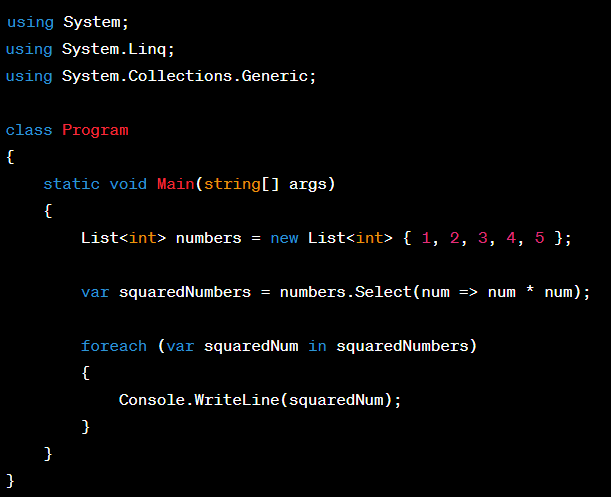
**Các hàm lọc và phân trang trong linq**

1. **Select:**

* Phương thức **Select** trong LINQ được sử dụng để chọn, ánh xạ và biến đổi dữ liệu từ một tập hợp gốc thành một tập hợp mới. Cú pháp cơ bản của phương thức **Select** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu ban đầu mà bạn muốn thực hiện phép chọn lên.
* **item** là biến để tham chiếu đến từng phần tử trong tập hợp.
* **Biểu thức biến đổi** là một biểu thức được sử dụng để biến đổi từng phần tử trong **collection**.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn tạo ra một danh sách mới chứa bình phương của từng số:



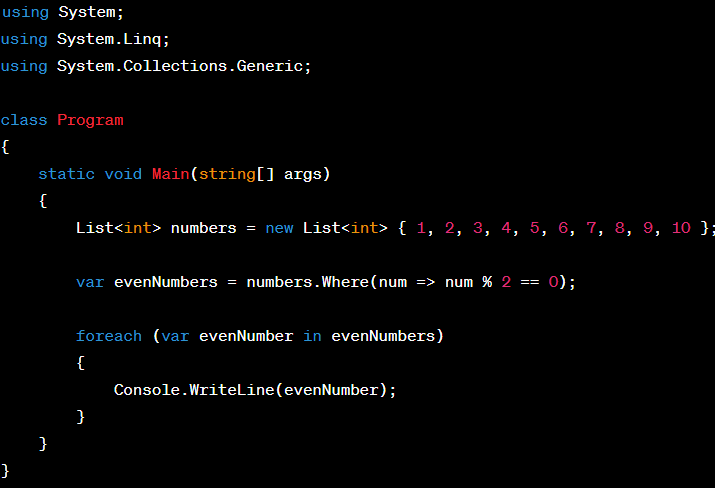
* Trong ví dụ trên, phương thức **Select** được sử dụng để biến đổi từng số trong danh sách **numbers** bằng cách tính bình phương của chúng. Kết quả của phép biến đổi được lưu trong tập hợp mới **squaredNumbers**, và sau đó nó được duyệt qua để in ra kết quả.
* **Select** có thể được sử dụng để thực hiện nhiều loại biến đổi khác nhau trên các tập hợp dữ liệu, giúp bạn trích xuất thông tin cần thiết và tạo ra các tập hợp mới dựa trên dữ liệu gốc.

1. **Where:**

* Phương thức **Where** trong LINQ được sử dụng để lọc các phần tử trong tập hợp dữ liệu dựa trên một điều kiện đã cho. Phương thức này cho phép bạn chọn ra các phần tử thoả mãn điều kiện cụ thể và tạo ra một tập hợp con mới chứa các phần tử đã lọc. Cú pháp cơ bản của phương thức **Where** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu ban đầu mà bạn muốn thực hiện việc lọc.
* **item** là biến để tham chiếu đến từng phần tử trong tập hợp.
* **Điều kiện** là biểu thức logic được sử dụng để kiểm tra xem phần tử có thoả mãn điều kiện lọc không.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn lọc ra các số chẵn:



- Trong ví dụ trên, phương thức **Where** được sử dụng để lọc ra các số chẵn từ danh sách **numbers** bằng cách kiểm tra xem số đó có chia hết cho 2 hay không. Các số thoả mãn điều kiện được lưu trong tập hợp mới **evenNumbers**, và sau đó nó được duyệt qua để in ra kết quả.

