### Кам’янець-Подільський національний університет імені Івана Огієнка

### КАФЕДРА КОМП’ЮТЕРНИХ НАУК

### Навчальна дисципліна «Об’єктно-орієнтоване програмування»

### ЛАБОРАТОРНА РОБОТА #0106

## Тема:

**Класи колекцій і протоколи ітерацій**

Варіант №1

**Виконав**:  
студент 1-го курсу  
групи KNms1-B24  
Білик Я. Ю.

**Прийняв**:  
старший викладач,  
Слободянюк О. В.

### Кам’янець-Подільський – 2025

1. **Короткі теоретичні відомості.**

Використання колекцій.

Будь-який набір даних, об'єктів можна назвати колекцією.

Масив з простору імен System.Array - це теж свого роду колекція, але статична, його не можна ні розширити, ні стиснути при необхідності.

.NET Являє собою безліч класів колекцій, що дозволяють працювати з наборами даних більш гнучко ніж масив.

Кожен об'єкт, що міститься в колекції, називається її елементом. Деякі колекції зберігають дані як прямий список елементів, інші містять списки пар ключів і значень.

**Generics.**

У C# є два окремих механізми для створення коду, який повторно використовується: спадкування та узагальнення.

Спадкування висловлює можливість повторного використання базового типу, а узагальнення висловлюють ідею повторного використання «шаблону», що містить типи — «заглушки».

У .NET Framework, починаючи з версії 2.0, з’явилася можливість використовувати в якості параметрів типи даних. Ця можливість реалізується за допомогою Generics.

Generics дозволяють створювати узагальнені: класи, структури, інтерфейси, делегати, методи.

Параметри типів використовуються для вказівки типів: змінних класу, параметрів функції, повертаємих значень функцій, локальних змінних.

Наприклад: у колекції ArrayList зберігаються дані типу object для того щоб в цю колекцію можна було помістити змінні будь-якого типу.

Найчастіше в колекції зберігаються змінні одного і того ж типу.

При отриманні даних з колекції треба вказувати явне приведення типу.

Можна легко допустити помилку приведення при отриманні даних з колекції, тобто помістити в колекцію змінну одного типу, а при добуванні виконати приведення до іншого типу.

Іншим недоліком є зниження продуктивності через виконання упаковки і розпаковки при зберіганні в колекції змінних структурних типів.

Методи Generics колекцій приймають і повертають параметри узагальненого типу. Тому при розміщенні даних у колекцію і при зверненні до цих даних упаковка і розпаковка не відбувається.

Якщо в неузагальнених колекціях зберігати об'єкти посилальних типів даних, то зниження продуктивності відбуватися не буде.

Використання Generics дає наступні переваги:

безпеку типів. З Generics колекцію можна помістити тільки об'єкти певного типу (зазначеного при розкритті шаблону);

більш простий і зрозумілий код; підвищення продуктивності.

**Створення Generic інтерфейсів.**

При створенні інтерфейсів, як і при створенні класів, можна використовувати узагальнені параметри типу. Особливо зручно використовувати узагальнені інтерфейси при роботі зі структурними типами, тому робота зі структурними типами через неузагальнений інтерфейс привела б до упаковки і розпаковки, а також до втрати безпеки типів під час компіляції.

При реалізації generic інтерфейсу в неузагальненому класі необхідно конкретизувати аргументи типу.

У generic класі можна реалізувати інтерфейс, використовуючи узагальнені параметри типу.

Створення Generic делегатів.

Підтримка узагальнених делегатів дозволяє передавати методам зворотнього виклику будь-які типи об'єктів, забезпечуючи при цьому безпеку викликів.

Створення Generic методів.

Метод, параметризований якимось типом даних, може знаходитися в узагальненому класі.

При виклику методу можна після імені вказувати реальний тип даних, але можна цього і не робити, тому тип даних визначається автоматично за типом параметрів, переданих в метод.

