Level UP

IT-образование
в Петербурге
и удалённо

Towards the .NET Junior Developer

The extremely solid course

Lesson 5

Data structures.
Language-Integrated Query (LINQ)

Agenda

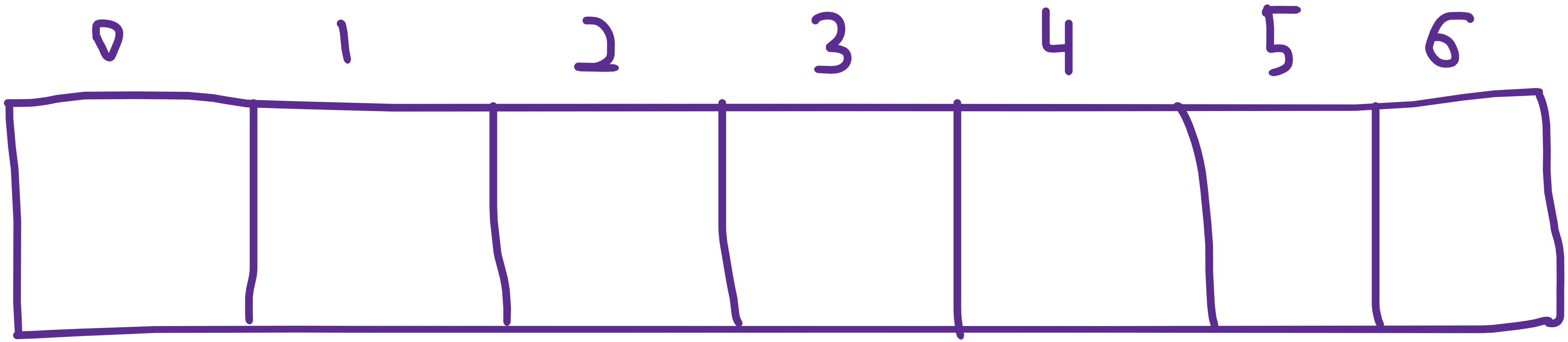
- Data structures
  - Array
  - List
  - Linked List
  - Hash Table
  - Stack
  - Queue
- Extension Methods
- LINQ
  - Extension methods
  - SQL-like syntax
  - Lazy calculation
- Books of the day
- Links of the day
- Hometask

Data structures

Data structures

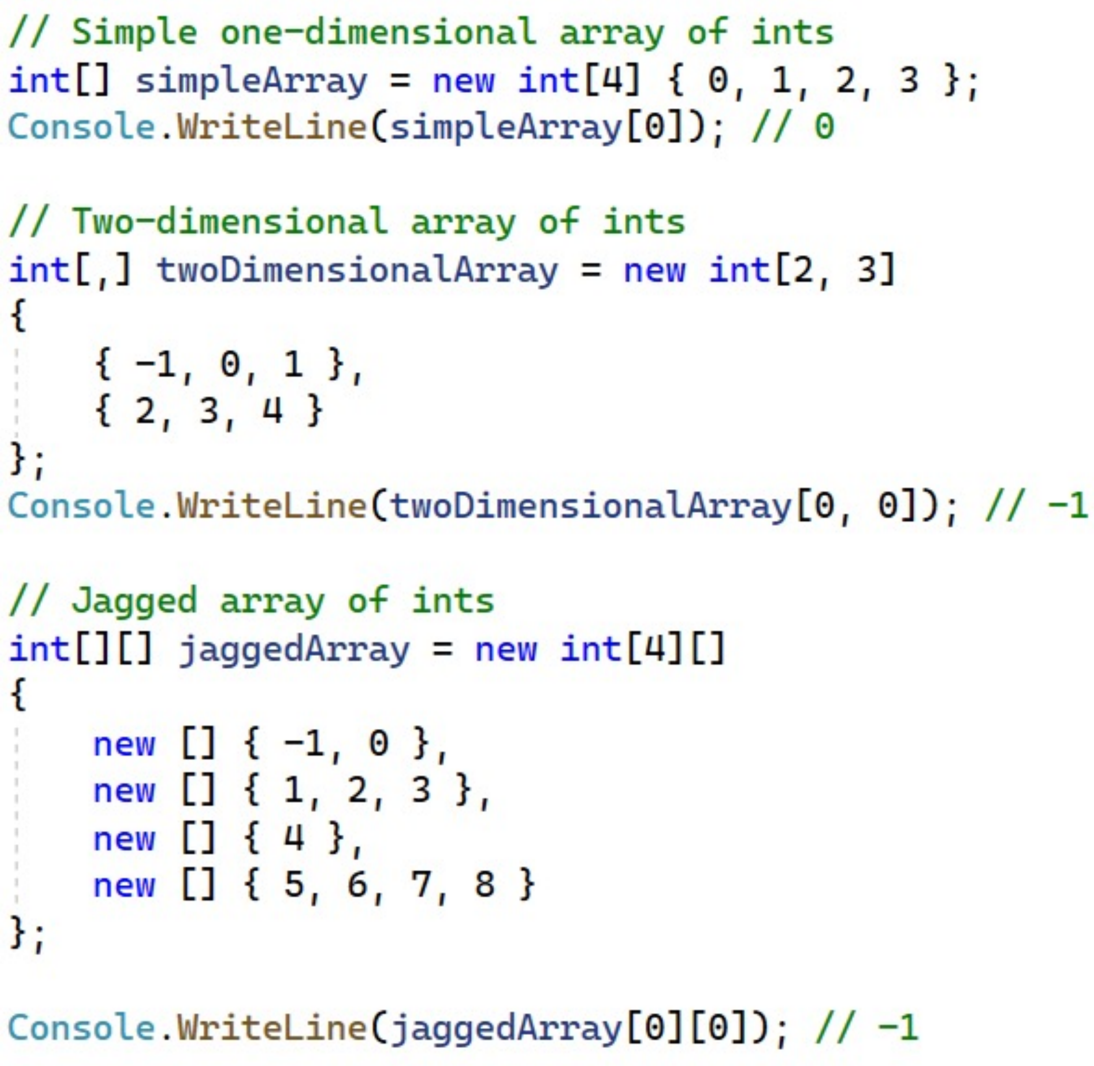
- Array
- List
- Linked List
- Hash Table
- Stack
- Queue

Data structures - array



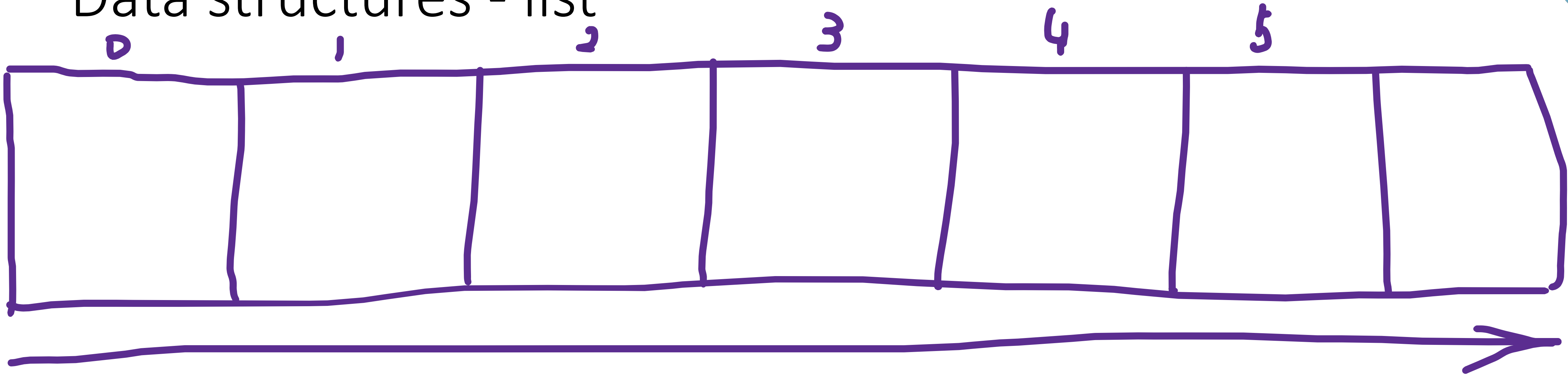
- Has constant length (items number)
- Can store only objects of the same type
- Can be accessed via index
- Get item by index – O(1)
- Set item by index – O(1)
- Search element – O(n)
- Can be N-dimensional
- Can be “jagged”
- Reference types
- Inherits System.Array (all)

Data structures - array



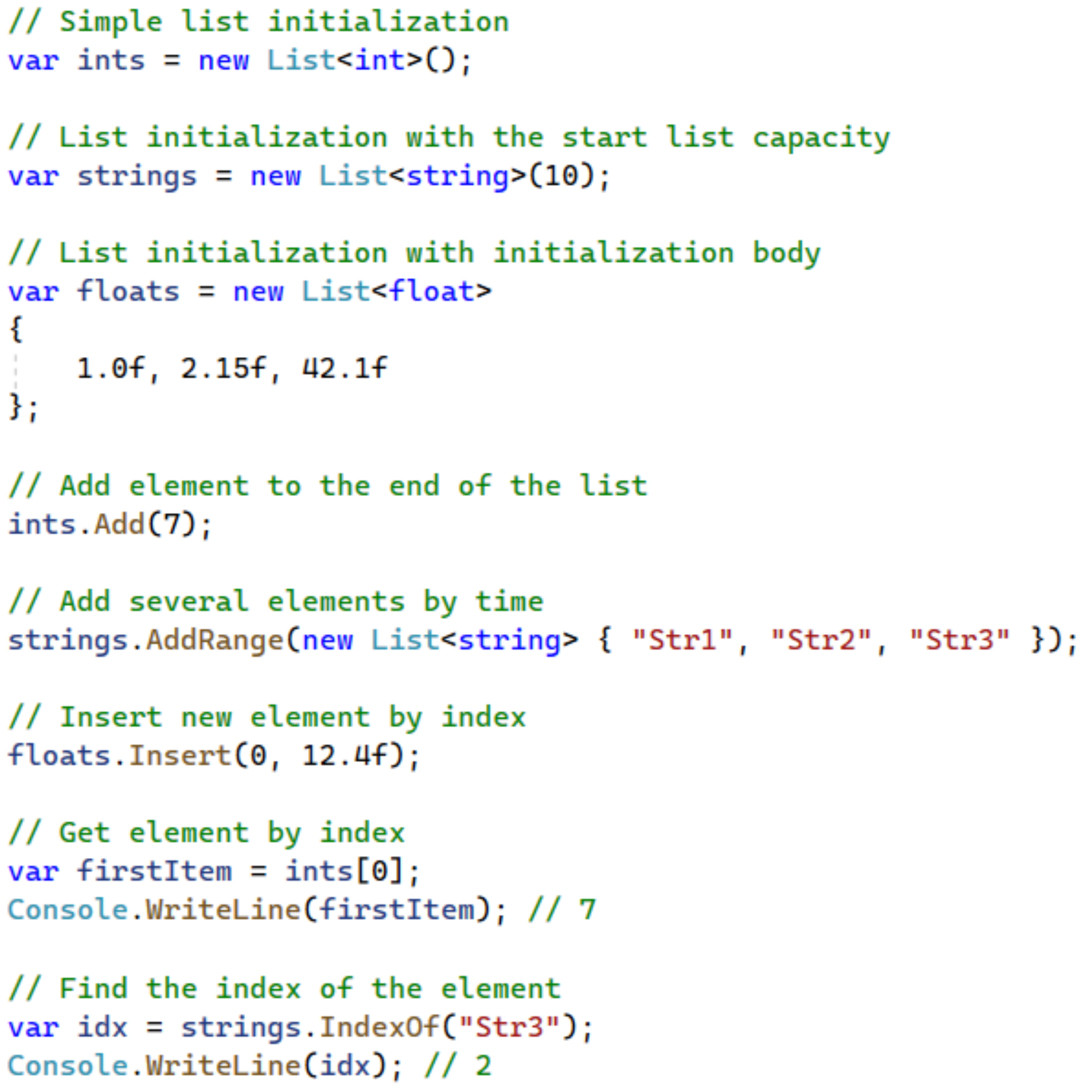
```
// Simple one-dimensional array of ints
int[] simpleArray = new int[4] { 0, 1, 2, 3 };
Console.WriteLine(simpleArray[0]); // 0

// Two-dimensional array of ints
int[,] twoDimensionalArray = new int[2, 3]
{
    { -1, 0, 1 },
    { 2, 3, 4 }
};
Console.WriteLine(twoDimensionalArray[0, 0]); // -1

// Jagged array of ints
int[][] jaggedArray = new int[4][]
{
    new [] { -1, 0 },
    new [] { 1, 2, 3 },
    new [] { 4 },
    new [] { 5, 6, 7, 8 }
};

Console.WriteLine(jaggedArray[0][0]); // -1
```

