LINQ in C# — 50+ Exercises

(Target: .NET Framework 4.6; snippets explicitly marked `(net6+)` require .NET 6+)

Prepared for practice

November 3, 2025

How to use this booklet

Each exercise gives a short task and a minimal *solution* in C#. The target is **.NET Framework 4.6**. Where a snippet uses a newer API, it is labeled `(net6+)` and, when easy, a 4.6-friendly alternative is shown in a comment.

```
// Common usings for all snippets
using System;
using System.Linq;
using System.Collections.Generic;

// Sample domain models (classes for .NET Framework 4.6)
public class Person
{
  public int Id { get; set; }
  public string Name { get; set; }
  public int Age { get; set; }
  public string City { get; set; }
  public Person(int id, string name, int age, string city)
  { Id=id; Name=name; Age=age; City=city; }
}

public class Order
{
  public int Id { get; set; }
  public int PersonId { get; set; }
  public string Product { get; set; }
  public int Quantity { get; set; }
  public decimal Price { get; set; }
  public Order(int id, int personId, string product, int quantity, decimal price)
  { Id=id; PersonId=personId; Product=product; Quantity=quantity; Price=price; }
}
```

You can paste multiple snippets into a single `Main` by reusing the same sample data (below).

Sample data (reuse as needed)

```
// People
var people = new List<Person>{
  new Person(1,"Ana",28,"Belgrade"), new Person(2,"Marko",41,"Novi Sad"),
  new Person(3,"Ivana",33,"Niš"),    new Person(4,"Petar",28,"Belgrade"),
  new Person(5,"Mina",22,"Kragujevac"), new Person(6,"Nikola",41,"Novi Sad")
};

```

```
// Orders
var orders = new List<Order>{
  new Order(101,1,"Keyboard",1, 45.90m),  new Order(102,1,"Mouse",2, 15.50m),
  new Order(103,2,"Monitor",1, 199.99m),  new Order(104,3,"USB-C Cable",3, 8.90m),
  new Order(105,3,"Headphones",1, 59.00m),new Order(106,4,"Mouse",1, 15.50m),
  new Order(107,5,"Laptop",1, 999.00m),   new Order(108,6,"Mousepad",4, 5.00m),
  new Order(109,6,"Webcam",1, 79.00m)
};

// Simple sequences
int[] nums = { 1,2,3,4,5,6,7,8,9,10 };
string[] words = { "apple","banana","pear","apricot","plum","grape","peach" };
```

Exercises & Solutions

1) Filter evens

Return all even numbers from ***nums****.*

```
// Solution (method)
var evens = nums.Where(n => n % 2 == 0).ToArray();
```

2) Squares of odds

Square only odd numbers.

```
// Solution
var oddSquares = nums.Where(n => n % 2 == 1).Select(n => n * n).ToList();
```

3) First starting with 'ap'

Find the first word starting with "ap", or null if none.

```
// Solution
var firstAp = words.FirstOrDefault(w => w.StartsWith("ap"));
```

4) Last number < 7

Get the last number less than 7, default 0.

```
// Solution (.NET 4.6)
var lastLt7 = nums.Where(n => n < 7).DefaultIfEmpty(0).Last();

// Alternative (net6+):
// var lastLt7 = nums.LastOrDefault(n => n < 7, 0);
```

5) Any long word?

Check if any word length ≥ 6.

```
// Solution
bool anyLong = words.Any(w => w.Length >= 6);
```

6) All positive?

Are all numbers > 0?

```
// Solution
bool allPositive = nums.All(n => n > 0);
```

7) Count Belgrade people

How many people live in Belgrade?

```
// Solution
int belgradeCount = people.Count(p => p.City == "Belgrade");
```

8) Distinct cities

List distinct cities of `people`.

```
// Solution
var cities = people.Select(p => p.City).Distinct().OrderBy(c => c).ToList();
```

9) Sort by age desc, then name asc

Order people by Age desc, then Name asc.

```
// Solution
var sorted = people.OrderByDescending(p => p.Age).ThenBy(p => p.Name).ToList();
```

10) Skip/Take paging

Get page 2 (size 3) from `words`.

```
// Solution
int page = 2, size = 3;
var page2 = words.Skip((page-1)*size).Take(size).ToArray();
```

11) Sum of even squares

Sum (n^2) for even `nums`.

```
// Solution
int sumEvenSquares = nums.Where(n => n%2==0).Select(n => n*n).Sum();
```

