Лабораторна робота № 4.

Тема: Об'єкти та класи (*деструктори, індексатори, операції класу, операції перетворення типів****).***

Мета роботи:

Ознайомлення з функціями класів: деструктори, індексатори, операції класу, операції перетворення типів. Доступ до методів базових.

Теоретичні відомості

Індексатори

Індексатор є різновидністю властивості та звичайно застосовується для організації доступу до схованих полів класу за індексом, наприклад, так само, як ми звертаємося до елемента масиву. Синтаксис індексатора аналогічний синтаксису властивості:

```
[атрибути] [специфікатори] тип this [список параметрів] // останні [ ] є елементами синтаксису
{
 [get код_доступу]
 [set код_доступу]
}
```

Індексатори найчастіше оголошуються зі специфікатором `public`, оскільки вони входять в інтерфейс об'єкта. Код доступу це блоки операторів, які виконуються при одержанні (`get`) або установці (`set`) значення деякого елемента класу. Може бути відсутня частина `get` або `set`, але не обидві одночасно. Якщо відсутня частина `set`, індексатор доступний тільки для читання, якщо відсутня частина `get`, індексатор доступний тільки для запису.

Список параметрів може містити один або кілька описів індексів, за яким виконується доступ до елемента. Найчастіше використовується один індекс цілого типу.

Як приклад розглянемо індексатор, який дозволяє одержати n-член послідовності Фиббоначи:

```
class Demofib
{

 public int this[int i] //індексатор, доступний тільки для читання
 {
  get
  {
   if (i <=0) throw new Exception("неприпустиме значення індексу");
   else if (i==1 || i==2) return 1;
   else
   {int a=1, b=1, c;
    for (int j=3; j<=i; ++j)
    {
     c=a+b;
     a=b;
     b=c;
     }
    return b;
   }
  }
 }
```

```
}

class Program
{
 static void Main()
 {
  Console.Write("n=");
  int n=int.Parse(Console.Readline());
  Demofib a=new Demofib();
  try
  {
   Console.Writeline("a[{0}]={1}",n,a[n]);
  }
  catch (Exception e)
  {
   Console.Writeline(e.Message);
  }
 }
}
```

Мова C# допускає використання багатомірних індексаторів. Вони застосовуються для роботи з багатомірними масивами. Розглянемо на прикладі попередню задачу за умови, що організує двовимірний масив.

```
class Demoarray
{
 int[,] Myarray;//закритий масив
 int n, m;//закриті поля: розмірність масиву

 public Demoarray(int sizen, int sizem)//конструктор
 {
  Myarray = new int[sizen, sizem];
  this.n = sizen;
  this.m = sizem;
 }

 public int Lengthn //властивість, що повертає кількість рядків
 {
  get { return n; }
 }

 public int Lengthm //властивість, що повертає кількість рядків
 {
  get { return m; }
 }

 public int this[int i, int j] //індексатор
 {
  get
  {
   if (i < 0 || i >= n || j < 0 || j >= m)
                throw new Exception("вихід за межі масиву");
   else return Myarray[i, j];
  }
  set
  {
   if (i < 0 || i >= n || j < 0 || j >= m)
                throw new Exception("вихід за межі масиву");
   else if (value >= 0 && value <= 100) Myarray[i, j] = value;
   else throw new Exception("привласнюється неприпустиме значення");
  }
 }
}

 class Program
 {
  static void Main()
```

```
{
 Demoarray a = new Demoarray(3, 3);
 for (int i = 0; i < a.Lengthn; i++,Console.Writeline())
 {
  for (int j = 0; j < a.Lengthm; j++)
   {
    a[i, j] = i *j;    // використання індексатора в режимі запису
    Console.Write("{0,5}", a[i, j]);// використання індексатора в режимі читання
   }
 }
 Console.Writeline();
 try
 {
  //Console.Writeline(a[3,3]);
  //a[0,0]=200;
 }
 catch (Exception e)
 {
  Console.Writeline(e.Message);
 }

 }
}
```

Операції класу

С # дозволяє перевизначити більшість операцій так, щоб при використанні їх об'єктами конкретного класу виконувалися дії, відмінні від стандартних. Це дає можливість застосовувати об'єкти власних типів даних у складі виразів, наприклад:

```
newobject x, y, z;
…
z = x+y; // використовується операція додавання, класу newobject
```

Визначення власних операцій класу називають перевантаженням операцій. Перевантаження операцій звичайно застосовується для класів, для яких семантика операцій робить програму більш зрозумілої. Якщо призначення операції інтуїтивно незрозуміло, перевантажувати таку операцію не рекомендується.

Операції класу описуються за допомогою методів спеціального виду, синтаксис яких виглядає в такий спосіб:

```
[ атрибути] специфікатори operator операція (параметри )
{тіло}
```

У якості специфікаторів одночасно використовуються ключові слова `public` та `static`. Крім того, операцію можна оголосити як зовнішню - `extern`.

При описі операцій необхідно дотримувати наступних правил:

1. операція повинна бути описана як відкритий статичний метод класу (**`public static`**);
2. параметри в операцію повинні передаватися за значенням (тобто неприпустимо використовувати параметри **`ref`** і **`out`**);
3. сигнатури всіх операцій класу повинні розрізнятися;
4. типи, що використовуються в операції повинні бути доступні).

Унарні операції

У класах можна перевизначити наступні унарні операції: **+ - ! ~ ++ --**, а також сталі **true** і **false**. При цьому, якщо була перевантажена стали **true**, то повинна бути перевантажена і стала **false**, і навпаки.

Синтаксис оголошення унарної операції:

```
тип operator унарная_операція (параметр)
```

Приклади заголовків унарних операцій:

```
public static int operator + (Demoarray m)
public static Demoarray operator --(Demoarray m)
public static bool operator true (Demoarray m)
```

Параметр, переданий в операцію, повинен мати тип класу, для якого вона визначається. При цьому операція повинна повертати:

1. для операцій **+, -, !, ~** величину будь-якого типу;
2. для операцій **++, --** величину типу класу, для якого вона визначається;
3. для операцій **true** і **false** величину типу **bool**.

Операції не повинні змінювати значення операнда, що передається. Операція, що повертає величину типу класу, для якого вона визначається, повинна створити новий об'єкт цього класу, виконати з ним необхідні дії й передати його як результату.

