. `/=` присвоєння з діленням
i. Присвоення з векторним множення елементів масивів **PointArray** для двох матриць;
ii. для матриці та скаляра типу `int`;
iii. для матриці та скаляра типу `double`;
iv. для матриці та скаляра типу `float`;

j. `%=` присвоєння з остачею від ділення
i. для матриці та скаляра типу `int`;
ii. для матриці та скаляра типу `double`;
iii. для матриці та скаляра типу `float`;

– операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
a. `>` (більше) для двох матриць;
b. `>=` (більше рівне) для двох матриць;
c. `<` (менше) для двох матриць;
d. `<=` (менше рівне) для двох матриць.
– операцію індексації `[]` – функцію, яка звертається до елементу `Vector3D`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError.**
– розподілу пам'яті `new` та `delete`;
– виклику функції `()`;
– побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
a. введення `>>` (побітовий зсув право) ;
b. введення `<<` (побітовий зсув ліво);

• Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 2.6. Створити клас `MatrixShort` (матриця цілих чисел), використовуючи клас `VectorShort`. Розробити такі елементи класу:

• Поля (захищені):
– `VectorShort` * **ShortArray**; // масив

- `int` **n,size**; // розміри матриці
- `int` **codeError**; // код помилки
- `static int` **num_matrix**; // кількість матриць (загально доступних)
- У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:

• конструктори:
- конструктор без параметрів(**ShortArray** = `nullptr`, **n** =**size** = **codeError** = `0`);
- конструктор з одним параметрів типу `int` **`size`**(створює одиничну матрицю **`n=size`** на **`size`**);
- конструктор із двома параметрами - розміри матриці (виділяє місце та ініціалізує значенням нуль);
- конструктор із трьома параметрами - розміри матриці та значення ініціалізації;
- копіювання;

• функції доступу до полів : **n, size**та **codeError.**
• деструктор (деструктор звільняє пам'ять);
• перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
- унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
- унарної логічної `!` (заперечення): повертає значення **`true`**, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше **`false`**;
- унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу матриця;
- унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
- присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
- арифметичних бінарні:
  a. `+` додавання:
     i. для двох матриць
     ii. для матриці та скаляра типу `short`
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `short`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `VectorShort` ,
     iii. для матриці та скаляра типу `short`;
  d. `/`(ділення)
     i. для матриці та скаляра типу `short`;
  e. `%`(остача від ділення)
     i. для матриці та скаляра типу `short`;
  f. `+=` присвоєння з додаванням:
     i. для двох матриць
     ii. для матриці та скаляра типу `short`
  g. `-=` присвоєння з відніманням:
     i. для двох матриць

ii. для матриці та скаляра типу `short`;

h. `*=` присвоєння з множенням
   i. для двох матриць,
   ii. для матриці та вектора `VectorShort`,
   iii. для матриці та скаляра типу `short`;

i. `/=` присвоєння з діленням
   i. для матриці та скаляра типу `short`;

j. `%=` присвоєння з остачею від ділення
   i. для матриці та скаляра типу `short`;

– побітові бінарні операції
  a. `|` (побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `short`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `short`;
  c. `&` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `short`;
  d. `>>=` (присвоєння з побітовим зсувом в право)
     i. для двох матриць
     ii. для матриці та скаляра типу `short`;
  e. `<<=` (присвоєння з побітовим зсувом в ліво)
     i. для двох матриць
     ii. для матриці та скаляра типу `short`;
  f. `|=` (присвоєння з побітовим додаванням)
     i. для двох матриць
     ii. для матриці та скаляра типу `short`;
  g. `^=` (присвоєння з побітовим додаванням за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `short`;
  h. `&=` (присвоєння з побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `short`;

– операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
  a. `>` (більше) для двох матриць;
  b. `>=` (більше рівне) для двох матриць;
  c. `<` (менше) для двох матриць;
  d. `<=` (менше рівне) для двох матриць.
– операцію індексації `[]` – функцію, яка звертається до елементу `VectorShort`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError**.
– розподілу пам'яті `new` та `delete`;
– виклику функції `()`;
– побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)

a. введення >> (побітовий зсув право) ;
b. введення << (побітовий зсув ліво);

- Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 2.7. Створити клас `MatrixLong` (матриця цілих чисел), використовуючи клас `VectorLong`. Розробити такі елементи класу:

- Поля (захищені):
  - `VectorLong` * **LongArray**; // масив
  - `int` **n,size**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_matrix**; // кількість матриць (загально доступних)
  - У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:
- конструктори:
  - конструктор без параметрів(**LongArray** = `nullptr`, **n =m = codeError** = `0`);
  - конструктор з одним параметрів типу `int size`(створює одиничну матрицю `n=size` на `size`);
  - конструктор із двома параметрами - розміри матриці (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри матриці та значення ініціалізації;
  - копіювання;
- функції доступу до полів : **n, size**та **codeError.**
- деструктор (деструктор звільняє пам'ять);
- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше `false`;
  - унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу матриця;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
  - присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
  - арифметичних бінарні:
    a. `+` додавання:
       i. для двох матриць
       ii. для матриці та скаляра типу `long`
    b. `-` (віднімання):
       i. для двох матриць
       ii. для матриці та скаляра типу `long`;
    c. `*`(множення)
       i. для двох матриць,
       ii. для матриці та вектора `VectorLong`,
       iii. для матриці та скаляра типу `long`;
    d. `/`(ділення)

i. для матриці та скаляра типу `long`;

e. `%`(остача від ділення)
   i. для матриці та скаляра типу `long`;

f. `+=` присвоєння з додаванням:
   i. для двох матриць
   ii. для матриці та скаляра типу `long`

g. `-=` присвоєння з відніманням:
   i. для двох матриць
   ii. для матриці та скаляра типу `long`;

h. `*=` присвоєння з множенням
   i. для двох матриць,
   ii. для матриці та вектора `VectorLong`,
   iii. для матриці та скаляра типу `long`;

i. `/=` присвоєння з діленням
   i. для матриці та скаляра типу `long`;

j. `%=` присвоєння з остачею від ділення
   i. для матриці та скаляра типу `long`;

– побітові бінарні операції
   a. `|`(побітове додавання)
      i. для двох матриць
      ii. для матриці та скаляра типу `long`;
   b. `^` (побітове додавання за модулем 2)
      i. для двох матриць
      ii. для матриці та скаляра типу `long`;
   c. `&` (побітове множення)
      i. двох векторів
      ii. вектора і скаляра типу `long`;
   d. `>>=` (присвоєння з побітовим зсувом в право)
      i. для двох матриць
      ii. для матриці та скаляра типу `long`;
   e. `<<=` (присвоєння з побітовим зсувом в ліво)
      i. для двох матриць
      ii. для матриці та скаляра типу `long`;
   f. `|=` (присвоєння з побітовим додаванням)
      i. для двох матриць
      ii. для матриці та скаляра типу `long`;
   g. `^=` (присвоєння з побітовим додаванням за модулем 2)
      i. для двох матриць
      ii. для матриці та скаляра типу `long`;
   h. `&=` (присвоєння з побітове множення)
      i. двох векторів
      ii. вектора і скаляра типу `long`;

– операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;

– порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
   a. `>` (більше) для двох матриць;
   b. `>=` (більше рівне) для двох матриць;
   c. `<` (менше) для двох матриць;

d. `<=` (менше рівне) для двох матриць.

– операцію індексації `[]` – функцію, яка звертається до елементу `VectorLong`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError.**
– розподілу пам'яті `new` та `delete`;
– виклику функції `()`;
– побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
a. введення `>>` (побітовий зсув право) ;
b. введення `<<` (побітовий зсув ліво);

• Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 2.8. Створити клас `MatrixUShort` (матриця цілих чисел), використовуючи клас `VectorUShort`. Розробити такі елементи класу:

• Поля (захищені):
– `VectorUShort` * **UShortArray**; // масив
– `int` **n,size**; // розміри матриці
– `int` **codeError**; // код помилки
– `static int` **num_matrix**; // кількість матриць (загально доступних)
– У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:

• конструктори:
– конструктор без параметрів(**UShortArray** = `nullptr`, **n** = **codeError** = `0`);
– конструктор з одним параметрів типу `int size`(створює одиничну матрицю `n=size` на `size`);
– конструктор із двома параметрами - розміри матриці (виділяє місце та ініціалізує значенням нуль);
– конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
– копіювання;

• функції доступу до полів : **n, size** та **codeError.**
• деструктор (деструктор звільняє пам'ять);
• перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
– унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
– унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше `false`;
– унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
– унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
– присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
– арифметичних бінарні:
a. `+` додавання:
i. для двох матриць
ii. для матриці та скаляра типу `unsigned short`

b. `-` (віднімання):
   i. для двох матриць
   ii. для матриці та скаляра типу `unsigned short`;
c. `*`(множення)
   i. для двох матриць,
   ii. для матриці та вектора `VectorUShort`,
   iii. для матриці та скаляра типу `unsigned short`;
d. `/`(ділення)
   i. для матриці та скаляра типу `unsigned short`;
e. `%`(остача від ділення)
   i. для матриці та скаляра типу `unsigned short`;
f. `+=` присвоєння з додаванням:
   i. для двох матриць
   ii. для матриці та скаляра типу `unsigned short`
g. `-=` присвоєння з відніманням:
   i. для двох матриць
   ii. для матриці та скаляра типу `unsigned short`;
h. `*=` присвоєння з множенням
   i. для двох матриць,
   ii. для матриці та вектора `VectorUShort`,
   iii. для матриці та скаляра типу `unsigned short`;
i. `/=` присвоєння з діленням
   i. для матриці та скаляра типу `unsigned short`;
j. `%=` присвоєння з остачею від ділення
   i. для матриці та скаляра типу `unsigned short`;

– побітові бінарні операції
   a. `|`(побітове додавання)
      i. для двох матриць
      ii. для матриці та скаляра типу `unsigned short`;
   b. `^` (побітове додавання за модулем 2)
      i. для двох матриць
      ii. для матриці та скаляра типу `unsigned short`;
   c. `&` (побітове множення)
      i. двох векторів
      ii. вектора і скаляра типу `unsigned short`;
   d. `>>=` (присвоєння з побітовим зсувом в право)
      i. для двох матриць
      ii. для матриці та скаляра типу `unsigned short`;
   e. `<<=` (присвоєння з побітовим зсувом в ліво)
      i. для двох матриць
      ii. для матриці та скаляра типу `unsigned short`;
   f. `|=` (присвоєння з побітовим додаванням)
      i. для двох матриць
      ii. для матриці та скаляра типу `unsigned short`;
   g. `^=` (присвоєння з побітовим додаванням за модулем 2)
      i. для двох матриць
      ii. для матриці та скаляра типу `unsigned short`;
   h. `&=` (присвоєння з побітове множення)
      i. двох векторів

ii. вектора і скаляра типу `unsigned short`;

– операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
  a. `>` (більше) для двох матриць;
  b. `>=` (більше рівне) для двох матриць;
  c. `<` (менше) для двох матриць;
  d. `<=` (менше рівне) для двох матриць.
– операцію індексації `[]` – функцію, яка звертається до елементу `VectorUShort`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError.**
– розподілу пам'яті `new` та `delete`;
– виклику функції `()`;
– побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  a. введення `>>` (побітовий зсув право) ;
  b. введення `<<` (побітовий зсув ліво);

• Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 2.9. Створити клас `MatrixUInt` (матриця цілих чисел), використовуючи клас `VectorUInt`. Розробити такі елементи класу:

• Поля (захищені):
– `VectorUInt` * **UIntArray**; // масив
– `int` **n,size**; // розміри матриці
– `int` **codeError**; // код помилки
– `static int` **num_matrix**; // кількість матриць (загально доступних)
– У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:

• конструктори:
– конструктор без параметрів(**UIntArray** = `nullptr`, **n** = **codeError** = `0`);
– конструктор з одним параметрів типу `int size`(створює одиничну матрицю `n=size` на `size`);
– конструктор із двома параметрами - розміри матриці (виділяє місце та ініціалізує значенням нуль);
– конструктор із трьома параметрами - розміри матриці та значення ініціалізації;
– копіювання;