**Ітератори**

Для циклічного звернення до елементів колекції найчастіше простіше (та й краще) організувати цикл foreach.

Якщо потрібно створити клас, що містить об'єкти, що перераховуються в циклі foreach, то в цьому класі слід реалізувати інтерфейси IEnumerator і IEnumerable.

Іншими словами, для того щоб звернутися в циклі foreach до об'єкта класу, що визначається користувачем, необхідно реалізувати інтерфейси IEnumerator і IEnumerable в їх узагальненій або неузагальненій формі.

Простіше скористатися ітератором, який являє собою метод, оператор або аксесуар, який повертає по черзі члени сукупності об'єктів від початку до кінця.

Реалізувавши ітератор, можна звертатися до об'єктів класу, визначеним користувачем, в циклі foreach.

Позначення **yield** служить в мові С# в якості контекстного ключового слова. Це означає, що воно має спеціальне призначення тільки в блоці ітератора — повернути.

Для передчасного переривання ітератора служить наступна форма оператора yield: yield break;

Коли цей оператор виконується, ітератор повідомляє про те, що досягнутий кінець колекції. А це, по суті, зупиняє сам ітератор.

У ітераторі допускається застосування декількох операторів yield. Але кожен такий оператор повинен повертати наступний елемент в колекції.

**LINQ**

Мова інтегрованих запитів (Language Integrated Quary – LINQ) вбудовує синтаксис запитів в мову програмування C# і забезпечує можливість доступу до різних джерел даних за дапомогою одного і того ж синтаксису.

1. **Повні умови завдань.**
   1. Як у мові C# використовується узагальнення?

Узагальнення (generics) дозволяють створювати класи, методи, інтерфейси, які працюють з будь-яким типом даних. У цьому прикладі узагальнення використовується в класі:

public class HumanContainer<T> : IEnumerable<T> where T : Human

Це означає, що HumanContainer може містити будь-які типи, що є нащадками класу Human.

* 1. Що робить ключове слово where при визначенні класу HumanContainer?

Ключове слово where T : Human обмежує тип-параметр T лише тими класами, які є спадкоємцями Human. Це гарантує, що у контейнері будуть тільки "людські" об’єкти, які мають властивості Height, Weight тощо.

* 1. З якою метою клас Human реалізує інтерфейс IComparable? Що описує цей інтерфейс?

Інтерфейс IComparable<T> описує метод CompareTo, який використовується для порівняння об’єктів. Клас Human реалізує його, щоб об’єкти можна було сортувати за повним іменем:

public int CompareTo(Human other)

{

return string.Compare(other.FullName, FullName, StringComparison.InvariantCultureIgnoreCase);

}

Це дозволяє, наприклад, використовувати List<T>.Sort().

* 1. Поясніть призначення інтерфейсу IEnumerable. Які методи доведеться реалізувати для того, щоб скористатися цим інтерфейсом?

Інтерфейс IEnumerable дозволяє об'єкту бути ітерованим у foreach. Потрібно реалізувати метод:

public IEnumerator<T> GetEnumerator()

та неявну реалізацію:

IEnumerator IEnumerable.GetEnumerator()

Це дає можливість перераховувати елементи у контейнері HumanContainer.

* 1. Що таке «Ітератор». Який інтерфейс описує властивості та поведінку об'єкта-ітератора?

Ітератор — це об'єкт, який дозволяє поетапно перебирати колекцію. Він реалізує інтерфейс IEnumerator, який має:

Current — поточний елемент;

MoveNext() — перехід до наступного елемента;

Reset() — скидання позиції.

Ітератори у C# часто реалізуються через yield return.

* 1. Поясніть принцип роботи індексатора.

Індексатор дозволяє доступ до елементів класу за індексом: Він працює як масив. У прикладі доступ обмежено перевіркою діапазону індексу.

* 1. Навіщо використовується мова інтегрованих запитів (LINQ)?

LINQ дозволяє писати запити до колекцій прямо в коді C#, уніфікуючи доступ до даних (масиви, списки, XML, БД). Це зручно, компактно та безпечно завдяки типізації.