Data structures - list



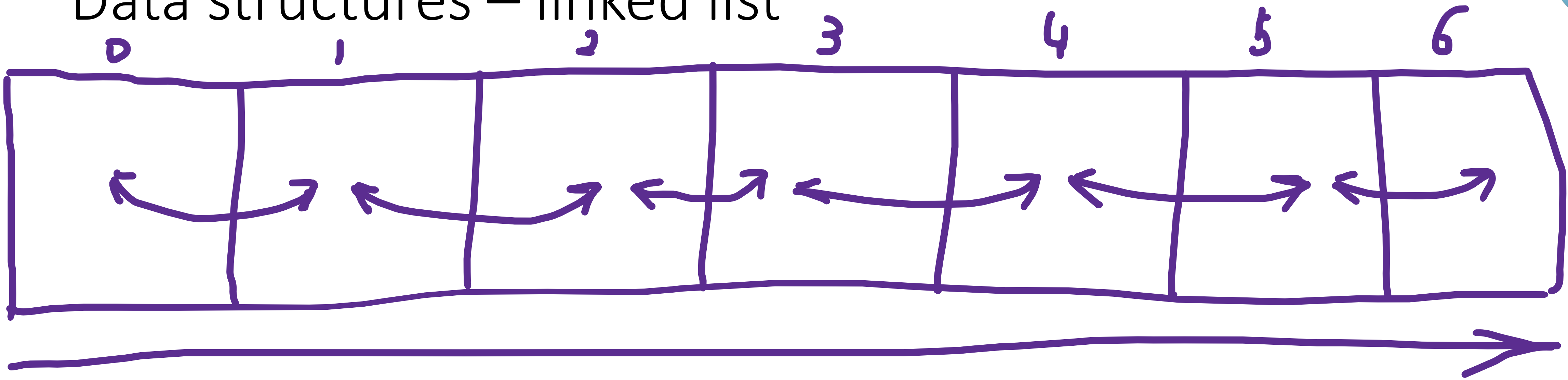
- Has dynamic length (items number)
- Can store only objects of the same type
- Can be accessed via index
- Get item by index – O(1)
- Set item by index – O(1)
- Search item – O(n)
- Add item to the end – O(1)
- Add item to the custom position – O(n)

Data structures - list



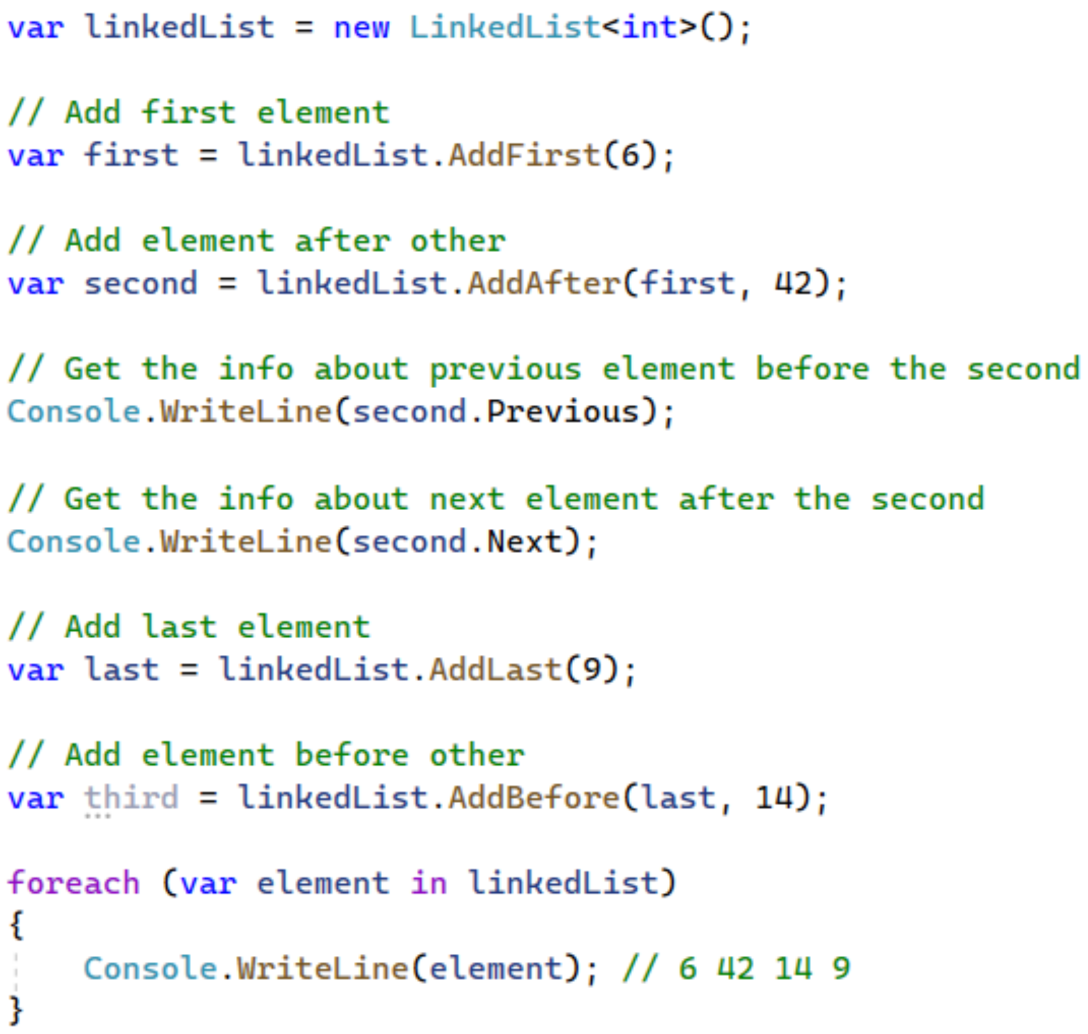
```
// Simple list initialization
var ints = new List<int>();

// List initialization with the start list capacity
var strings = new List<string>(10);

// List initialization with initialization body
var floats = new List<float>
{
    1.0f, 2.15f, 42.1f
};

// Add element to the end of the list
ints.Add(7);

// Add several elements by time
strings.AddRange(new List<string> { "Str1", "Str2", "Str3" });

// Insert new element by index
floats.Insert(0, 12.4f);

// Get element by index
var firstItem = ints[0];
Console.WriteLine(firstItem); // 7

// Find the index of the element
var idx = strings.IndexOf("Str3");
Console.WriteLine(idx); // 2
```

Data structures – linked list



- Has dynamic length (items number)
- Can store only objects of the same type
- Can be accessed via index
- Search item – O(n)
- Add item to any position – O(1)
- Delete item from any position – O(1)