12) Average age by city

Group by City and get average Age.

```
// Solution
var avgByCity = people
  .GroupBy(p => p.City)
  .Select(g => new { City = g.Key, AvgAge = g.Average(p => p.Age) })
  .ToList();
```

13) Max priced order per person

For each PersonId, find their most expensive order (price).

```
// Solution
var maxOrderPerPerson = orders
  .GroupBy(o => o.PersonId)
  .Select(g => g.OrderByDescending(o => o.Price).First())
  .ToList();
```

14) Total spend per person

*Compute total spend (Quantity*Price) per Person.*

```
// Solution
var totalSpend = orders
  .GroupBy(o => o.PersonId)
  .Select(g => new {
    PersonId = g.Key,
    Total = g.Sum(o => o.Quantity * o.Price)
  })
  .ToList();
```

15) Join People with Orders

Join by PersonId; produce {Name, Product}.

```
// Solution (method)
var personOrders = people.Join(
  orders, p => p.Id, o => o.PersonId,
  (p,o) => new { p.Name, o.Product }
).ToList();
```

16) GroupJoin (left join)

People with their list of products (may be empty).

```
// Solution
var leftJoin = people.GroupJoin(
  orders, p => p.Id, o => o.PersonId,
  (p, os) => new { p.Name, Products = os.Select(x => x.Product).ToList() }
).ToList();
```

17) Query syntax: group by city

Same as (12) but query syntax.

```
// Solution (query)
var avgByCityQ =
  from p in people
  group p by p.City into g
  select new { City = g.Key, AvgAge = g.Average(x => x.Age) };
```

18) Query syntax: inner join

Same idea as (15) using query syntax.

```
// Solution (query)
var personOrdersQ =
  from p in people
  join o in orders on p.Id equals o.PersonId
  select new { p.Name, o.Product };
```

19) Composite projection

Select anonymous objects of {Word, Len}.

```
// Solution
var meta = words.Select(w => new { Word = w, Len = w.Length }).ToList();
```

20) TakeWhile/SkipWhile

From nums, take while < 6*, skip while* < 6*.*

```
// Solution
var head = nums.TakeWhile(n => n < 6).ToArray();
var tail = nums.SkipWhile(n => n < 6).ToArray();
```

21) DistinctBy City (net6+)

Distinct persons by City, keep first occurrence.

```
// Solution (net6+)
var distinctCity = people.DistinctBy(p => p.City).ToList();

// .NET 4.6 alternative:
var distinctCity46 = people.GroupBy(p => p.City).Select(g => g.First()).ToList();
```

22) MaxBy/MinBy (net6+)

Oldest and youngest person.

```
// Solution (net6+)
var oldest = people.MaxBy(p => p.Age);
var youngest = people.MinBy(p => p.Age);

// .NET 4.6 alternatives:
var oldest46 = people.OrderByDescending(p => p.Age).FirstOrDefault();
var youngest46 = people.OrderBy(p => p.Age).FirstOrDefault();
```

23) ElementAt with bounds

3rd element or default (-1).

```
// Solution
var third = nums.Skip(2).DefaultIfEmpty(-1).First();
```

24) DefaultIfEmpty

If no numbers >20, return {0}.

```
// Solution
var gt20OrZero = nums.Where(n => n > 20).DefaultIfEmpty(0).ToArray();
```

25) Set operations

Union, Intersect, Except.

```
// Solution
int[] a = {1,2,3,4}, b = {3,4,5,6};
var u = a.Union(b).ToArray();
var i = a.Intersect(b).ToArray();
var e = a.Except(b).ToArray();
```

26) Zip

Pair nums with words (shortest length wins).

```
// Solution
var pairs = nums.Zip(words, (n,w) => n + ":" + w).ToList();
```

27) SelectMany (flatten)

Split words into letters and flatten.

```
// Solution
var letters = words.SelectMany(w => w.ToCharArray()).ToList();
```

28) Cartesian product (SelectMany)

All pairs (word, num) where num $\leq$ word.Length.

```
// Solution
var pairs2 = words
  .SelectMany(w => nums.Where(n => n <= w.Length), (w,n) => new { w, n })
  .ToList();
```

29) Aggregate (factorial)

Compute $1 \cdot 2 \cdot \ldots \cdot 6$.

```
// Solution
int fact6 = Enumerable.Range(1,6).Aggregate(1, (acc,x) => acc * x);
```

30) Running sum (scan)

