Danh sách trong C#: ArrayList, List, SortedList, Dictionary

Trong bài học này chúng ta sẽ chuyển sang nội dung về các loại danh sách (List) cơ bản thường dùng trong C#, bao gồm lớp ArrayList, SortedList, danh sách tổng quát (Generic List) List<T>, và từ điển (Dictionary). Lưu ý thấy ngay rằng đây là các implementation sẵn có của C#. Chúng ta có thể học và sử dụng ngay mà không cần phải tự implement. Đây cũng là các loại dữ liệu danh sách được sử dụng rất phổ biến trong lập trình C#.

NỘI DUNG CỦA BÀI [Ẩn]

Mảng và Danh sách trong C#

Mảng là một cấu trúc dữ liệu tập hợp rất phổ biến và hữu dụng trong nhiều thuật toán (như các thuật toán sắp xếp). Tuy nhiên, trong nhiều trường hợp khả năng ứng dụng của mảng bị hạn chế hoặc phức tạp hơn do bản chất của cấu trúc mảng: kích thước của mảng là cố định sau khi khởi tạo.

Giả sử bạn không biết trước được tổng số phần tử của mảng (và đây cũng là tình huống rất thường gặp), bạn sẽ phải tạo ra một mảng rất lớn để có đủ chỗ cho dữ liệu về sau. Đây cũng là một giải pháp rất thường thấy khi các bạn mới bắt đầu nhập môn lập trình. Dĩ nhiên đây chỉ là một giải pháp tình thế và nó rất không hiệu quả.

Giải pháp triệt để là sử dụng những cấu trúc dữ liệu khác tương tự mảng nhưng cho phép tăng giảm số lượng phần tử linh động khi có nhu cầu. Các cấu trúc dữ liệu như vậy thường được gọi là *danh sách* (List).

C# và .NET Framework xây dựng sẵn nhiều loại danh sách (list) khác nhau cho các loại nhu cầu. Trong bài học này chúng ta sẽ học lớp ArrayList, một loại danh sách (list) đơn giản nhất trong C#. Tiếp theo chúng ta sẽ làm quen với lớp List<T> – loại danh sách tổng quát được sử dụng rộng rãi bậc nhất trong C#. Cuối cùng chúng ta sẽ làm quen với SortedList – loại danh sách (list) tổng quát đặc biệt luôn luôn duy trì sắp xếp cho phần tử.

Kiểu (lớp) ArrayList trong C#

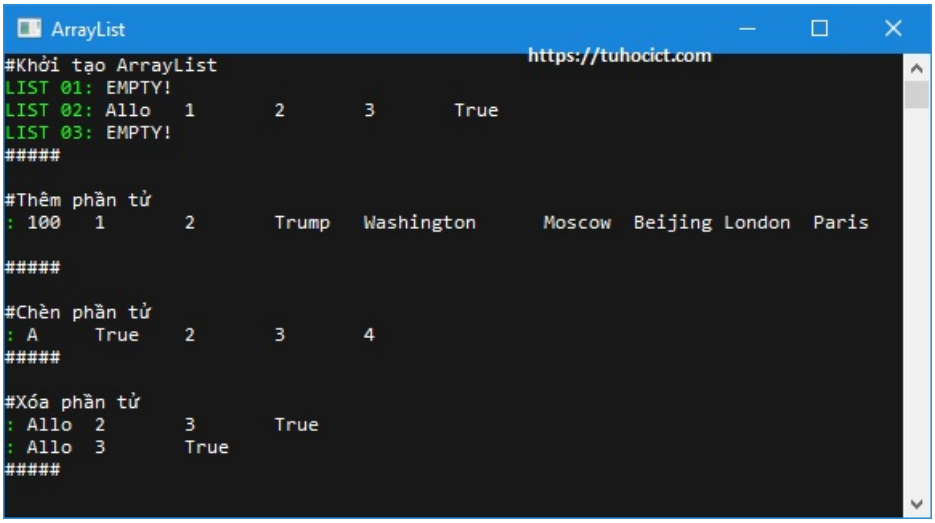
`ArrayList` là lớp thực thi cho một loại danh sách (list) trong C# đặc biệt có khả năng: (1) chứa dữ liệu thuộc bất kỳ kiểu cơ sở nào, (2) dễ dàng thêm-bớt-tìm kiếm phần tử trong danh sách, (3) các phần tử có thể có kiểu cơ sở khác nhau. Lớp `ArrayList` nằm trong không gian tên `System.Collections`.

Chúng ta hãy cùng thực một số ví dụ nhỏ với lớp ArrayList.