Data structures – linked list

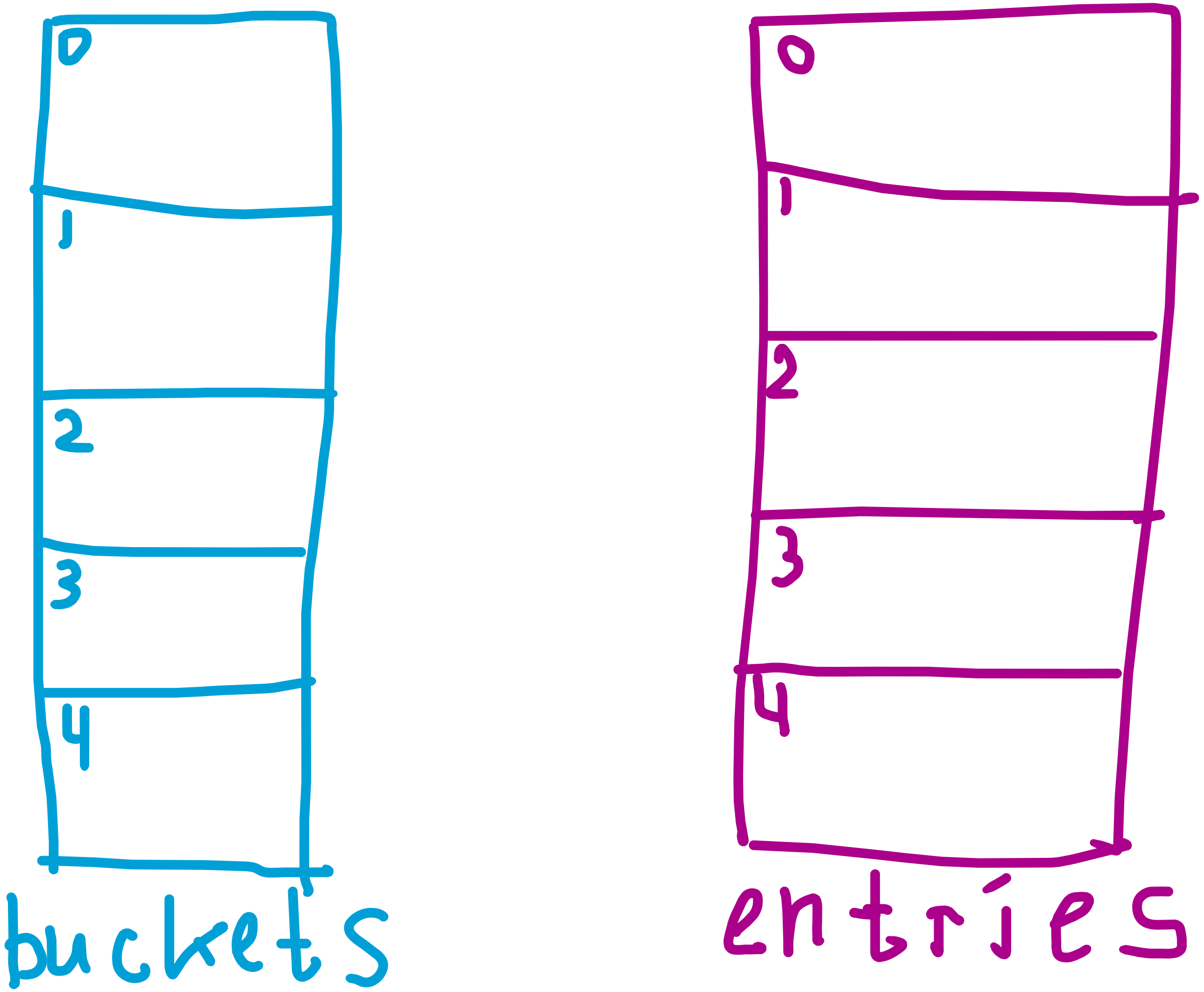


```
var linkedList = new LinkedList<int>();

// Add first element
var first = linkedList.AddFirst(6);

// Add element after other
var second = linkedList.AddAfter(first, 42);

// Get the info about previous element before the second
Console.WriteLine(second.Previous);

// Get the info about next element after the second
Console.WriteLine(second.Next);

// Add last element
var last = linkedList.AddLast(9);

// Add element before other
var third = linkedList.AddBefore(last, 14);

foreach (var element in linkedList)
{
    Console.WriteLine(element); // 6 42 14 9
}
```

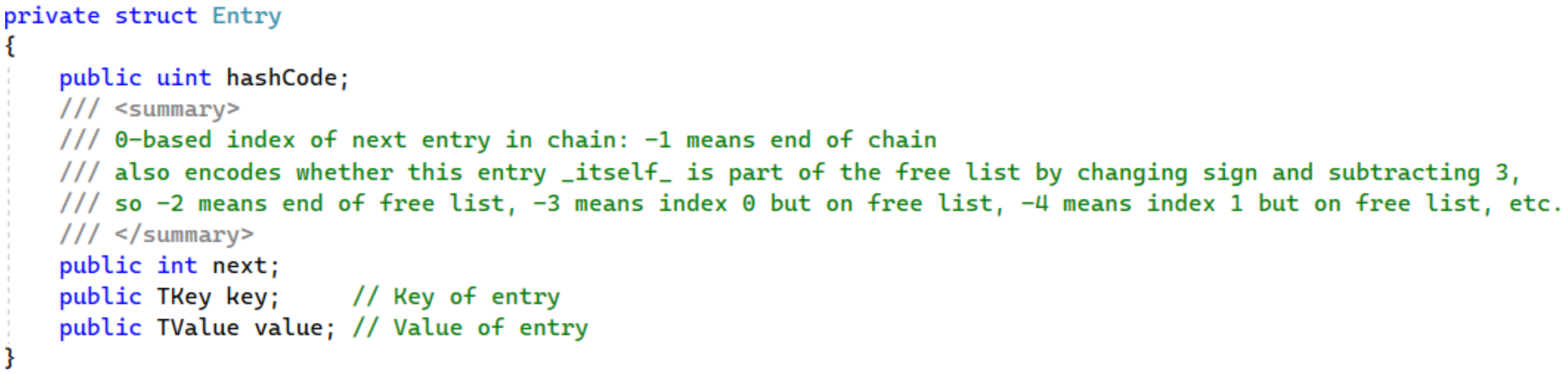
Data structures – hash table

keys	values
63	Samara
64	Saratov
69	Tver
73	Ulyanovsk
78	SPb
99	Moscow

Data structures – hash table

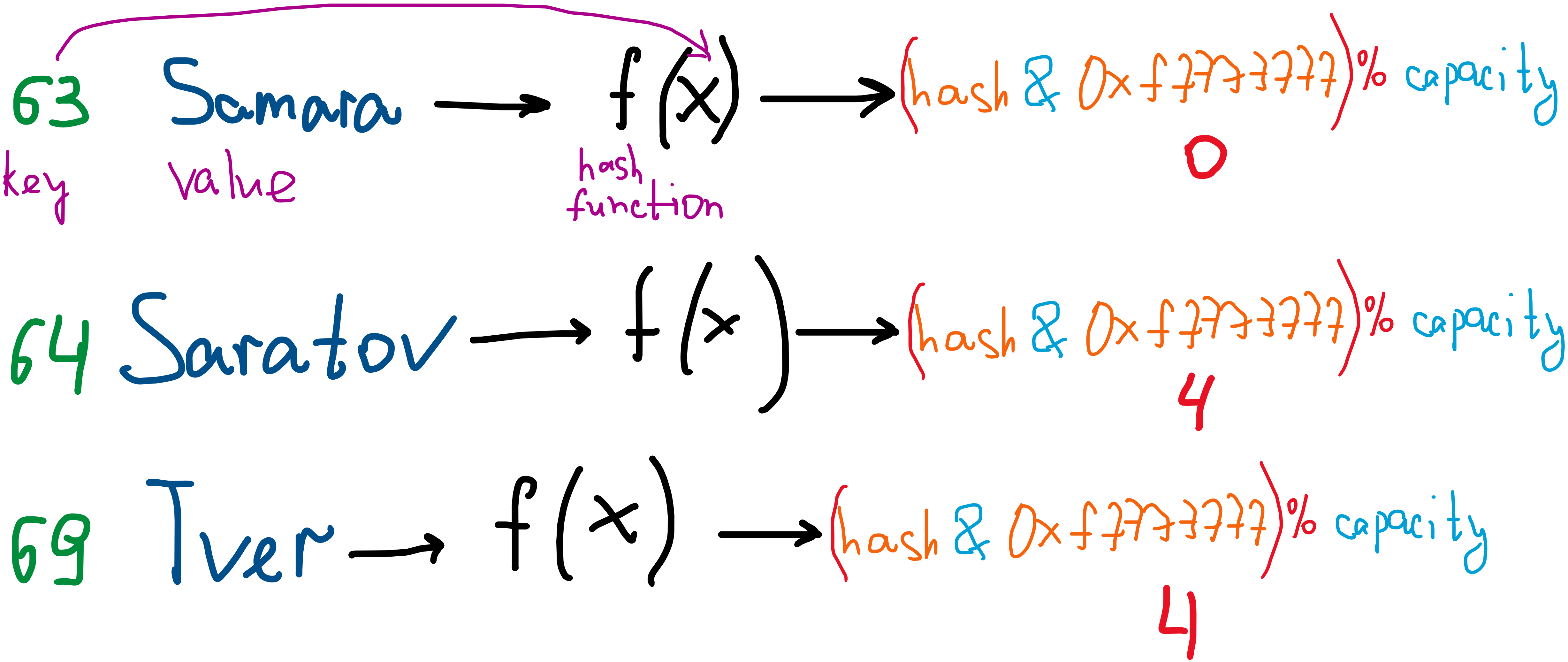


Data structures – hash table

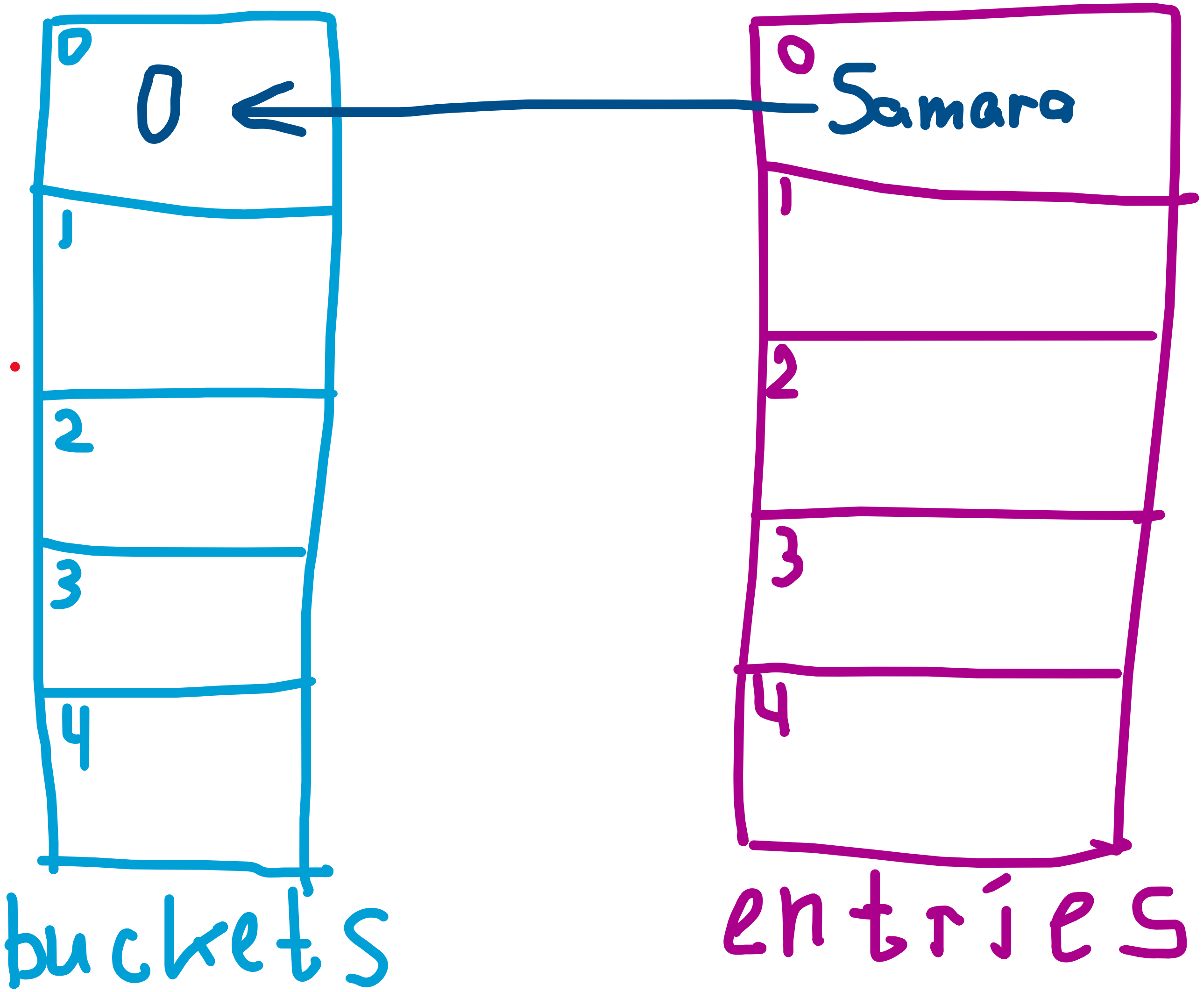


```
private struct Entry
{
    public uint hashCode;
    /// <summary>
    /// 0-based index of next entry in chain: -1 means end of chain
    /// also encodes whether this entry _itself_ is part of the free list by changing sign and subtracting 3,
    /// so -2 means end of free list, -3 means index 0 but on free list, -4 means index 1 but on free list, etc.
    /// </summary>
    public int next;
    public TKey key;     // Key of entry
    public TValue value; // Value of entry
}
```