* 1. Наведіть приклад відкладених запитів та тих, що виконуються одразу, у мові LINQ

Відкладений запит:

var query = container1.Where(h => h.Height > 170);

— виконується лише під час ітерації (foreach, ToList() тощо).

Негайний запит:

var result = container1.Where(h => h.Height > 170).ToList();

— результат негайно обчислюється й зберігається в result.

* 1. У чому переваги відкладених запитів?
* Ефективність: обробка виконується тільки тоді, коли потрібна.
* Свіжість даних: зміни в колекції враховуються під час виконання запиту.
* Композиційність: запити можна ланцюжити.
  1. Як LINQ використовує лямбда-вирази?

Лямбда-вирази — це компактні функції, що передаються як аргументи.

* 1. Поясніть принцип роботи всіх LINQ-операцій, які у прикладі.

OrderBy, ThenBy

Сортування за Height, потім за Weight.

Where

Фільтрація за умовою.

Select

Проєкція в нову форму (анонімний об'єкт).

SelectMany

Плоске розгортання вкладених колекцій (наприклад, частин FullName).

Skip / SkipWhile

Пропускає елементи: за кількістю або до певної умови.

Take / TakeWhile

Бере елементи: за кількістю або до виконання умови.

Concat

Об'єднання двох колекцій.

GroupBy

Групування студентів за University.

First / FirstOrDefault

Повертає перший елемент або null (якщо не знайдено).

DefaultIfEmpty

Повертає елемент за замовчуванням, якщо колекція порожня.

Min / Max

Знаходить мінімальне/максимальне значення (наприклад, Weight, Height).

Join

З'єднання елементів за спільною ознакою (Height), створює нові об'єкти.

GroupJoin

Групування з приєднанням — елементи групуються та підраховується їх кількість і вага.

All / Any

Перевірка умов на всі або хоча б один елемент у колекції.

Contains

Перевірка, чи містить колекція певний елемент (wr3).

Задача 1

Створіть узагальнений клас CollectionType<T>. Визначити в класі конструктори, методи додавання та видалення елементів, інші необхідні методи та, якщо потрібно, перевантажені операції. Визначити індексатори та властивості. CollectionType можна реалізувати на основі стандартних колекцій (List, Stack, Array тощо).

Задача 2

Візьміть, створений тип (клас) з лабораторної №1, та реалізувати в ньому інтерфейс IComparable<T>. Використовуйте цей клас як параметр вашого узагальненого класу. Створіть кілька колекцій. Виконайте сортування, LINQ-запити відповідно до варіанта.

Задача 3

Виконайте кілька складних LINQToObject запитів (мінімум 5) до колекції об'єктів, використовуючи одночасно більше трьох операцій (приклад: where + select + orderBy, first + any + min).

Задача 4

Створіть узагальнену стандартну колекцію з простору імен System.Collections вказану у варіанті з рядками та виконайте введення-виведення, пошук рядків, що містять певне значення, підрахунок кількості рядків довжини n, сортування у зростаючому та спадному порядку.

1. **Лістинги програм.**

**Задача 1**

using System;

using System.Collections;

using System.Collections.Generic;

public class CollectionType<T> : IEnumerable<T>

{

private List<T> \_items;

// Конструктор за замовчуванням

public CollectionType()

{

\_items = new List<T>();

}

// Конструктор із початковою колекцією

public CollectionType(IEnumerable<T> collection)

{

\_items = new List<T>(collection);

}

// Додавання елемента

public void Add(T item)

{

\_items.Add(item);

}

// Видалення елемента

public bool Remove(T item)

{

return \_items.Remove(item);

}

// Отримання кількості елементів

public int Count => \_items.Count;

// Індексатор

public T this[int index]

{

get => \_items[index];

set => \_items[index] = value;

}

// Перевірка наявності елемента

public bool Contains(T item)

{

return \_items.Contains(item);