Як приклад розглянемо клас `Demoarray`, що реалізує одномірний масив.

```
class Demoarray
 {
  int[] Myarray;//закритий масив

  public Demoarray(int size)//конструктор 1
  {
   Myarray = new int[size];
  }

  public Demoarray(params int[] arr)//конструктор 2
  {
   Myarray = new int[arr.Length];
   for (int i = 0; i < Myarray.Length; i++) Myarray[i] = arr[i];
  }

  public int Lengtharray //властивість, що повертає розмірність
  {
   get { return Myarray.Length; }
  }

  public int this[int i] //індексатор
  {
   get
   {
    if (i < 0 || i >= Myarray.Length) throw new Exception("вихід за межі
масиву");
```

```
    return Myarray[i];
   }
   set
   {
    if (i < 0 || i >= Myarray.Length) throw new Exception("вихід за межі
масиву");
    else Myarray[i] = value;
   }
  }

  public static Demoarray operator -(Demoarray x) //перевантаження операції
унарный мінус
  {
   Demoarray temp = new Demoarray(x.Lengtharray);
   for (int i = 0; i < x.Lengtharray; ++i)
    temp[i] = -x[i];
   return temp;
  }

  public static Demoarray operator ++(Demoarray x) //перевантаження операції
инкремента
  {
   Demoarray temp = new Demoarray(x.Lengtharray);
   for (int i = 0; i < x.Lengtharray; ++i)
    temp[i] = x[i]+1;
   return temp;
  }

  public static bool operator true(Demoarray a) //перевантаження сталихи true
  {
   foreach (int i in a.Myarray)
   {
    if (i<0)
    {
     return false;
    }
   }
   return true;
  }

  public static bool operator false(Demoarray a)//перевантаження сталихи
false
  {
   foreach (int i in a.Myarray)
   {
    if (i>0)
    {
     return true;
    }
   }
   return false;
  }

  public void Print(string name) //метод - виводить поле-масив на екран
  {
   Console.Writeline(name + ": ");
   for (int i = 0; i < Myarray.Length; i++)
    Console.Write(Myarray[i] + " ");
   Console.Writeline();
  }
 }
```

```
class Program
{
 static void Main()
 {
  try
  {
   Demoarray Mas = new Demoarray(1, -4, 3, -5, 0); //виклик конструктора 2
   Mas.Print("Вихідний масив");
   Console.Writeline("\nунарный мінус");
   Demoarray newmas=-Mas; //застосування операції унарного мінуса
   Mas.Print("Массив Mas"); // зверніть увагу, що створюється новий об'єкт і
знаки міняються
   newmas.Print("Масив newmas"); //тільки в нового масиву
   Console.Writeline("\nоперация префиксного инкремента");
   Demoarray Mas1=++Mas;
 Mas.Print("Массив Mas");
   Mas1.Print("Массив Mas1=++Mas");
   Console.Writeline("\nоперация постфиксного инкремента");
   Demoarray Mas2=Mas++;
   Mas.Print("Массив Mas");
   Mas2.Print("Массив Mas2=Mas++");
   if (Mas)
    Console.Writeline("\nв масиві всі елементи позитивні\n");
   else Console.Writeline("\nв масиві є не позитивні елементи\n");
  }
  catch (Exception e)
  {
   Console.Writeline(e.Message);
  }
 }
```

Бінарні операції

При розробці класу можна перевантажити наступні бінарні операції: **+ - * / % & | << >> == != < > <= >=**. Зверніть увагу, операцій присвоювання в цьому списку немає.

Синтаксис оголошення бінарної операції:

```
тип operator бінарна_операція (параметр1, параметр 2)
```

Приклади заголовків бінарних операцій:

```
public static Demoarray operator + (Demoarray a, Demoarray b)
public static bool operator == (Demoarray a, Demoarray b)
```

При перевизначені бінарних операцій потрібно враховувати наступні правила:

1. Хоча б один параметр, переданий в операцію, повинен мати тип класу, для якого вона визначається.
2. Операція може повертати величину будь-якого типу.
3. Операції рівність і нерівність визначаються тільки парами та звичайно повертають логічне значення. Найчастіше пере визначаються операції порівняння на рівність і нерівність для того, щоб забезпечити порівняння значення деяких полів об'єктів, а не посилань на об'єкт.

Приклад. Повернемося до класу Demoarray, що реалізує одномірний масив, і додамо в нього дві версії перевантаженої операції +:

- Варіант 1: додає до кожного елемента масиву задане число;
- Варіант 2: по-елементно складає два масиви

```
class Demoarray
 {
  …
  public static Demoarray operator +(Demoarray x, int a) //варіант 1
  {
   Demoarray temp = new Demoarray(x.Lengtharray);
   for (int i = 0; i < x.Lengtharray; ++i)
    temp[i]=x[i]+a;
   return temp;
  }

  public static Demoarray operator +(Demoarray x, Demoarray y) //варіант 2
  {
   if (x.Lengtharray == y.Lengtharray)
   {
    Demoarray temp = new Demoarray(x.Lengtharray);
    for (int i = 0; i < x.Lengtharray; ++i)
     temp[i] = x[i] + y[i];
    return temp;
   }
   else throw new Exception("невідповідність размерностей");
  }
 }

 class Program
 {
  static void Main()
  {
   try
   {
    Demoarray a = new Demoarray(1, -4, 3, -5, 0);
    a.Print("Масиву a");
    Demoarray b=a+10;
    b.Print("\nмассива b");
    Demoarray c = a+b;
    c.Print("\nмассива c");
   }
   catch (Exception e)
   {
    Console.Writeline(e.Message);
   }
  }
 }
```

Операції перетворення типів

Операції перетворення типів забезпечують можливість явного та неявного перетворення між користувацькими типами даних. Синтаксис оголошення операції перетворення типів виглядає в такий спосіб:

```
explicit operator цільовий_тип (параметр)  //явне перетворення
implicit operator цільовий_тип (параметр)  //неявне перетворення
```

Ці операції виконують перетворення з типу параметра в тип, зазначений у заголовку операції. Одним із цих типів повинен бути клас, для якого виконується перетворення.

Неявне перетворення виконується автоматично в наступних ситуаціях:

1. при присвоюванні об'єкта змінної цільового типу;
2. при використанні об'єкта у вираженні, що містить змінні цільового типу;
3. при передачі об'єкта в метод параметра цільового типу;
4. при явнім приведенні типу.

Явне перетворення виконується при використанні операції приведення типу.

При визначенні операції перетворення типу слід ураховувати наступні особливості:

1. тип значення, що вертається (цільовий_тип) включається в сигнатуру об'явника операції;
2. ключові слова **`explicit`** і **`implicit`** не включаються в сигнатуру об'явника операції.

Отже, для одного й того класу не можна визначити одночасно і явну, і неявну версію. Однак, тому що неявне перетворення автоматично виконаються при явнім використанні операції приведення типу, те досить розробити тільки неявну версію операції перетворення типу.

Як приклад повернемося до класу `Demoarray`, що реалізує одномірний масив, і додамо в нього неявну версію перевизначення типу `Demoarray` у тип одномірний масив і навпаки:

```
class Demoarray
 {
  …

  public static implicit operator Demoarray (int []a) //неявне перетворення
типу int [] в Demoarray
  {
   return new Demoarray(a);
  }

  public static implicit operator int [](Demoarray a) //неявне перетворення
типу Demoarray в int []
  {
   int []temp=new int[a.Lengtharray];
   for (int i = 0; i < a.Lengtharray; ++i)
    temp[i] = a[i];
   return temp;
  }
 }

 class Program
 {
  static void arrayprint(string name, int[]a) //метод, який дозволяє вивести
на екран одномірний масив
  {
   Console.Writeline(name + ": ");
   for (int i = 0; i < a.Length; i++)
    Console.Write(a[i] + " ");
   Console.Writeline();
  }
```

```
  static void Main()
  {
   try
   {
    Demoarray a = new Demoarray(1, -4, 3, -5, 0);
    int []mas1=a; //неявне перетворення типу Demoarray в int []
    int []mas2=(int []) a; //явне перетворення типу Demoarray в int []
    Demoarray b1 =mas1; //неявне перетворення типу int [] в Demoarray
    Demoarray b2 =(Demoarray)mas2; //явне перетворення типу int [] в
Demoarray

    //зміна значень
    mas1[0]=0;  mas2[0]=-1;
    b1[0]=100;  b2[0]=-100;

    //висновок на екран
    a.Print("Масиву a");
    arrayprint("Массив mas1", mas1);
    arrayprint("Массив mas2", mas2);
    b1.Print("Масиву b1");
    b2.Print("Масиву b2");
    }
   catch (Exception e)
   {
    Console.Writeline(e.Message);
   }
  }
 }
```

Завдання до лабораторної роботи:

Порядок виконання роботи:

1) Створити проект C#.
2) Визначити класи в яких реалізуються методи, деструктори, індексатори та операції згідно з варіантом завдання.
3) Розробити програму мовою C# тестування всіх можливосте створених класів із виведення відповідної інформації.
4) Підготувати звіт у твердій копії та в електронному виді.

