# **Типи кортежів**

# **(Tuple types)**

Функція кортежів надає стислий синтаксис для групування кількох елементів даних у спрощеній структурі даних. У наступному прикладі показано, як можна оголосити змінну кортежу, ініціалізувати її та отримати доступ до її елементів даних.

(double, int) t1 = (4.5, 3);

Console.WriteLine($"Tuple with elements {t1.Item1} and {t1.Item2}.");

// Output:

// Tuple with elements 4.5 and 3.

(double Sum, int Count) t2 = (4.5, 3);

Console.WriteLine($"Sum of {t2.Count} elements is {t2.Sum}.");

// Output:

// Sum of 3 elements is 4.5.

Як показано в попередньому прикладі, для визначення типу кортежу необхідно вказати типи всіх його елементів даних і, за необхідності, імена полів. У типі кортежа неможливо визначити методи, але можна використовувати методи, що надаються .NET, як показано в наступному прикладі.

(double, int) t = (4.5, 3);

Console.WriteLine(t.ToString());

Console.WriteLine($"Hash code of {t} is {t.GetHashCode()}.");

// Output:

// (4.5, 3)

// Hash code of (4.5, 3) is 718460086

Типи кортежів підтримують оператори == та !=

Типи кортежів є типами значень, а елементи кортежа - загальнодоступними полями. Тому кортежі являють собою змінювані типи значень.Можна визначити кортежі з як завгодно великим числом елементів.

var t =

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

11, 12, 13, 14, 15, 16, 17, 18,

19, 20, 21, 22, 23, 24, 25, 26);

Console.WriteLine(t.Item26); // output: 26

**Варіанти використання кортежів**

Найчастіше кортежі використовуються як тип, що повертається методом. Тобто замість визначення параметрів методу out можна згрупувати результати методу в тип кортежа, що повертається, як показано в наступному прикладі.

var xs = new[] { 4, 7, 9 };

var limits = FindMinMax(xs);

Console.WriteLine($"Limits of [{string.Join(" ", xs)}] are {limits.min} and {limits.max}");

// Output:

// Limits of [4 7 9] are 4 and 9

var ys = new[] { -9, 0, 67, 100 };

var (minimum, maximum) = FindMinMax(ys);

Console.WriteLine($"Limits of [{string.Join(" ", ys)}] are {minimum} and {maximum}");

// Output:

// Limits of [-9 0 67 100] are -9 and 100

(int min, int max) FindMinMax(int[] input)

{

if (input is null || input.Length == 0)

{

throw new ArgumentException("Cannot find minimum and maximum of a null or empty array.");

}

var min = int.MaxValue;

var max = int.MinValue;

foreach (var i in input)

{

if (i < min)

{

min = i;

}

if (i > max)

{

max = i;

}

}

return (min, max);

}

Як показано в попередньому прикладі, з екземпляром кортежу, що повертається, можна працювати безпосередньо або деконструювати його в окремі змінні.

Як правило, кортежі використовуються для групування слабко пов'язаних елементів даних. Це доцільно в закритих і внутрішніх службових методах.

**Імена полів кортежів**

Імена полів кортежів вказуються явно або у виразі ініціалізації кортежа, або у визначенні типу кортежа, як показано в наступному прикладі.

var t = (Sum: 4.5, Count: 3);

Console.WriteLine($"Sum of {t.Count} elements is {t.Sum}.");

(double Sum, int Count) d = (4.5, 3);

Console.WriteLine($"Sum of {d.Count} elements is {d.Sum}.");

Якщо ім'я поля не вказано, його можна вивести з імені відповідної змінної у виразі ініціалізації кортежу, як показано в наступному прикладі:

var sum = 4.5;

var count = 3;

var t = (sum, count);

Console.WriteLine($"Sum of {t.count} elements is {t.sum}.");

Це називається ініціалізаторами проекцій кортежу. Ім'я змінної не проектується на ім'я поля кортежу в таких випадках:

* Ім'я кандидата - це ім'я елемента типу кортежа, наприклад Item3, ToString або Rest.
* Ім'я кандидата є дублікатом іншого імені поля кортежа, явного або неявного.

У цих випадках необхідно або явно вказати ім'я поля, або отримати доступ до поля за ім'ям за замовчуванням. За замовчуванням поля кортежу мають імена ***Item1***, ***Item2***, ***Item3*** тощо. Завжди можна використовувати ім'я поля за замовчуванням, навіть якщо ім'я поля вказано явно або воно є виведеним, як показано в наступному прикладі.

var a = 1;

var t = (a, b: 2, 3);

Console.WriteLine($"The 1st element is {t.Item1} (same as {t.a}).");

Console.WriteLine($"The 2nd element is {t.Item2} (same as {t.b}).");

Console.WriteLine($"The 3rd element is {t.Item3}.");

// Output:

// The 1st element is 1 (same as 1).

// The 2nd element is 2 (same as 2).

// The 3rd element is 3.

Імена полів не враховуються під час присвоєння кортежу та порівняння кортежів на рівність. Під час компіляції компілятор замінює імена полів не за замовчуванням відповідними іменами за замовчуванням. У результаті явно вказані або виведені імена полів будуть недоступні під час виконання.

