**Виконав: Мельник Б. В.; КН-922в**

**Лабораторна робота №2**

**Тема:** Використання масивів і колекцій у мові C#.

**Мета роботи:** набути навички в написанні програм з використання масивів і колекцій.

**Індивідуальні завдання**

**Варіант №12**

**Завдання 1**

Використовуючи завдання власного варіанта до лабораторної роботи 8 з електронних методичних вказівок з основ програмування та алгоритмічних мов створити проект з реалізації цих завдань засобами мови C#.

***Завдання 8 лабораторної роботи:*** Треба оголосити масив цілих чисел і заповнити його випадковими значеннями. Розмір масиву і діапазон значень його елементів задані у виданому варіанті індивідуального завдання (див. п.6). В індивідуальних завданнях вказано також, яку обробку масиву слід провести.

Для всіх варіантів завдання слід мати на увазі наступне:

* нуль вважається додатним числом, якщо в завданні не обумовлений якийсь інший його статус;
* коли мова йде про якісь послідовності чисел, мається на увазі послідовність з довжиною, більшою одиниці;
* у тих випадках, коли завдання вимагає виконання якихось обчислень, дозволяється виконувати їх з тією точністю, яку забезпечують операції цілочисельної арифметики.

***Індивідуальне завдання:***

|  |  |  |  |
| --- | --- | --- | --- |
| **№ варіанту** | **Розмірність масиву** | **Діапазон** | **Що потрібно зробити** |
| 12 | 200 | -100 — 100 | У всіх послідовностях додатних чисел замінити значення елементів з максимальним і мінімальним значенням на середнє для цієї послідовності |

***Текст програми:***

using System; // Підключення простору імен System

class Task\_1 // Оголошення класу Task\_1

{

static void Main() // Головний метод програми

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для вводу

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для виводу

int[] array = new int[200]; // Оголошення масиву з 200 елементів

Random rand = new Random(); // Створення об'єкта для генерації випадкових чисел

// Заповнення масиву випадковими значеннями від -100 до 100

for (int i = 0; i < array.Length; i++)

{

array[i] = rand.Next(-100, 101); // Присвоєння випадкового значення елементу масиву

}

Console.WriteLine("Початковий масив:"); // Виведення повідомлення

PrintArray(array); // Виклик методу для виведення масиву

// Обробка масиву

ProcessArray(array); // Виклик методу для обробки масиву

Console.WriteLine("\nМасив після обробки:"); // Виведення повідомлення

PrintArray(array); // Виклик методу для виведення масиву

Console.ReadKey(); // Очікування натискання клавіші

}

static void ProcessArray(int[] array) // Метод для обробки масиву

{

int start = -1; // Початковий індекс послідовності невід'ємних чисел

int end = -1; // Кінцевий індекс послідовності невід'ємних чисел

int sum = 0; // Сума невід'ємних чисел

int count = 0; // Кількість невід'ємних чисел

for (int i = 0; i < array.Length; i++) // Прохід по всіх елементах масиву

{

if (array[i] >= 0) // Якщо елемент невід'ємний

{

if (start == -1) // Якщо це початок нової послідовності

{

start = i; // Встановлення початкового індексу

}

end = i; // Оновлення кінцевого індексу

sum += array[i]; // Додавання елемента до суми

count++; // Збільшення лічильника

}

else // Якщо елемент від'ємний

{

if (count > 1) // Якщо в послідовності більше одного елемента

{

int average = sum / count; // Обчислення середнього значення

int maxIndex = start; // Індекс максимального елемента

int minIndex = start; // Індекс мінімального елемента

for (int j = start; j <= end; j++) // Прохід по послідовності

{

if (array[j] > array[maxIndex]) // Якщо елемент більший за поточний максимум

{

maxIndex = j; // Оновлення індексу максимуму

}

if (array[j] < array[minIndex]) // Якщо елемент менший за поточний мінімум

{

minIndex = j; // Оновлення індексу мінімуму

}

}

array[maxIndex] = average; // Заміна максимального елемента на середнє значення

array[minIndex] = average; // Заміна мінімального елемента на середнє значення

}

start = -1; // Скидання початкового індексу

end = -1; // Скидання кінцевого індексу

sum = 0; // Скидання суми

count = 0; // Скидання лічильника

}

}

// Обробка останньої послідовності, якщо вона існує

if (count > 1) // Якщо в послідовності більше одного елемента

{

int average = sum / count; // Обчислення середнього значення

int maxIndex = start; // Індекс максимального елемента

int minIndex = start; // Індекс мінімального елемента

for (int j = start; j <= end; j++) // Прохід по послідовності

{

if (array[j] > array[maxIndex]) // Якщо елемент більший за поточний максимум

{

maxIndex = j; // Оновлення індексу максимуму

}

if (array[j] < array[minIndex]) // Якщо елемент менший за поточний мінімум

{

minIndex = j; // Оновлення індексу мінімуму

}

}

array[maxIndex] = average; // Заміна максимального елемента на середнє значення

array[minIndex] = average; // Заміна мінімального елемента на середнє значення

}

}

static void PrintArray(int[] array) // Метод для виведення масиву

{

foreach (int value in array) // Прохід по всіх елементах масиву

{

Console.Write(value + " "); // Виведення елемента

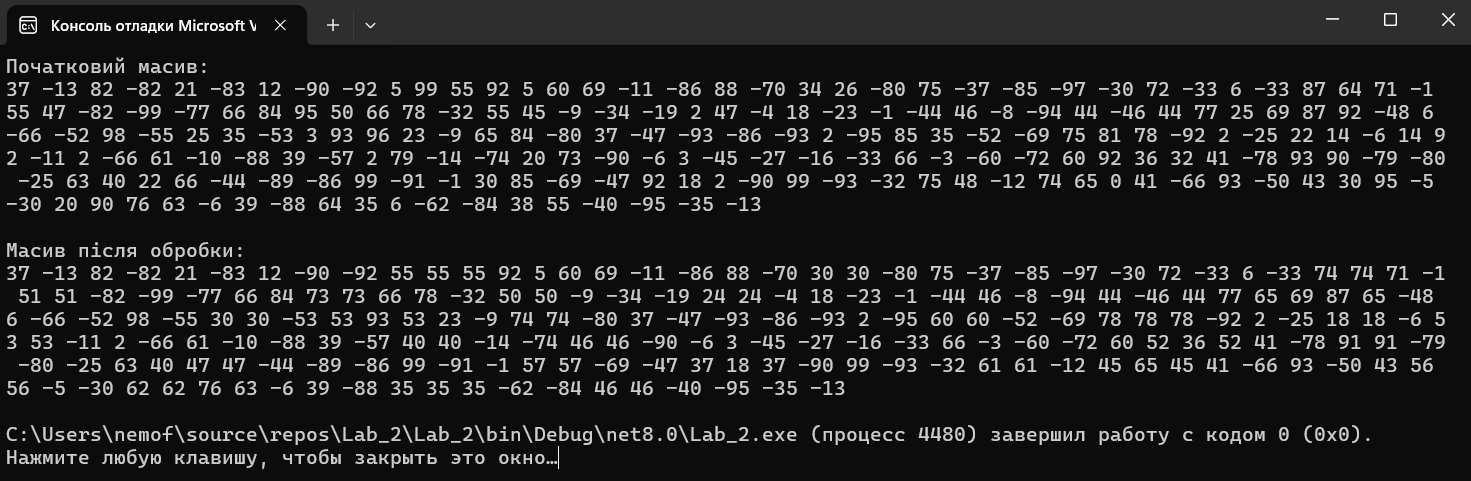
}

Console.WriteLine(); // Перехід на новий рядок

}

}

***Результат роботи програми:***



**Алгоритм побудови коду:**

1. **Оголошення масиву**: Створюється масив із 200 елементів, які заповнюються випадковими числами в діапазоні від -100 до 100.
2. **Обробка масиву**:
   * Програма проходить по масиву та виявляє послідовності невід'ємних чисел.
   * Для кожної послідовності з більш ніж одного елемента обчислюється середнє значення.
   * Максимальний і мінімальний елементи цієї послідовності замінюються середнім значенням.
3. **Виведення результатів**: Програма виводить початковий та оброблений масиви.