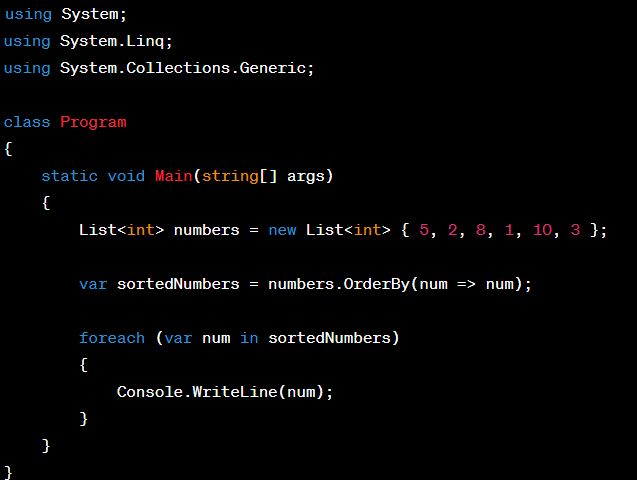
- **Where** có thể được sử dụng để lọc dữ liệu dựa trên nhiều điều kiện khác nhau, giúp bạn trích xuất và lựa chọn các phần tử theo yêu cầu.

1. **OderBy:**

* Phương thức **OrderBy** trong LINQ được sử dụng để sắp xếp các phần tử trong một tập hợp dữ liệu theo một thứ tự cụ thể. Phương thức này cho phép bạn chọn một trường hoặc biểu thức để sắp xếp các phần tử trong tập hợp. Cú pháp cơ bản của phương thức **OrderBy** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu ban đầu mà bạn muốn thực hiện việc sắp xếp.
* **item** là biến để tham chiếu đến từng phần tử trong tập hợp.
* **Trường** sắp xếp là biểu thức mô tả trường hoặc giá trị mà bạn muốn sử dụng để sắp xếp.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn sắp xếp chúng theo thứ tự tăng dần:



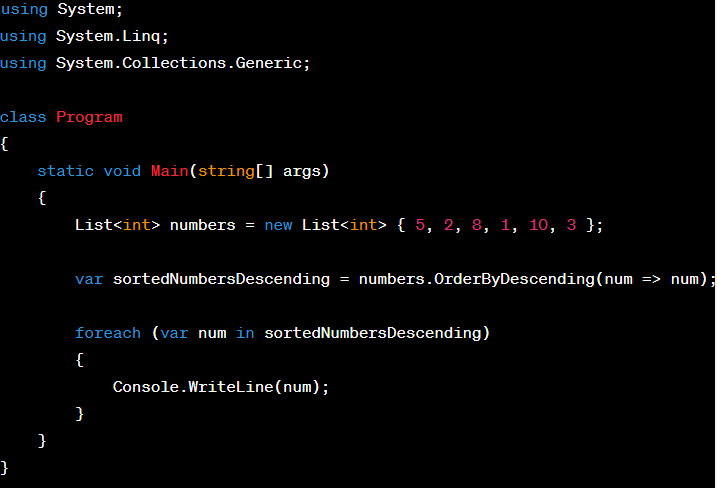
* Trong ví dụ trên, phương thức **OrderBy** được sử dụng để sắp xếp các số trong danh sách **numbers** theo thứ tự tăng dần. Kết quả sau khi sắp xếp được lưu trong tập hợp mới sortedNumbers, và sau đó nó được duyệt qua để in ra kết quả.
* **OrderBy** cũng có thể được sử dụng để sắp xếp dữ liệu theo thứ tự giảm dần bằng cách sử dụng **OrderByDescending** thay vì OrderBy

1. **OrderByDescending:**

* Phương thức **OrderByDescending** trong LINQ được sử dụng để sắp xếp các phần tử trong một tập hợp dữ liệu theo thứ tự giảm dần. Tương tự như phương thức OrderBy, phương thức **OrderByDescending** cũng cho phép bạn chọn một trường hoặc biểu thức để sắp xếp các phần tử trong tập hợp.
* Cú pháp cơ bản của phương thức **OrderByDescending** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu ban đầu mà bạn muốn thực hiện việc sắp xếp.
* **item** là biến để tham chiếu đến từng phần tử trong tập hợp.
* **Trường** sắp xếp là biểu thức mô tả trường hoặc giá trị mà bạn muốn sử dụng để sắp xếp.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn sắp xếp chúng theo thứ tự giảm dần:



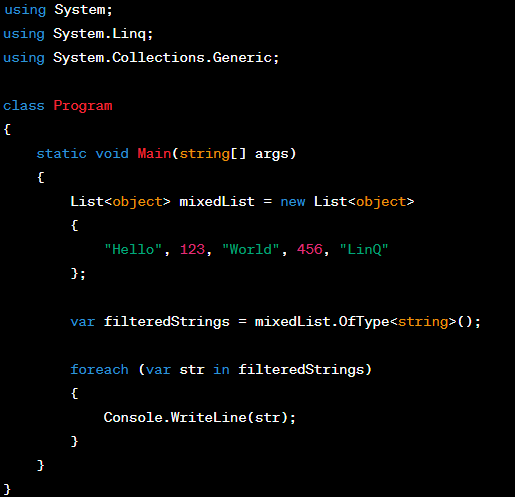
* Trong ví dụ trên, phương thức **OrderByDescending** được sử dụng để sắp xếp các số trong danh sách numbers theo thứ tự giảm dần. Kết quả sau khi sắp xếp được lưu trong tập hợp mới **sortedNumbersDescending**, và sau đó nó được duyệt qua để in ra kết quả.
* Phương thức **OrderByDescending** giúp bạn sắp xếp dữ liệu theo thứ tự ngược lại so với phương thức OrderBy.

1. **OfType:**

* Phương thức **OfType** trong LINQ được sử dụng để lọc các phần tử trong một tập hợp dữ liệu chỉ định về kiểu cụ thể. Phương thức này cho phép bạn lấy ra những phần tử trong tập hợp gốc mà có thể được ép kiểu (cast) thành kiểu cụ thể mà bạn quan tâm.
* Cú pháp cơ bản của phương thức **OfType** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn thực hiện việc lọc.
* **T** là kiểu dữ liệu mà bạn muốn lọc.
* Ví dụ, giả sử bạn có một danh sách chứa các đối tượng có kiểu dữ liệu khác nhau, và bạn muốn lọc ra chỉ những đối tượng có kiểu string:



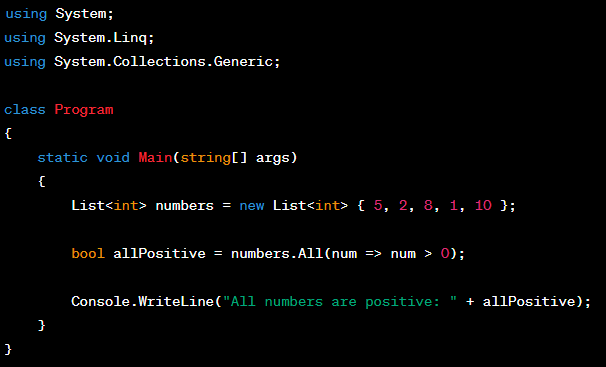
* Trong ví dụ trên, phương thức **OfType<string>()** được sử dụng để lọc ra chỉ những phần tử có kiểu string từ danh sách mixedList. Kết quả sau khi lọc được lưu trong tập hợp mới **filteredStrings**, và sau đó nó được duyệt qua để in ra kết quả.
* Phương thức **OfType** giúp bạn tách ra các phần tử có kiểu cụ thể từ một danh sách chứa các kiểu khác nhau, và loại bỏ các phần tử không thể ép kiểu thành kiểu mong muốn.

1. **All:**

* Phương thức **All** trong LINQ được sử dụng để kiểm tra xem tất cả các phần tử trong một tập hợp dữ liệu có thoả mãn một điều kiện cụ thể hay không.
* Phương thức này trả về true nếu tất cả các phần tử đều thoả mãn điều kiện, và trả về false nếu ít nhất một phần tử không thoả mãn điều kiện. Cú pháp cơ bản của phương thức **All** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn kiểm tra.
* **item** là biến để tham chiếu đến từng phần tử trong tập hợp.
* **Điều kiện** là biểu thức logic để kiểm tra xem phần tử có thoả mãn điều kiện hay không.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn kiểm tra xem tất cả các số trong danh sách có lớn hơn 0 không:



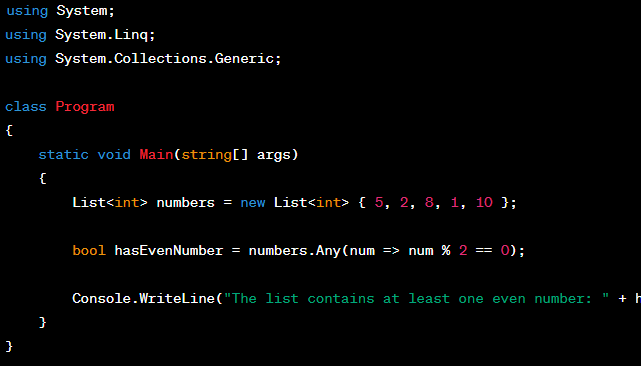
* Trong ví dụ trên, phương thức **All** được sử dụng để kiểm tra xem tất cả các số trong danh sách numbers có lớn hơn 0 hay không. Kết quả của việc kiểm tra được gán vào biến allPositive, và sau đó kết quả được in ra màn hình.
* Phương thức **All** là một cách tiện lợi để kiểm tra tính đúng đắn của tất cả các phần tử trong một tập hợp dữ liệu theo một điều kiện cụ thể.

1. **Any:**

* Phương thức **Any** trong LINQ được sử dụng để kiểm tra xem có ít nhất một phần tử trong một tập hợp dữ liệu thoả mãn một điều kiện cụ thể hay không. Phương thức này trả về **true** nếu ít nhất một phần tử thoả mãn điều kiện, và trả về **false** nếu không có phần tử nào thoả mãn điều kiện. Cú pháp cơ bản của phương thức **Any** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn kiểm tra.
* **item** là biến để tham chiếu đến từng phần tử trong tập hợp.
* **Điều kiện** là biểu thức logic để kiểm tra xem phần tử có thoả mãn điều kiện hay không.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn kiểm tra xem có ít nhất một số chẵn trong danh sách hay không:



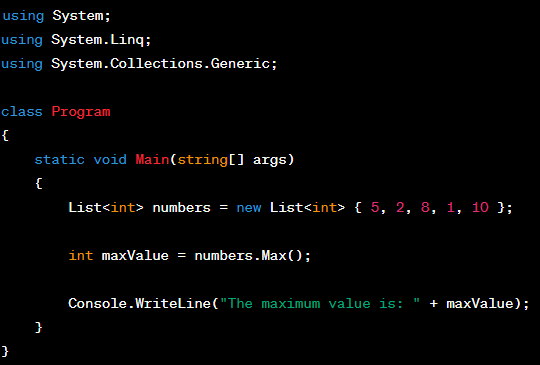
* Trong ví dụ trên, phương thức **Any** được sử dụng để kiểm tra xem có ít nhất một số chẵn trong danh sách **numbers** hay không. Kết quả của việc kiểm tra được gán vào biến **hasEvenNumber**, và sau đó kết quả được in ra màn hình.
* Phương thức **Any** là một cách tiện lợi để kiểm tra sự tồn tại của ít nhất một phần tử thoả mãn một điều kiện trong tập hợp dữ liệu.

1. **Max,Min:**

* Trong LINQ, có hai phương thức **Max** và **Min** được sử dụng để tìm giá trị lớn nhất và nhỏ nhất trong một tập hợp dữ liệu, tương ứng. Dưới đây là cách bạn có thể sử dụng chúng:
* **Phương thức Max**:
* Phương thức Max được sử dụng để tìm giá trị lớn nhất trong một tập hợp dữ liệu.



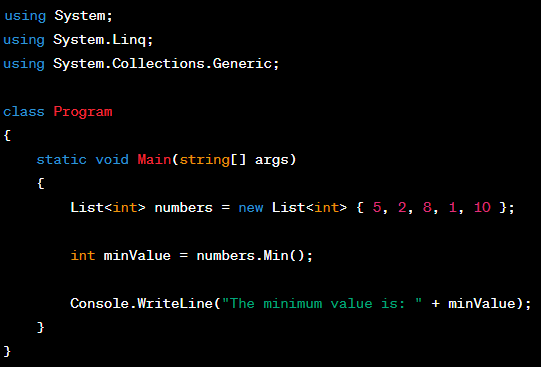
* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn tìm giá trị lớn nhất.
* **maxValue** là giá trị lớn nhất trong tập hợp.
* Ví dụ, bạn có một danh sách các số nguyên và bạn muốn tìm số lớn nhất trong danh sách:



* Phương thức **Min**:
* Phương thức **Min** được sử dụng để tìm giá trị nhỏ nhất trong một tập hợp dữ liệu.