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace P01_ArrayList
{
    class Program
    {
        static void Print(ArrayList list, string label)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.Write($"{label}: ");
            Console.ResetColor();

            if (list.Count == 0)
                Console.Write("EMPTY!");

            // duyệt danh sách và in các phần tử ra console
            foreach (object item in list)
            {
                Console.Write($"{item}\t");
            }

            // hoặc
            //for (var i = 0; i < list.Count; i++)
            //{
            //    Console.Write($"{list[i]}\t");
            //}

            Console.WriteLine();
        }

        static void CreateInitialize()
        {
            Console.WriteLine("#Khởi tạo ArrayList");

            // khởi tạo array list
            var list1 = new ArrayList();

            // khởi tạo array list và cung cấp sẵn dữ liệu ban đầu
            var list2 = new ArrayList(new object[] { "Allo", 1, 2, 3, true });

            // khởi tạo và cung cấp kích thước ban đầu (sau có thể thêm bớt thoải mái)
            var list3 = new ArrayList(5);

            Print(list1, "LIST 01");
            Print(list2, "LIST 02");
            Print(list3, "LIST 03");

            Console.WriteLine("#####");
        }

        static void Add()
        {
            Console.WriteLine("#Thêm phần tử");
```

```
56.
57.             // khởi tạo array list
58.             var arrayList = new ArrayList();
59.
60.             // thêm một số nguyên (vào cuối)
61.             arrayList.Add(100);
62.             // thêm tiếp một mảng số nguyên
63.             arrayList.AddRange(new[] { 1, 2 });
64.
65.             // thêm một chuỗi
66.             arrayList.Add("Trump");
67.             // thêm một mảng string
68.             arrayList.AddRange(new[] { "Washington", "Moscow", "Beijing", "London", "Par
69.
70.             Print(arrayList, "");
71.
72.             Console.WriteLine("#####");
73.         }
74.
75.         static void Insert()
76.         {
77.             Console.WriteLine("#Chèn phần tử");
78.
79.             var list = new ArrayList(5);
80.             list.Insert(0, 'A'); // lưu ý arrayList khởi tạo với 5 phần tử
81.             list.InsertRange(1, new[] {2, 3, 4 });
82.             // chèn 1 bool vào vị trí số 1
83.             list.Insert(1, true);
84.             Print(list, "");
85.
86.             Console.WriteLine("#####");
87.         }
88.
89.         static void Remove()
90.         {
91.             Console.WriteLine("#Xóa phần tử");
92.
93.             var arrayList = new ArrayList(new object[] { "Allo", 1, 2, 3, true });
94.             // xóa phần tử có giá trị 1
95.             arrayList.Remove(1);
96.             Print(arrayList, "");
97.             // xóa phần tử ở vị trí số 1
98.             arrayList.RemoveAt(1);
99.             Print(arrayList, "");
100.            Console.WriteLine("#####");
101.        }
102.
103.        static void Main(string[] args)
104.        {
105.            Console.OutputEncoding = System.Text.Encoding.Unicode;
106.
107.            CreateInitialize(); Console.WriteLine();
108.            Add(); Console.WriteLine();
109.            Insert(); Console.WriteLine();
110.            Remove(); Console.WriteLine();
111.
112.            Console.ReadKey();
113.        }
114.    }
115. }
```

Dịch và chạy chương trình sẽ thu được kết quả như dưới đây



Khai báo và khởi tạo ArrayList

ArrayList cung cấp 3 overload để khởi tạo object.

```
// khởi tạo array list
var list1 = new ArrayList();

// khởi tạo array list và cung cấp sẵn dữ liệu ban đầu
var list2 = new ArrayList(new object[] { "Allo", 1, 2, 3, true });

// khởi tạo và cung cấp kích thước ban đầu (sau có thể thêm bớt thoải mái)
var list3 = new ArrayList(5);
```

Cách đơn giản nhất là dùng overload không tham số.

Overload thứ hai nhận một tập hợp giá trị bất kỳ làm tham số. Khi khởi tạo, các phần tử của tập hợp này sẽ được sao chép sang và trở thành phần tử của arraylist.

Overload thứ ba nhận một số nguyên làm tham số. Số nguyên này thể hiện dung lượng (capacity) tạm thời của arraylist, nghĩa là số phần tử có thể chứa được. Lưu ý rằng, không giống với mảng, dung lượng của arraylist có thể tiếp tục tăng lên theo nhu cầu. Số phần tử thực chứa trong arraylist luôn nhỏ hơn hoặc bằng với dung lượng.

Thêm/chèn phần tử vào ArrayList

Để **thêm phần tử** vào cuối danh sách có thể dùng hai phương thức: Add để thêm từng phần tử đơn; AddRange để thêm danh sách phần tử. AddRange nhận tham số là một mảng các object.