**Функціональність:**

* **Метод ProcessArray** виконує обробку масиву: знаходить послідовності невід'ємних чисел, обчислює середнє значення, а також замінює максимальний і мінімальний елементи в послідовності на середнє.
* **Метод PrintArray** виводить масив на екран.
* Програма працює з кодуванням UTF-8, що дозволяє коректно відображати кириличні символи в консолі.

**Висновок:**

Програма успішно реалізує обробку масиву та виконує всі необхідні операції згідно з завданням. Вона демонструє вміння працювати з масивами, обчислювати середнє значення та використовувати умовні оператори для обробки послідовностей даних.

**Завдання 2\***

Виконати минуле завдання, зробивши похідний масив динамічним, задавши його розмір з консолі.***Текст програми:***

using System; // Підключення простору імен System

class Task\_2 // Оголошення класу Task\_2

{

static void Main() // Головний метод програми

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування вводу UTF-8

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування виводу UTF-8

Console.Write("Введіть розмір масиву: "); // Вивід запиту на введення розміру масиву

int size = int.Parse(Console.ReadLine()); // Зчитування розміру масиву з консолі

int[] array = new int[size]; // Створення масиву заданого розміру

Random rand = new Random(); // Створення об'єкта для генерації випадкових чисел

// Заповнення масиву випадковими значеннями від -100 до 100

for (int i = 0; i < array.Length; i++)

{

array[i] = rand.Next(-100, 101); // Присвоєння випадкового значення елементу масиву

}

Console.WriteLine("Початковий масив:"); // Вивід повідомлення про початковий масив

PrintArray(array); // Виклик методу для виводу масиву

// Обробка масиву

ProcessArray(array); // Виклик методу для обробки масиву

Console.WriteLine("\nМасив після обробки:"); // Вивід повідомлення про оброблений масив

PrintArray(array); // Виклик методу для виводу масиву

Console.ReadKey(); // Очікування натискання клавіші перед завершенням програми

}

static void ProcessArray(int[] array) // Метод для обробки масиву

{

int start = -1; // Початковий індекс послідовності невід'ємних чисел

int end = -1; // Кінцевий індекс послідовності невід'ємних чисел

int sum = 0; // Сума невід'ємних чисел у послідовності

int count = 0; // Кількість невід'ємних чисел у послідовності

for (int i = 0; i < array.Length; i++) // Прохід по всіх елементах масиву

{

if (array[i] >= 0) // Якщо елемент невід'ємний

{

if (start == -1) // Якщо це початок нової послідовності

{

start = i; // Встановлення початкового індексу

}

end = i; // Оновлення кінцевого індексу

sum += array[i]; // Додавання елемента до суми

count++; // Збільшення лічильника елементів

}

else // Якщо елемент від'ємний

{

if (count > 1) // Якщо в послідовності більше одного елемента

{

int average = sum / count; // Обчислення середнього значення

int maxIndex = start; // Індекс максимального елемента

int minIndex = start; // Індекс мінімального елемента

for (int j = start; j <= end; j++) // Прохід по послідовності

{

if (array[j] > array[maxIndex]) // Пошук максимального елемента

{

maxIndex = j;

}

if (array[j] < array[minIndex]) // Пошук мінімального елемента

{

minIndex = j;

}

}

array[maxIndex] = average; // Замінити максимальний елемент на середнє значення

array[minIndex] = average; // Замінити мінімальний елемент на середнє значення

}

start = -1; // Скидання початкового індексу

end = -1; // Скидання кінцевого індексу

sum = 0; // Скидання суми

count = 0; // Скидання лічильника

}

}

// Обробка останньої послідовності, якщо вона існує

if (count > 1)

{

int average = sum / count; // Обчислення середнього значення

int maxIndex = start; // Індекс максимального елемента

int minIndex = start; // Індекс мінімального елемента

for (int j = start; j <= end; j++) // Прохід по послідовності

{

if (array[j] > array[maxIndex]) // Пошук максимального елемента

{

maxIndex = j;

}

if (array[j] < array[minIndex]) // Пошук мінімального елемента

{

minIndex = j;

}

}

array[maxIndex] = average; // Замінити максимальний елемент на середнє значення

array[minIndex] = average; // Замінити мінімальний елемент на середнє значення

}

}

static void PrintArray(int[] array) // Метод для виводу масиву

{

foreach (int value in array) // Прохід по всіх елементах масиву

{

Console.Write(value + " "); // Вивід елемента масиву

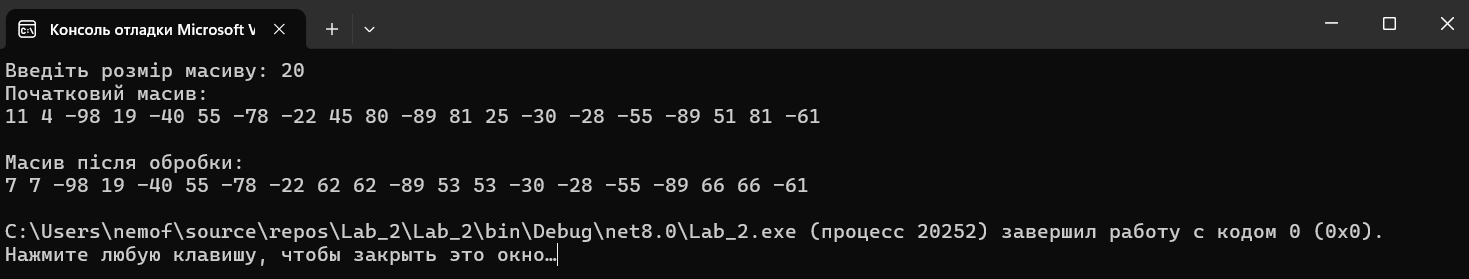
}

Console.WriteLine(); // Перехід на новий рядок

}

}

***Результат роботи програми:***



**Опис алгоритму**

1. **Введення даних**: Програма зчитує з консолі розмір масиву та створює масив випадкових чисел заданого розміру. Використовується клас Random для генерації чисел в діапазоні від -100 до 100.
2. **Заповнення масиву**: За допомогою циклу масив заповнюється випадковими числами. Потім він виводиться на екран для ознайомлення.
3. **Обробка масиву**: Метод ProcessArray шукає послідовності невід'ємних чисел в масиві. Для кожної послідовності:
   * Визначаються її початковий та кінцевий індекси.
   * Обчислюється сума та кількість елементів.
   * Якщо кількість невід'ємних чисел в послідовності більша за 1, програма знаходить максимальний і мінімальний елементи та замінює їх на середнє арифметичне цієї послідовності.
4. **Результат**: Після обробки масив знову виводиться на екран, де видно зміни.
5. **Методи**:
   * PrintArray: Використовується для виводу елементів масиву на консоль.
   * ProcessArray: Виконує обробку масиву, замінюючи максимальні та мінімальні елементи послідовностей на середнє арифметичне.

**Функціональність**

* Програма дозволяє користувачу задати розмір масиву, заповнити його випадковими числами та обробити послідовності невід'ємних чисел.
* В обробці враховуються лише ті послідовності невід'ємних чисел, що містять більше одного елемента.
* Максимальні та мінімальні елементи в кожній послідовності замінюються на середнє арифметичне цієї послідовності.

**Висновок**

Програма продемонструвала роботу з масивами динамічного розміру та застосування методів для обробки даних. Алгоритм коректно визначає послідовності невід'ємних чисел та виконує необхідні заміни.