Завдання 1. Варіанти задач. Створити клас із полями, конструкторами, методами та властивостями. До запропонованих полів, методів та властивосте можна додавати власні.

1.1. У клас **`Point`** додати:

- Індексатор, що дозволяє по індексу **`0`** звертатися до поля x, по індексу **`1`** - до поля y, по індексу **`2`** – до поля колір, а при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **`++`** (**`--`**): одночасно збільшує (зменшує) значення полів x та y на 1;

- сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо значення полів **x** та **y** рівне, інакше **false**;
- операції бінарний **+**: одночасно додає до полів x та y значення скаляра;
- перетворення типу **Point** в **string** (і навпаки).

1.2. У клас **Triangle** додати:

- Індексатор, що дозволяє по індексу **0** звертатися до поля a, по індексу **1** - до поля **b**, по індексу **2** - до поля **c** , по індексу **3** – до поля колір, а при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **++** (**--**): одночасно збільшує (зменшує) значення полів **a**, **b** і **c** на **1**;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо трикутник із заданими довжинами сторін існує, інакше **false**;
  - операції *****: одночасно множить поля **a**, **b** і **c** на скаляр;
- перетворення типу **Triangle** в **string** (і навпаки).

1.3. У клас **Rectangle** додати:

- Індексатор, що дозволяє по індексу **0** звертатися до поля **a**, по індексу **1** - до поля **b**, по індексу **2** – до поля колір, а при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **++** (**--**): одночасно збільшує (зменшує) значення полів **a** і **b**;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо прямокутник із заданими довжинами сторін є квадратом, інакше **false**;
  - операції *: одночасно множить поля a і b на скаляр;
- перетворення типу **Rectangle** в **string** (і навпаки).

1.4. У клас **Rom**b (ромб) додати:

- Індексатор, що дозволяє по індексу **0** звертатися до поля a, по індексу **1** - до поля **d1**, по індексу **2** – до поля колір, а при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **++** (**--**): одночасно збільшує (зменшує) значення полів **a** і **d1**;

- сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо ромб із заданими параметрами є квадратом, інакше **false**;
- операції *****: одночасно множить по **a** і **d1** на скаляр;
- перетворення типу **Romb** в **string** (і навпаки).

1.5. У клас **Money** додати:

- Індексатор, що дозволяє по індексу **0** звертатися до поля **first**, по індексу **1** - до поля **second**, при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **++** (**--**): одночасно збільшує (зменшує) значення полів **first** і **second**;
  - операції **!**: повертає значення **true**, якщо поле **second** не нульове, інакше **false**;
  - операції бінарний **+**: додає до значення поля **second** значення скаляра;
- перетворення типу **Money** в **string** (і навпаки).

1.6. У клас **DRomb** (ромб) додати:

- Індексатор, що дозволяє по індексу **0** звертатися до поля **d1**, по індексу **1** - до поля **d2**, по **індексу 2** – до поля колір, а при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **++** (**--**): одночасно збільшує (зменшує) значення полів **a** і **d1**;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо ромб із заданими параметрами є квадратом, інакше **false**;
  - операції **+**: одночасно додає **d1** та **d2** скаляр;
  - перетворення типу **DRomb** в **string** (і навпаки).

1.7. У клас **Trapeze** (трапеція) додати:

- Індексатор, що дозволяє по індексу **0** звертатися до поля **a**, по індексу **1** - до поля **b**, по індексу **2** - до поля **h**, по індексу **3** – до поля колір, а при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **++** (**--**): одночасно збільшує (зменшує) значення полів **a** і **b** на 1;

- сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо із заданими параметрами існує, інакше **false**;
- операції *****: одночасно множить поля a і h на скаляр;

- перетворення типу **Trapeze** в **string** (і навпаки).

1.8. У клас **ITriangle** (isosceles triangle, рівнобедрений трикутник), додати:

- Індексатор, що дозволяє по індексу **0** звертатися до поля a, по індексу **1** - до поля **b**, по індексу **2** – до поля колір, а при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **++** (**--**): одночасно збільшує (зменшує) значення полів a і b на 1;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо трикутник із заданими довжинами сторін існує, інакше **false**;
  - операції *****: одночасно множить поля a і b на скаляр;
- перетворення типу **ITriangle** в **string** (і навпаки).

1.9. У клас **ATriangle** (angled triangle, прямокутний трикутник), додати:

- Індексатор, що дозволяє по індексу **0** звертатися до поля a, по індексу **1** - до поля **b**, по індексу **2** – до поля колір, а при інших значеннях індексу видається повідомлення про помилку.
- Перевантаження:
  - операції **++** (**--**): одночасно збільшує (зменшує) значення полів a і b на 1;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення true, якщо трикутник із заданими довжинами сторін існує, інакше **false**;
  - операції **+**: одночасно додає скаляр до полів a і b;
- перетворення типу **ATriangle** в **string** (і навпаки).

1.10. Додати в клас **Date** (дата):

- Індексатор, що дозволяє визначити дату і-того по рахункові дня щодо встановленої дати (при від'ємних значеннях індексу відлік ведеться у зворотному порядку).
- Перевантаження:
  - операції **!**: повертає значення true, якщо встановлена дата не є останнім днем місяця, інакше **false**;

- сталих **true** і **false**: звертання до екземпляра класу дає значення true, якщо встановлена дата є початком року, інакше **false**;
- операції **&**: повертає значення **true**, якщо поля двох об'єктів рівні, інакше **false**;
- перетворення класу **Data** у тип s**tring** (і навпаки).

Завдання 2. Варіанти задач. Створити клас вектор. Визначити поля, конструктори, деструктор, методи та перевантажити операції. При перевантаженні бінарних операцій функції-операції виконує певні дії над кожною парою елементів масивів в об’єктах класу вектор за індексом, якщо в функції-операції в параметрі скаляр то операція відбувається на елементом масиву класу вектор та скаляром; у випадку коли розміри векторів, тоді повертати більший за розміром вектор, також можна виконати операцію над елементами з індексами, які допустимі у меншому векторі. Функції-операції рівності та порівняння виконують дії(порівняння, рівності) над кожною парою елементів векторів за індексом, повертає значення **true**, якщо умова виконується для кожної пари, інакше **false**. Розробити програму тестування даного класу.

2.1. Створити клас **VectorInt** (вектор цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `int [] IntArray;` // масив
  - `uint size;` // розмір вектора
  - `int codeError;` // код помилки
  - `static uint num_vec;` // кількість векторів
- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;

- присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
- статичний метод, що підраховує кількість векторів даного типу;
- присвоїти елементам масиву деяке значення (параметр);

- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля `codeError` (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле `codeError` записується -1(при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле `codeError`); .
- Перевантаження:
  - унарних операції `++` `(--)`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - сталих `true` і `false`: звертання до екземпляра класу дає значення `true`, якщо `size` не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше `false`;
  - унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
  - унарної побітової операції `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
  - арифметичних бінарні операції ():
    a. `+` додавання:
       i. для двох векторів
       ii. для вектора і скаляра типу `int`
    b. `-` (віднімання):
       i. для двох векторів
       ii. для вектора і скаляра типу `int`;
    c. `*`(множення)
       i. для двох векторів
       ii. для вектора і скаляра типу `int`;
    d. `/` (ділення)
       i. для двох векторів
       ii. для вектора і скаляра типу `int`;
    e. `%` (остача від ділення)
       i. для двох векторів

ii. для вектора і скаляра типу `int`;

– побітові бінарні операції
  a. `|` (побітове додавання)
    i. для двох векторів
    ii. для вектора і скаляра типу `int`;
  b. `^` (побітове додавання за модулем 2)
    i. для двох векторів
    ii. для вектора і скаляра типу `int`;
  c. `|` (побітове множення)
    i. двох векторів
    ii. вектора і скаляра типу `int`;
  d. `>>` (побітовий зсув право)
    i. для двох векторів
    ii. для вектора і скаляра типу `int`;
  e. `<<` (побітовий зсув ліво)
    i. для двох векторів
    ii. для вектора і скаляра типу `int`;
– операцій `==`(рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом)
  a. `>` (більше) для двох векторів;
  b. `>=` (більше рівне) для двох векторів;
  c. `<` (менше) для двох векторів;
  d. `<=`(менше рівне) для двох векторів.