```
// thêm một số nguyên (vào cuối)
list.Add(100);
// thêm tiếp một mảng số nguyên
list.AddRange(new[] { 1, 2 });

// thêm một chuỗi
list.Add("Trump");
// thêm một mảng string
list.AddRange(new[] { "Washington", "Moscow", "Beijing", "London", "Paris" });
```

ArrayList chấp nhận kiểu của phần tử là `object`. Điều này có nghĩa là nó chấp nhận phần tử thuộc tất cả các kiểu dữ liệu của C#.

> Trong C#, lớp `object` (hoặc `Object`) là lớp tổ tông của tất cả các lớp khác, kể cả lớp do người dùng xây dựng. Theo cơ chế đa hình và kế thừa, một đối tượng tạo ra từ lớp con cũng được xem là đối tượng của lớp cha (biến thuộc kiểu con đồng thời cũng là biến thuộc kiểu cha). Do vậy, bất kỳ đối tượng nào trong C# cũng đồng thời là đối tượng của lớp object. Nếu phương thức chấp nhận tham số thuộc kiểu `object`, nó chấp nhận tham số thuộc bất kỳ kiểu nào của C#.

Chúng ta cũng có thể **chèn phần tử** vào vị trí bất kỳ trong danh sách với phương thức Insert hoặc InsertRange

```
var list = new ArrayList(5);
list.Insert(0, 'A'); // lưu ý arrayList khởi tạo với 5 phần tử
list.InsertRange(1, new[] {2, 3, 4 });
// chèn 1 bool vào vị trí số 1
list.Insert(1, true);
```

Khi chèn vào một vị trí, tất cả các phần tử đứng sau sẽ bị dồn về cuối danh sách.

Xóa phần tử khỏi ArrayList

Để **xóa phần tử** khỏi danh sách có thể sử dụng một trong các phương thức: Remove, RemoveAt, RemoveRange.

```
var list = new ArrayList(new object[] { "Allo", 1, 2, 3, true });
// xóa phần tử có giá trị 1
list.Remove(1);
Print(list, "");
// xóa phần tử ở vị trí số 1
list.RemoveAt(1);
```

- Remove sẽ xóa phần tử đầu tiên trong danh sách có giá trị trùng với giá trị cung cấp.
- RemoveAt xóa bỏ phần tử ở một vị trí xác định.
- RemoveRange xóa bỏ một nhóm phần tử.

Ngoài ra, để xóa bỏ tất cả các phần tử, có thể dùng phương thức Clear.

Truy xuất ArrayList

Để truy xuất từng phần tử, chúng ta dùng pháp toán chỉ số (index operator) tương tự mảng:

```
var a = (int)list[1];
var b = (int)list[2];
var str = list[0] as string;
```

Sự khác biệt ở chỗ, khi đọc phần tử của danh sách, chúng ta đồng thời phải thực hiện ép kiểu (type casting) từ kiểu object sang kiểu cụ thể khác. Trong ví dụ trên, chúng ta ép từ object sang int và string.

Để duyệt danh sách, chúng ta sử dụng vòng lặp tương tự như mảng. Một cách khác thường gặp là sử dụng vòng lặp foreach:

```
foreach(object item in list)
{
      Console.Write($"{item}\t");
```

```
}

for (var i = 0; i < list.Count; i++)
{
    Console.Write($"{list[i]}\t");
}
```

Để truy xuất một phần (danh sách con) của ArrayList có thể sử dụng phương thức GetRange.

Một số tính năng khác

Lấy thông tin về số lượng phần tử đang chứa trong danh sách: thuộc tính Count

Lấy thông tin về dung lượng hiện tại: thuộc tính Capacity

Xác định xem một giá trị có nằm trong danh sách: phương thức Contains

Xác định chỉ số của phần tử theo giá trị: phương thức IndexOf

Sắp xếp danh sách: phương thức Sort

Chuyển đổi danh sách thành mảng: phương thức ToArray

Ưu nhược điểm của ArrayList

Ưu điểm

Có thể thấy việc sử dụng ArrayList rất đơn giản và không khác biệt nhiều với mảng. ArrayList cho phép truy xuất trực tiếp phần tử theo chỉ số tương tự như mảng. Do đó, các thuật toán đã biết với mảng hoàn toàn có thể thực hiện được trên ArrayList.

ArrayList có ưu thế so với mảng là khả năng chèn-thêm mới-xóa bỏ phần tử rất linh hoạt. Đây là lợi thế chung của các loại danh sách so với mảng tĩnh.

ArrayList không giới hạn kiểu dữ liệu của phần tử. Hoàn toàn có thể sử dụng ArrayList như một kho dữ liệu trong bộ nhớ để chứa các object.

Nhược điểm

Các phần tử của ArrayList đều thuộc kiểu object. Do vậy, khi dùng để lưu trữ các giá trị không tham chiếu (như int, bool) sẽ xảy ra quá trình boxing/unboxing. Hai quá trình này mất thời gian để thực hiện. Nếu làm boxing/unboxing với lượng dữ liệu lớn sẽ rất không hiệu quả.

Khi truy xuất phần tử sẽ phải thực hiện biến đổi kiểu (type casting). Khi thêm (chèn) phần tử, mọi phần tử đều trải qua boxing về kiểu object. Khi lấy dữ liệu ra chúng ta phải thực hiện biến đổi kiểu (type casting) về dạng ban đầu. Do đó, người lập trình phải tự theo dõi kiểu dữ liệu của từng phần tử. Nếu không theo dõi được kiểu của phần tử, quá trình biến đổi

kiểu có thể bị lỗi. Đặc điểm này của ArrayList không phù hợp với đặc thù strong-typing (định kiểu mạnh) của C#.

Tạm kết

Trong phần này của bài học chúng ta đã xem xét chi tiết cách sử dụng loại danh sách (list) đầu tiên và đơn giản nhất trong C#.

Lưu ý rằng ArrayList là kiểu dữ liệu không được khuyến khích sử dụng nữa. Tuy nhiên, chúng ta vẫn xem xét rất kỹ cách sử dụng ArrayList vì các loại danh sách khác có cách sử dụng cơ bản hoàn toàn tương tự.

> Bạn hãy tự mình viết code sử dụng các phương thức và thuộc tính đã trình bày ở trên để nắm rõ cách thức hoạt động của ArrayList. Khi đã nắm được ArrayList, bạn hoàn toàn có thể tự sử dụng được kiểu List sẽ trình bày ở phần tiếp theo.

Kiểu dữ liệu List<T> sau đây sẽ giải quyết các vấn đề của ArrayList và cũng là kiểu dữ liệu nên sử dụng để thay thế mảng.

Generic List trong C#: List<T>

List<T> là một kiểu dữ liệu rất mạnh trong C# và được sử dụng đặc biệt rộng rãi. List<T> cũng là kiểu cơ sở để tạo ra các kiểu tập hợp cao cấp hơn trong C# (như BindingList). Để hiểu được kiểu dữ liệu này, bạn cần nắm được khái niệm và kỹ thuật lập trình generic trong C#.

List<T> được định nghĩa sẵn với các phương thức và thuộc tính tương tự như mảng và ArrayList. List<T> có các phương thức và thuộc tính với tên gọi và tính năng giống hệt của ArrayList, bao gồm: Count, Capacity, Add/AddRange, Clear, Contains, IndexOf/LastIndexOf, Insert/InsertRange, Remove, RemoveAt, RemoveRange, Reverse, ToArray. Việc truy xuất phần tử của List<T> giống hệt mảng và ArrayList.

Sự khác biệt của List<T> là ở chỗ kiểu dữ liệu cơ sở (của các phần tử) chỉ được xác định ở giai đoạn sử dụng class, thay vì ở giai đoạn định nghĩa như các kiểu none-generic.

Để dễ hình dung, trong C# List<T> tương tự như ArrayList, nhưng các phần tử của nó bắt buộc phải cùng kiểu T (giống như mảng). Trong đó T được xác định khi khai báo object của class List<T>. T có thể là bất kỳ kiểu dữ liệu C# hợp lệ nào, dù là kiểu định nghĩa sẵn (built-in types) hay kiểu do người dùng định nghĩa.

Hãy cùng thực hiện ví dụ nhỏ sau:

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.
6.  namespace P02_List
7.  {
8.      class Program
```

```
    {
        static void CreateInitialize()
        {
            // khởi tạo danh sách trống, các phần tử phải có kiểu int
            var list1 = new List<int>();
            Print(list1, "List of Ints");

            // khai báo và khởi tạo danh sách có thể chứa ngay 3 phần tử, các phần tử ph
            var list2 = new List<string>(3);
            Print(list2, "List of Strings");

            // khai báo và khởi tạo danh sách ký tự, đồng thời cung cấp luôn dữ liệu ban
            var list3 = new List<char>(new[] { 'a', 'b', 'c', 'd' });
            Print(list3, "List of Chars");

            // khai báo và khởi tạo sử dụng cú pháp object initialization
            var list4 = new List<int> { 1, 2, 3, 4, 5 };
            Print(list4, "List of Ints");
        }

        static void Print<T>(List<T> list, string label)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.Write($"{label}: ");
            Console.ResetColor();

            if (list.Count == 0)
                Console.Write("EMPTY!");

            // duyệt danh sách và in các phần tử ra console
            foreach (object item in list)
            {
                Console.Write($"{item}    ");
            }

            // hoặc
            //for (var i = 0; i < list.Count; i++)
            //{
            //    Console.Write($"{list[i]}\t");
            //}

            Console.WriteLine();
        }

        static void Access()
        {
            Console.WriteLine("# Truy xuất phần tử");
            var list = new List<int> { 1, 2, 3, 4, 5 };
            var a = list[0];
            var b = list[1];
            // để ý rằng chúng ta không cần ép kiểu cho a và b do List<T> có đặc tính st
            var c = a + b;
            Console.WriteLine($"Type of a: {a.GetType().Name}\r\nType of a: {b.GetType()
        }

        static void Main(string[] args)
        {
            Console.OutputEncoding = Encoding.Unicode;

            CreateInitialize(); Console.WriteLine();
            Access(); Console.WriteLine();

            Console.ReadKey();
        }
    }
}
```

List<T> nằm trong không gian tên `System.Collections.Generic` (khác với ArrayList nằm trong `System.Collections`).