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn tìm giá trị nhỏ nhất.
* **minValue** là giá trị nhỏ nhất trong tập hợp.
* Ví dụ, bạn có một danh sách các số nguyên và bạn muốn tìm số nhỏ nhất trong danh sách:



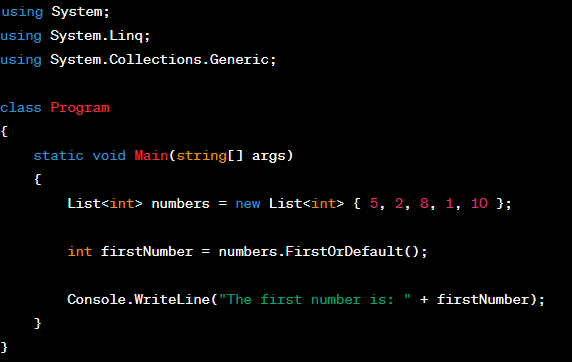
* Cả hai phương thức **Max** và **Min** đều trả về giá trị tương ứng mà bạn đang tìm kiếm.

1. **FirstOrDefault:**

* Phương thức **FirstOrDefault** trong LINQ được sử dụng để trả về phần tử đầu tiên trong một tập hợp dữ liệu hoặc trả về một giá trị mặc định nếu tập hợp không có phần tử nào.
* Phương thức này thường được sử dụng khi bạn muốn truy xuất phần tử đầu tiên trong tập hợp hoặc khi bạn không chắc chắn liệu tập hợp có chứa phần tử hay không.
* Cú pháp cơ bản của phương thức **FirstOrDefault** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn truy xuất phần tử đầu tiên.
* **firstItem** là phần tử đầu tiên trong tập hợp hoặc giá trị mặc định nếu tập hợp không có phần tử nào.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn truy xuất phần tử đầu tiên trong danh sách:



* Nếu tập hợp không có phần tử nào, phương thức **FirstOrDefault** sẽ trả về giá trị mặc định của kiểu dữ liệu tương ứng (ví dụ: 0 cho kiểu int, null cho kiểu tham chiếu, ...). Để xác định giá trị mặc định khác, bạn có thể cung cấp tham số thứ hai cho phương thức **FirstOrDefault**, như sau:



* Ví dụ, nếu bạn muốn trả về -1 nếu tập hợp không có phần tử nào:



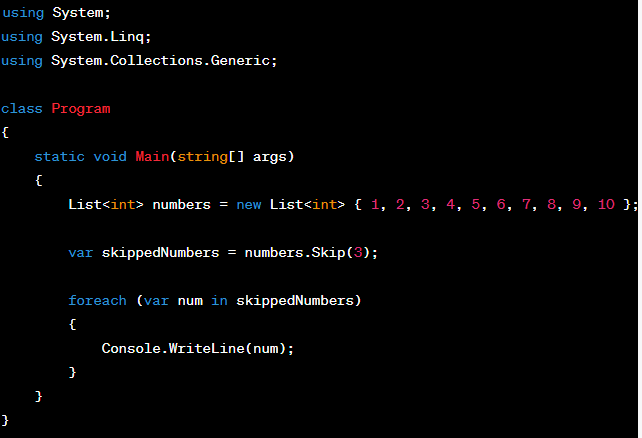
* Phương thức **FirstOrDefault** thường được sử dụng khi bạn muốn truy xuất phần tử đầu tiên trong tập hợp mà không cần kiểm tra trước liệu tập hợp có phần tử hay không.