**Завдання 3\***

Виконати минуле завдання, використовуючи кольорові можливості консолі для покращення інтерфейсу роботи програми. Наприклад, виділяти різними кольорами знайдені в масивах за завданням елементи чи їхні послідовності, розраховані параметри, тощо.

***Текст програми:***

using System; // Підключення простору імен System

class Task\_3 // Оголошення класу Task\_3

{

static void Main() // Головний метод програми

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для введення

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для виведення

Console.Write("Введіть розмір масиву: "); // Виведення запиту на введення розміру масиву

int size = int.Parse(Console.ReadLine()); // Зчитування розміру масиву з консолі

int[] array = new int[size]; // Оголошення масиву заданого розміру

Random rand = new Random(); // Створення об'єкта для генерації випадкових чисел

// Заповнення масиву випадковими значеннями від -100 до 100

for (int i = 0; i < array.Length; i++)

{

array[i] = rand.Next(-100, 101); // Присвоєння випадкового значення елементу масиву

}

Console.WriteLine("Початковий масив:"); // Виведення повідомлення про початковий масив

PrintArray(array); // Виклик методу для виведення масиву

// Обробка масиву

ProcessArray(array); // Виклик методу для обробки масиву

Console.WriteLine("\nМасив після обробки:"); // Виведення повідомлення про оброблений масив

PrintArray(array); // Виклик методу для виведення масиву

Console.ReadKey(); // Очікування натискання клавіші перед завершенням програми

}

static void ProcessArray(int[] array) // Метод для обробки масиву

{

int start = -1; // Початковий індекс послідовності невід'ємних чисел

int end = -1; // Кінцевий індекс послідовності невід'ємних чисел

int sum = 0; // Сума невід'ємних чисел у послідовності

int count = 0; // Кількість невід'ємних чисел у послідовності

for (int i = 0; i < array.Length; i++) // Прохід по всіх елементах масиву

{

if (array[i] >= 0) // Перевірка, чи є елемент невід'ємним

{

if (start == -1) // Якщо це початок нової послідовності

{

start = i; // Встановлення початкового індексу

}

end = i; // Оновлення кінцевого індексу

sum += array[i]; // Додавання елемента до суми

count++; // Збільшення лічильника

}

else // Якщо елемент від'ємний

{

if (count > 1) // Якщо в послідовності більше одного елемента

{

int average = sum / count; // Обчислення середнього значення

int maxIndex = start; // Індекс максимального елемента

int minIndex = start; // Індекс мінімального елемента

for (int j = start; j <= end; j++) // Прохід по послідовності

{

if (array[j] > array[maxIndex]) // Пошук максимального елемента

{

maxIndex = j;

}

if (array[j] < array[minIndex]) // Пошук мінімального елемента

{

minIndex = j;

}

}

HighlightArray(array, start, end, maxIndex, minIndex, average); // Виділення послідовності

array[maxIndex] = average; // Замінити максимальний елемент на середнє значення

array[minIndex] = average; // Замінити мінімальний елемент на середнє значення

}

start = -1; // Скидання початкового індексу

end = -1; // Скидання кінцевого індексу

sum = 0; // Скидання суми

count = 0; // Скидання лічильника

}

}

// Обробка останньої послідовності, якщо вона існує

if (count > 1)

{

int average = sum / count; // Обчислення середнього значення

int maxIndex = start; // Індекс максимального елемента

int minIndex = start; // Індекс мінімального елемента

for (int j = start; j <= end; j++) // Прохід по послідовності

{

if (array[j] > array[maxIndex]) // Пошук максимального елемента

{

maxIndex = j;

}

if (array[j] < array[minIndex]) // Пошук мінімального елемента

{

minIndex = j;

}

}

HighlightArray(array, start, end, maxIndex, minIndex, average); // Виділення послідовності

array[maxIndex] = average; // Замінити максимальний елемент на середнє значення

array[minIndex] = average; // Замінити мінімальний елемент на середнє значення

}

}

static void HighlightArray(int[] array, int start, int end, int maxIndex, int minIndex, int average) // Метод для виділення послідовності

{

Console.WriteLine("\nОбробка послідовності:"); // Виведення повідомлення про обробку послідовності

for (int i = start; i <= end; i++) // Прохід по послідовності

{

if (i == maxIndex || i == minIndex) // Якщо елемент є максимальним або мінімальним

{

Console.ForegroundColor = ConsoleColor.Red; // Встановлення червоного кольору для виведення

}

else

{

Console.ForegroundColor = ConsoleColor.Green; // Встановлення зеленого кольору для виведення

}

Console.Write(array[i] + " "); // Виведення елемента

}

Console.ResetColor(); // Скидання кольору

Console.ForegroundColor = ConsoleColor.Yellow; // Встановлення жовтого кольору для виведення

Console.WriteLine($"\nСереднє значення: {average}"); // Виведення середнього значення

Console.ResetColor(); // Скидання кольору

}

static void PrintArray(int[] array) // Метод для виведення масиву

{

foreach (int value in array) // Прохід по всіх елементах масиву

{

Console.Write(value + " "); // Виведення елемента

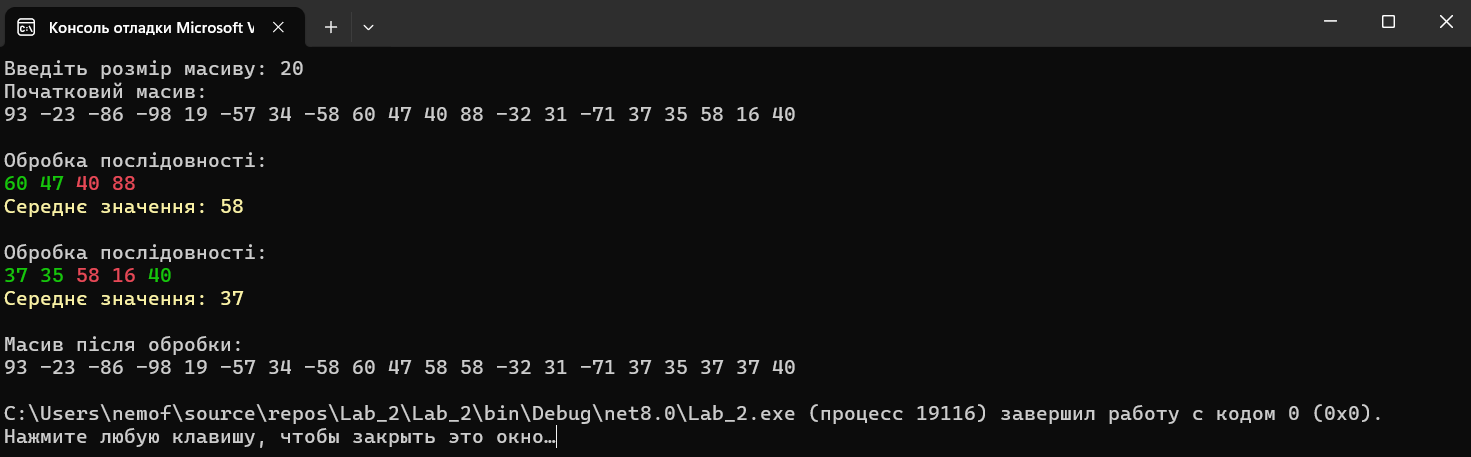
}

Console.WriteLine(); // Перехід на новий рядок

}

}

***Результат роботи програми:***



**Алгоритм побудови коду:**

1. **Ініціалізація програми:**
   * Встановлюємо правильне кодування для введення та виведення у консолі (UTF-8).
   * Запитуємо у користувача розмір масиву та створюємо масив випадкових цілих чисел в діапазоні від -100 до 100.
2. **Обробка масиву:**
   * Програма знаходить послідовності невід'ємних чисел у масиві. Для кожної такої послідовності обчислюється сума і кількість елементів.
   * Визначаються індекси мінімального та максимального елементів послідовності.
   * Якщо послідовність містить більше одного елемента, то максимальний та мінімальний елементи замінюються середнім значенням цієї послідовності.
3. **Кольорове оформлення виводу:**
   * Максимальні та мінімальні елементи виділяються червоним кольором.
   * Інші елементи послідовності виділяються зеленим.
   * Середнє значення для кожної послідовності виводиться жовтим кольором.
4. **Закінчення програми:**
   * Після обробки та виведення модифікованого масиву, програма чекає на натискання будь-якої клавіші для завершення.

