Bài 10.2: Tìm hiểu lớp List<T>

- ✓ Tổng quan về List<T>
- ✓ Các thuộc tính
- ✓ Các phương thức thường dùng
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Tổng quan về List<T>

- ✓Đại diện cho một danh sách các đối tượng cùng kiểu T nào đó.
- ✓Có thể truy cập bằng chỉ số.
- ✓Cung cấp sẵn các phương thức hỗ trợ sắp xếp, tìm kiếm, quản lý các phần tử trong list.
- ✓Kích thước của đối tượng sinh ra từ List<T> có thể tự động co dãn tùy theo hành vi cần thực hiện.
- ✓Cụ thể nếu ta thêm mới đối tượng vào list, kích thước của list sẽ tự động tăng lên nếu cần thêm chỗ chứa phần tử mới.
- ✓Ngược lại, nếu ta xóa bớt phần tử trong list, kích thước của list sẽ tự động giảm xuống cho phù hợp.
- ✓Các phần tử trong list theo mặc định có thứ tự giống như thứ tự ta thêm chúng vào list.
- ✓Các phần tử trong list có thể trùng lặp và có thể null.
- ✓List<T> và ArrayList có cùng chức năng nhưng khuyến nghị nên dùng List<T>.

Các thuộc tính

- ✓Capacity: dùng để lấy hoặc thiết lập số lượng phần tử mà cấu trúc dữ liệu nội tại bên trong list có thể chứa trước khi cần co dãn kích thước của nó. Capacity luôn lớn hơn Count.
- ✓Count: số lượng phần tử hiện có trong list, thuộc tính chỉ đọc. Nếu Count > Capacity thì việc cấp phát lại bộ nhớ sẽ xảy ra để đảm bảo có đủ chỗ cho các phần tử cũ và phần tử mới thêm vào list. Lúc này các phần tử cũ được sao chép sang mảng nội tại trước, sau đó mới đến lượt phần tử mới thêm vào.
- ✓Item[index]: lấy ra hoặc thiết lập giá trị cho phần tử tại vị trí index trong mảng nội tại của list kiểu T.

Các phương thức khởi tạo

- ✓List<T>(): tạo một đối tượng của List<T> rỗng với Capacity = 0.
- ✓List<T>(IEnumerable<T>): khởi tạo một đối tượng mới chứa các phần tử sao chép từ tập hợp cung cấp trong đối số. Số lượng phần tử, thứ tự phần tử sau khi sao chép bằng đúng số phần tử, thứ thự như trong tập hợp cung cấp bởi tham số. Xảy ra ngoại lệ nếu tham số là null.
- ✓List<T>(Int32): tạo một đối tượng mới với 0 phần tử và có Capacity khởi tạo bằng giá trị cung cấp trong tham số. Xảy ra ngoại lệ nếu tham số < 0.

Ví dụ về List<T>

```
Console.OutputEncoding = Encoding.UTF8;
List<int> list1 = new List<int>();
List<string> list2 = new List<string>(new string[] {"Loan", "Phong", "Minh"});
List<float> list3 = new List<float>(20);
Console.WriteLine("Khả năng lưu trữ của list1: " + list1.Capacity);
Console.WriteLine("Số lượng phần tử của list1: " + list1.Count);
Console.WriteLine("Khả năng lưu trữ của list2: " + list2.Capacity);
Console.WriteLine("Số lượng phần tử của list2: " + list2.Count);
Console.WriteLine("Khả năng lưu trữ của list3: " + list3.Capacity);
Console.WriteLine("Số lượng phần tử của list3: " + list3.Count);
```

```
C:\WINDOWS\system32\cmd.exe
Khả năng lưu trữ của list1: 0
Số lượng phần tử của list1: 0
Khả năng lưu trữ của list2: 3
Số lượng phần tử của list2: 3
Khả năng lưu trữ của list3: 20
Số lượng phần tử của list3: 0
```