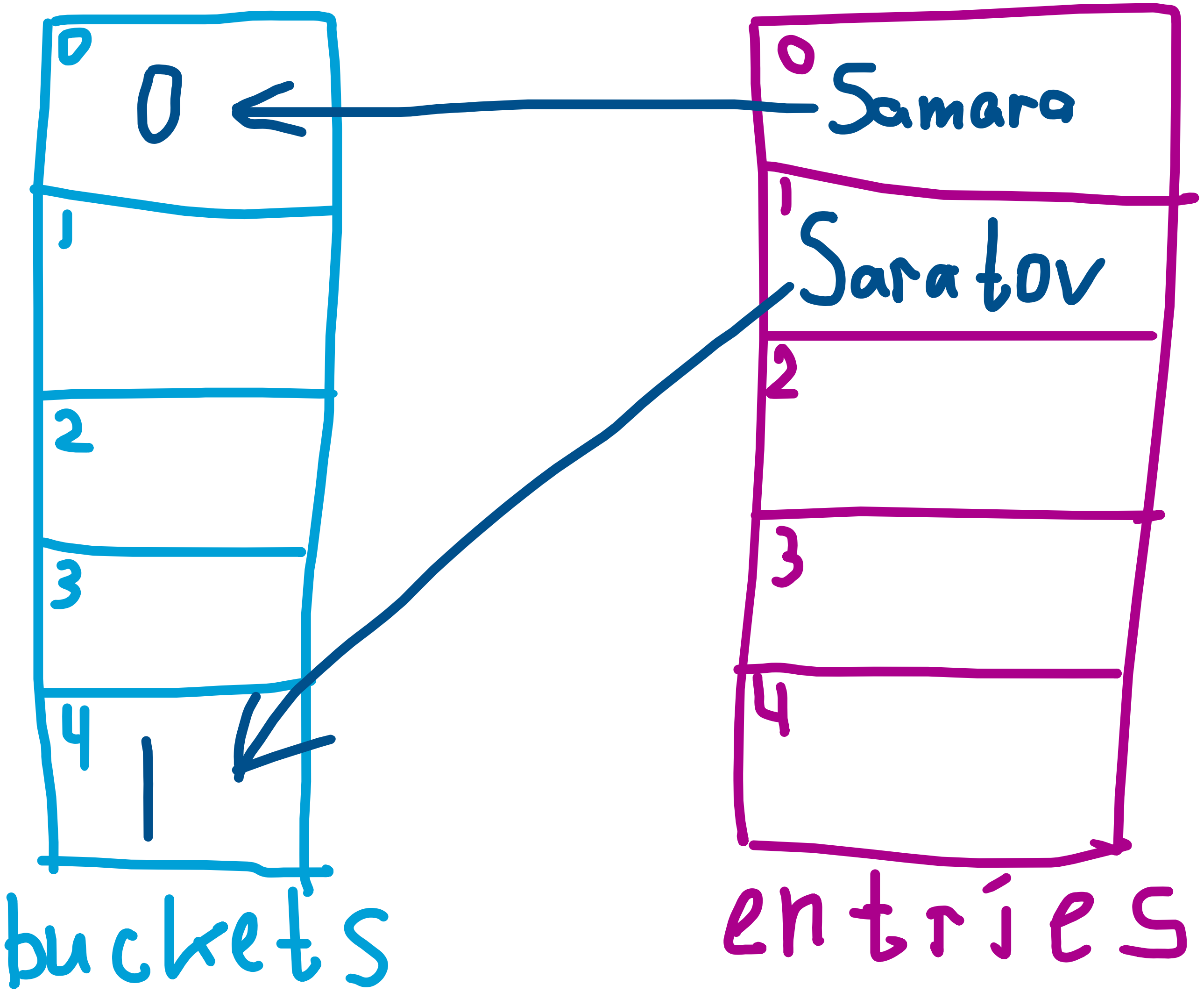
Data structures – hash table


63
key
Samara
value
f(x)
hash function
(hash & 0xf7777777)% capacity
0
64
Saratov
f(x)
(hash & 0xf7777777)% capacity
4
69
Tver
f(x)
(hash & 0xf7777777)% capacity
4