Produce running sums over nums.

```
// Solution (simple loop with LINQ idea)
var running = new List<int>();
nums.Aggregate(0, (acc,x) => { var s = acc + x; running.Add(s); return s; });
```

31) ToDictionary

Map `Person.Id` → `Name`.

```
// Solution
var byId = people.ToDictionary(p => p.Id, p => p.Name);
```

32) ToLookup (multi-map)

`City` → people living there.

```
// Solution
var lookup = people.ToLookup(p => p.City, p => p.Name);
// e.g., lookup["Belgrade"] is IEnumerable<string>
```

33) Chunk (`net6+`)

Split nums into chunks of size 3.

```
// Solution (net6+)
var chunks = nums.Chunk(3).ToList();

// .NET 4.6 alternative (group by index / size):
var chunks46 = nums
  .Select((x,i) => new { x, i })
  .GroupBy(t => t.i / 3)
  .Select(g => g.Select(t => t.x))
  .ToList();
```

34) Order stability

Show that `OrderBy` is stable with `ThenBy`.

```
// Solution (illustration)
var stable = people.OrderBy(p => p.Age).ThenBy(p => p.Name).ToList();
```

35) Group by composite key

Group orders by {PersonId, Product}.

```
// Solution
var byPersonProduct = orders
  .GroupBy(o => new { o.PersonId, o.Product })
  .Select(g => new { g.Key.PersonId, g.Key.Product, Qty = g.Sum(o => o.Quantity) })
  .ToList();
```

36) Left anti-join

People with no orders.

```
// Solution (.NET 4.6, avoid O(n^2) Contains by using HashSet)
var withOrdersSet = new HashSet<int>(orders.Select(o => o.PersonId));
var noOrders = people.Where(p => !withOrdersSet.Contains(p.Id)).ToList();
```

37) Inner join with composite condition

Match only if same city and age parity.

```
// Solution (self-join illustration)
var pairsSameCityParity =
  from p1 in people
  from p2 in people
  where p1.Id < p2.Id && p1.City==p2.City && (p1.Age % 2) == (p2.Age % 2)
  select new { p1.Name, p2.Name, p1.City };
```

38) Grouping & ordering inside groups

For each city: top-1 oldest person.

```
// Solution
var oldestPerCity = people
  .GroupBy(p => p.City)
  .Select(g => g.OrderByDescending(p => p.Age).First())
  .ToList();
```

39) `Select` index overload

Project word with its index.

```
// Solution
var withIndex = words.Select((w,i) => new { Index = i, Word = w }).ToList();
```

40) Stable paging after ordering

Order by Name, then take page 1 (size 2).

```
// Solution
var pg = people.OrderBy(p => p.Name).Skip(0).Take(2).ToList();
```

41) Quantifiers on groups

Cities where all residents are ≥ 30.

```
// Solution
var matureCities = people
  .GroupBy(p => p.City)
  .Where(g => g.All(p => p.Age >= 30))
  .Select(g => g.Key)
  .ToList();
```

42) Sliding window (size 3)

Triplets (n_i, n_{i+1}, n_{i+2}).

```
// Solution (.NET 4.6-friendly anonymous types)
var triples = nums
  .Zip(nums.Skip(1), (a,b) => new { a, b })
  .Zip(nums.Skip(2), (ab,c) => new { a = ab.a, b = ab.b, c })
  .ToList();
```

43) Median (odd count)

Median of odd-length sequence.

```
// Solution (nums length odd assumed)
var med = nums.OrderBy(n => n).ElementAt(nums.Length/2);
```

44) Top-N by key

Top 3 longest words.

```
// Solution
var top3 = words.OrderByDescending(w => w.Length).Take(3).ToList();
```

45) GroupJoin with aggregation

People with total spend (0 if none).

```
// Solution
var spendByPerson = people.GroupJoin(
  orders, p => p.Id, o => o.PersonId,
  (p, os) => new {
    p.Name,
    Total = os.Sum(x => x.Price * x.Quantity)
  }).ToList();
```

46) Query syntax: let/into

Words grouped by first letter (lowercased), filtered by group size ≥ 2.

```
// Solution (query)
var groupsQ =
  from w in words
  let key = char.ToLowerInvariant(w[0])
  group w by key into g
  where g.Count() >= 2
  select new { Letter = g.Key, Items = g.OrderBy(x => x).ToList() };
```