**Функціональність:**

Програма дозволяє:

* Генерувати випадковий масив із цілих чисел.
* Виявляти та обробляти послідовності невід'ємних чисел у масиві.
* Замінювати максимальні та мінімальні елементи послідовності на її середнє значення.
* Виводити оброблені послідовності з кольоровими акцентами для зручності візуального сприйняття.

**Висновок:**

Завдяки використанню кольорових можливостей консолі програма стає більш інтерактивною та зручною для користувача. Кольорове виділення дозволяє швидко ідентифікувати важливі елементи та результати обробки масиву. Це підвищує інформативність та покращує користувацький досвід.

**Завдання 4\***

Виконати минуле завдання, використовуючи замість масивів одну з колекцій мови C# (List, LinkedList, ArrayList, чи ін.).

***Текст програми:***

using System; // Підключення простору імен System

using System.Collections.Generic; // Підключення простору імен для роботи з колекціями

class Task\_4 // Оголошення класу Task\_4

{

static void Main() // Головний метод програми

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для введення

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для виведення

Console.Write("Введіть розмір масиву: "); // Запит розміру масиву у користувача

int size = int.Parse(Console.ReadLine()); // Зчитування розміру масиву з консолі

List<int> list = new List<int>(); // Створення списку цілих чисел

Random rand = new Random(); // Створення об'єкта для генерації випадкових чисел

// Заповнення списку випадковими значеннями від -100 до 100

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

{

list.Add(rand.Next(-100, 101)); // Додавання випадкового числа до списку

}

Console.WriteLine("Початковий список:"); // Виведення початкового списку

PrintList(list); // Виклик методу для виведення списку

// Обробка списку

ProcessList(list); // Виклик методу для обробки списку

Console.WriteLine("\nСписок після обробки:"); // Виведення списку після обробки

PrintList(list); // Виклик методу для виведення списку

Console.ReadKey(); // Очікування натискання клавіші перед завершенням програми

}

static void ProcessList(List<int> list) // Метод для обробки списку

{

int start = -1; // Початковий індекс послідовності невід'ємних чисел

int end = -1; // Кінцевий індекс послідовності невід'ємних чисел

int sum = 0; // Сума невід'ємних чисел у послідовності

int count = 0; // Кількість невід'ємних чисел у послідовності

for (int i = 0; i < list.Count; i++) // Проходження по всіх елементах списку

{

if (list[i] >= 0) // Якщо елемент невід'ємний

{

if (start == -1) // Якщо це початок нової послідовності

{

start = i; // Встановлення початкового індексу

}

end = i; // Оновлення кінцевого індексу

sum += list[i]; // Додавання елемента до суми

count++; // Збільшення лічильника елементів

}

else // Якщо елемент від'ємний

{

if (count > 1) // Якщо в послідовності більше одного елемента

{

int average = sum / count; // Обчислення середнього значення

int maxIndex = start; // Індекс максимального елемента

int minIndex = start; // Індекс мінімального елемента

for (int j = start; j <= end; j++) // Проходження по послідовності

{

if (list[j] > list[maxIndex]) // Якщо елемент більший за поточний максимум

{

maxIndex = j; // Оновлення індексу максимального елемента

}

if (list[j] < list[minIndex]) // Якщо елемент менший за поточний мінімум

{

minIndex = j; // Оновлення індексу мінімального елемента

}

}

HighlightList(list, start, end, maxIndex, minIndex, average); // Виклик методу для виділення послідовності

list[maxIndex] = average; // Замінити максимальний елемент на середнє значення

list[minIndex] = average; // Замінити мінімальний елемент на середнє значення

}

start = -1; // Скидання початкового індексу

end = -1; // Скидання кінцевого індексу

sum = 0; // Скидання суми

count = 0; // Скидання лічильника

}

}

// Обробка останньої послідовності, якщо вона існує

if (count > 1)

{

int average = sum / count; // Обчислення середнього значення

int maxIndex = start; // Індекс максимального елемента

int minIndex = start; // Індекс мінімального елемента

for (int j = start; j <= end; j++) // Проходження по послідовності

{

if (list[j] > list[maxIndex]) // Якщо елемент більший за поточний максимум

{

maxIndex = j; // Оновлення індексу максимального елемента

}

if (list[j] < list[minIndex]) // Якщо елемент менший за поточний мінімум

{

minIndex = j; // Оновлення індексу мінімального елемента

}

}

HighlightList(list, start, end, maxIndex, minIndex, average); // Виклик методу для виділення послідовності

list[maxIndex] = average; // Замінити максимальний елемент на середнє значення

list[minIndex] = average; // Замінити мінімальний елемент на середнє значення

}

}

static void HighlightList(List<int> list, int start, int end, int maxIndex, int minIndex, int average) // Метод для виділення послідовності

{

Console.WriteLine("\nОбробка послідовності:"); // Виведення повідомлення про обробку

for (int i = start; i <= end; i++) // Проходження по послідовності

{

if (i == maxIndex || i == minIndex) // Якщо елемент є максимальним або мінімальним

{

Console.ForegroundColor = ConsoleColor.Red; // Встановлення червоного кольору для виведення

}

else

{

Console.ForegroundColor = ConsoleColor.Green; // Встановлення зеленого кольору для виведення

}

Console.Write(list[i] + " "); // Виведення елемента

}

Console.ResetColor(); // Скидання кольору

Console.ForegroundColor = ConsoleColor.Yellow; // Встановлення жовтого кольору для виведення

Console.WriteLine($"\nСереднє значення: {average}"); // Виведення середнього значення