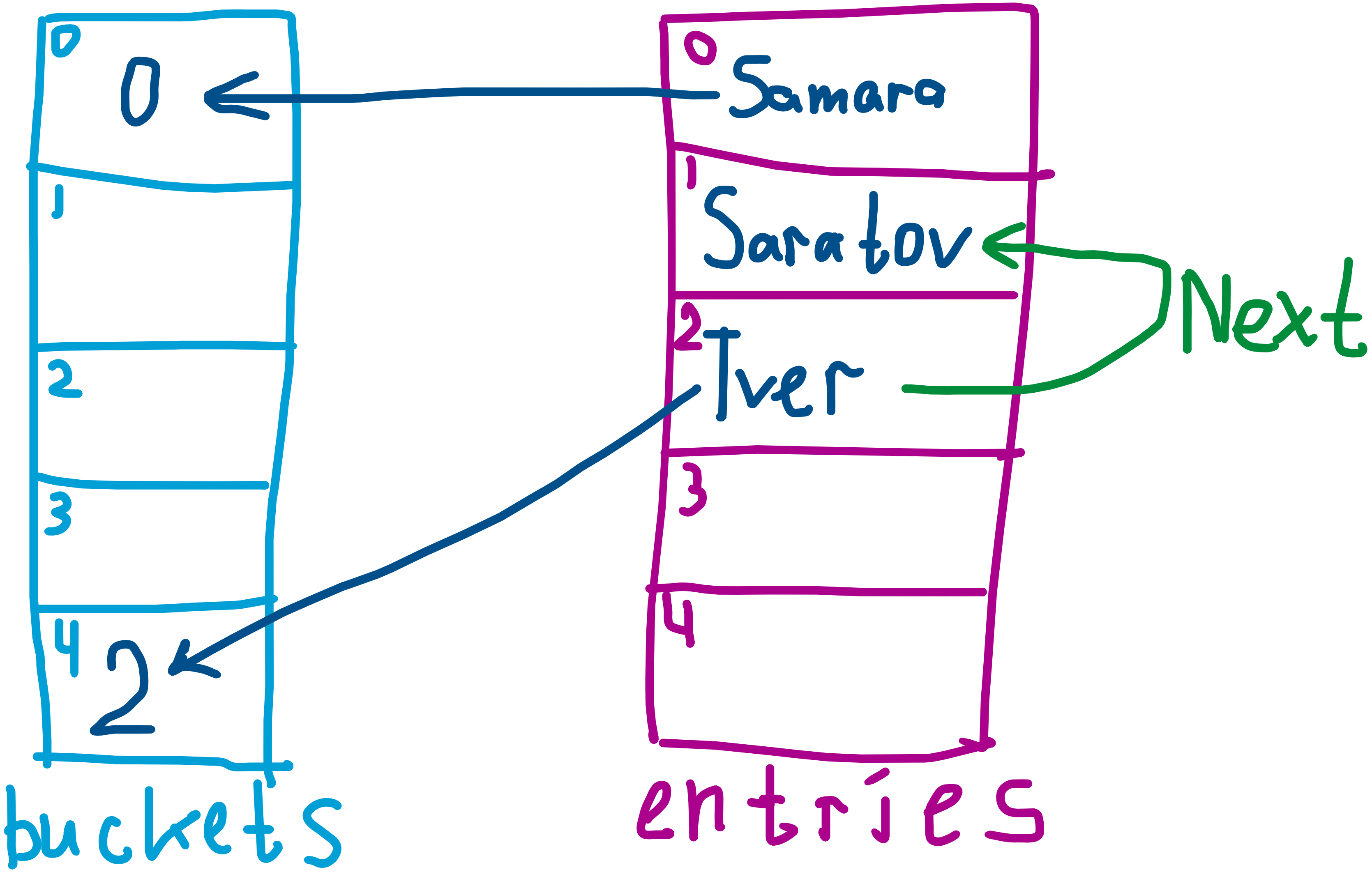
Data structures – hash table



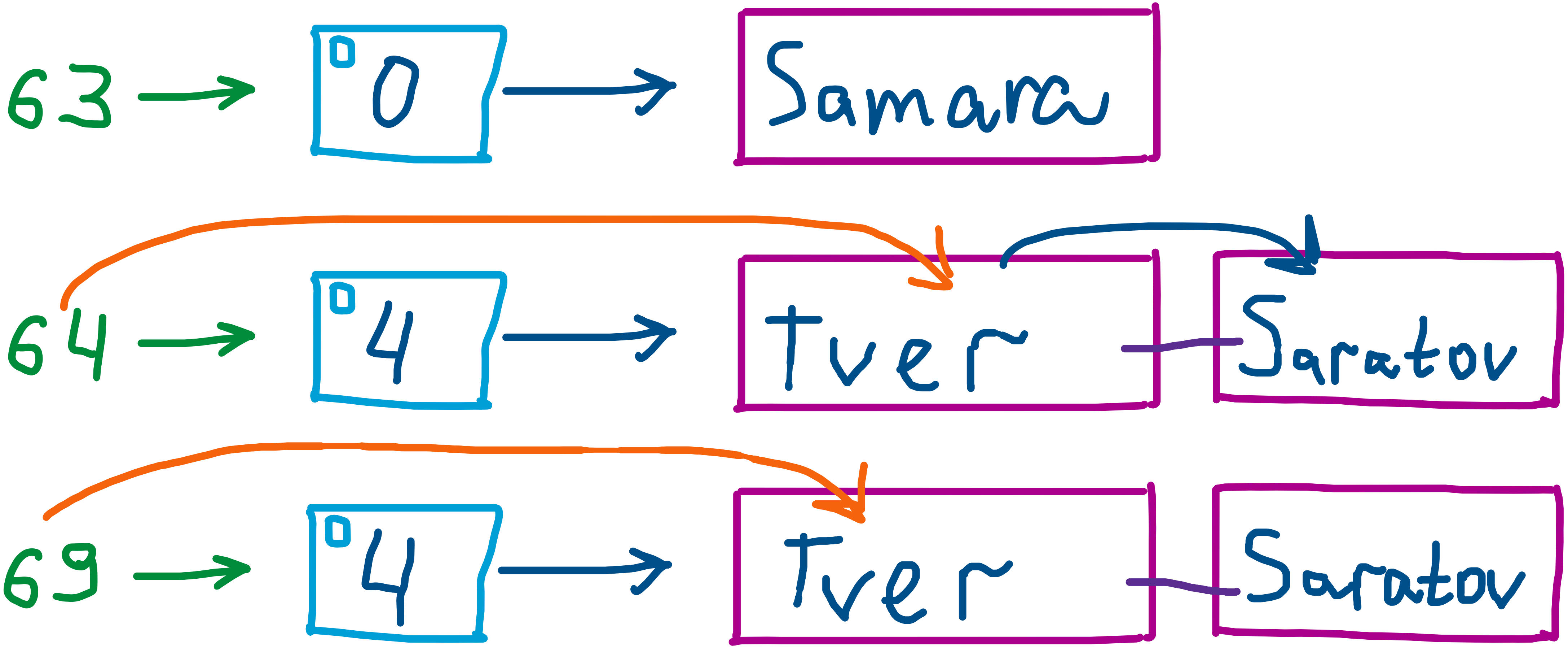
Data structures – hash table



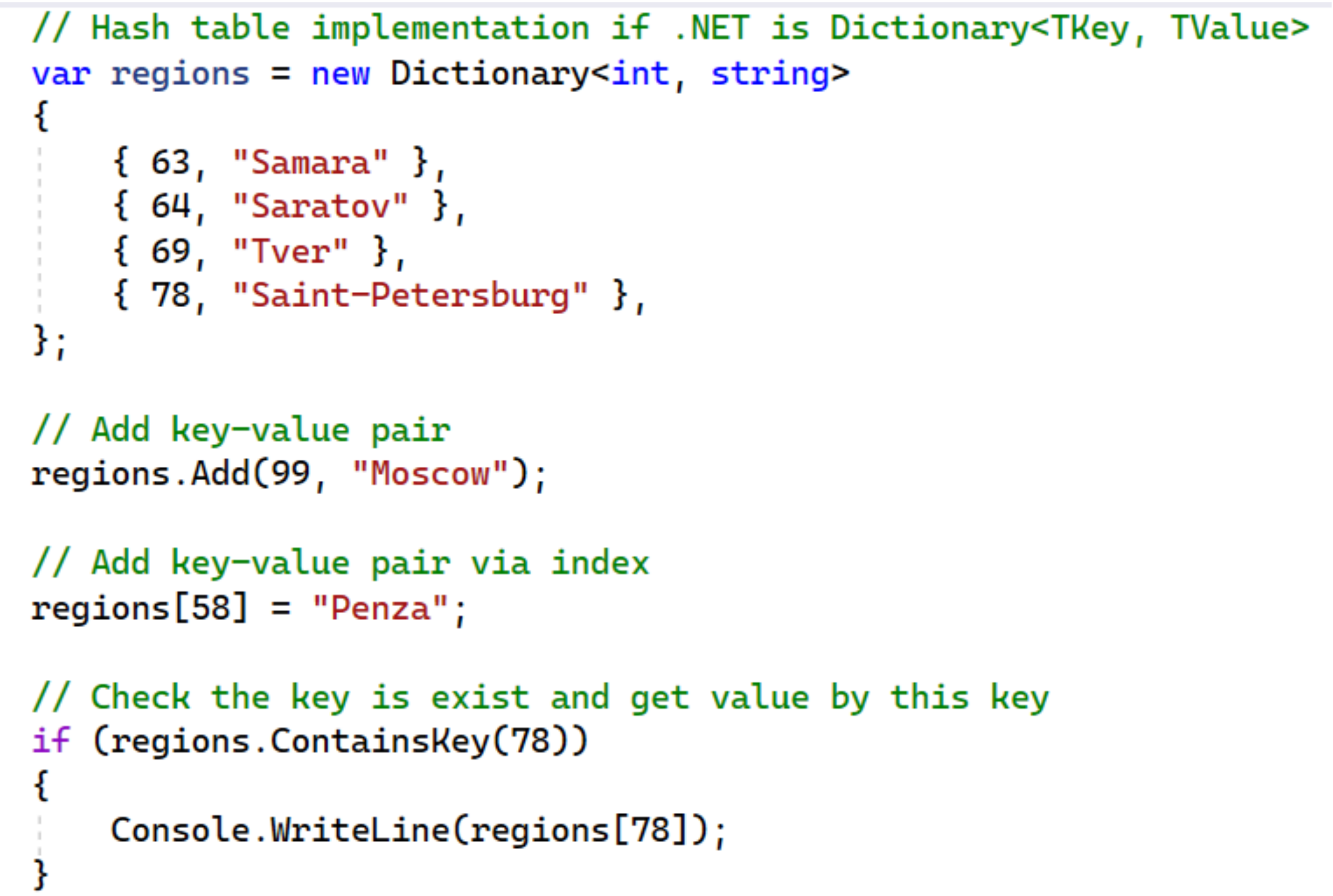
Data structures – hash table



Data structures – hash table

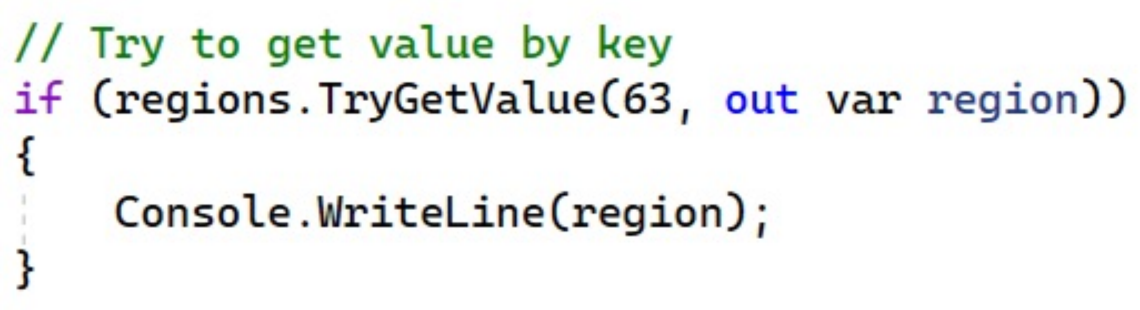


Data structures – hash table

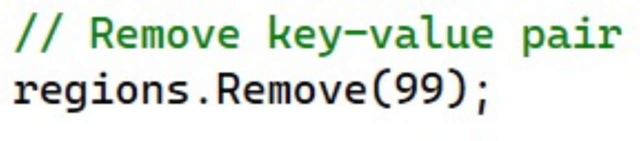


```
// Hash table implementation if .NET is Dictionary<TKey, TValue>
var regions = new Dictionary<int, string>
{
    { 63, "Samara" },
    { 64, "Saratov" },
    { 69, "Tver" },
    { 78, "Saint-Petersburg" },
};

// Add key-value pair
regions.Add(99, "Moscow");

// Add key-value pair via index
regions[58] = "Penza";

// Check the key is exist and get value by this key
if (regions.ContainsKey(78))
{
    Console.WriteLine(regions[78]);
}
```

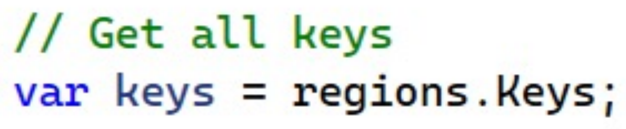
Data structures – hash table



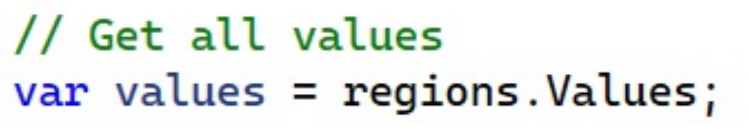
```
// Try to get value by key
if (regions.TryGetValue(63, out var region))
{
    Console.WriteLine(region);
}
```



```
// Remove key-value pair
regions.Remove(99);
```



```
// Get all keys
var keys = regions.Keys;
```

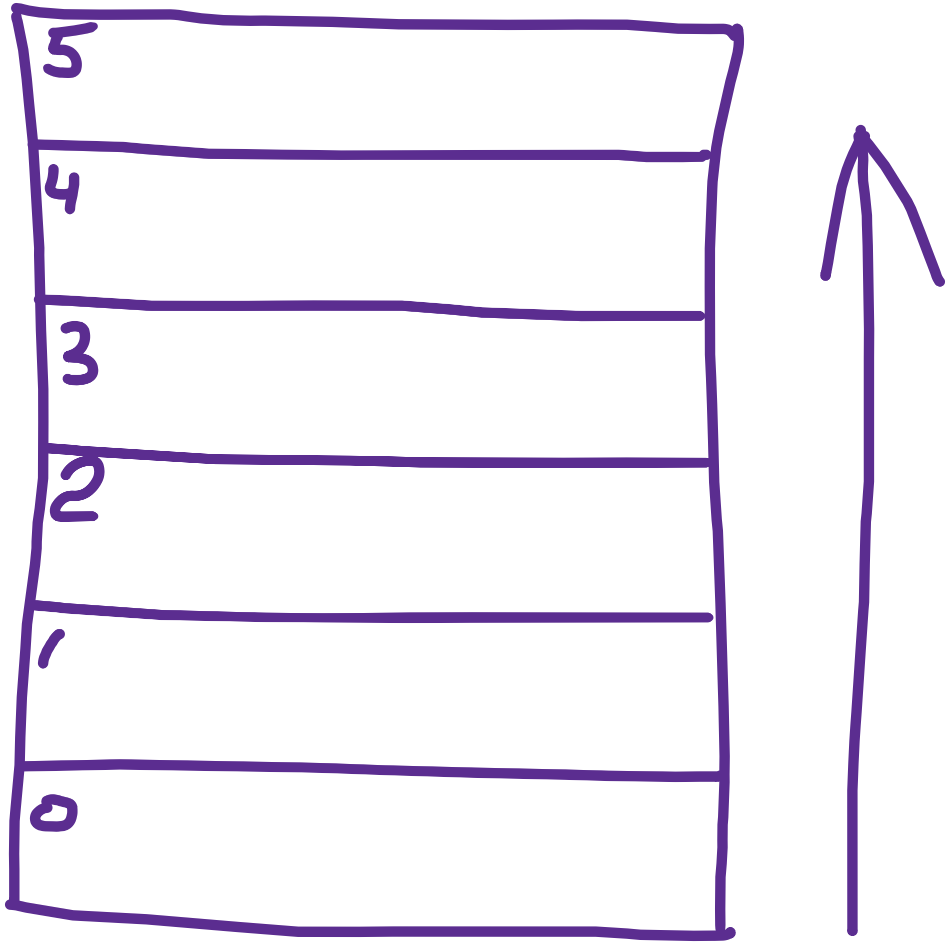


```
// Get all values
var values = regions.Values;
```