}

// Очистити колекцію

public void Clear()

{

\_items.Clear();

}

// Перевантаження оператора + для об'єднання колекцій

public static CollectionType<T> operator +(CollectionType<T> a, CollectionType<T> b)

{

var result = new CollectionType<T>(a.\_items);

result.\_items.AddRange(b.\_items);

return result;

}

// Реалізація IEnumerable<T>

public IEnumerator<T> GetEnumerator()

{

return \_items.GetEnumerator();

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

// Переоприділення ToString для виводу елементів

public override string ToString()

{

return $"[{string.Join(", ", \_items)}]";

}

}

class Program

{

static void Main()

{

var collection1 = new CollectionType<int>();

collection1.Add(1);

collection1.Add(2);

var collection2 = new CollectionType<int>(new[] { 3, 4, 5 });

var combined = collection1 + collection2;

Console.WriteLine(combined);

combined.Remove(2);

Console.WriteLine(combined[1]);

Console.WriteLine($"Count: {combined.Count}");

}

}

**Задача 2**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

public class CollectionType<T> : IEnumerable<T>

{

private List<T> \_items;

public CollectionType() => \_items = new List<T>();

public CollectionType(IEnumerable<T> collection) => \_items = new List<T>(collection);

public void Add(T item) => \_items.Add(item);

public bool Remove(T item) => \_items.Remove(item);

public void Clear() => \_items.Clear();

public bool Contains(T item) => \_items.Contains(item);

public int Count => \_items.Count;

public T this[int i]

{

get => \_items[i];

set => \_items[i] = value;

}

// Метод сортування: для T, що реалізує IComparable<T>

public void SortItems()

{

\_items.Sort();

}

public IEnumerator<T> GetEnumerator() => \_items.GetEnumerator();

IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();

public override string ToString() => $"[{string.Join(", ", \_items)}]";

}

// Клас з лабораторної #1, доповнений IComparable<T>

public class KindofAntelopes : IComparable<KindofAntelopes>

{

public string Name { get; set; }

public char Group { get; set; }

public string Area { get; set; }

public int Population

{

get => \_pop;

set => \_pop = value < 0 ? throw new ArgumentException("Population must be ≥ 0") : value;

}

private int \_pop;

public KindofAntelopes()

: this("Невідома", '-', "Невідоме", 0) { }

public KindofAntelopes(string name, char group, string area, int population)

{

Name = name; Group = group; Area = area; Population = population;

}

// Копіювальний конструктор

public KindofAntelopes(KindofAntelopes other)

: this(other.Name, other.Group, other.Area, other.Population) { }

public void Display()

{

Console.WriteLine($"Назва: {Name}, Група: {Group}, Житло: {Area}, Чисельність: {Population}");

}

// Порівняння за Population

public int CompareTo(KindofAntelopes other)

{

if (other == null) return 1;

return this.Population.CompareTo(other.Population);

}

public override string ToString() => $"{Name}({Group})[{Population}]";

}

class Program

{

static void Main()

{

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

var herd1 = new CollectionType<KindofAntelopes>(new[]

{

new KindofAntelopes("Джейран", 'А', "Азія", 30000),

new KindofAntelopes("Гну", 'В', "Африка", 560000),

new KindofAntelopes("Бейза", 'Н', "Африка", 2500)

});

var herd2 = new CollectionType<KindofAntelopes>(new[]

{

new KindofAntelopes("Ардиковий", 'С', "Європа", 12000),

new KindofAntelopes("Імпалa", 'D', "Африка", 100000)

});

var herd3 = new CollectionType<KindofAntelopes>(new[]

{

new KindofAntelopes("Сайга", 'Е', "Азія", 85000),

new KindofAntelopes("Морський козел", 'F', "Європа", 5000),

new KindofAntelopes("Північний олень", 'G', "Північна Америка", 400000),

new KindofAntelopes("Кулан", 'H', "Азія", 15000)

});