Console.ResetColor(); // Скидання кольору

}

static void PrintList(List<int> list) // Метод для виведення списку

{

foreach (int value in list) // Проходження по всіх елементах списку

{

Console.Write(value + " "); // Виведення елемента

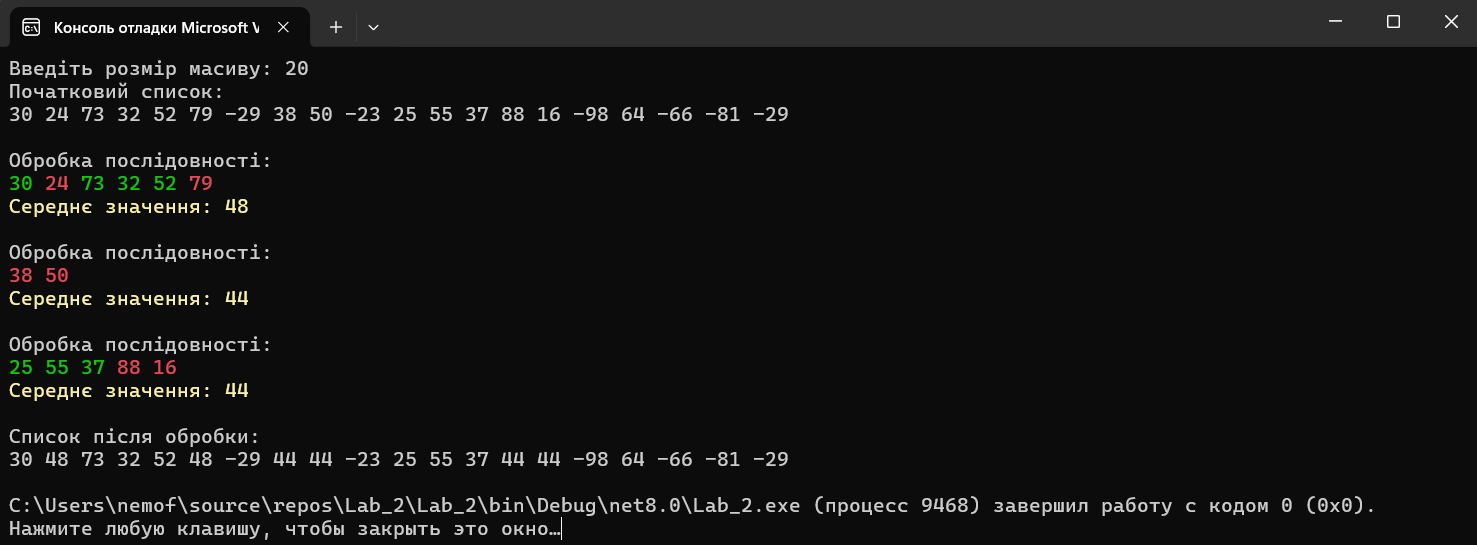
}

Console.WriteLine(); // Перехід на новий рядок

}

}

***Результат роботи програми:***



**Алгоритм побудови коду:**

1. **Ініціалізація кодування консолі**: Установлюємо кодування для правильного відображення кирилиці в консолі.
2. **Запит розміру списку у користувача**: Вводимо розмір списку.
3. **Генерація списку випадкових значень**: Використовуємо клас List<int> для створення списку, заповнюючи його випадковими цілими числами в діапазоні від -100 до 100.
4. **Виведення початкового списку**: Виводимо згенерований список на екран.
5. **Обробка списку**:
   * Шукаємо послідовності з двох і більше невід'ємних чисел.
   * Для кожної послідовності обчислюємо середнє значення, після чого міняємо місцями максимальний і мінімальний елемент послідовності з цим значенням.
   * Використовується метод HighlightList для виділення елементів послідовності кольором.
6. **Виведення обробленого списку**: Показуємо список після обробки.
7. **Завершення роботи програми**: Очікуємо натискання клавіші перед закриттям програми.

**Функціональність:**

Програма:

* Створює список випадкових цілих чисел.
* Шукає послідовності невід'ємних чисел у списку.
* Обчислює середнє значення для кожної послідовності, та замінює ним максимальні і мінімальні елементи цієї послідовності.
* Виводить початковий та оброблений список, виділяючи кольором максимальні та мінімальні елементи перед заміною.

**Висновок:**

Програма демонструє роботу з колекціями на прикладі списку (List<int>) у C#. Вона ефективно обробляє послідовності невід'ємних чисел, замінюючи максимальні та мінімальні елементи на середнє значення. Такий підхід зручніший і більш гнучкий у порівнянні з масивами, оскільки колекції забезпечують автоматичне керування розміром та додаткові можливості для обробки даних.

**Завдання 5**

Використовуючи завдання власного варіанта до лабораторної роботи 9 з електронних методичних вказівок з основ програмування та алгоритмічних мов створити проект з реалізації цих завдань засобами мови C#.

***Завдання лабораторної роботи №9:*** Створити квадратну матрицю розміром 9х9. В індивідуальних завданнях (див. п.6) вказано, який тип даних використати та яку обробку матриці потрібно виконати.

Якщо по умові завдання матрицю слід заповнити випадковими числами, рекомендується вибирати ці числа з діапазону 0 – 99. Якщо за умовою завдання в матрицю слід записати ЛПЧ – лінійну послідовність чисел, мається на увазі послідовність: 1,2,3...

***Індивідуальні завдання:***

|  |  |  |
| --- | --- | --- |
| **№ варіанту** | **Зміст завдання** | **Ілюстрація** |
| 12 | Відобразити верхню половину матриці на нижню дзеркально симетрично щодо горизонтальної осі.  Значення елементів матриці – випадкові числа. |  |

***Текст програми:***

using System; // Підключення простору імен System

class Task\_5 // Оголошення класу Task\_5

{

static void Main()

{

// Встановлення кодування для введення та виведення в консоль

Console.InputEncoding = System.Text.Encoding.UTF8;

Console.OutputEncoding = System.Text.Encoding.UTF8;

int size = 9; // Розмір матриці

int[,] matrix = new int[size, size]; // Оголошення двовимірного масиву

Random rand = new Random(); // Створення об'єкта для генерації випадкових чисел

// Заповнення матриці випадковими числами

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

{

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

{

matrix[i, j] = rand.Next(0, 100); // Присвоєння випадкового числа від 0 до 99

}

}

// Відображення верхньої половини на нижню

for (int i = 0; i < size / 2; i++)

{

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

{

matrix[size - 1 - i, j] = matrix[i, j]; // Копіювання значень з верхньої половини в нижню

}

}

// Виведення матриці

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

{

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

{

Console.Write(matrix[i, j] + "\t"); // Виведення значення елемента матриці з табуляцією

}

Console.WriteLine(); // Перехід на новий рядок після виведення кожного рядка матриці

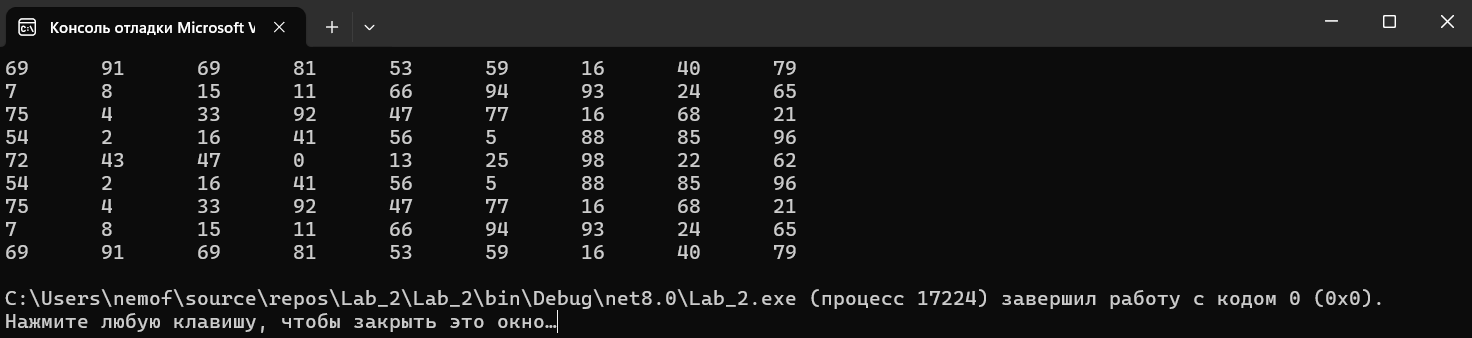
}

Console.ReadKey(); // Очікування натискання клавіші перед завершенням програми

}

}

***Результат роботи програми:***



**Алгоритм побудови коду**

1. **Оголошення класу і підключення простору імен:**
   * Підключено простір імен System для використання основних функцій C#.
   * Оголошено клас Task\_5, який містить метод Main.
2. **Налаштування кодування:**
   * Для підтримки кирилиці встановлено кодування UTF8 для введення та виведення в консолі.
3. **Оголошення змінних:**
   * Створено двовимірний масив matrix розміром 9x9.
   * Об'єкт класу Random використовується для генерації випадкових чисел.
4. **Заповнення матриці:**
   * Цикли for заповнюють кожен елемент матриці випадковими числами від 0 до 99.
5. **Копіювання верхньої половини на нижню:**
   * За допомогою циклу верхня половина матриці копіюється в нижню, забезпечуючи симетрію відносно горизонтальної осі.
6. **Виведення результату:**
   * Матриця виводиться на екран із табуляцією для кожного елемента, кожен рядок виводиться на нову лінію.
7. **Очікування завершення програми:**
   * Програма очікує натискання клавіші перед завершенням, щоб дати користувачу час переглянути результат.