47) Custom comparer in `OrderBy`

Order words by last char using custom comparer.

```
// Solution (.NET 4.6-safe; avoid index-from-end operator)
IComparer<string> lastChar = Comparer<string>.Create((a,b) =>
{
  char la = a[a.Length - 1];
  char lb = b[b.Length - 1];
  return la.CompareTo(lb);
});
var byLast = words.OrderBy(w => w, lastChar).ToList();
```

48) `GroupBy` with key selector & element selector

Group orders by product; collect just quantities.

```
// Solution
var qtyByProduct = orders
  .GroupBy(o => o.Product, o => o.Quantity)
  .Select(g => new { Product = g.Key, TotalQty = g.Sum() })
  .ToList();
```

49) `SelectMany` for one-to-many (tokens)

Split phrases into words and normalize.

```
// Solution
string[] phrases = { "red green", "blue  red", "green" };
var tokens = phrases
  .SelectMany(p => p.Split(new[]{' '}, StringSplitOptions.RemoveEmptyEntries))
  .Select(t => t.Trim().ToLowerInvariant())
  .Distinct()
  .OrderBy(t => t)
  .ToList();
```

50) Pipeline with materialization

Demonstrate deferred vs. immediate execution.

```
// Solution
var query = nums.Where(n => {
  Console.WriteLine("Filtering " + n);
  return n % 2 == 0;
}); // deferred
var materialized = query.ToList(); // executes now
```

51) Join on computed keys

Join people and orders by city initial = product initial. (Toy example)

```
// Solution
var funJoin =
  from p in people
  join o in orders
    on p.City[0] equals o.Product[0]
  select new { p.Name, p.City, o.Product };
```

52) Group & top-k within each group

For each city, list up to 2 youngest names.

```
// Solution
var k=2;
var youngestPerCity = people
  .GroupBy(p => p.City)
  .Select(g => new {
    g.Key,
    Names = g.OrderBy(p => p.Age).ThenBy(p => p.Name)
             .Take(k).Select(p => p.Name).ToList()
  }).ToList();
```

53) Stable dedup with key + prefer condition

Deduplicate by name, keep the oldest age for that name.

```
// Solution
var byNameOldest = people
  .GroupBy(p => p.Name)
  .Select(g => g.OrderByDescending(p => p.Age).First())
  .ToList();
```

54) Partitioning: `Chunk` + aggregate (`net6+`)

Sum every chunk of 4 numbers.

```
// Solution (net6+)
var sumChunks = nums.Chunk(4).Select(c => c.Sum()).ToList();

// .NET 4.6 alternative:
var sumChunks46 = nums
  .Select((x,i) => new { x, i })
  .GroupBy(t => t.i / 4)
  .Select(g => g.Sum(t => t.x))
  .ToList();
```

55) Sliding average

Average over windows of size 3.

```
// Solution (works with the anonymous-type triples from #42)
var slidingAvg = triples.Select(t => (t.a + t.b + t.c)/3.0).ToList();
```

56) `OfType<T>` filter

Keep only strings from mixed list.

```
// Solution
IEnumerable<object> mixed = new object[]{ "x", 1, "y", 2.0, "z" };
var onlyStrings = mixed.OfType<string>().ToList();
```

57) `Cast<T>` to force enumeration

Cast objects to ints (will throw if not int).

```
// Solution
IEnumerable<object> objs = new object[]{ 1,2,3 };
var asInts = objs.Cast<int>().ToArray();
```

58) `Prepend/Append` (`netcore 3.0+`)

Add sentinels around ***nums****.*

```
// Solution (netcore 3.0+)
var padded = nums.Prepend(0).Append(999).ToArray();

// .NET 4.6 alternative:
var padded46 = new[]{0}.Concat(nums).Concat(new[]{999}).ToArray();
```

59) `SequenceEqual`

Check if two sequences equal ignoring order (sort both).

```
// Solution
int[] a1={3,1,2}, a2={2,3,1};
bool same = a1.OrderBy(x=>x).SequenceEqual(a2.OrderBy(x=>x));
```

60) `ExceptBy/IntersectBy` (net6+)

Remove people whose Ids appear in a banned list.

```
// Solution (net6+)
int[] bannedIds = {2,5};
var allowed = people.ExceptBy(bannedIds, p => p.Id).ToList();

// .NET 4.6 alternative:
var banned = new HashSet<int>(bannedIds);
var allowed46 = people.Where(p => !banned.Contains(p.Id)).ToList();
```

Tip. All exercises are easily testable by adding `Console.WriteLine` or materializing with `ToList()` to inspect results.