• функції доступу до полів : **n, size**та **codeError.**
• деструктор (деструктор звільняє пам'ять);
• перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
– унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
– унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше `false`;
– унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу матриця;

- унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
- присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
- арифметичних бінарні:
  a. `+` додавання:
     i. для двох матриць
     ii. для матриці та скаляра типу `unsigned int`
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `unsigned int`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `VectorUInt`,
     iii. для матриці та скаляра типу `unsigned int`;
  d. `/`(ділення)
     i. для матриці та скаляра типу `unsigned int`;
  e. `%`(остача від ділення)
     i. для матриці та скаляра типу `unsigned int`;
  f. `+=` присвоєння з додаванням:
     i. для двох матриць
     ii. для матриці та скаляра типу `unsigned int`
  g. `-=` присвоєння з відніманням:
     i. для двох матриць
     ii. для матриці та скаляра типу `unsigned int`;
  h. `*=` присвоєння з множенням
     i. для двох матриць,
     ii. для матриці та вектора `VectorUInt`,
     iii. для матриці та скаляра типу `unsigned int`;
  i. `/=` присвоєння з діленням
     i. для матриці та скаляра типу `unsigned int`;
  j. `%=` присвоєння з остачею від ділення
     i. для матриці та скаляра типу `unsigned int`;
- побітові бінарні операції
  a. `|`(побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `unsigned int`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `unsigned int`;
  c. `&` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `unsigned int`;
  d. `>>=` (присвоєння з побітовим зсувом в право)
     i. для двох матриць
     ii. для матриці та скаляра типу `unsigned int`;
  e. `<<=` (присвоєння з побітовим зсувом в ліво)
     i. для двох матриць
     ii. для матриці та скаляра типу `unsigned int`;
  f. `|=` (присвоєння з побітовим додаванням)

i. для двох матриць
ii. для матриці та скаляра типу `unsigned int`;

g. `^=` (присвоєння з побітовим додаванням за модулем 2)
i. для двох матриць
ii. для матриці та скаляра типу `unsigned int`;

h. `&=` (присвоєння з побітове множення)
i. двох векторів
ii. вектора і скаляра типу `unsigned int`;

- операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;
- порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
  a. `>` (більше) для двох матриць;
  b. `>=` (більше рівне) для двох матриць;
  c. `<` (менше) для двох матриць;
  d. `<=` (менше рівне) для двох матриць.
- операцію індексації `[]` – функцію, яка звертається до елементу `VectorUInt`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError.**
- розподілу пам'яті `new` та `delete`;
- виклику функції `()`;
- побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
  a. введення `>>` (побітовий зсув право) ;
  b. введення `<<` (побітовий зсув ліво);

• Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Задача 2.10. Створити клас `MatrixULong` (матриця цілих чисел), використовуючи клас `VectorULong`. Розробити такі елементи класу:

• Поля (захищені):
- `VectorULong` * **ULongArray**; // масив
- `int` **n,size**; // розміри матриці
- `int` **codeError**; // код помилки
- `static int` **num_matrix**; // кількість матриць (загально доступних)
- У змінну стану встановлювати код помилки, коли не вистачає пам'яті, виходить за межі матриці. У класі визначити:

• конструктори:
- конструктор без параметрів(**ULongArray** = `nullptr`, **n** = **codeError** = `0`);
- конструктор з одним параметрів типу `int size`(створює одиничну матрицю `n=size` на `size`);
- конструктор із двома параметрами - розміри матриці (виділяє місце та ініціалізує значенням нуль);
- конструктор із трьома параметрами - розміри матриці та значення ініціалізації;
- копіювання;

• функції доступу до полів : **n, size**та **codeError.**
• деструктор (деструктор звільняє пам'ять);