2.2. Створити клас `VectorUInt` (вектор цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  – `uint [] IntArray;` // масив
  – `uint size;` // розмір вектора
  – `int codeError;` // код помилки
  – `static uint num_vec;` // кількість векторів
- Конструктори:
  – конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  – конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);

- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).

- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість векторів даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле **codeError** записується -1(при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле **codeError**); .
- Перевантаження:
  - унарних операції **++** **(--)**: одночасно збільшує (зменшує) значення елементів масиву на **1**;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **size** не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
  - унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **size** не дорівнює – нулю, інакше **false**;
  - унарної побітової операції **~** (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
  - арифметичних бінарні операції ():
    a. **+** додавання:
       i. для двох векторів
       ii. для вектора і скаляра типу **int**
    b. **-** (віднімання):
       i. для двох векторів
       ii. для вектора і скаляра типу **int**;
    c. *****(множення)

i. для двох векторів
ii. для вектора і скаляра типу `int`;

d. `/` (ділення)
i. для двох векторів
ii. для вектора і скаляра типу `short`;

e. `%` (остача від ділення)
i. для двох векторів
ii. для вектора і скаляра типу `short`;

– побітові бінарні операції

a. `|` (побітове додавання)
i. для двох векторів
ii. для вектора і скаляра типу `uint`;

b. `^` (побітове додавання за модулем 2)
i. для двох векторів
ii. для вектора і скаляра типу `uint`;

c. `|` (побітове множення)
i. двох векторів
ii. вектора і скаляра типу `uint`;

d. `>>` (побітовий зсув право)
i. для двох векторів
ii. для вектора і скаляра типу `uint`;

e. `<<` (побітовий зсув ліво)
i. для двох векторів
ii. для вектора і скаляра типу `uint`;

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

a. `>` (більше) для двох векторів;
b. `>=` (більше рівне) для двох векторів;
c. `<` (менше) для двох векторів;
d. `<=`(менше рівне) для двох векторів.

2.3. Створити клас `VectorShort` (вектор цілих чисел). Розробити такі елементи класу:

• Поля (захищені):

– `short [] ShortArray;` // масив
– `uint n;` // розмір вектора
– `uint codeError;` // код помилки
– `static uint num_v;` // кількість векторів

- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість векторів даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **`codeError`** (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле **`codeError`** записується – 10 (при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле **`codeError`**); .
- Перевантаження:
  - унарних операції `++` `(--)`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - сталих `true` і `false`: звертання до екземпляра класу дає значення `true`, якщо **`size`** не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше `false`;
  - унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо **`size`** не дорівнює – нулю, інакше `false`;
  - унарної побітової операції `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
  - арифметичних бінарні операції ():
    a. `+` додавання:
       i. для двох векторів

ii. для вектора і скаляра типу **short**

b. **-** (віднімання):
   i. для двох векторів
   ii. для вектора і скаляра типу **short**;

c. ***** (множення)
   i. для двох векторів
   ii. для вектора і скаляра типу **short**;

d. **/** (ділення)
   i. для двох векторів
   ii. для вектора і скаляра типу **ushort**;

e. **%** (остача від ділення)
   i. для двох векторів
   ii. для вектора і скаляра типу **short**;

– побітові бінарні операції
   a. **|** (побітове додавання)
      i. для двох векторів
      ii. для вектора і скаляра типу **short**;
   b. **^** (побітове додавання за модулем 2)
      i. для двох векторів
      ii. для вектора і скаляра типу **short**;
   c. **|** (побітове множення)
      i. двох векторів
      ii. вектора і скаляра типу **short**;
   d. **>>** (побітовий зсув право)
      i. для двох векторів
      ii. для вектора і скаляра типу **ushort**;
   e. **<<** (побітовий зсув ліво)
      i. для двох векторів
      ii. для вектора і скаляра типу **ushort**;

– операцій **==**(рівності) та **!=** (нерівності), функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом)
   a. **>** (більше) для двох векторів;
   b. **>=** (більше рівне) для двох векторів;
   c. **<** (менше) для двох векторів;
   d. **<=**(менше рівне) для двох векторів.

2.4. Створити клас **VectorUshort** (вектор цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `ushort [] ArrayUShort;` // масив
  - `uint num;` // розмір вектора
  - `uint codeError;` // код помилки
  - `static uint num_vs;` // кількість векторів
- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість векторів даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле **codeError** записується – 1 (при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле **codeError**); .
- Перевантаження:
  - унарних операції `++` `(--)`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - сталих `true` і `false`: звертання до екземпляра класу дає значення `true`, якщо **size** не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше `false`;

- унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
- унарної побітової операції `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
- арифметичних бінарні операції ():
    a. `+` додавання:
        i. для двох векторів
        ii. для вектора і скаляра типу `ushort;`
    b. `-` (віднімання):
        i. для двох векторів
        ii. для вектора і скаляра типу `ushort`;
    c. `*`(множення)
        i. для двох векторів
        ii. для вектора і скаляра типу `ushort`;
    d. `/` (ділення)
        i. для двох векторів
        ii. для вектора і скаляра типу `ushort`;
    e. `%` (остача від ділення)
        i. для двох векторів
        ii. для вектора і скаляра типу `ushort`;
- побітові бінарні операції
    a. `|` (побітове додавання)
        i. для двох векторів
        ii. для вектора і скаляра типу `ushort`;
    b. `^` (побітове додавання за модулем 2)
        i. для двох векторів
        ii. для вектора і скаляра типу `ushort`;
    c. `|` (побітове множення)
        i. двох векторів
        ii. вектора і скаляра типу `ushort`;
    d. `>>` (побітовий зсув право)
        i. для двох векторів
        ii. для вектора і скаляра типу `short`;
    e. `<<` (побітовий зсув ліво)
        i. для двох векторів
        ii. для вектора і скаляра типу `short`;

- операцій `==`(рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом)
  a. `>` (більше) для двох векторів;
  b. `>=` (більше рівне) для двох векторів;
  c. `<` (менше) для двох векторів;
  d. `<=`(менше рівне) для двох векторів.

2.5. Створити клас **`VectorLong`** (вектор цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `long [] IntArray;` // масив
  - `uint size;` // розмір вектора
  - `int codeError;` // код помилки
  - `static uint num_vl;` // кількість векторів
- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість векторів даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **`codeError`** (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле **`codeError`**

записується -1(при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле **codeError**); .