So với ArrayList, List<T> có khác biệt nhỏ ở cách thức khởi tạo. Khi khởi tạo List<T> trong C# chúng ta đồng thời phải cung cấp kiểu dữ liệu của phần tử. Đặc điểm này làm nên tính strong-typed của List<T>, vốn là một đặc thù trong C#. Sau này khi truy xuất chúng ta không phải ép kiểu, do mỗi phần tử bắt buộc phải thuộc kiểu T do ta cung cấp khi khởi tạo.

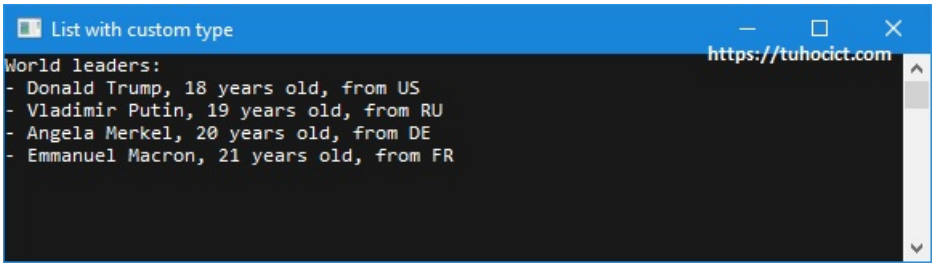
Tất cả các phương thức đã biết ở ArrayList đều có mặt trên List<T> với cùng tên gọi và chức năng. Do đó chúng ta không lặp lại chúng nữa. Bạn hoàn toàn có thể tự mình thử code với các phương thức đó.

Sử dụng List với kiểu do người dùng định nghĩa

Ở phần trên chúng ta đã sử dụng List<T> với T là các kiểu có sẵn trong C# (built-in types). Không chỉ vậy, T hoàn toàn có thể là các kiểu dữ liệu do người dùng định nghĩa.

Hãy cùng xem xét một ví dụ nhỏ khác.

```
using System;
using System.Collections.Generic;

namespace P03_ListWithCustomTypes
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "List with custom type";

            var people = Initialize();
            people.Add(new Person { Name = "Theresa May", Age = 22, Country = Country.UK
            Console.WriteLine("World leaders:");
            Print(people);

            Console.ReadKey();
        }

        static List<Person> Initialize()
        {
            var people = new List<Person> {
                new Person { Name = "Donald Trump", Age = 18, Country = Country.US },
                new Person { Name = "Vladimir Putin", Age = 19, Country = Country.RU },
                new Person { Name = "Angela Merkel", Age = 20, Country = Country.DE },
                new Person { Name = "Emmanuel Macron", Age = 21, Country = Country.FR },
            };
            return people;
        }

        static void Print(List<Person> people)
        {
            foreach (var p in people)
            {
                Console.WriteLine($"- {p.Name}, {p.Age} years old, from {p.Country}");
            }
        }
    }

    class Person
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public Country Country { get; set; }
    }

    enum Country
    {
        RU, VI, UK, US, DE, FR
    }
}
```



Có thể dễ dàng nhận thấy, làm việc với các kiểu tự định nghĩa không có gì khác biệt với các kiểu có sẵn. Đặc tính strong-typed của List<T> rất hữu ích khi làm việc với các kiểu dữ liệu phức tạp do người dùng định nghĩa. Không cần boxing/unboxing hay type casting, đồng thời đảm bảo hiệu suất cao khi truy xuất.

Một ưu thế rất lớn của List<T> là có thể sử dụng bộ thư viện LINQ để truy vấn dữ liệu.

List và LINQ

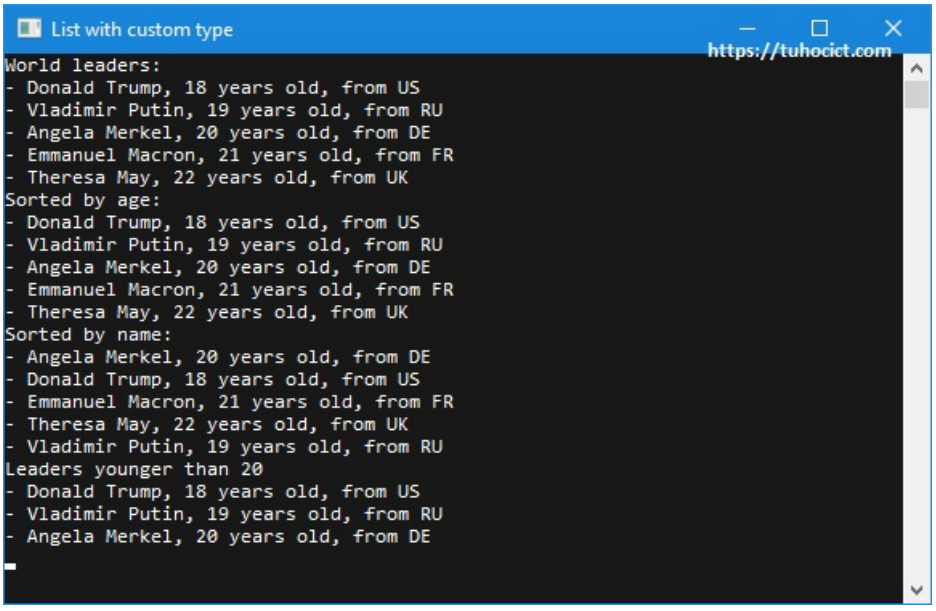
Các thuật toán sắp xếp mà chúng ta đã xem xét chỉ là một trong số những thuật toán hoạt động trên mảng và danh sách. Khi làm việc với các kiểu dữ liệu này có một loạt các yêu cầu rất thường gặp như tìm giá trí lớn nhất/nhỏ nhất, tính giá trị trung bình, trích danh sách con, trích một phần dữ liệu, nhóm dữ liệu, v.v..