Các phương thức thường dùng

Phương thức và mô tả
Add(T item); thêm mới đối tượng vào cuối danh sách hiện tại. Nếu số phần tử trước khi thêm đã đạt giới hạn Count == Capacity thì việc resize sẽ xảy ra.
AddRange(IEnumerable<T> collection); thêm các phần tử của tham số vào cuối danh sách hiện thời. Xảy ra ngoại lệ ArgumentNullException nếu tham số null. Các phần tử được sao chép và lưu đúng thứ tự như nguyên bản trong tham số.
AsReadOnly(); trả về một đối tượng của ReadOnlyCollection<T> từ đối tượng hiện tại. Sử dụng trong trường hợp không cho phép sửa đổi dữ liệu trong tập hợp.
BinarySearch(T item); tìm trong toàn bộ tập hợp hiện thời xem có tồn tại item không. Sử dụng comparer mặc định. Trả về vị trí phần tử đầu tiên tìm thấy hoặc trả về số âm nếu không tìm thấy. **BinarySearch(T item, IComparer<T> comparer);** giống phương thức trên ngoại trừ việc sử dụng comparer được cung cấp. Nếu cung cấp comparer là null thì phương thức sẽ sử dụng trình so sánh Comparer mặc định. **BinarySearch(int index, int count, T item, IComparer<T> comparer);** tìm trong count phần tử từ vị trí index xem có xuất hiện item không. Sử dụng comparer được cung cấp để so sánh. Trả về chỉ số phần tử đầu tiên tìm được hoặc số âm nếu không tìm thấy. Nếu comparer là null thì sử dụng trình so sánh comparer mặc định. Xảy ra ngoại lệ nếu bất kì 1 trong hai tham số đầu < 0. Tất cả các phương thức tìm kiếm nêu trên đều yêu cầu tập hợp đã được sắp xếp theo tiêu chí của comparer được triển khai. Nếu không kết quả tìm kiếm có thể không đúng.
Clear(); xóa tất cả các phần tử trong list. Capacity không bị reset.
Contains(T item); kiểm tra xem phần tử item có tồn tại trong tập hợp không. Phương thức này sử dụng trình so sánh mặc định, là triển khai của IEquatable<T>.Equals() để so sánh sự tương đương.
ConvertAll(Converter<T, TOutput> converter); chuyển tất cả các phần tử kiểu T trong danh sách sang kiểu TOuput rồi trả về danh sách các phần tử ở kiểu TOuput.

Các phương thức thường dùng

CopyTo(T[] arr, int index); copy toàn bộ các phần tử của danh sách hiện tại sang một mảng kiểu T, bắt đầu từ vị trí index trong mảng đích. Các phần tử được copy giữ nguyên thứ tự như trong tập hợp gốc. Xảy ra ngoại lệ nếu arr null hoặc không đủ chỗ chứa các phần tử cần copy sang tính từ index đến cuối mảng arr, hoặc khi index âm. **CopyTo(int index, T[] array, int arrIndex, int count);** copy một lượng count phần tử bắt đầu từ vị trí index trong tập hợp gốc sang mảng arr bắt đầu từ vị trí arrIndex. Xảy ra ngoại lệ nếu arr null, index hoặc arrIndex âm hoặc count âm hoặc index >= Count của tập hợp hiện tại hoặc số phần tử từ arrIndex không đủ để chứa count phần tử cần sao chép sang. **CopyTo(T[]);** copy toàn bộ tập hợp sang mảng arr, bắt đầu từ vị trí phần tử đầu tiên.
EnsureCapacity(int capacity); dùng để điều chỉnh khả năng lưu trữ của tập hợp về tối thiểu là capacity phần tử.
Exists(Predicate<T> match); dùng để kiểm tra xem tập hợp có các phần tử thỏa mãn điều kiện đưa ra bởi tham số match không. Trả về true nếu tập hợp có 1 hoặc nhiều phần tử thỏa mãn và false nếu không có phần tử nào. Xảy ra ngoại lệ nếu match là null.
Find(Predicate<T> match); tìm một phần tử thỏa mãn điều kiện chỉ ra trong tham số và trả về phần tử đầu tiên tìm được. Nếu không tìm thấy trả về giá trị mặc định của kiểu T. Xảy ra ngoại lệ nếu match là null. Các biến thể của nó: **FindAll(Predicate<T> match);** tìm và trả về list các phần tử của tập hợp hiện thời thỏa mãn điều kiện chỉ ra trong tham số match. **FindLast(Predicate<T> match);** tìm và trả về phần tử cuối thỏa mãn điều kiện trong tham số. Nếu không tìm thấy, trả về giá trị mặc định của kiểu T. **FindIndex(Predicate<T> match);** tìm và trả về chỉ số phần tử đầu tiên thỏa mãn điều kiện cho trước.