Data structures – hash table

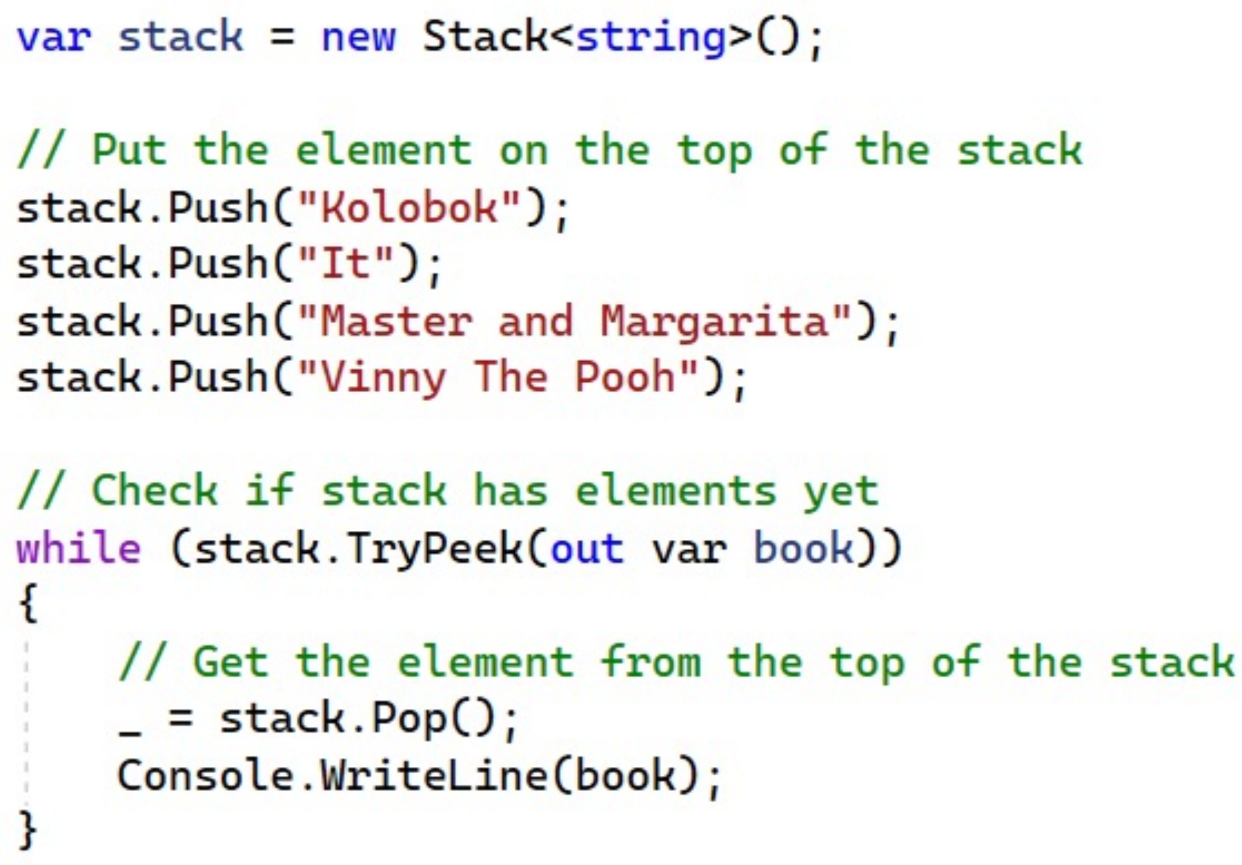
- Has dynamic length (key-value pairs number)
- Can store only objects of the same type
- Can be accessed via key
- Add item by key – O(1)
- Get item by key – O(1)/O(n)
- Remove item by key – O(1)

Data structures – stack



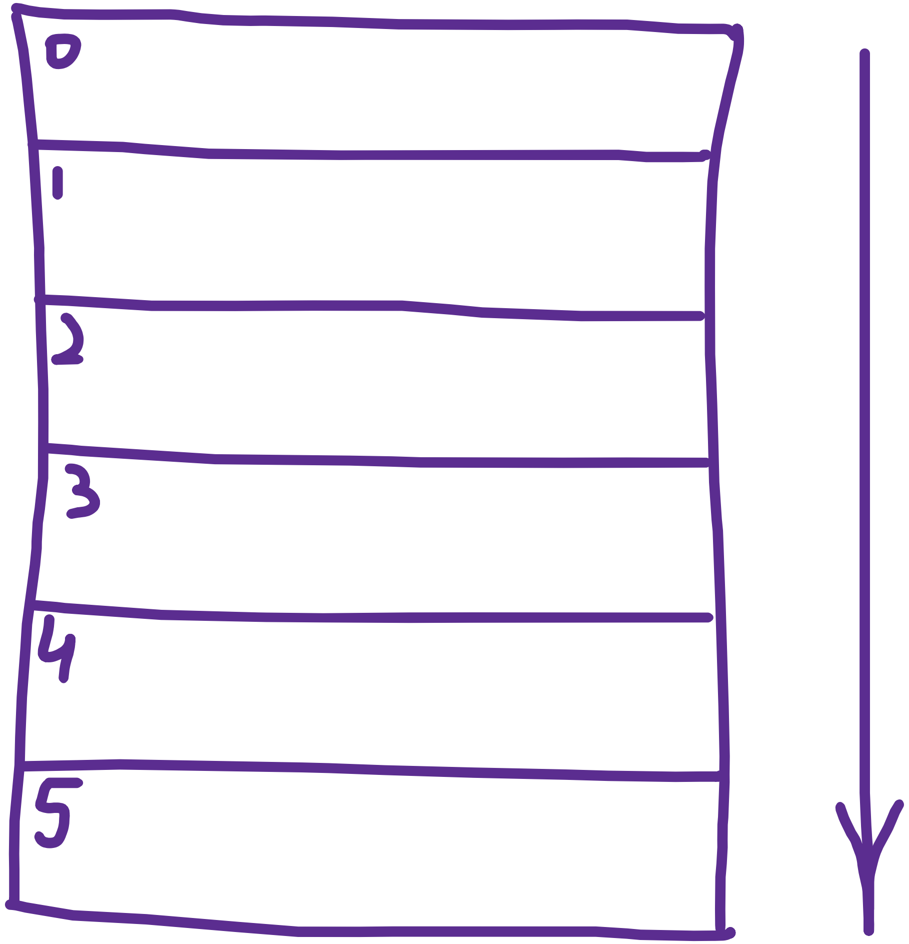
- Has dynamic length (items number)
- Can store only objects of the same type
- Add item on the top of the stack – O(1)
- Get item from the top of the stack – O(1)
- Search item – O(n)

Data structures – stack



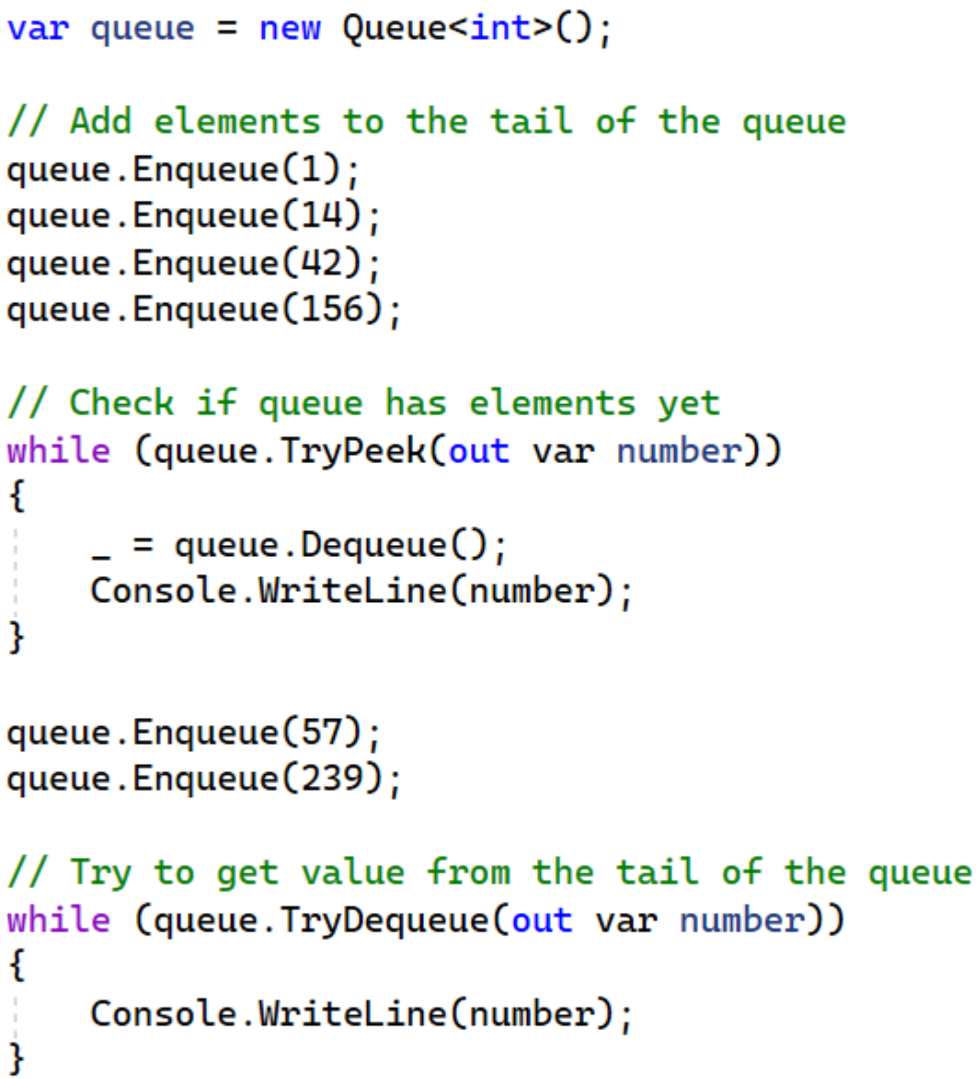
```
var stack = new Stack<string>();

// Put the element on the top of the stack
stack.Push("Kolobok");
stack.Push("It");
stack.Push("Master and Margarita");
stack.Push("Vinny The Pooh");

// Check if stack has elements yet
while (stack.TryPeek(out var book))
{
    // Get the element from the top of the stack
    _ = stack.Pop();
    Console.WriteLine(book);
}
```