**Функціональність**

* Програма генерує квадратну матрицю випадкових чисел, де елементи верхньої половини симетрично відображаються на нижню.
* Після цього програма виводить матрицю в зручному форматі з табуляцією для кожного числа, що робить матрицю легкою для візуального сприйняття.

**Висновок**

Дана програма демонструє роботу з двовимірними масивами, використання циклів для обробки даних, а також основи роботи з консоллю. Алгоритм простий та ефективний для задачі симетричного відображення матриці.

**Завдання 6\***

Виконати минуле завдання, зробивши матрицю динамічною, задавши її розміри з консолі, використовуючи для цього прямокутний ([ , ]) двовимірний масив.

***Текст програми:***

using System; // Підключення простору імен System

class Task\_6 // Оголошення класу Task\_6

{

static void Main()

{

// Встановлення кодування для введення та виведення в консоль

Console.InputEncoding = System.Text.Encoding.UTF8;

Console.OutputEncoding = System.Text.Encoding.UTF8;

// Запит розміру матриці у користувача

Console.Write("Введіть розмір матриці: ");

int size = int.Parse(Console.ReadLine());

// Створення двовимірного масиву для матриці

int[,] matrix = new int[size, size];

Random rand = new Random();

// Заповнення матриці випадковими числами

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

{

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

{

matrix[i, j] = rand.Next(0, 100); // Генерація випадкового числа від 0 до 99

}

}

// Відображення верхньої половини матриці на нижню

for (int i = 0; i < size / 2; i++)

{

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

{

matrix[size - 1 - i, j] = matrix[i, j]; // Копіювання значень з верхньої половини в нижню

}

}

// Виведення матриці на екран

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

{

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

{

Console.Write(matrix[i, j] + "\t"); // Виведення елементів матриці з табуляцією

}

Console.WriteLine(); // Перехід на новий рядок після кожного рядка матриці

}

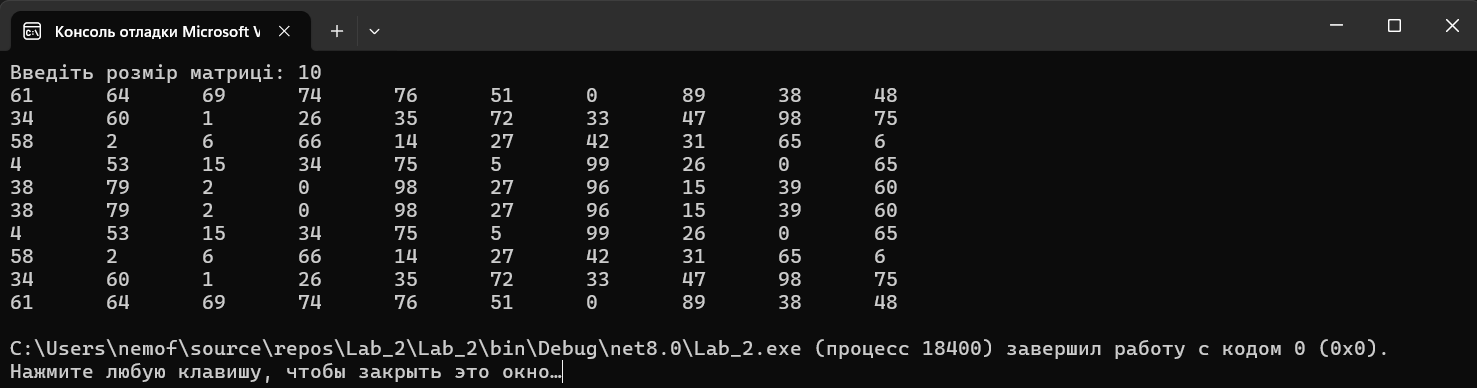
// Очікування натискання клавіші перед завершенням програми

Console.ReadKey();

}

}

***Результат роботи програми:***



**Алгоритм побудови коду:**

1. **Ініціалізація консолі**: Установлюються параметри для правильного введення та виведення інформації з використанням кодування UTF-8.
2. **Запит даних від користувача**: Програма запитує користувача ввести розмір матриці.
3. **Створення двовимірного масиву**: Використовується прямокутний масив типу int[,], де розмір масиву задається введеними даними.
4. **Заповнення матриці випадковими числами**: Для кожного елемента масиву з допомогою генератора випадкових чисел заповнюються значення у межах від 0 до 99.
5. **Відображення верхньої половини на нижню**: Елементи верхньої половини матриці копіюються в нижню так, що відбувається дзеркальне відображення по горизонталі.
6. **Виведення матриці на екран**: Матриця виводиться у вигляді таблиці з використанням табуляції для естетичного вигляду.
7. **Завершення програми**: Програма очікує натискання клавіші перед завершенням роботи.

**Функціональність:**

1. Програма динамічно створює двовимірний масив на основі введеного користувачем розміру.
2. Генерація випадкових чисел для заповнення матриці відбувається з використанням класу Random.
3. Дзеркальне відображення верхньої половини матриці в нижню реалізовано за допомогою двох вкладених циклів.
4. Виведення матриці відбувається з використанням формату табуляції, що робить матрицю читабельною.

**Висновок:**

Ця програма є прикладом використання двовимірних масивів для створення динамічної матриці з дзеркальним відображенням її частин. Основними перевагами є простота введення розміру та генерації випадкових даних, а також ефективне використання циклів для обробки елементів матриці.

**Завдання 7\***

Виконати минуле завдання, зробивши матрицю динамічною, задавши її розміри з консолі, використовуючи для цього зубчастий ([][]) двовимірний масив.

***Текст програми:***

using System; // Підключення простору імен System

class Task\_7 // Оголошення класу Task\_7

{

static void Main() // Головний метод програми

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для введення UTF-8

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для виведення UTF-8

Console.Write("Введіть розмір матриці: "); // Виведення запиту на введення розміру матриці

int size = int.Parse(Console.ReadLine()); // Зчитування розміру матриці з консолі та перетворення в ціле число

int[][] matrix = new int[size][]; // Оголошення зубчастого масиву (матриці)

Random rand = new Random(); // Створення об'єкта для генерації випадкових чисел

// Ініціалізація зубчастого масиву

for (int i = 0; i < size; i++) // Цикл для ініціалізації кожного рядка матриці

{

matrix[i] = new int[size]; // Ініціалізація рядка матриці

}

// Заповнення матриці випадковими числами

for (int i = 0; i < size; i++) // Цикл для проходження по рядках матриці

{

for (int j = 0; j < size; j++) // Цикл для проходження по стовпцях матриці

{

matrix[i][j] = rand.Next(0, 100); // Заповнення елемента матриці випадковим числом від 0 до 99

}

}

// Відображення верхньої половини на нижню

for (int i = 0; i < size / 2; i++) // Цикл для проходження по верхній половині матриці

{

for (int j = 0; j < size; j++) // Цикл для проходження по стовпцях матриці

{

matrix[size - 1 - i][j] = matrix[i][j]; // Копіювання елементів верхньої половини в нижню