- Перевантаження:
  - унарних операції **++** **(--)**: одночасно збільшує (зменшує) значення елементів масиву на 1;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **size** не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
  - унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **size** не дорівнює – нулю, інакше **false**;
  - унарної побітової операції **~** (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
  - арифметичних бінарні операції ():
    a. **+** додавання:
       i. для двох векторів
       ii. для вектора і скаляра типу **long**
    b. **–** (віднімання):
       i. для двох векторів
       ii. для вектора і скаляра типу **long**;
    c. *****(множення)
       i. для двох векторів
       ii. для вектора і скаляра типу **long**;
    d. **/** (ділення)
       i. для двох векторів
       ii. для вектора і скаляра типу **long**;
    e. **%** (остача від ділення)
       i. для двох векторів
       ii. для вектора і скаляра типу **long**;
  - побітові бінарні операції
    a. **|** (побітове додавання)
       i. для двох векторів
       ii. для вектора і скаляра типу **long**;
    b. **^** (побітове додавання за модулем 2)
       i. для двох векторів
       ii. для вектора і скаляра типу **long**;
    c. **|** (побітове множення)
       i. двох векторів
       ii. вектора і скаляра типу **int**;

d. `>>` (побітовий зсув право)
   i. для двох векторів
   ii. для вектора і скаляра типу `long`;
e. `<<` (побітовий зсув ліво)
   i. для двох векторів
   ii. для вектора і скаляра типу `long`;

– операцій `==`(рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом)
   a. `>` (більше) для двох векторів;
   b. `>=` (більше рівне) для двох векторів;
   c. `<` (менше) для двох векторів;
   d. `<=`(менше рівне) для двох векторів.

2.6. Створити клас `VectorULong` (вектор цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  – `ulong [] IntArray;` // масив
  – `uint size;` // розмір вектора
  – `int codeError;` // код помилки
  – `static uint num_vec;` // кількість векторів
- Конструктори:
  – конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  – конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);
  – конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  – ввести елементи вектора з клавіатури;
  – вивести елементи вектора на екран;
  – присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  – статичний метод, що підраховує кількість векторів даного типу;
  – присвоїти елементам масиву деяке значення (параметр);
- Властивості:

- повертає розмірність вектора (доступні лише для читання);
- дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).

- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле **codeError** записується -1(при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле **codeError**); .
- Перевантаження:
  - унарних операції **++** **(--)**: одночасно збільшує (зменшує) значення елементів масиву на **1**;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **size** не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
  - унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **size** не дорівнює – нулю, інакше **false**;
  - унарної побітової операції **~** (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
  - арифметичних бінарні операції ():
    a. **+** додавання:
       i. для двох векторів
       ii. для вектора і скаляра типу **ulong**
    b. **-** (віднімання):
       i. для двох векторів
       ii. для вектора і скаляра типу **ulong**;
    c. *****(множення)
       i. для двох векторів
       ii. для вектора і скаляра типу **ulong**;
    d. **/** (ділення)
       i. для двох векторів
       ii. для вектора і скаляра типу **ulong**;
    e. **%** (остача від ділення)
       i. для двох векторів
       ii. для вектора і скаляра типу **ulong**;
  - побітові бінарні операції
    a. **|** (побітове додавання)
       i. для двох векторів
       ii. для вектора і скаляра типу **ulong**;

b. ^ (побітове додавання за модулем 2)
   i. для двох векторів
   ii. для вектора і скаляра типу `ulong`;
c. | (побітове множення)
   i. двох векторів
   ii. вектора і скаляра типу `ulong`;
d. >> (побітовий зсув право)
   i. для двох векторів
   ii. для вектора і скаляра типу `uint`;
e. << (побітовий зсув ліво)
   i. для двох векторів
   ii. для вектора і скаляра типу `uint`;

- операцій ==(рівності) та != (нерівності), функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом)
  a. > (більше) для двох векторів;
  b. >= (більше рівне) для двох векторів;
  c. < (менше) для двох векторів;
  d. <=(менше рівне) для двох векторів.

2.7. Створити клас `VectorFloat` (вектор дійсних чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `float [] FArray;` // масив
  - `uint num;` // розмір вектора
  - `int codeError;` // код помилки
  - `static uint num_vec;` // кількість векторів
- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;

- присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
- статичний метод, що підраховує кількість векторів даного типу;
- присвоїти елементам масиву деяке значення (параметр);

- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля `codeError` (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле `codeError` записується -1(при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле `codeError`); .
- Перевантаження:
  - унарних операції `++` `(--)`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - сталих `true` і `false`: звертання до екземпляра класу дає значення `true`, якщо `size` не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше `false`;
  - унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
  - унарної побітової операції `~` (заперечення): повертає цілу частину дійсного числа заперечення для всіх елементів масиву класу вектор;
  - арифметичних бінарні операції ():
    a. `+` додавання:
       i. для двох векторів
       ii. для вектора і скаляра типу `float`
    b. `-` (віднімання):
       i. для двох векторів
       ii. для вектора і скаляра типу `float`;
    c. `*`(множення)
       i. для двох векторів
       ii. для вектора і скаляра типу `float`;
    d. `/` (ділення)
       i. для двох векторів
       ii. для вектора і скаляра типу `float`;
    e. `%` (остача від ділення)
       i. для двох векторів

ii. для вектора і скаляра типу `float`;
- побітові бінарні операції (виконувати операції на кодами символьного представлення дійсного числа)
  a. `|` (побітове додавання)
     i. для двох векторів
     ii. для вектора і скаляра типу `ubyte`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох векторів
     ii. для вектора і скаляра типу `ubyte`;
  c. `|` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `ubyte`;
  d. `>>` (побітовий зсув право)
     i. для двох векторів
     ii. для вектора і скаляра типу `uint`;
  e. `<<` (побітовий зсув ліво)
     i. для двох векторів
     ii. для вектора і скаляра типу `uint`;
- операцій `==`(рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом)
  a. `>` (більше) для двох векторів;
  b. `>=` (більше рівне) для двох векторів;
  c. `<` (менше) для двох векторів;
  d. `<=`(менше рівне) для двох векторів.

2.8. Створити клас `VectorDecimal` (вектор дійсних чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `decimal [] ArrayDecimal;` // масив
  - `uint num;` // розмір вектора
  - `int codeError;` // код помилки
  - `static uint num_vec;` // кількість векторів
- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);

- конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).

- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість векторів даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле **codeError** записується -1(при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле **codeError**); .
- Перевантаження:
  - унарних операції **++** **(--)**: одночасно збільшує (зменшує) значення елементів масиву на **1**;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **size** не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
  - унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **size** не дорівнює – нулю, інакше **false**;
  - унарної побітової операції **~** (заперечення): повертає цілу частину дійсного числа для всіх елементів масиву класу вектор;
  - арифметичних бінарні операції ():
    a. **+** додавання:
       i. для двох векторів
       ii. для вектора і скаляра типу **decimal**
    b. **-** (віднімання):
       i. для двох векторів
       ii. для вектора і скаляра типу **decimal**;
    c. *****(множення)

i. для двох векторів
ii. для вектора і скаляра типу `decimal`;

d. `/` (ділення)
i. для двох векторів
ii. для вектора і скаляра типу `decimal`;

e. `%` (остача від ділення)
i. для двох векторів
ii. для вектора і скаляра типу `decimal`;

– побітові бінарні операції (виконувати операції на кодами символьного представлення дійсного числа)

a. `|` (побітове додавання)
i. для двох векторів
ii. для вектора і скаляра типу `ubyte`;

b. `^` (побітове додавання за модулем 2)
i. для двох векторів
ii. для вектора і скаляра типу `ubyte`;

c. `|` (побітове множення)
i. двох векторів
ii. вектора і скаляра типу `ubyte`;

d. `>>` (побітовий зсув право)
i. для двох векторів
ii. для вектора і скаляра типу `uint`;

e. `<<` (побітовий зсув ліво)
i. для двох векторів
ii. для вектора і скаляра типу `uint`;

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

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

a. `>` (більше) для двох векторів;
b. `>=` (більше рівне) для двох векторів;
c. `<` (менше) для двох векторів;
d. `<=`(менше рівне) для двох векторів.