1. **Skip**:

* Phương thức **Skip** trong LINQ được sử dụng để bỏ qua một số lượng cụ thể các phần tử đầu tiên trong một tập hợp dữ liệu và trả về các phần tử còn lại sau số phần tử đã bỏ qua.
* Phương thức này thường được sử dụng để thực hiện phân trang (paging) hoặc lấy một phần của dữ liệu từ tập hợp lớn.
* Cú pháp cơ bản của phương thức **Skip** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn bỏ qua phần tử đầu tiên.
* **count** là số lượng phần tử mà bạn muốn bỏ qua.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn bỏ qua ba số đầu tiên trong danh sách:



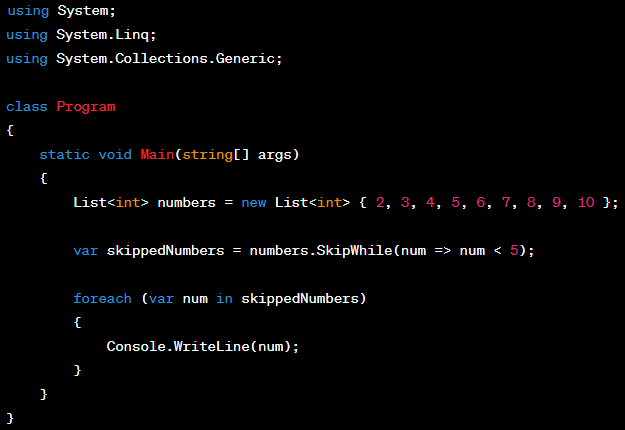
* Trong ví dụ trên, phương thức **Skip(3)** được sử dụng để bỏ qua ba số đầu tiên trong danh sách numbers. Các số còn lại sau phần tử thứ ba được lưu trong tập hợp mới **skippedNumbers**, và sau đó nó được duyệt qua để in ra kết quả.
* Phương thức **Skip** giúp bạn lấy một phần của tập hợp dữ liệu dựa trên số lượng phần tử bạn muốn bỏ qua.

1. **SkipWhile:**

* Phương thức **SkipWhile** trong LINQ được sử dụng để bỏ qua các phần tử đầu tiên trong một tập hợp dữ liệu dựa trên một điều kiện cụ thể.
* Phương thức này sẽ bỏ qua các phần tử cho đến khi điều kiện không còn đúng, sau đó trả về các phần tử còn lại trong tập hợp.
* Cú pháp cơ bản của phương thức **SkipWhile** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn bỏ qua phần tử đầu tiên.
* **item** là biến để tham chiếu đến từng phần tử trong tập hợp.
* Điều kiện là biểu thức logic để kiểm tra xem có tiếp tục bỏ qua các phần tử hay không. Khi điều kiện trả về **true**, phần tử đó sẽ bị bỏ qua. Khi điều kiện trả về **false**, việc bỏ qua dừng lại và các phần tử còn lại sẽ được trả về.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn bỏ qua các số nhỏ hơn 5 trong danh sách:



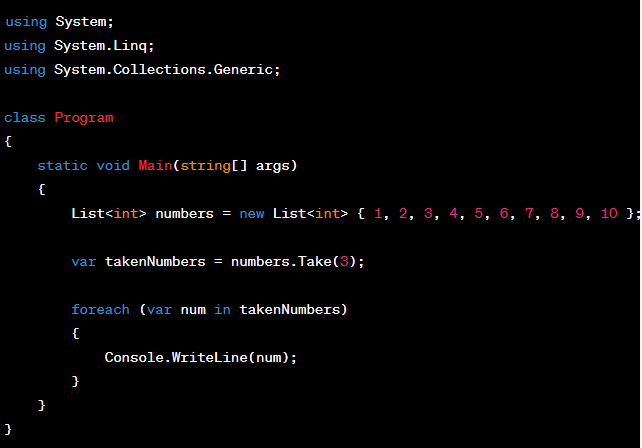
* Trong ví dụ trên, phương thức **SkipWhile** được sử dụng để bỏ qua các số nhỏ hơn 5 trong danh sách numbers. Khi đạt đến số 5, điều kiện trả về false và việc bỏ qua dừng lại. Các số còn lại sau số 5 được lưu trong tập hợp mới **skippedNumbers**, và sau đó nó được duyệt qua để in ra kết quả.
* Phương thức **SkipWhile** giúp bạn bỏ qua các phần tử đầu tiên trong tập hợp dữ liệu dựa trên một điều kiện cho đến khi điều kiện không còn đúng nữa.