}

}

// Виведення матриці

for (int i = 0; i < size; i++) // Цикл для проходження по рядках матриці

{

for (int j = 0; j < size; j++) // Цикл для проходження по стовпцях матриці

{

Console.Write(matrix[i][j] + "\t"); // Виведення елемента матриці з табуляцією

}

Console.WriteLine(); // Перехід на новий рядок після виведення рядка матриці

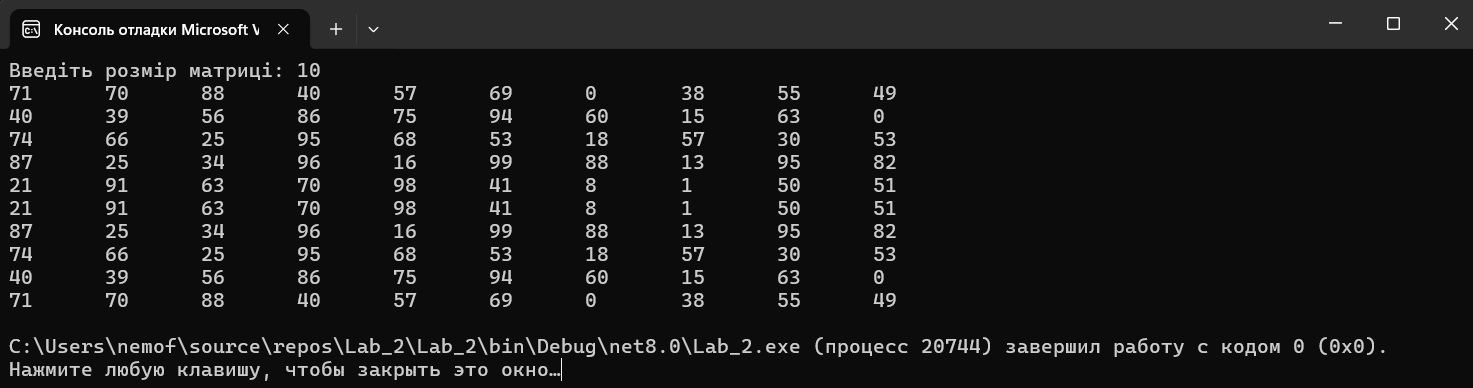
}

Console.ReadKey(); // Очікування натискання клавіші перед завершенням програми

}

}

***Результат роботи програми:***



**Алгоритм побудови коду:**

1. **Підключення простору імен:**
   * Використовується System для базових операцій введення та виведення.
2. **Оголошення класу та головного методу:**
   * Task\_7 — основний клас.
   * Main — головний метод, що виконується при запуску програми.
3. **Налаштування кодування:**
   * Встановлюється кодування UTF-8 для правильного відображення символів в консолі.
4. **Введення розміру матриці:**
   * Через консоль вводиться розмір матриці. Значення зчитується і перетворюється в ціле число.
5. **Оголошення зубчастого масиву:**
   * Створюється динамічна матриця (зубчастий двовимірний масив) на основі введеного розміру.
6. **Ініціалізація масиву:**
   * Кожен рядок масиву ініціалізується як новий масив відповідного розміру.
7. **Заповнення матриці випадковими числами:**
   * Використовується генератор випадкових чисел для заповнення матриці значеннями в діапазоні від 0 до 99.
8. **Копіювання верхньої половини на нижню:**
   * Верхня половина матриці копіюється на нижню для відображення симетричності.
9. **Виведення матриці:**
   * Матриця виводиться в консолі з табуляцією для кожного елемента, забезпечуючи форматований вигляд.
10. **Очікування завершення програми:**
    * Програма чекає на натискання клавіші перед завершенням.

**Функціональність:**

Ця програма дозволяє користувачу створити динамічну матрицю зубчастого типу з розміром, заданим з консолі. Випадкові числа заповнюють матрицю, після чого верхня половина відображається на нижню для отримання симетрії. Виведення відбувається у форматованому вигляді, що робить її зручною для перегляду.

**Висновок:**

Програма демонструє використання зубчастих масивів для створення динамічних структур даних, а також основи роботи з випадковими числами та базовою маніпуляцією матрицями. Вона може бути розширена для більш складних операцій, таких як робота з матрицями різної форми або додатковими функціями для користувача.

**Завдання 8\***

Виконати минуле завдання, використовуючи кольорові можливості консолі для покращення інтерфейсу роботи програми. Наприклад, виділяти різними кольорами знайдені в матриці за завданням елементи чи їхні послідовності, розраховані параметри, тощо.

***Текст програми:***

using System; // Підключення простору імен System

class Task\_8 // Оголошення класу Task\_8

{

static void Main() // Головний метод програми

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування вводу UTF-8

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування виводу UTF-8

Console.Write("Введіть розмір матриці: "); // Запит розміру матриці у користувача

int size = int.Parse(Console.ReadLine()); // Зчитування розміру матриці з консолі

int[][] matrix = new int[size][]; // Оголошення зубчастого масиву

Random rand = new Random(); // Створення об'єкта для генерації випадкових чисел

// Ініціалізація зубчастого масиву

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

{

matrix[i] = new int[size]; // Ініціалізація кожного рядка масиву

}

// Заповнення матриці випадковими числами

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

{

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

{

matrix[i][j] = rand.Next(0, 100); // Заповнення елементів матриці випадковими числами від 0 до 99

}

}

// Відображення верхньої половини на нижню

for (int i = 0; i < size / 2; i++)

{

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

{

matrix[size - 1 - i][j] = matrix[i][j]; // Копіювання верхньої половини матриці на нижню

}

}

// Виведення матриці з кольорами

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

{

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

{

if (i < size / 2)

{

// Верхня половина матриці

Console.ForegroundColor = ConsoleColor.Green; // Встановлення зеленого кольору для верхньої половини

}

else if (i >= size / 2)

{

// Нижня половина матриці

Console.ForegroundColor = ConsoleColor.Yellow; // Встановлення жовтого кольору для нижньої половини

}

Console.Write(matrix[i][j] + "\t"); // Виведення елемента матриці з табуляцією

}

Console.WriteLine(); // Перехід на новий рядок після виведення кожного рядка матриці

Console.ResetColor(); // Скидання кольору до стандартного

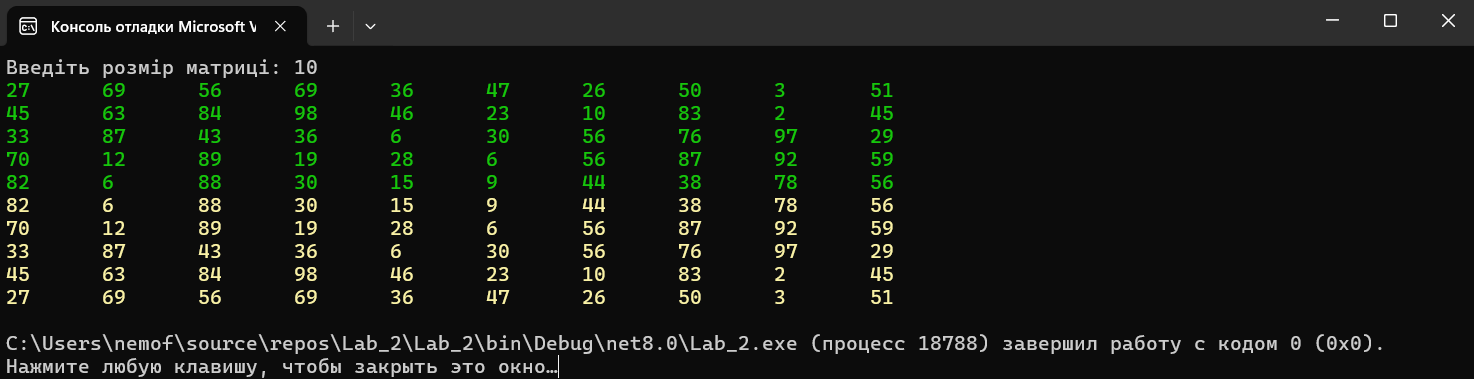
}

Console.ReadKey(); // Очікування натискання клавіші перед завершенням програми

}

}

***Результат роботи програми:***



**Алгоритм побудови коду**

1. **Ініціалізація**:
   * Програма починається з підключення простору імен System для використання основних функцій .NET.
   * Основний метод Main відповідає за виконання коду програми.
   * Налаштовується кодування UTF-8 для правильної роботи з введенням та виведенням даних в консолі.
2. **Введення користувача**:
   * Користувачу пропонується ввести розмір матриці, який зчитується і використовується для подальших обчислень.
3. **Ініціалізація матриці**:
   * Створюється зубчастий масив (матриця), яка ініціалізується випадковими числами від 0 до 99.
   * Верхня половина матриці копіюється на нижню, тобто утворюється симетрична матриця відносно горизонталі.
4. **Кольорове виділення**:
   * Програма використовує можливості консолі для зміни кольору тексту:
     + Елементи верхньої половини матриці відображаються зеленим кольором.
     + Елементи нижньої половини — жовтим кольором.
5. **Виведення матриці**:
   * Матриця виводиться у вигляді таблиці з табуляцією між елементами, де кожен рядок виводиться окремо з відповідним кольоровим оформленням.