LINQ là một bộ thư viện các phương thức mở rộng (extension method) hỗ trợ thực hiện hầu như tất cả các yêu cầu có thể phát sinh khi làm việc với dữ liệu tập hợp như mảng và danh sách. Tất các các phương thức LINQ chỉ sử dụng được nếu trong khối using chúng ta có lệnh `using System.Linq`.

Hãy cùng thực hiện ví dụ sau (mở rộng tiếp ví dụ ở phần trên)

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.
5.  namespace P03_ListWithCustomTypes
6.  {
7.      class Program
8.      {
9.          static void Main(string[] args)
10.         {
11.             Console.Title = "List with custom type";
12.
13.             var people = Initialize();
14.             people.Add(new Person { Name = "Theresa May", Age = 22, Country = Country.UK
15.             Console.WriteLine("World leaders:");
16.             Print(people);
17.
18.             Console.WriteLine("Sorted by age:");
19.             Print(Sort(people, "age"));
20.
21.             Console.WriteLine("Sorted by name:");
22.             Print(Sort(people, "name"));
23.
24.             Console.WriteLine("Leaders younger than 20");
25.             Print(GetYoungLeaders(people, 20));
26.
27.             Console.ReadKey();
28.         }
29.
30.         static List<Person> Initialize()
31.         {
32.             var people = new List<Person> {
33.                 new Person { Name = "Donald Trump", Age = 18, Country = Country.US },
34.                 new Person { Name = "Vladimir Putin", Age = 19, Country = Country.RU },
```

```
                new Person { Name = "Angela Merkel", Age = 20, Country = Country.DE },
                new Person { Name = "Emmanuel Macron", Age = 21, Country = Country.FR },
            };
            return people;
        }

        static void Print(List<Person> people)
        {
            foreach (var p in people)
            {
                Console.WriteLine($"- {p.Name}, {p.Age} years old, from {p.Country}");
            }
        }

        static List<Person> Sort(List<Person> people, string criteria)
        {
            if (criteria.ToLower().Equals("age"))
                return people.OrderBy(p => p.Age).ToList();
            if (criteria.ToLower().Equals("name"))
                return people.OrderBy(p => p.Name).ToList();
            return people;
        }

        static List<Person> GetYoungLeaders(List<Person> people, int age)
        {
            return people.Where(p => p.Age <= age).ToList();
        }
    }

    class Person
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public Country Country { get; set; }
    }