// Масив колекцій

var collections = new[] { herd1, herd2, herd3 };

// 2) Сортуємо елементи усередині кожної колекції за Population

foreach (var col in collections)

{

col.SortItems();

}

// Вивід після сортування

Console.WriteLine("=== Після сортування елементів у кожній колекції за Population ===");

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

{

Console.WriteLine($"Колекція #{i + 1} (Count = {collections[i].Count}): {collections[i]}");

}

// 3) LINQ-запити

int n = 3;

// знайти колекції розміру n

var sizeN = collections.Where(c => c.Count == n).ToList();

Console.WriteLine($"\nКолекції з кількістю елементів = {n}:");

if (sizeN.Any())

sizeN.ForEach(c => Console.WriteLine(c));

else

Console.WriteLine("Таких немає.");

// знайти максимальну та мінімальну колекцію за Count

var maxCol = collections.OrderByDescending(c => c.Count).First();

var minCol = collections.OrderBy(c => c.Count).First();

Console.WriteLine($"\nМаксимальна колекція (Count = {maxCol.Count}): {maxCol}");

Console.WriteLine($"Мінімальна колекція (Count = {minCol.Count}): {minCol}");

}

}

**Задача 3**

using System;

using System.Collections.Generic;

using System.Linq;

public class KindofAntelopes : IComparable<KindofAntelopes>

{

public string Name { get; set; }

public char Group { get; set; }

public string Area { get; set; }

public int Population

{

get => \_pop;

set => \_pop = value < 0

? throw new ArgumentException("Population must be ≥ 0")

: value;

}

private int \_pop;

public KindofAntelopes()

: this("Невідома", '-', "Невідоме", 0) { }

public KindofAntelopes(string name, char group, string area, int population)

{

Name = name;

Group = group;

Area = area;

Population = population;

}

public int CompareTo(KindofAntelopes other)

{

if (other == null) return 1;

return Population.CompareTo(other.Population);

}

public override string ToString()

=> $"{Name}({Group}) in {Area}: {Population}";

}

class Program

{

static void Main()

{

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

// Підготовка даних

var antelopes = new List<KindofAntelopes>

{

new KindofAntelopes("Гну", 'A', "Африка", 560000),

new KindofAntelopes("Джейран", 'A', "Азія", 30000),

new KindofAntelopes("Бейза", 'B', "Африка", 2500),

new KindofAntelopes("Сайга", 'A', "Азія", 85000),

new KindofAntelopes("Імпала", 'C', "Африка", 100000),

new KindofAntelopes("Північний олень", 'D', "Америка", 400000),

new KindofAntelopes("Кулан", 'E', "Азія", 15000),

new KindofAntelopes("Антилопа каама", 'F', "Африка", 12000),

new KindofAntelopes("Газель Томпсона", 'G', "Африка", 550000)

};

// 1) Африканські антилопи з популяцією > 10000, відсортувати за спаданням, вибрати назви

var africanBig = antelopes

.Where(a => a.Area == "Африка" && a.Population > 10000)

.OrderByDescending(a => a.Population)

.Select(a => a.Name);

Console.WriteLine("1) Африканські антилопи з популяцією > 10000 (спадання):");

foreach (var name in africanBig)

Console.WriteLine($" {name}");

// 2) Перша антилопа групи 'A' в Азії з Population < 100000

var firstMatch = antelopes

.Where(a => a.Group == 'A' && a.Area == "Азія" && a.Population < 100000)

.OrderBy(a => a.Population)

.FirstOrDefault();

Console.WriteLine("\n2) Перша азійська антилопа групи A з Population < 100000:");

Console.WriteLine(firstMatch != null

? $" {firstMatch}"

: " Не знайдено");

// 3) Групування по регіону, сортування груп за кількістю, вивід елементів кожної групи

var groupedByArea = antelopes

.GroupBy(a => a.Area)

.OrderByDescending(g => g.Count());

Console.WriteLine("\n3) Групування по регіону (за кількістю у спадному порядку):");

foreach (var group in groupedByArea)