**Функціональність**

* Програма генерує симетричну матрицю, де верхня частина копіюється на нижню.
* Застосування кольорів для виділення елементів дозволяє легко розрізняти частини матриці:
  + Зеленим позначається верхня половина.
  + Жовтим — нижня половина.
* Це покращує візуальне сприйняття даних, полегшуючи роботу з програмою і підвищуючи її інтуїтивність.

**Висновок**

Завдання виконано успішно: програма працює коректно, використовуючи кольорові можливості консолі для покращення візуального відображення даних. Це робить інтерфейс більш зрозумілим і зручним для користувача.

**Завдання 9**

Виконати підсумкове завдання згідно із номером свого варіанту

12. Дана цілочислова квадратна матриця. Визначити кількість рядків, елементи яких утворюють арифметичну прогресію.

***Текст програми:***

using System; // Підключення простору імен System

class Task\_9 // Оголошення класу Task\_9

{

static void Main() // Головний метод програми

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для вводу

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для виводу

int[,] matrix = { // Оголошення та ініціалізація двовимірного масиву

{ 1, 3, 5, 7 },

{ 2, 4, 6, 8 },

{ 1, 2, 3, 4 },

{ 10, 20, 30, 40 }

};

int rowCount = matrix.GetLength(0); // Отримання кількості рядків у матриці

int colCount = matrix.GetLength(1); // Отримання кількості стовпців у матриці

int arithmeticRowsCount = 0; // Лічильник рядків, що утворюють арифметичну прогресію

Console.WriteLine("Матриця:"); // Виведення заголовку для матриці

for (int i = 0; i < rowCount; i++) // Цикл по рядках матриці

{

for (int j = 0; j < colCount; j++) // Цикл по стовпцях матриці

{

Console.ForegroundColor = ConsoleColor.Cyan; // Встановлення кольору тексту

Console.Write(matrix[i, j] + "\t"); // Виведення елемента матриці з табуляцією

}

Console.WriteLine(); // Перехід на новий рядок після виведення всіх елементів рядка

}

Console.ResetColor(); // Скидання кольору тексту до стандартного

for (int i = 0; i < rowCount; i++) // Цикл по рядках матриці для перевірки арифметичної прогресії

{

if (IsArithmeticProgression(matrix, i, colCount)) // Перевірка, чи утворює рядок арифметичну прогресію

{

arithmeticRowsCount++; // Збільшення лічильника, якщо рядок утворює арифметичну прогресію

}

}

Console.ForegroundColor = ConsoleColor.Green; // Встановлення кольору тексту

Console.WriteLine($"\nКількість рядків, елементи яких утворюють арифметичну прогресію: {arithmeticRowsCount}"); // Виведення результату

Console.ResetColor(); // Скидання кольору тексту до стандартного

Console.ReadKey(); // Очікування натискання клавіші перед завершенням програми

}

static bool IsArithmeticProgression(int[,] matrix, int row, int colCount) // Метод для перевірки, чи утворює рядок арифметичну прогресію

{

if (colCount < 2) return false; // Якщо кількість стовпців менше 2, повернути false

int difference = matrix[row, 1] - matrix[row, 0]; // Обчислення різниці між першим і другим елементами рядка

for (int j = 1; j < colCount; j++) // Цикл по елементах рядка

{

if (matrix[row, j] - matrix[row, j - 1] != difference) // Якщо різниця між сусідніми елементами не дорівнює початковій різниці

{

return false; // Повернути false, якщо рядок не утворює арифметичну прогресію

}

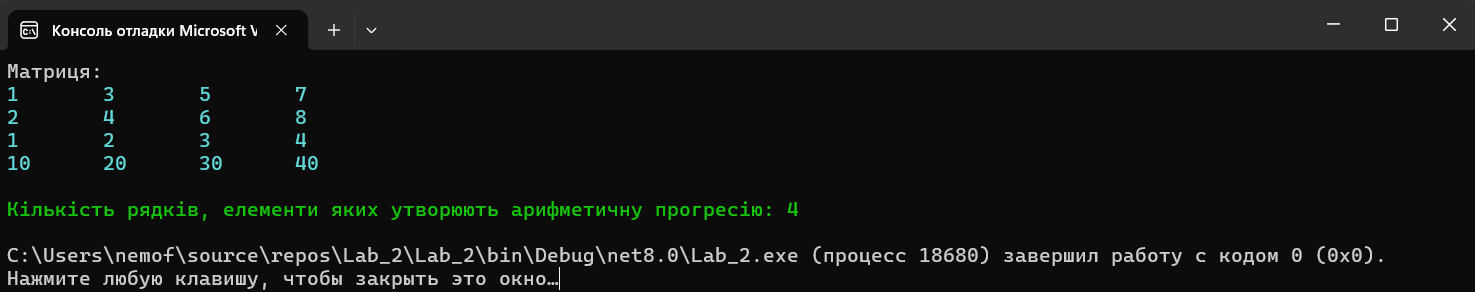
}

return true; // Повернути true, якщо рядок утворює арифметичну прогресію

}

}

***Результат роботи програми:***



**Алгоритм побудови коду:**

1. **Ініціалізація матриці**: Створюється двовимірний масив цілих чисел, який представляє квадратну матрицю. Матриця заповнена значеннями.
2. **Виведення матриці на екран**: Спочатку матриця виводиться на консоль. Для кожного елемента застосовується кольорове форматування для кращої візуалізації.
3. **Перевірка арифметичної прогресії**:
   * Для кожного рядка матриці викликається функція IsArithmeticProgression, яка перевіряє, чи утворюють елементи рядка арифметичну прогресію.
   * У разі позитивного результату збільшується лічильник таких рядків.
4. **Обчислення різниці між елементами**: В основі перевірки лежить обчислення різниці між першим і другим елементами рядка. Далі ця різниця порівнюється з різницями між всіма наступними елементами. Якщо всі різниці однакові, рядок вважається таким, що утворює арифметичну прогресію.
5. **Виведення результату**: Виводиться кількість рядків, що утворюють арифметичну прогресію, з використанням зеленого кольору для виділення результату.

**Функціональність коду:**

* Код обчислює кількість рядків у квадратній матриці, елементи яких утворюють арифметичну прогресію.
* Він перевіряє кожен рядок матриці за допомогою функції IsArithmeticProgression.
* Функція порівнює різниці між елементами, і якщо вони рівні, повертається true, що означає, що рядок є частиною арифметичної прогресії.
* В результаті програма виводить на екран кількість таких рядків.

**Висновок:**

Даний код коректно вирішує поставлене завдання. Він забезпечує візуалізацію матриці та її елементів, а також підраховує кількість рядків, що утворюють арифметичну прогресію.

**Загальний висновок**

В ході лабораторної роботи №2 з використання масивів і колекцій у мові C#, було успішно реалізовано завдання з обробки масивів різного типу. Основні операції включали створення масивів та списків, їх заповнення випадковими значеннями, а також обробку послідовностей елементів відповідно до індивідуальних завдань.

Програми демонструють використання таких основних концепцій як:

* Робота з масивами та колекціями;
* Обчислення середніх значень і заміна елементів масивів;
* Динамічне завдання розмірів масивів;
* Використання кольорового форматування для зручності виводу в консоль.

Алгоритми, розроблені в рамках цієї лабораторної роботи, дозволяють не лише ефективно обробляти масиви, але й застосовувати нові знання для більш складних задач, таких як обробка колекцій і дзеркальне відображення матриць.