2.9. Створити клас `VectorDouble` (вектор дійсних чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `double [] FArray;` // масив
  - `uint num;` // розмір вектора

- `int codeError;` // код помилки
- `static uint num_vd;` // кількість векторів

- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість векторів даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **`codeError`** (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле **`codeError`** записується -1(при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле **`codeError`**); .
- Перевантаження:
  - унарних операції `++` `(--)`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - сталих `true` і `false`: звертання до екземпляра класу дає значення `true`, якщо **`size`** не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше `false`;
  - унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо **`size`** не дорівнює – нулю, інакше `false`;
  - унарної побітової операції `~` (заперечення): повертає цілу частину дійсного числа для всіх елементів масиву класу вектор;
  - арифметичних бінарні операції ():

a. `+` додавання:
   i. для двох векторів
   ii. для вектора і скаляра типу `double`
b. `-` (віднімання):
   i. для двох векторів
   ii. для вектора і скаляра типу `double`;
c. `*`(множення)
   i. для двох векторів
   ii. для вектора і скаляра типу `double`;
d. `/` (ділення)
   i. для двох векторів
   ii. для вектора і скаляра типу `double`;
e. `%` (остача від ділення)
   i. для двох векторів
   ii. для вектора і скаляра типу `double`;

– побітові бінарні операції (виконувати операції на кодами символьного представлення дійсного числа)
a. `|` (побітове додавання)
   i. для двох векторів
   ii. для вектора і скаляра типу `ubyte`;
b. `^` (побітове додавання за модулем 2)
   i. для двох векторів
   ii. для вектора і скаляра типу `ubyte`;
c. `|` (побітове множення)
   i. двох векторів
   ii. вектора і скаляра типу `ubyte`;
d. `>>` (побітовий зсув право)
   i. для двох векторів
   ii. для вектора і скаляра типу `uint`;
e. `<<` (побітовий зсув ліво)
   i. для двох векторів
   ii. для вектора і скаляра типу `uint`;

– операцій `==`(рівності) та `!=` (нерівності), функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
– порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом)
a. `>` (більше) для двох векторів;
b. `>=` (більше рівне) для двох векторів;
c. `<` (менше) для двох векторів;

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

2.10. Створити клас **`VectorByte`** (вектор цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `byte [] BArray;` // масив
  - `uint n;` // розмір вектора
  - `int codeError;` // код помилки
  - `static uint num_vec;` // кількість векторів
- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор з одним параметром - розмір вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із двома параметрами - розмір вектора та значення ініціалізації(виділяє місце - значення першого аргументу та ініціалізує значенням другого аргументу).
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість векторів даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертає розмірність вектора (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **`codeError`** (доступні для читання і запису).
- Індексатор, що дозволяє звертатися по індексу до масиву, якщо значення індексу невірне в поле **`codeError`** записується -1(при читанні повертається значення – 0, при записі – запис здійснюється тільки в поле **`codeError`**); .
- Перевантаження:
  - унарних операції `++` `(--)`: одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - сталих `true` і `false`: звертання до екземпляра класу дає значення `true`, якщо **`size`** не дорівнює – нулю, або всі елементи масиву не рівні – нулю, інакше `false`;

- унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо `size` не дорівнює – нулю, інакше `false`;
- унарної побітової операції `~` (заперечення): повертає побітове заперечення для всіх елементів масиву класу вектор;
- арифметичних бінарні операції ():
    a. `+` додавання:
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`
    b. `-` (віднімання):
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`;
    c. `*`(множення)
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`;
    d. `/` (ділення)
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`;
    e. `%` (остача від ділення)
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`;
- побітові бінарні операції
    a. `|` (побітове додавання)
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`;
    b. `^` (побітове додавання за модулем 2)
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`;
    c. `|` (побітове множення)
        i. двох векторів
        ii. вектора і скаляра типу `byte`;
    d. `>>` (побітовий зсув право)
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`;
    e. `<<` (побітовий зсув ліво)
        i. для двох векторів
        ii. для вектора і скаляра типу `byte`;

- операцій ==(рівності) та != (нерівності), функція-операція виконує певні дії над кожною парою елементів векторів за індексом;
- порівняння (функція-операція виконує певні дії над кожною парою елементів векторів за індексом)
  a. > (більше) для двох векторів;
  b. >= (більше рівне) для двох векторів;
  c. < (менше) для двох векторів;
  d. <=(менше рівне) для двох векторів.

Завдання 3. Варіанти задач. Створити клас матриця. Визначити поля, конструктори, деструктор, методи та перевантажити операції. При перевантаженні бінарних операцій функції-операції виконує, крім операції множення, певні дії над кожною парою елементів масивів в об'єктах класів за індексами, якщо в функції-операції в параметрі скаляр то операція відбувається на елементом масиву класу та скаляром; у випадку коли розміри матриць різні тоді повертати першу матрицю, також можна виконати операцію над елементами з індексами, які допустимі. Функції-операції рівності та порівняння виконують дії(порівняння, рівності) над кожною парою елементів за індексами, повертає значення `true`, якщо умова виконується для кожної пари, інакше `false`. Розробити програму тестування даного класу.

3.1. Створити клас `MatrixInt` (матриця цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `int` **[,] IntArray**; // масив
  - `int` **n,m**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_vec**; // кількість матриць
- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;

- присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
- статичний метод, що підраховує кількість матриць даного типу;
- присвоїти елементам масиву деяке значення (параметр);

• Властивості:
- повертають розмірність матриці (доступні лише для читання);
- дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).

• Індексатори:
- з двома індексами, які відповідають індексам масиву;
- з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
  Якщо значення індексів невірне в поле **codeError** записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .

• Перевантаження:
- унарних операції **++** (**--**): одночасно збільшує (зменшує) значення елементів масиву на 1;
- сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
- унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше **false**;
- унарної побітової операції **~** (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;
- арифметичних бінарні операці
  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 **MatrixInt** ,

iii. для матриці та скаляра типу `int`;

d. `/`(ділення)
   i. для двох матриць
   ii. для матриці та скаляра типу `int`;

e. `%`(остача від ділення)
   i. для двох матриць
   ii. 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. `>>` (побітовий зсув право)
     i. для двох матриць
     ii. для матриці та скаляра типу `int`;
  e. `<<` (побітовий зсув ліво)
     i. для двох матриць
     ii. для матриці та 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.

3.2. Створити клас `MatrixUint` (матриця цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  – `uint` **[,] IntArray**; // масив
  – `int` **n,m**; // розміри матриці
  – `int` **codeError**; // код помилки
  – `static int` **num_m**; // кількість матриць

- Конструктори: конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість матриць даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертають розмірність матриці (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **`codeError`** (доступні для читання і запису).
- Індексатори:
  - з двома індексами, які відповідають індексам масиву;
  - з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
    Якщо значення індексів невірне в поле **`codeError`** записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .
- Перевантаження:
  - унарних операції **`++`** (**`--`**): одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - сталих **`true`** і **`false`**: звертання до екземпляра класу дає значення **`true`**, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше **`false`**;
  - унарної логічної операції **`!`** (заперечення): повертає значення **`true`**, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше **`false`**;
  - унарної побітової операції **`~`** (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;

- арифметичних бінарні операці
  a. `+` додавання:
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `VectorUint` ,
     iii. для матриці та скаляра типу `uint`;
  d. `/`(ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`;
  e. `%`(остача від ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`;
- побітові бінарні операції
  a. `|` (побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`;
  c. `|` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `uint`;
  d. `>>` (побітовий зсув право)
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`;
  e. `<<` (побітовий зсув ліво)
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`;
- 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.