{

Console.WriteLine($" {group.Key} ({group.Count()} видів):");

foreach (var a in group.OrderBy(a => a.Name))

Console.WriteLine($" - {a.Name}");

}

// 4) Три антилопи з найменшою Population, де група ≠ 'A', відсортувати по зростанню

var lowestPop = antelopes

.Where(a => a.Group != 'A')

.OrderBy(a => a.Population)

.Take(3)

.Select(a => new { a.Name, a.Population });

Console.WriteLine("\n4) 3 антилопи з найменшою чисельністю (група ≠ 'A'):");

foreach (var item in lowestPop)

Console.WriteLine($" {item.Name}: {item.Population}");

// 5) Чи є хоча б одна антилопа в Африці з Population < 10000 і групою 'F' або 'G'?

bool hasRare = antelopes

.Where(a => a.Area == "Африка")

.Any(a => a.Population < 10000 && (a.Group == 'F' || a.Group == 'G'));

Console.WriteLine("\n5) Чи є рідкісна африканська антилопа (група F або G, <10000)?");

Console.WriteLine(hasRare ? " Так, є." : " Ні, немає.");

}

}

**Задача 4**

using System;

using System.Collections.Generic;

using System.Linq;

public class MyLinkedList<T>

{

private LinkedList<T> list = new LinkedList<T>();

// Додавання елемента

public void Add(T item) => list.AddLast(item);

// Видалення елемента

public bool Remove(T item) => list.Remove(item);

// Виведення елементів

public void PrintAll()

{

foreach (var item in list)

Console.WriteLine(" " + item);

}

// Пошук елементів, що містять певне значення (для string)

public IEnumerable<T> FindContaining(string value)

{

if (typeof(T) != typeof(string))

throw new InvalidOperationException("Пошук за підстрокою доступний лише для рядків.");

return list.Cast<string>()

.Where(s => s.Contains(value, StringComparison.CurrentCultureIgnoreCase))

.Cast<T>();

}

// Підрахунок кількості рядків певної довжини

public int CountByLength(int length)

{

if (typeof(T) != typeof(string))

throw new InvalidOperationException("Фільтрація за довжиною доступна лише для рядків.");

return list.Cast<string>().Count(s => s.Length == length);

}

// Сортування за зростанням

public IEnumerable<T> SortAscending()

=> list.OrderBy(x => x);

// Сортування за спаданням

public IEnumerable<T> SortDescending()

=> list.OrderByDescending(x => x);

}

class Program

{

static void Main()

{

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

var myList = new MyLinkedList<string>();

// Введення рядків

Console.Write("Скільки рядків ввести? ");

if (!int.TryParse(Console.ReadLine(), out int n) || n <= 0)

{

Console.WriteLine("Некоректне число.");

return;

}

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

{

Console.Write($" Рядок {i + 1}: ");

string input = Console.ReadLine() ?? "";

myList.Add(input);

}

// Виведення

Console.WriteLine("\nВсі рядки:");

myList.PrintAll();

// Пошук підрядка

// Пошук підрядка

Console.Write("\nПошук підрядка: ");

string search = Console.ReadLine() ?? "";

var found = myList.FindContaining(search).ToList(); // Матеріалізуємо результат

if (found.Count == 0)

{

Console.WriteLine($"Рядки, що містять \"{search}\", не знайдено.");

}

else

{

Console.WriteLine($"Знайдено рядки, що містять \"{search}\":");

foreach (var s in found)

Console.WriteLine(" " + s);

}

// Підрахунок за довжиною

Console.Write("\nВведіть довжину рядка: ");

if (int.TryParse(Console.ReadLine(), out int len))

{

int count = myList.CountByLength(len);

Console.WriteLine($"Кількість рядків довжини {len}: {count}");

}

// Сортування за зростанням

Console.WriteLine("\nСортування за зростанням:");

foreach (var s in myList.SortAscending())