Các phương thức thường dùng

FindLastIndex(Predicate<T> match); tìm và trả về vị trí của phần tử cuối thỏa mãn điều kiện chỉ ra trong tham số. Trả về -1 nếu không tìm thấy. **FindLastIndex(int startIndex, Predicate<T> match);** giống phương thức trên nhưng tìm từ vị trí startIndex. **FindIndex(int startIndex, int count, Predicate<T> match);** tìm vị trí phần tử đầu thỏa mãn điều kiện trong tham số. Việc tìm kiếm diễn ra trong count phần tử bắt đầu từ vị trí startIndex. Trả về -1 nếu không tìm thấy và giá trị >= 0 là chỉ số phần tử đầu tiên tìm được. **FindLastIndex(int startIndex, int count, Predicate<T> match);** tìm vị trí phần tử cuối thỏa mãn điều kiện trong match bắt đầu từ vị trí startIndex, trong count phần tử. Trả về vị trí phần tử tìm được hoặc -1 nếu không tìm thấy.
ForEach(Action<T> action); thực hiện một hành động cụ thể trên mỗi phần tử của tập hợp. Xảy ra ngoại lệ nếu action là null.
GetEnumerator(); trả về một Enumerator lặp trên các phần tử của tập hợp.
GetRange(int index, int count); tạo mọt bản sao của một khoảng các phần tử từ tập hợp hiện tại tính từ vị trí index đến khi đủ count phần tử. Xảy ra ngoại lệ nếu các tham số âm hoặc vượt quá biên của tập hợp.
IndexOf(T item, int index); trả về chỉ số phần tử đầu tiên có giá trị tương đương item trong tập hợp. Việc tìm kiếm bắt đầu từ vị trí index đến hết tập hợp. Trả về -1 nếu không tìm thấy. Ngoại lệ nếu index ngoài biên tập hợp. **IndexOf(T item, int index, int count);** tìm đối tượng item trong tập hợp, bắt đầu từ vị trí index, tìm trong count phần tử. Trả về chỉ số của đối tượng đầu tiên thỏa mãn. Nếu không tìm thấy, trả về -1. Xảy ra ngoại lệ nếu index, count vượt biên tập hợp hoặc nhận các giá trị không hợp lệ. **IndexOf(T item);** tìm và trả về chỉ số của đối tượng item trong tập hợp hiện tại. Trả về -1 nếu không tìm thấy. Phương thức này triển khai tìm kiếm tuyến tính.

Các phương thức thường dùng

Insert(int index, T item); chèn phần tử item vào tập hợp tại vị trí index. Xảy ra ngoại lệ nếu index vượt biên tập hợp(âm hoặc index > Count).
InsertRange(int index, IEnumerable<T> collection); chèn các phần tử của collection vào tập hợp hiện thời tại vị trí index. Xảy ra ngoại lệ nếu collection null hoặc index ngoài biên tập hợp.
LastIndexOf(T item, int index); trả về chỉ số phần tử cuối cùng có giá trị tương đương item trong tập hợp. Việc tìm kiếm bắt đầu từ vị trí index đến hết tập hợp. Trả về -1 nếu không tìm thấy. Ngoại lệ nếu index ngoài biên tập hợp. **LastIndexOf(T item, int index, int count);** tìm đối tượng item trong tập hợp, bắt đầu từ vị trí index, tìm trong count phần tử. Trả về chỉ số của đối tượng cuối cùng thỏa mãn. Nếu không tìm thấy, trả về -1. Xảy ra ngoại lệ nếu index, count vượt biên tập hợp hoặc nhận các giá trị không hợp lệ. **LastIndexOf(T item);** tìm và trả về chỉ số của đối tượng item cuối cùng xuất hiện trong tập hợp hiện tại. Trả về -1 nếu không tìm thấy. Phương thức này triển khai tìm kiếm tuyến tính.
Remove(T item); xóa phần tử đầu tiên trong tập hợp hiện thời có giá trị trùng với item. Trả về true nếu xóa thành công và false nếu thất bại.
RemoveAll(Predicate<T> match); xóa tất cả các phần tử thỏa mãn điều kiện trong tham số. trả về số phần tử bị xóa khỏi tập hợp. Xảy ra ngoại lệ nếu match null.
RemoveAt(int index); xóa phần tử tại vị trí index trong tập hợp. Xảy ra ngoại lệ nếu index vượt biên tập hợp.
RemoveRange(int index, int count); xóa một khoảng count phần tử bắt đầu từ vị trí index trong tập hợp. Xảy ra ngoại lệ nếu index ngoài biên tập hợp.
Reverse(); đảo ngược toàn bộ tập hợp. **Reverse(int index, int count);** đảo ngược count phần tử từ vị trí index trong tập hợp. xảy ra ngoại lệ nếu index vượt biên tập hợp hoặc index, count không chỉ ra khoảng hợp lệ trong tập hợp.