3.3. Створити клас `MatrixShort` (матриця цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `uint` **[,] ShortArray**; // масив
  - `int` **n,m**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_m**; // кількість матриць
- Конструктори: конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість матриць даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертають розмірність матриці (доступні лише для читання);
  - дозволяє отримати-встановити значення поля `codeError` (доступні для читання і запису).
- Індексатори:
  - з двома індексами, які відповідають індексам масиву;
  - з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
    Якщо значення індексів невірне в поле `codeError` записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .
- Перевантаження:
  - унарних операції `++` (`--`): одночасно збільшує (зменшує) значення елементів масиву на `1`;

- сталих `true` і `false`: звертання до екземпляра класу дає значення `true`, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше `false`;
- унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше `false`;
- унарної побітової операції `~` (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;
- арифметичних бінарні операці
  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 двох матриць
     ii. для матриці та скаляра типу `short`;
  e. `%`(остача від ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `short`;
- побітові бінарні операції
  a. `|` (побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `ushort`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `ushort`;
  c. `|` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `ushort`;
  d. `>>` (побітовий зсув право)
     i. для двох матриць
     ii. для матриці та скаляра типу `uint`;

e. `<<` (побітовий зсув ліво)
   i. для двох матриць
   ii. для матриці та скаляра типу `uint`;
- 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.

3.4. Створити клас `MatrixUshort` (матриця цілих чисел).
Розробити такі елементи класу:

- Поля (захищені):
  - `ushort` **[,] ShortIntArray**; // масив
  - `int` **n,m**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_m**; // кількість матриць
- Конструктори: конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість матриць даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертають розмірність матриці (доступні лише для читання);
  - дозволяє отримати-встановити значення поля `codeError` (доступні для читання і запису).
- Індексатори:

- з двома індексами, які відповідають індексам масиву;
- з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
  Якщо значення індексів невірне в поле **codeError** записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .

• Перевантаження:

- унарних операції **++** (**--**): одночасно збільшує (зменшує) значення елементів масиву на 1;
- сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
- унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше **false**;
- унарної побітової операції **~** (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;
- арифметичних бінарні операці
  a. **+** додавання:
     i. для двох матриць
     ii. для матриці та скаляра типу **ushort**
  b. **-** (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу **ushort**;
  c. *****(множення)
     i. для двох матриць,
     ii. для матриці та вектора **VectorUshort** ,
     iii. для матриці та скаляра типу **ushort**;
  d. **/**(ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу **ushort**;
  e. **%**(остача від ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу **ushort**;
- побітові бінарні операції
  a. **|** (побітове додавання)
     i. для двох матриць

ii. для матриці та скаляра типу `ushort`;
b. `^` (побітове додавання за модулем 2)
i. для двох матриць
ii. для матриці та скаляра типу `ushort`;
c. `|` (побітове множення)
i. двох векторів
ii. вектора і скаляра типу `ushort`;
d. `>>` (побітовий зсув право)
i. для двох матриць
ii. для матриці та скаляра типу `uint`;
e. `<<` (побітовий зсув ліво)
i. для двох матриць
ii. для матриці та скаляра типу `uint`;

– 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.

3.5. Створити клас `MatrixLong` (матриця цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `long` **[,] LongArray**; // масив
  - `uint` **n,m**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_m**; // кількість матриць
- Конструктори: конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;

- присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
- статичний метод, що підраховує кількість матриць даного типу;
- присвоїти елементам масиву деяке значення (параметр);

• Властивості:
- повертають розмірність матриці (доступні лише для читання);
- дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).

• Індексатори:
- з двома індексами, які відповідають індексам масиву;
- з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
  Якщо значення індексів невірне в поле **codeError** записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .

• Перевантаження:
- унарних операції **++** (**--**): одночасно збільшує (зменшує) значення елементів масиву на 1;
- сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
- унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше **false**;
- унарної побітової операції **~** (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;
- арифметичних бінарні операці
  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 двох матриць
ii. для матриці та скаляра типу `long`;
e. `%`(остача від ділення)
i. для двох матриць
ii. для матриці та скаляра типу `long`;
- побітові бінарні операції
a. `|` (побітове додавання)
i. для двох матриць
ii. для матриці та скаляра типу `ulong`;
b. `^` (побітове додавання за модулем 2)
i. для двох матриць
ii. для матриці та скаляра типу `ulong`;
c. `|` (побітове множення)
i. двох векторів
ii. вектора і скаляра типу `ulong`;
d. `>>` (побітовий зсув право)
i. для двох матриць
ii. для матриці та скаляра типу `uint`;
e. `<<` (побітовий зсув ліво)
i. для двох матриць
ii. для матриці та скаляра типу `uint`;
- 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.

3.6. Створити клас `MatrixUlong` (матриця цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `ulong` **[,] ULArray**; // масив
  - `uint` **n,m**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_m**; // кількість матриць

- Конструктори: конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість матриць даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертають розмірність матриці (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **`codeError`** (доступні для читання і запису).
- Індексатори:
  - з двома індексами, які відповідають індексам масиву;
  - з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
    Якщо значення індексів невірне в поле **`codeError`** записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .
- Перевантаження:
  - унарних операції **`++`** (**`--`**): одночасно збільшує (зменшує) значення елементів масиву на `1`;
  - сталих **`true`** і **`false`**: звертання до екземпляра класу дає значення **`true`**, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше **`false`**;
  - унарної логічної операції **`!`** (заперечення): повертає значення **`true`**, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше **`false`**;
  - унарної побітової операції **`~`** (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;

- арифметичних бінарні операці
  a. `+` додавання:
     i. для двох матриць
     ii. для матриці та скаляра типу `ulong`
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `ulong`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `VectorULong` ,
     iii. для матриці та скаляра типу `ulong`;
  d. `/`(ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `ulong`;
  e. `%`(остача від ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `ulong`;
- побітові бінарні операції
  a. `|` (побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `ulong`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `ulong`;
  c. `|` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `ulong`;
  d. `>>` (побітовий зсув право)
     i. для двох матриць
     ii. для матриці та скаляра типу `ushort`;
  e. `<<` (побітовий зсув ліво)
     i. для двох матриць
     ii. для матриці та скаляра типу `ushort`;
- 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.

3.7. Створити клас `FloatMatrix` (матриця дійсних чисел).
Розробити такі елементи класу:

- Поля (захищені):
  - `float` **[,] FMArray**; // масив
  - `uint` **n,m**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_mf**; // кількість матриць
- Конструктори: конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість матриць даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертають розмірність матриці (доступні лише для читання);
  - дозволяє отримати-встановити значення поля `codeError` (доступні для читання і запису).
- Індексатори:
  - з двома індексами, які відповідають індексам масиву;
  - з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j**);
    Якщо значення індексів невірне в поле `codeError` записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .
- Перевантаження:
  - унарних операції `++` (`--`): одночасно збільшує (зменшує) значення елементів масиву на `1`;

- сталих `true` і `false`: звертання до екземпляра класу дає значення `true`, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше `false`;
- унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше `false`;
- унарної побітової операції `~` (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;
- арифметичних бінарні операці
  a. `+` додавання:
     i. для двох матриць
     ii. для матриці та скаляра типу `float;`
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `float`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `VectorFloat` ,
     iii. для матриці та скаляра типу `float`;
  d. `/`(ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `float`;
  e. `%`(остача від ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `float`;
- побітові бінарні операції (виконувати операції на кодами символьного представлення дійсного числа)
  a. `|` (побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `float`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `float`;
  c. `|` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `float`;
  d. `>>` (побітовий зсув право)
     i. для двох матриць

ii. для матриці та скаляра типу **ushort**;

e. **<<** (побітовий зсув ліво)

i. для двох матриць

ii. для матриці та скаляра типу **ushort**;

- 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.