- перевантаження операцій (операції перевантажувати через функції класу або дружні функції, якщо не вказано яким чином це робити):
  - унарних префіксних та постфіксних `++` та `--`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - унарної логічної `!` (заперечення): повертає значення `true`, якщо елементи якщо **n та size** не дорівнюють – нулю, інакше `false`;
  - унарної побітової `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу матриця;
  - унарний арифметичний `-` (мінус) : повертає всі елементи масиву класу матриця з протилежним знаком;
  - присвоєння `=`: копіює матрицю (перевантажити через функцію класу);
  - арифметичних бінарні:
    a. `+` додавання:
       i. для двох матриць
       ii. для матриці та скаляра типу `unsigned long`
    b. `-` (віднімання):
       i. для двох матриць
       ii. для матриці та скаляра типу `unsigned long`;
    c. `*`(множення)
       i. для двох матриць,
       ii. для матриці та вектора `VectorULong`,
       iii. для матриці та скаляра типу `unsigned long`;
    d. `/`(ділення)
       i. для матриці та скаляра типу `unsigned long`;
    e. `%`(остача від ділення)
       i. для матриці та скаляра типу `unsigned long`;
    f. `+=` присвоєння з додаванням:
       i. для двох матриць
       ii. для матриці та скаляра типу `unsigned long`
    g. `-=` присвоєння з відніманням:
       i. для двох матриць
       ii. для матриці та скаляра типу `unsigned long`;
    h. `*=` присвоєння з множенням
       i. для двох матриць,
       ii. для матриці та вектора `VectorULong`,
       iii. для матриці та скаляра типу `unsigned long`;
    i. `/=` присвоєння з діленням
       i. для матриці та скаляра типу `unsigned long`;
    j. `%=` присвоєння з остачею від ділення
       i. для матриці та скаляра типу `unsigned long`;
  - побітові бінарні операції
    a. `|`(побітове додавання)
       i. для двох матриць
       ii. для матриці та скаляра типу `unsigned long`;
    b. `^` (побітове додавання за модулем 2)
       i. для двох матриць
       ii. для матриці та скаляра типу `unsigned long`;
    c. `&` (побітове множення)

i. двох векторів
ii. вектора і скаляра типу `unsigned long`;

d. `>>=` (присвоєння з побітовим зсувом в право)
i. для двох матриць
ii. для матриці та скаляра типу `unsigned long`;

e. `<<=` (присвоєння з побітовим зсувом в ліво)
i. для двох матриць
ii. для матриці та скаляра типу `unsigned long`;

f. `|=` (присвоєння з побітовим додаванням)
i. для двох матриць
ii. для матриці та скаляра типу `unsigned long`;

g. `^=` (присвоєння з побітовим додаванням за модулем 2)
i. для двох матриць
ii. для матриці та скаляра типу `unsigned long`;

h. `&=` (присвоєння з побітове множення)
i. двох векторів
ii. вектора і скаляра типу `unsigned long`;

– операцій `==` (рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів матриці за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів матриць за індексом)
a. `>` (більше) для двох матриць;
b. `>=` (більше рівне) для двох матриць;
c. `<` (менше) для двох матриць;
d. `<=` (менше рівне) для двох матриць.
– операцію індексації `[]` – функцію, яка звертається до елементу `VectorULong`, якщо індекс невірний вказує на останній елемент масиву та встановлює код помилки у змінну **CodeError.**
– розподілу пам'яті `new` та `delete`;
– виклику функції `()`;
– побітові операції зсувів, як дружні операції введення та виведення вектора у потік (перевантажувати через дружні функції)
a. введення `>>` (побітовий зсув право) ;
b. введення `<<` (побітовий зсув ліво);

• Передбачити можливість підрахунку числа об'єктів даного типу. Перевірити роботу цього класу.

Завдання 3. Варіанти задач. **Асоціативні масиви..**

Задача 3.1. Побудувати асоційований клас збереження цілих чисел від 1 до 100. Написати програму, яка використовує асоційований клас, для друку ціни товару прописом. Ціна товару не перевищує 100 гривень. Наприклад : шапка - 8 грн. 05 коп. надрукувати : шапка вісім гривень п'ять копійок.

Задача 3.2. Побудувати асоційований клас збереження назви країни та столиці. Використовуючи асоційований клас, написати програму, яка по

введеній назві столиці країни друкує її назву країни, або вводить назву країни, якщо такої країни немає в базі.

Задача 3.3. Побудувати асоційований клас збереження телефонний довідник.

Задача 3.4. Побудувати асоційований клас збереження доменних імен та IP - адреса.

Задача 3.5. Побудувати асоційований клас збереження електронна пошта телефон.