Data structures – queue



- Has dynamic length (items number)
- Can store only objects of the same type
- Add item to the tail of the queue – O(1)
- Get item from the tale of the queue – O(1)
- Search item – O(n)

Data structures – queue



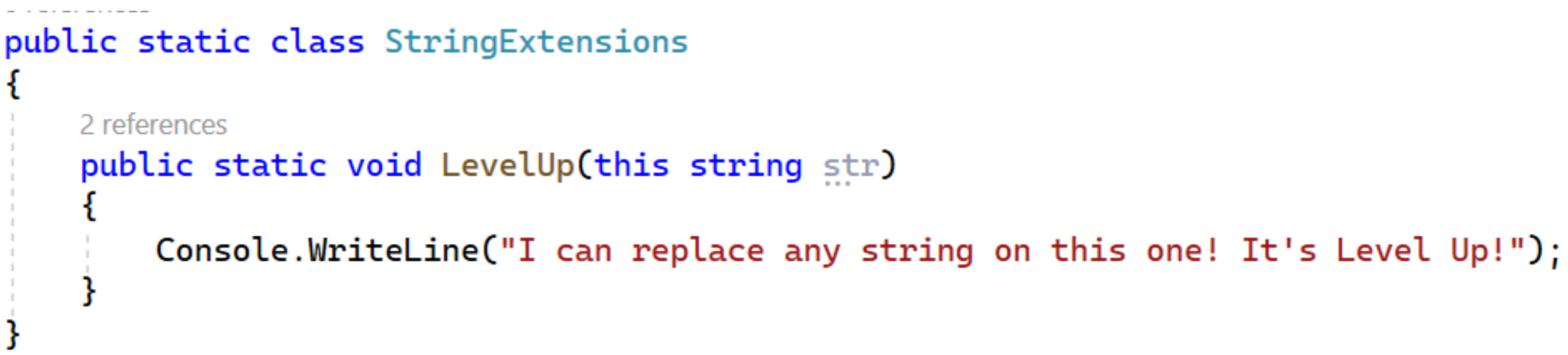
```
var queue = new Queue<int>();

// Add elements to the tail of the queue
queue.Enqueue(1);
queue.Enqueue(14);
queue.Enqueue(42);
queue.Enqueue(156);

// Check if queue has elements yet
while (queue.TryPeek(out var number))
{
    _ = queue.Dequeue();
    Console.WriteLine(number);
}

queue.Enqueue(57);
queue.Enqueue(239);

// Try to get value from the tail of the queue
while (queue.TryDequeue(out var number))
{
    Console.WriteLine(number);
}
```

Data structures demo

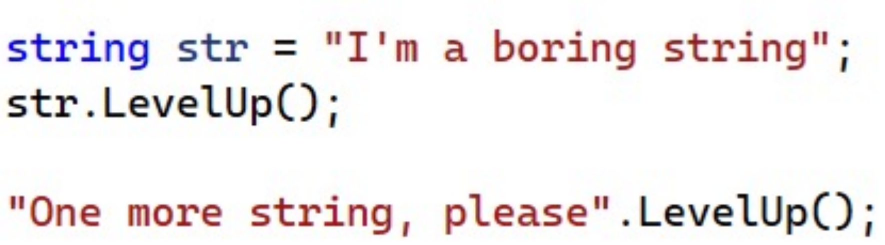
Visual Studio

Extension methods

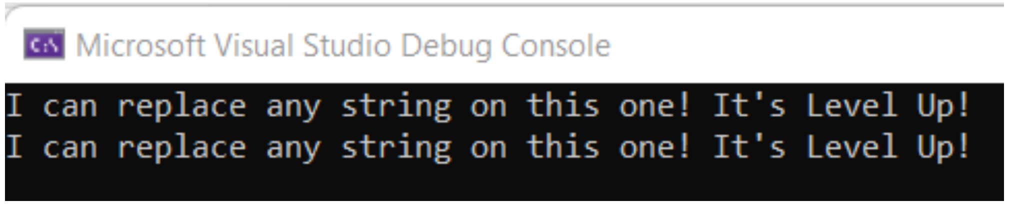
Extension methods



```
public static class StringExtensions
{
    2 references
    public static void LevelUp(this string str)
    {
        Console.WriteLine("I can replace any string on this one! It's Level Up!");
    }
}
```



```
string str = "I'm a boring string";
str.LevelUp();

"One more string, please".LevelUp();
```



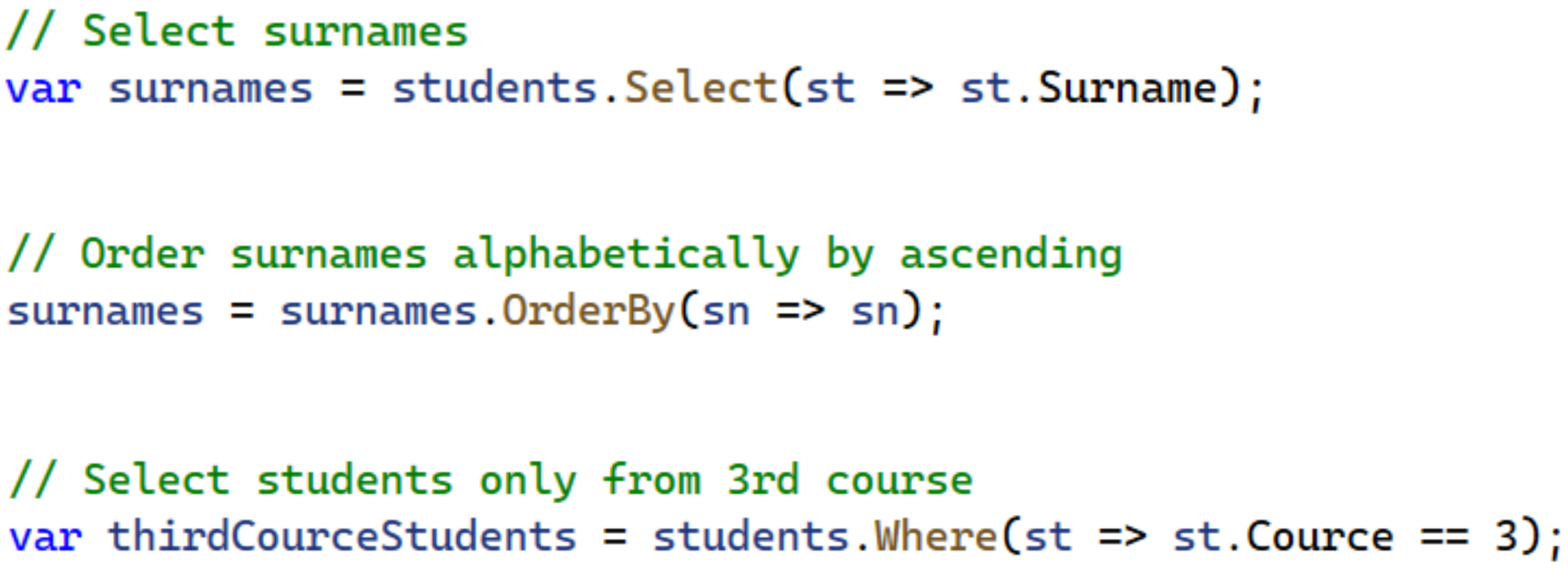
```
I can replace any string on this one! It's Level Up!
I can replace any string on this one! It's Level Up!
```

Extension methods demo

Visual Studio

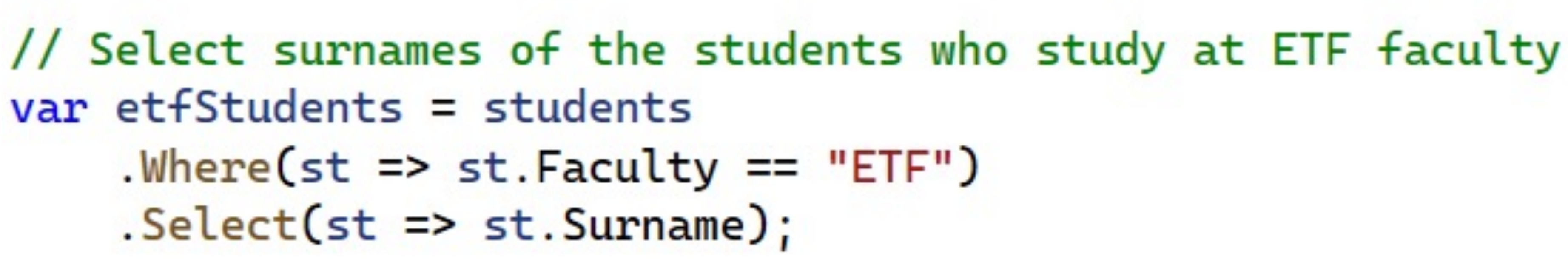
Language-Integrated Query (LINQ)

LINQ – extension methods

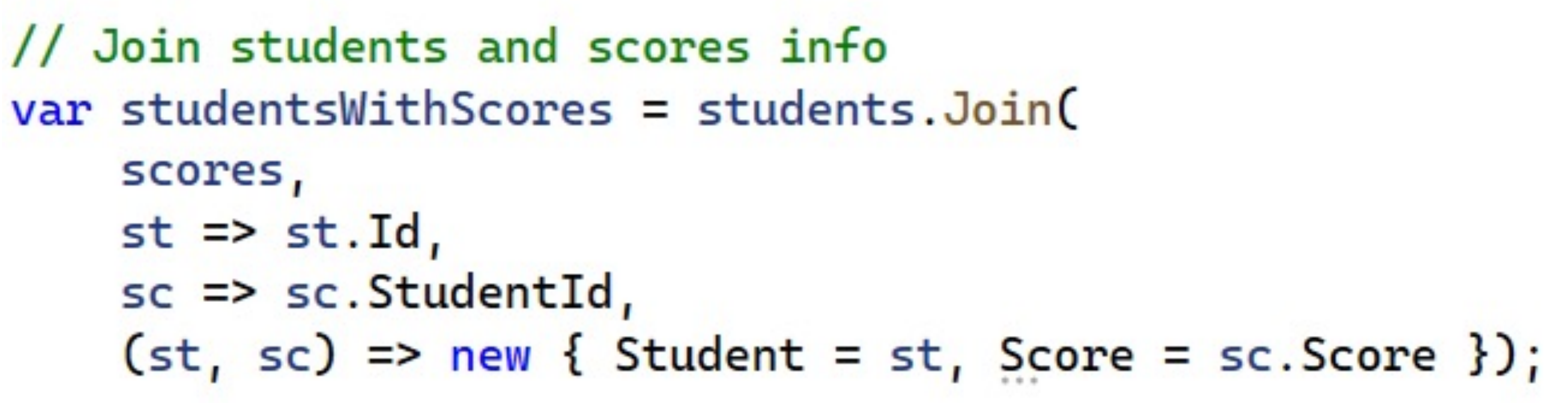


```
// Select surnames
var surnames = students.Select(st => st.Surname);


// Order surnames alphabetically by ascending
surnames = surnames.OrderBy(sn => sn);


// Select students only from 3rd course
var thirdCourceStudents = students.Where(st => st.Cource == 3);
```