    enum Country
    {
        RU, VI, UK, US, DE, FR
    }
}
```



Nếu sử dụng Intellisense với một biến List<T> bạn sẽ nhận thấy số lượng phương thức có thể sử dụng lớn hơn rất nhiều so với ArrayList. Đó chính là các phương thức mở rộng của LINQ bổ sung vào kiểu List<T>. Dễ dàng nhận thấy rất nhiều phương thức có tên gần giống với truy vấn SQL. Các thao tác thường gặp nhất với dữ liệu tập hợp cũng có mặt đầy đủ trong thư viện LINQ.

> LINQ là một chủ đề hoàn toàn khác và chúng ta không trình bày chi tiết trong bài học này.

Tạm kết

Có thể nói, List<T> là giải pháp hoàn hảo thay thế cho mảng để đồng thời đảm bảo tính linh hoạt và hiệu suất. Thực tế, List<T> cũng là kiểu tập hợp được dùng rộng rãi hàng đầu trong C#.

Tiếp theo đây chúng ta sẽ xem xét loại danh sách đơn giản cuối cùng: danh sách sắp xếp (SortedList).

Danh sách sắp xếp (SortedList)

Giới thiệu chung về SortedList

SortedList là một loại *danh sách generic* (như List<T>) nhưng **lưu các cặp khóa-giá trị**, đồng thời luôn **tự động sắp xếp dữ liệu theo khóa**. Yêu cầu bắt buộc là khóa không được trùng lặp và không được nhận giá trị null.

SortedList nằm trong không gian tên `System.Collections.Generic` (tương tự List).

Dưới đây là một số phương thức và thuộc tính của SortedList:

- Add: thêm một phần tử vào cuối danh sách
- Remove: bỏ phần tử có giá trị tương ứng khỏi danh sách
- ContainsKey: xác định xem một khóa có mặt trong danh sách hay không
- ContainsValue: xác định xem một giá trị có mặt trong danh sách hay không
- IndexOfKey: trả lại chỉ số của khóa trong danh sách
- IndexOfValue: trả lại chỉ số của giá trị trong danh sách
- Thuộc tính Keys: trả lại danh sách khóa
- Values: trả lại danh sách giá trị

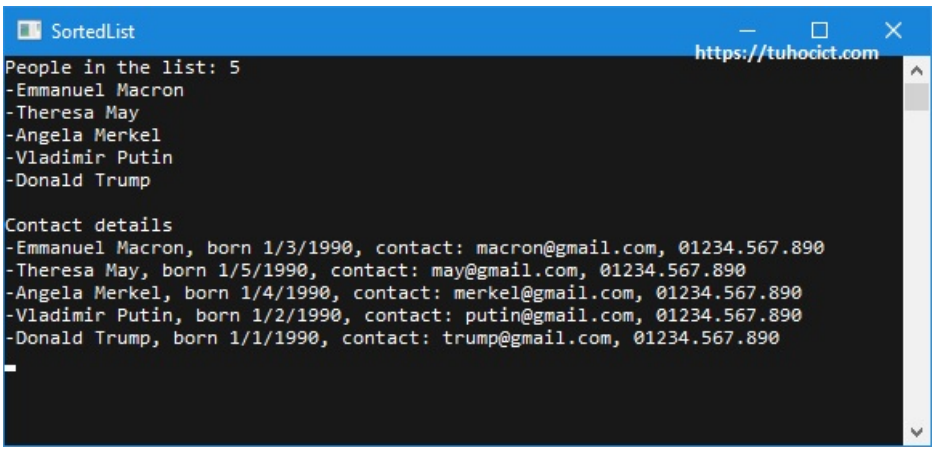
Việc truy xuất các phần tử sử dụng phép toán [] tương tự như mảng.

Ví dụ minh họa việc sử dụng SortedList trong C#

Để dễ hình dung cách làm việc với SortedList, hãy cùng thực hiện ví dụ sau:

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.
7.  namespace P04_SortedList
8.  {
9.      class Program
10.     {
```

```
11.          static void Main(string[] args)
12.          {
13.              Console.Title = "SortedList";
14.
15.              var people = CreateAddressBook();
16.              Print(people);
17.
18.              Console.ReadKey();
19.          }
20.
21.          static void Print(SortedList<string, Person> people)
22.          {
23.              Console.WriteLine($"People in the list: {people.Count}");
24.              foreach (var key in people.Keys)
25.              {
26.                  Console.WriteLine($"-{people[key].Name}");
27.              }
28.
29.              Console.WriteLine("\r\nContact details");
30.              foreach (var p in people)
31.              {
32.                  Console.WriteLine($"-{p.Value.Name}, born {p.Value.DateOfBirth.ToShortDa
33.              }
34.          }
35.
36.          static SortedList<string, Person> CreateAddressBook()
37.          {
38.              var addressBook = new SortedList<string, Person>
39.              {
40.                  {"trump", new Person { Name = "Donald Trump", DateOfBirth = new DateTime
41.                  {"putin", new Person { Name = "Vladimir Putin", DateOfBirth = new DateTi
42.                  {"macron", new Person { Name = "Emmanuel Macron", DateOfBirth = new Date
43.                  {"merkel", new Person { Name = "Angela Merkel", DateOfBirth = new DateTi
44.              };
45.
46.              addressBook.Add("may", new Person { Name = "Theresa May", DateOfBirth = new
47.
48.              return addressBook;
49.          }
50.      }
51.
52.      class Person
53.      {
54.          public string Name { get; set; }
55.          public DateTime DateOfBirth { get; set; }
56.          public string Email { get; set; }
57.          public string Phone { get; set; }
58.      }
59.  }
```



Có thể để ý thấy rằng, khi khai báo object thuộc kiểu SortedList, chúng ta phải cung cấp hai kiểu dữ liệu: TKey và TValue. TKey là kiểu dữ liệu của khóa, TValue là kiểu dữ liệu của giá trị. Chúng ta đã lựa chọn kiểu của khóa là string, kiểu của giá trị là Person.

Khi khởi tạo cũng như thêm phần tử, chúng ta phải cung cấp cả khóa và object thuộc kiểu Person.

Chúng ta cũng đã thấy, danh sách luôn được sắp xếp theo khóa. Trong trường hợp ví dụ trên, khóa là thông tin về họ được viết thường. Các chính khách trong danh sách luôn được sắp xếp theo họ.