1. **Take:**

* Phương thức Take trong LINQ được sử dụng để lấy một số lượng cụ thể các phần tử đầu tiên từ một tập hợp dữ liệu và trả về chúng dưới dạng một tập hợp mới.
* Phương thức này thường được sử dụng để thực hiện phân trang (paging) hoặc lấy một phần của dữ liệu từ tập hợp lớn.
* Cú pháp cơ bản của phương thức **Take** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn lấy phần tử đầu tiên.
* **count** là số lượng phần tử mà bạn muốn lấy.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn lấy ba số đầu tiên trong danh sách:



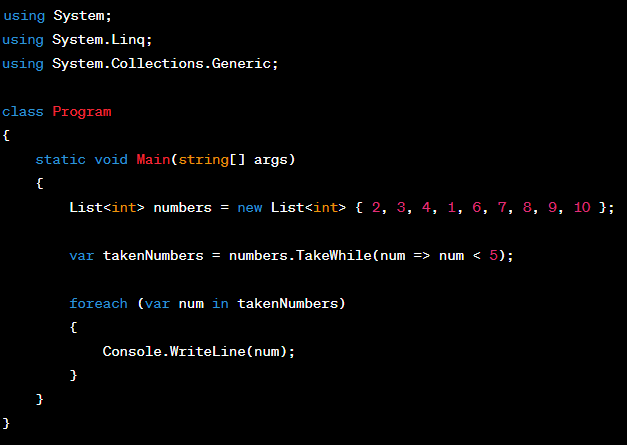
* Trong ví dụ trên, phương thức **Take**(3) được sử dụng để lấy ba số đầu tiên trong danh sách **numbers**. Ba số đầu tiên sẽ được lưu trong tập hợp mới **takenNumbers**, và sau đó nó được duyệt qua để in ra kết quả.
* Phương thức **Take** giúp bạn lấy một phần của tập hợp dữ liệu dựa trên số lượng phần tử bạn muốn lấy.

1. **TakeWhile:**

* Phương thức TakeWhile trong LINQ được sử dụng để lấy các phần tử đầu tiên từ một tập hợp dữ liệu dựa trên một điều kiện cụ thể.
* Phương thức này sẽ lấy các phần tử cho đến khi điều kiện không còn đúng, sau đó trả về các phần tử đã lấy dưới dạng một tập hợp mới.
* Cú pháp cơ bản của phương thức **TakeWhile** như sau:



* **collection** là tập hợp dữ liệu mà bạn muốn lấy các phần tử đầu tiên.
* **item** là biến để tham chiếu đến từng phần tử trong tập hợp.
* **Điều kiện** là biểu thức logic để kiểm tra xem có tiếp tục lấy các phần tử hay không. Khi điều kiện trả về true, phần tử đó sẽ được lấy. Khi điều kiện trả về false, việc lấy dừng lại và các phần tử đã lấy sẽ được trả về.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn lấy các số nhỏ hơn 5 từ đầu của danh sách:



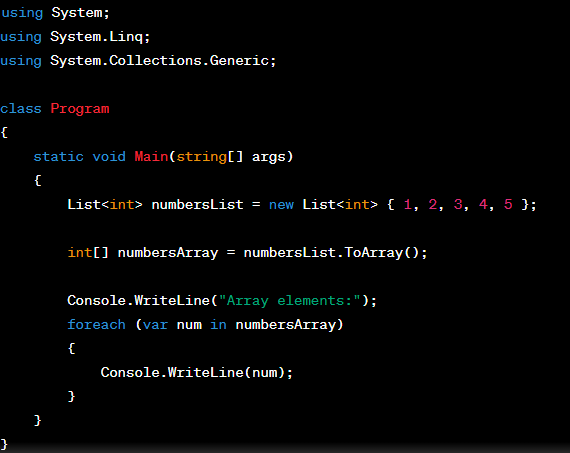
* Trong ví dụ trên, phương thức **TakeWhile** được sử dụng để lấy các số nhỏ hơn 5 từ đầu của danh sách numbers. Khi đạt đến số 6, điều kiện trả về false và việc lấy dừng lại. Các số đã lấy trước đó được lưu trong tập hợp mới **takenNumbers**, và sau đó nó được duyệt qua để in ra kết quả.
* Phương thức **TakeWhile** giúp bạn lấy các phần tử đầu tiên trong tập hợp dữ liệu dựa trên một điều kiện cho đến khi điều kiện không còn đúng nữa.