Console.WriteLine(" " + s);

// Сортування за спаданням

Console.WriteLine("\nСортування за спаданням:");

foreach (var s in myList.SortDescending())

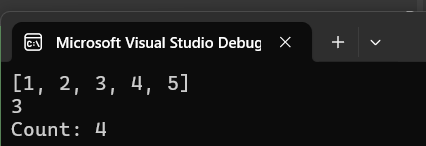
Console.WriteLine(" " + s);

}

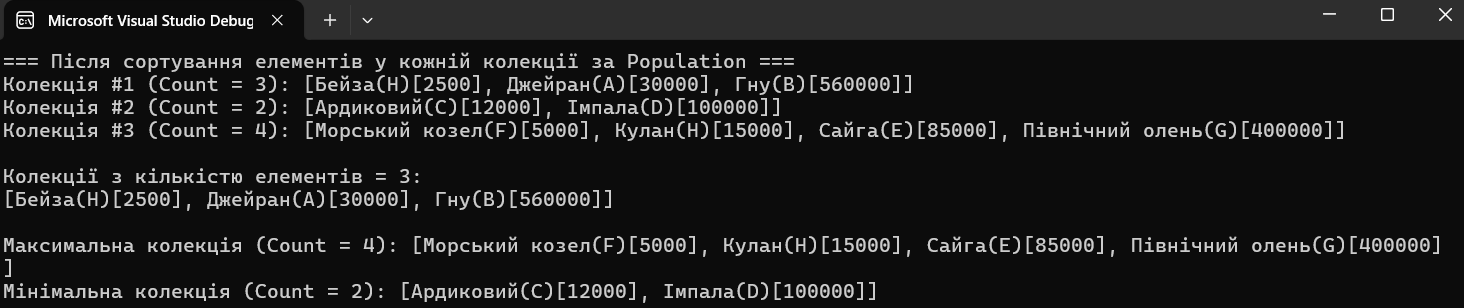
}

1. **Результати роботи програм.**

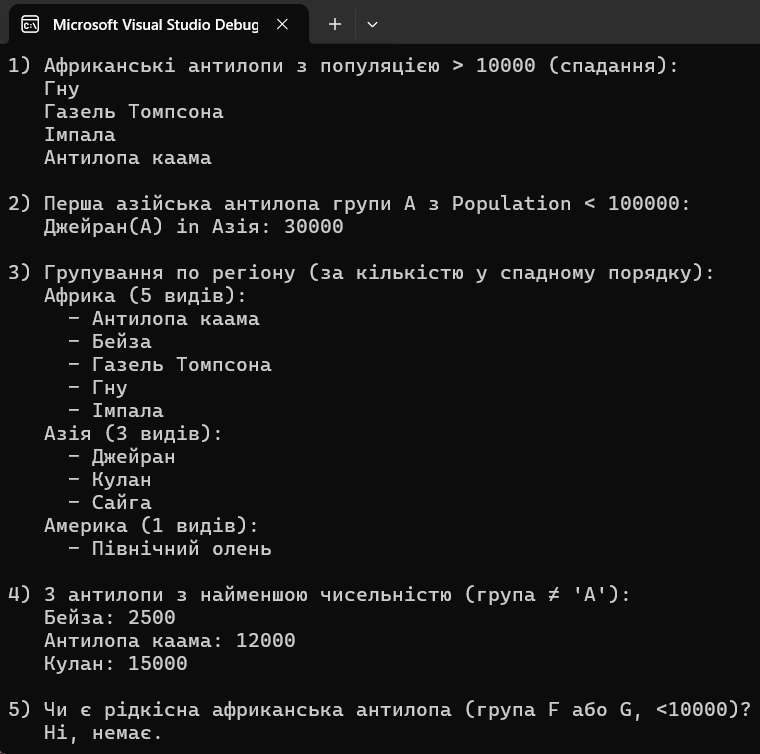
Задача 1

****

Задача 2



Задача 3



Задача 4