Các phương thức thường dùng

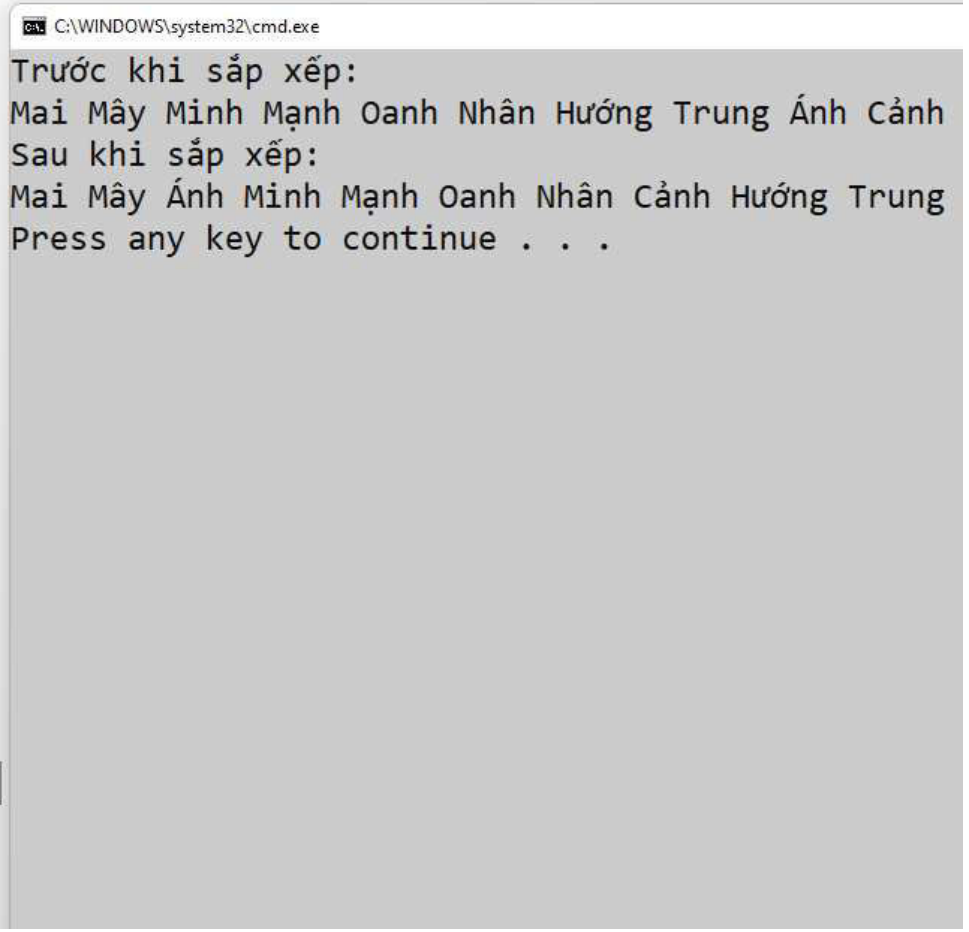
Sort(); Sắp xếp các phần tử trong toàn bộ tập hợp. Sử dụng comparer mặc định. **Sort(Comparison<T> comp);** sắp xếp toàn bộ tập hợp sử dụng comparison cho trước. Xảy ra ngoại lệ nếu comp bằng null. **Sort(IComparer<T>? comparer);** Sắp xếp các phần tử của tập hợp sử dụng comparer cho trước hoặc nếu comparer null thì sử dụng comparer mặc định. Xảy ra ngoại lệ nếu tham số null và không có comparer mặc định. **Sort(int index, int count, IComparer<T>? comparer);** sắp xếp count phần tử của tập hợp tính từ vị trí index, sử dụng comparer cho trước. Nếu comparer là null thì sử dụng comparer mặc định. Xảy ra ngoại lệ nếu index vượt biên tập hợp hoặc index, count không chỉ định một khoảng hợp lệ trong tập hợp. Hoặc comparer cho trước là null nhưng không có comparer mặc định để sử dụng. Hoặc comparer gây lỗi khi đang sắp xếp.
ToArray(); trả về một mảng các phần tử của tập hợp hiện thời.
TrimExcess(); thiết lập khả năng lưu trữ về số lượng phần tử hiện có trong tập hợp, nếu giá trị đó nhỏ hơn ngưỡng.
TrueForAll(Predicate<T> match); xác định xem liệu mọi phần tử trong tập hợp khớp với điều kiện chỉ ra bởi tham số hay không. Trả về true nếu mọi phần tử thỏa mãn điều kiện và false nếu ngược lại. Nếu tập hợp rỗng, kết quả là true. Xảy ra ngoại lệ nếu match là null.

Ví dụ

```
    Console.OutputEncoding = Encoding.UTF8;
    List<string> friends = new List<string>();
    friends.Add("Mai");
    friends.Add("Mây");
    friends.Add("Minh");
    friends.Add("Mạnh");
    friends.Add("Oanh");
    friends.Add("Nhân");
    friends.Add("Hướng");
    friends.Add("Trung");
    friends.Add("Ánh");
    friends.Add("Cảnh");
    Console.WriteLine("Trước khi sắp xếp: ");
    ShowElement(friends);
    friends.Sort(CompareByLength);
    Console.WriteLine("Sau khi sắp xếp: ");
    ShowElement(friends);
}

1 reference
static int CompareByLength(string x, string y)...

2 references
static void ShowElement<T>(List<T> list)...
```



Nội dung tiếp theo

Tìm hiểu về delegate