Khi duyệt SortedList có chút khác biệt nhỏ so với List hoặc ArrayList ở chỗ, chúng ta phải duyệt từng cặp khóa/giá trị. Object cần truy xuất nằm trong phần giá trị của cặp này.

Kiểu dữ liệu Dictionary

Khái niệm

`Dictionary` là một kiểu dữ liệu tập hợp tổng quát (generic collection) tương tự như `List<T>` nhưng được dùng cho lưu trữ danh sách các cặp khóa – giá trị. Khóa và giá trị có thể thuộc bất kỳ kiểu dữ liệu nào của .NET.

Kiểu dữ liệu này được mô tả đầy đủ là `Dictionary<TKey, TValue>`, trong đó `TKey` là kiểu của khóa, `TValue` là kiểu của giá trị. Lớp Dictionary<TKey, TValue> được định nghĩa trong không gian tên `System.Collection.Generics`.

Dictionary có thể hình dung như bộ từ điển song ngữ, ví dụ, từ điển Anh – Việt, trong đó từ tiếng Anh là khóa, nghĩa trong tiếng Việt là giá trị.

Lưu ý, khi sử dụng từ điển không được phép sử dụng lặp khóa hoặc để khóa có giá trị null. Khóa bắt buộc phải là duy nhất (tương tự như trong từ điển song ngữ). Nếu trùng lặp khóa sẽ báo lỗi ở giai đoạn runtime.

Thuộc tính và phương thức của Dictionary

Một số thuộc tính quan trọng của Dictionary:

Thuộc tính	**Mô tả**
Count	Cung cấp tổng số phần tử đang có trong Dictionary<TKey,TValue>.
IsReadOnly	Trả về true nếu Dictionary<TKey,TValue> không cho phép ghi
Keys	Trả về mảng các khóa của Dictionary<TKey,TValue>.
Values	Trả về mảng các giá trị của Dictionary<TKey,TValue>.

Một số phương thức quan trọng của Dictionary:

Phương thức	**Mô tả**
Add	Thêm một phần tử mới (cặp khóa/giá trị) vào từ điển.
Remove	Xóa một phần tử khỏi từ điển (theo khóa) Dictionary<TKey, TValue>.
ContainsKey	Kiểm tra xem trong Dictionary<TKey, TValue> có chứa khóa cần tìm không.
ContainsValue	Kiểm tra xem trong Dictionary<TKey, TValue> có chứa giá trị cần tìm không.

Phương thức	Mô tả
Clear	Xóa tất cả các phần tử khỏi Dictionary<TKey, TValue>.
TryGetValue	Thử lấy một giá trị ra theo khóa. Nếu khóa tồn tại, giá trị được lấy ra theo một biến ra (out parameter). Nếu khóa không tồn tại trả về giá trị false.

Sử dụng kiểu từ điển

Khởi tạo biến từ điển

```
using System;
using System.Collections.Generic;

namespace ConsoleApp
{
    class Department
    {
        public string Name { get; set; }
        public string Office { get; set; }
    }
    internal class _12_dictionary
    {
        private static void Main(string[] args)
        {
            var dict = new Dictionary<string, Department>
            {
                ["Donald Trump"] = new Department { Name = "Sale", Office = "New York"},
                ["Bill Clinton"] = new Department { Name = "Technical", Office = "Califo
                ["George Bush"] = new Department {  Name = "Service", Office = "Alaska"}
            };

            Console.ReadKey();
        }
    }
}
```

Thêm cặp khóa/giá trị vào từ điển

```
dict.Add("Jimmy Carter", new Department { Name = "Staff", Office = "Washington DC" });
dict["Barrack Obama"] = new Department { Name = "Personnel", Office = "Hawaii" };
```

Truy xuất phần tử trong từ điển

```
Console.WriteLine($"Bill Clinton: working at {dict["Bill Clinton"].Name} department in {
```

Xóa bỏ phần tử của từ điển

```
dict.Remove("Donald Trump");
```

Duyệt danh sách phần tử

```
foreach(var i in dict)
{
    Console.WriteLine($"Key = {i.Key}, Value = {i.Value}");
}
```

Kiểm tra sự tồn tại của khóa hoặc giá trị

```
if (dict.ContainsKey("Donald Trump"))
    Console.WriteLine("Donald Trump is working here");
else
    Console.WriteLine("Donald Trump has quit the job");
```

Kết luận

Bài học này đã cung cấp cho bạn những thông tin đầy đủ về các cấu trúc danh sách trong C#.NET, bao gồm ArrayList, List<T>, SortedList<TKey, TValue> và Dictonary<TKey, TValue>. Trong đó ArrayList thuộc loại none-generic; những class còn lại thuộc loại generic.

Các cấu trúc dữ liệu này rất phổ biến và có thể thường xuyên gặp trong lập trình C#.

Chúc bạn học tốt!