**Присвоювання та деконструкція кортежів**

У C# підтримується присвоювання між типами кортежів, які відповідають обом таким умовам:

* обидва типи кортежів повинні містити однакову кількість елементів;
* для кожної позиції кортежа тип правого елемента кортежа аналогічний типу відповідного лівого елемента кортежа або може бути неявно перетворений у цей тип.

Значення елементів кортежу присвоюються в порядку розташування елементів кортежу. Імена полів кортежа не враховуються і не присвоюються, як показано в наступному прикладі.

(int, double) t1 = (17, 3.14);

(double First, double Second) t2 = (0.0, 1.0);

t2 = t1;

Console.WriteLine($"{nameof(t2)}: {t2.First} and {t2.Second}");

// Output:

// t2: 17 and 3.14

(double A, double B) t3 = (2.0, 3.0);

t3 = t2;

Console.WriteLine($"{nameof(t3)}: {t3.A} and {t3.B}");

// Output:

// t3: 17 and 3.14

Оператор присвоювання **“*=*”** можна також використовувати для деконструкції екземпляра кортежу в окремі змінні. Існує три способи деконструкції кортежу.

* Ви можете явно оголосити тип кожної змінної у дужках.

var t = ("post office", 3.6);

(string destination, double distance) = t;

Console.WriteLine($"Distance to {destination} is {distance} kilometers.");

// Output:

// Distance to post office is 3.6 kilometers.

* Ви можете використовувати ключове слово ***var*** за межами круглих дужок, щоб оголосити неявно набрані змінні та дозволити компілятору вивести їхні типи.

var t = ("post office", 3.6);

var (destination, distance) = t;

Console.WriteLine($"Distance to {destination} is {distance} kilometers.");

// Output:

// Distance to post office is 3.6 kilometers.

* Ви можете використовувати існуючі змінні.

var destination = string.Empty;

var distance = 0.0;

var t = ("post office", 3.6);

(destination, distance) = t;

Console.WriteLine($"Distance to {destination} is {distance} kilometers.");

// Output:

// Distance to post office is 3.6 kilometers.

**Рівність кортежів**

Типи кортежів **“==”** підтримують оператори та **“!=”** оператори. Ці оператори порівнюють елементи лівого операнда з відповідними елементами правого операнда відповідно до порядку розташування елементів кортежа.

(int a, byte b) left = (5, 10);

(long a, int b) right = (5, 10);

Console.WriteLine(left == right); // output: True

Console.WriteLine(left != right); // output: False

var t1 = (A: 5, B: 10);

var t2 = (B: 5, A: 10);

Console.WriteLine(t1 == t2); // output: True

Console.WriteLine(t1 != t2); // output: False

Як показано в попередньому прикладі, в операціях **==** і **!=** не враховуються імена полів кортежів.

Два кортежі можна порівняти, якщо виконано обидві такі умови:

* обидва кортежі містять однакову кількість елементів. Наприклад, t1 != t2 не компілюється, якщо t1 і t2 мають різну кількість елементів.
* Для кожної позиції кортежа відповідні елементи з лівого і правого операндів кортежа порівнюються за допомогою операторів == і !=. Наприклад, (1, (2, 3)) == ((1, 2), 3) не компілюється, оскільки 1 не порівнюється за допомогою (1, 2).

Оператори **==** і **!=** порівнюють кортежі зі скороченою обробкою. Це означає, що операція зупиняється, щойно вона відповідає парі нерівних елементів або досягає кінця кортежів. Однак перед будь-яким порівнянням усі елементи кортежа обчислюються, як показано в наступному прикладі.

Console.WriteLine((Display(1), Display(2)) == (Display(3), Display(4)));

int Display(int s)

{

Console.WriteLine(s);

return s;

}

// Output:

// 1

// 2

// 3

// 4

// False

**Кортежі та System.Tuple**

Кортежі C# з типами System.ValueTuple, відрізняються від кортежів, представлених типами System.Tuple.

Основні відмінності полягають у такому:

* Типи System.ValueTuple є типами значень. Типи System.Tuple є посилальними типами.
* Типи System.ValueTuple є змінюваними. Типи System.Tuple є незмінними.
* Елементами даних типів System.ValueTuple є поля. Елементами даних типів System.Tuple є властивості.

**Кортежі як параметри виведення**

Як правило, ви виконуєте рефакторинг методу, що має параметри ***out***, у метод, що повертає кортеж. Однак бувають випадки, коли параметр out може мати тип кортежу. У наступному прикладі показано, як працювати з кортежами у вигляді параметрів ***out***.

var limitsLookup = new Dictionary<int, (int Min, int Max)>()

{

[2] = (4, 10),

[4] = (10, 20),

[6] = (0, 23)

};

if (limitsLookup.TryGetValue(4, out (int Min, int Max) limits))

{

Console.WriteLine($"Found limits: min is {limits.Min}, max is {limits.Max}");

}

// Output:

// Found limits: min is 10, max is 20