3.8. Створити клас **DecimalMatrix** (матриця дійсних чисел).
Розробити такі елементи класу:

- Поля (захищені):
  - **decimal [,] DCArray**; // масив
  - **uint n,m**; // розміри матриці
  - **int codeError**; // код помилки
  - **static int num_mf**; // кількість матриць
- Конструктори: конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість матриць даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертають розмірність матриці (доступні лише для читання);

- дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).

- Індексатори:
  - з двома індексами, які відповідають індексам масиву;
  - з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
    Якщо значення індексів невірне в поле **codeError** записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .
- Перевантаження:
  - унарних операції **++** (**--**): одночасно збільшує (зменшує) значення елементів масиву на 1;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
  - унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше **false**;
  - унарної побітової операції **~** (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;
  - арифметичних бінарні операці
    a. **+** додавання:
       i. для двох матриць
       ii. для матриці та скаляра типу **decimal;**
    b. **-** (віднімання):
       i. для двох матриць
       ii. для матриці та скаляра типу **decimal**;
    c. *****(множення)
       i. для двох матриць,
       ii. для матриці та вектора **VectorDecimal** ,
       iii. для матриці та скаляра типу **decimal**;
    d. **/**(ділення)
       i. для двох матриць
       ii. для матриці та скаляра типу **float**;
    e. **%**(остача від ділення)
       i. для двох матриць
       ii. для матриці та скаляра типу **uint**;

- побітові бінарні операції (виконувати операції на кодами символьного представлення дійсного числа)
  a. `|` (побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `decimal`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `decimal`
  c. `|` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `decimal`;
  d. `>>` (побітовий зсув право)
     i. для двох матриць
     ii. для матриці та скаляра типу `ushort`;
  e. `<<` (побітовий зсув ліво)
     i. для двох матриць
     ii. для матриці та скаляра типу `ushort`;
- 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.

3.9. Створити клас `MatrixDouble` (матриця дійсних чисел). Розробити такі елементи класу:

- Поля (захищені):
  - `double` **[,] DArray**; // масив
  - `uint` **n,m**; // розміри матриці
  - `int` **codeError**; // код помилки
  - `static int` **num_mf**; // кількість матриць
- Конструктори: конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);

- конструктор із трьома параметрами - розміри вектора та значення ініціалізації;

- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість матриць даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертають розмірність матриці (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).
- Індексатори:
  - з двома індексами, які відповідають індексам масиву;
  - з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
    Якщо значення індексів невірне в поле **codeError** записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .
- Перевантаження:
  - унарних операції **++** (**--**): одночасно збільшує (зменшує) значення елементів масиву на 1;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **n, m** не дорівнюють – нулю, або всі елементи масиву не рівні – нулю, інакше **false**;
  - унарної логічної операції **!** (заперечення): повертає значення **true**, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше **false**;
  - унарної побітової операції **~** (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;
  - арифметичних бінарні операці
    a. **+** додавання:
       i. для двох матриць
       ii. для матриці та скаляра типу **double;**

b. `-` (віднімання):
   i. для двох матриць
   ii. для матриці та скаляра типу `double`;
c. `*`(множення)
   i. для двох матриць,
   ii. для матриці та вектора `VectorDouble` ,
   iii. для матриці та скаляра типу `double`;
d. `/`(ділення)
   i. для двох матриць
   ii. для матриці та скаляра типу `double`;
e. `%`(остача від ділення)
   i. для двох матриць
   ii. для матриці та скаляра типу `uint`;

- побітові бінарні операції (виконувати операції на кодами символьного представлення дійсного числа)
  a. `|` (побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `double`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `double`;
  c. `|` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `double`;
  d. `>>` (побітовий зсув право)
     i. для двох матриць
     ii. для матриці та скаляра типу `ushort`;
  e. `<<` (побітовий зсув ліво)
     i. для двох матриць
     ii. для матриці та скаляра типу `ushort`;
- 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.

3.10. Створити клас **MatrixByte** (матриця цілих чисел). Розробити такі елементи класу:

- Поля (захищені):
  - **byte [,] ByteArray**; // масив
  - **uint n,m**; // розміри матриці
  - **int codeError**; // код помилки
  - **static int num_vec**; // кількість матриць
- Конструктори:
  - конструктор без параметрів(виділяє місце для одного елемента та ініціалізує його в нуль);
  - конструктор із двома параметрами - розміри вектора (виділяє місце та ініціалізує значенням нуль);
  - конструктор із трьома параметрами - розміри вектора та значення ініціалізації;
- Деструктор (виводить повідомлення в консоль).
- Методи, що дозволяють:
  - ввести елементи вектора з клавіатури;
  - вивести елементи вектора на екран;
  - присвоєння елементам масиву вектора деякого значення, яке задається як параметр;
  - статичний метод, що підраховує кількість матриць даного типу;
  - присвоїти елементам масиву деяке значення (параметр);
- Властивості:
  - повертають розмірність матриці (доступні лише для читання);
  - дозволяє отримати-встановити значення поля **codeError** (доступні для читання і запису).
- Індексатори:
  - з двома індексами, які відповідають індексам масиву;
  - з одним індексом, що дозволяє звертатися за індексом **k** до двомірного масиву **(k = i*m + j)**;
    Якщо значення індексів невірне в поле **codeError** записується -1 (при читанні повертається значення – **0**, при записі – запис здійснюється тільки в поле codeError); .
- Перевантаження:
  - унарних операції **++** (**--**): одночасно збільшує (зменшує) значення елементів масиву на 1;
  - сталих **true** і **false**: звертання до екземпляра класу дає значення **true**, якщо **n, m** не дорівнюють – нулю,

або всі елементи масиву не рівні – нулю, інакше `false`;

- унарної логічної операції `!` (заперечення): повертає значення `true`, якщо елементи якщо **n, m** не дорівнюють – нулю, інакше `false`;
- унарної побітової операції `~` (заперечення): повертає побітове заперечення для всіх елементів масиву в класі матриця;
- арифметичних бінарні операці
  a. `+` додавання:
     i. для двох матриць
     ii. для матриці та скаляра типу `byte;`
  b. `-` (віднімання):
     i. для двох матриць
     ii. для матриці та скаляра типу `byte`;
  c. `*`(множення)
     i. для двох матриць,
     ii. для матриці та вектора `VectorByte` ,
     iii. для матриці та скаляра типу `byte`;
  d. `/`(ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `byte`;
  e. `%`(остача від ділення)
     i. для двох матриць
     ii. для матриці та скаляра типу `byte`;
- побітові бінарні операції
  a. `|` (побітове додавання)
     i. для двох матриць
     ii. для матриці та скаляра типу `byte`;
  b. `^` (побітове додавання за модулем 2)
     i. для двох матриць
     ii. для матриці та скаляра типу `byte`;
  c. `|` (побітове множення)
     i. двох векторів
     ii. вектора і скаляра типу `byte`;
  d. `>>` (побітовий зсув право)
     i. для двох матриць
     ii. для матриці та скаляра типу `sbyte`;
  e. `<<` (побітовий зсув ліво)
     i. для двох матриць

ii. для матриці та скаляра типу `sbyte`;

- 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.