1. **ToDictionary:**

* Phương thức **ToArray** trong LINQ được sử dụng để chuyển đổi một tập hợp dữ liệu (ví dụ: danh sách, trình lặp, hoặc tập hợp LINQ) thành một mảng.
* Phương thức này rất hữu ích khi bạn muốn làm việc với mảng thay vì một tập hợp LINQ hoặc một loại dữ liệu khác.
* Cú pháp cơ bản của phương thức **ToArray** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn chuyển đổi thành mảng.
* **array** là mảng chứa các phần tử từ tập hợp dữ liệu.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn chuyển đổi nó thành một mảng:



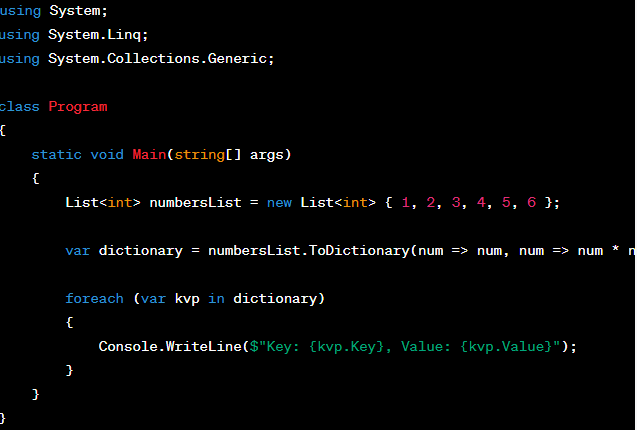
* Trong ví dụ trên, phương thức **ToArray** được sử dụng để chuyển đổi danh sách **numbersList** thành một mảng numbersArray. Sau đó, mảng được duyệt qua để in ra từng phần tử.
* Phương thức **ToArray** giúp bạn chuyển đổi dữ liệu từ tập hợp LINQ hoặc một loại dữ liệu khác thành một mảng, giúp dễ dàng thực hiện các thao tác mảng truyền thống.

1. **ToDictionary:**

* Phương thức **ToDictionary** trong LINQ được sử dụng để chuyển đổi một tập hợp dữ liệu thành một từ điển (**dictionary**) trong C#.
* Phương thức này cho phép bạn tạo một từ điển từ các phần tử của tập hợp, sử dụng một khóa duy nhất để xác định mỗi phần tử và một giá trị tương ứng.
* Cú pháp cơ bản của phương thức **ToDictionary** như sau:



* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn chuyển đổi thành từ điển.
* **keySelector** là một biểu thức delegate hoặc lambda expression để xác định khóa của từng phần tử.
* **valueSelector** là một biểu thức delegate hoặc **lambda** **expression** để xác định giá trị tương ứng với từng phần tử.
* Ví dụ, giả sử bạn có một danh sách các số nguyên và bạn muốn chuyển đổi nó thành một từ điển, với số chẵn làm khóa và bình phương của số đó làm giá trị:



* Trong ví dụ trên, phương thức **ToDictionary** được sử dụng để chuyển đổi danh sách **numbersList** thành một từ điển, với số chẵn làm khóa và bình phương của số đó làm giá trị. Kết quả là mỗi phần tử trong danh sách được thêm vào từ điển với khóa và giá trị tương ứng.
* Phương thức **ToDictionary** là một cách tiện lợi để chuyển đổi tập hợp dữ liệu thành một từ điển với các khóa và giá trị tùy chỉnh.

1. **ToList:**

* Phương thức **ToList** trong LINQ được sử dụng để chuyển đổi một tập hợp dữ liệu (ví dụ: danh sách, trình lặp, hoặc tập hợp LINQ) thành một danh sách (List) trong C#.
* Phương thức này cho phép bạn làm việc với dữ liệu dưới dạng danh sách thay vì tập hợp LINQ hoặc một loại dữ liệu khác.
* Cú pháp cơ bản của phương thức **ToList** như sau:
* Trong đó:
* **collection** là tập hợp dữ liệu mà bạn muốn chuyển đổi thành danh sách.
* **list** là danh sách chứa các phần tử từ tập hợp dữ liệu.
* Ví dụ, giả sử bạn có một trình lặp và bạn muốn chuyển đổi nó thành một danh sách:



* Trong ví dụ trên, phương thức **ToList** được sử dụng để chuyển đổi trình lặp **numberIterator** thành một danh sách **numberList**. Sau đó, danh sách được duyệt qua để in ra từng phần tử.
* Phương thức **ToList** giúp bạn chuyển đổi dữ liệu từ tập hợp LINQ hoặc một loại dữ liệu khác thành một danh sách, giúp dễ dàng thực hiện các thao tác danh sách truyền thống.