LINQ – extension methods

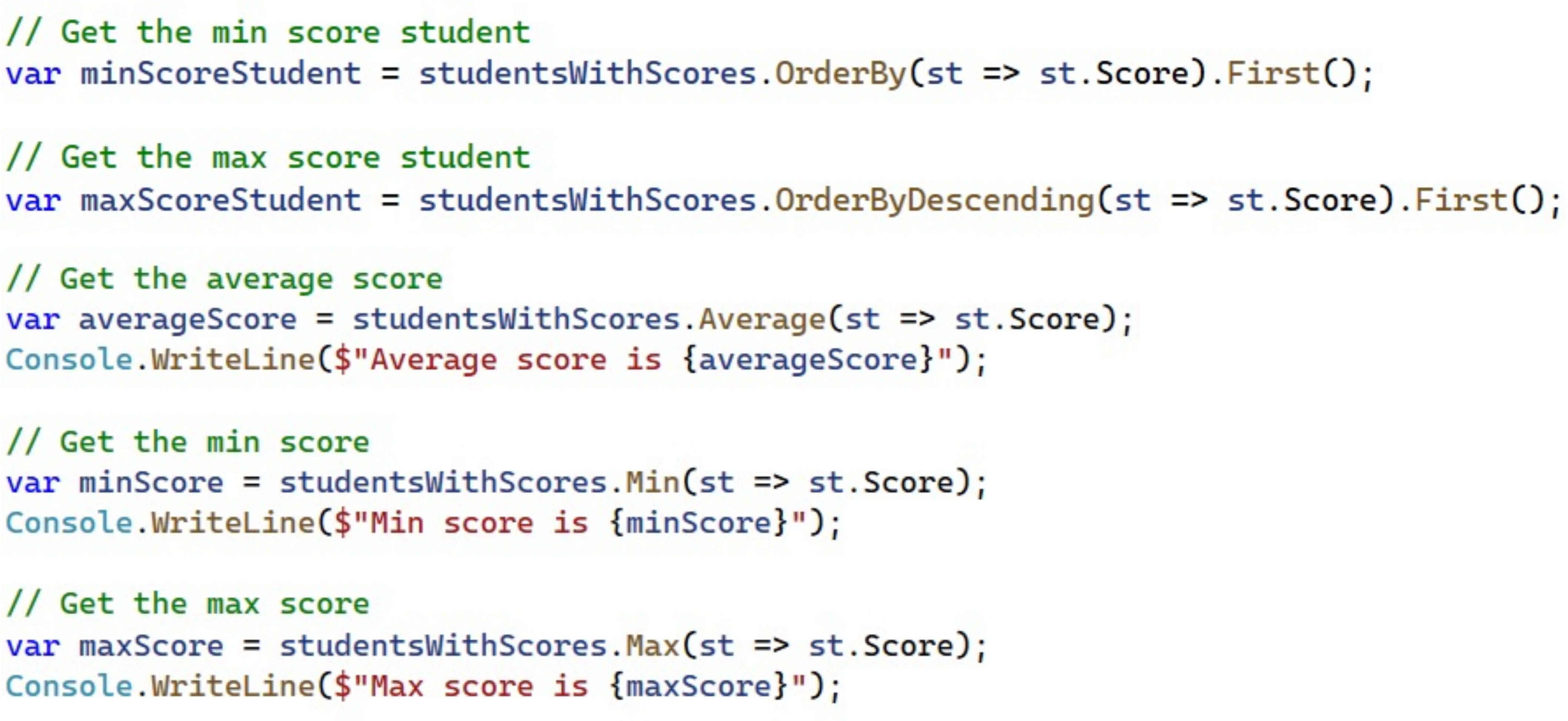


```
// Select surnames of the students who study at ETF faculty
var etfStudents = students
    .Where(st => st.Faculty == "ETF")
    .Select(st => st.Surname);
```



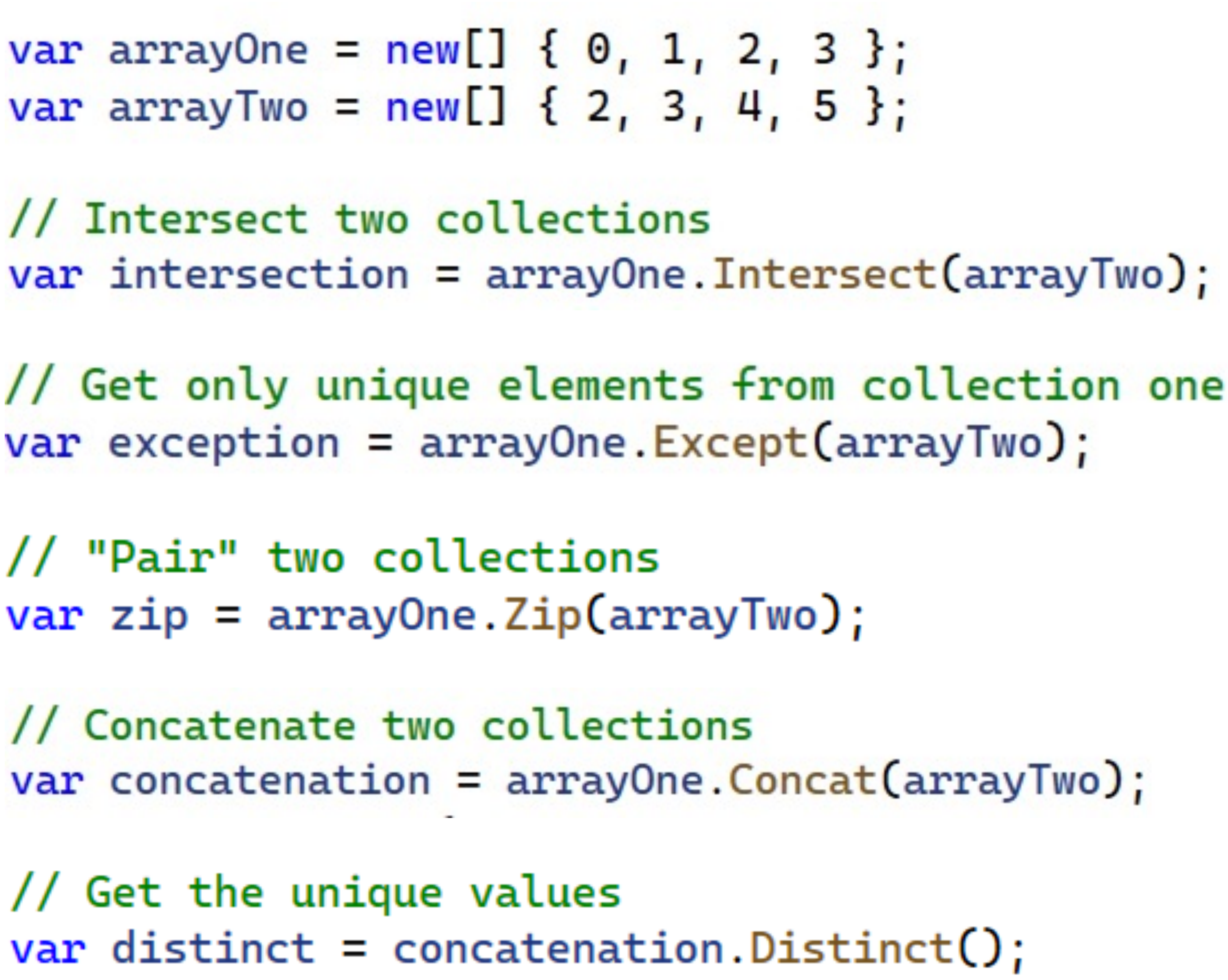
```
// Join students and scores info
var studentsWithScores = students.Join(
    scores,
    st => st.Id,
    sc => sc.StudentId,
    (st, sc) => new { Student = st, Score = sc.Score });
```

LINQ – extension methods



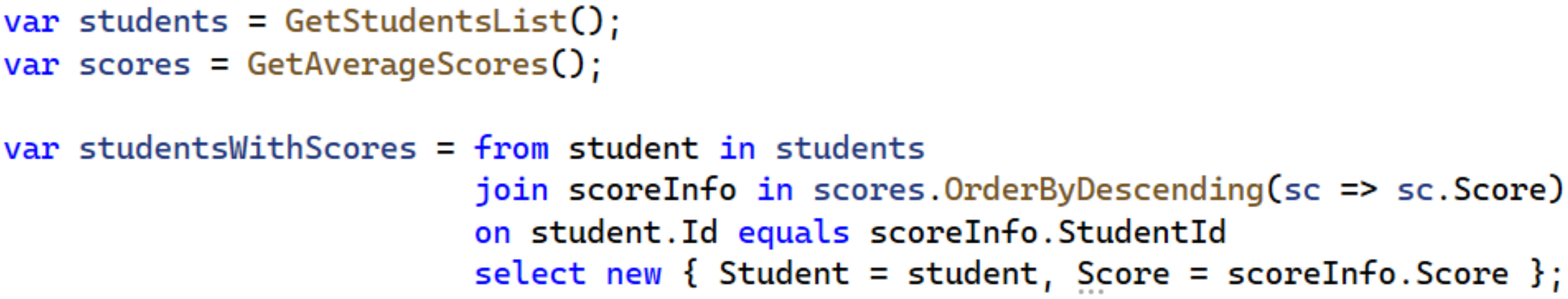
```
// Get the min score student
var minScoreStudent = studentsWithScores.OrderBy(st => st.Score).First();

// Get the max score student
var maxScoreStudent = studentsWithScores.OrderByDescending(st => st.Score).First();

// Get the average score
var averageScore = studentsWithScores.Average(st => st.Score);
Console.WriteLine($"Average score is {averageScore}");

// Get the min score
var minScore = studentsWithScores.Min(st => st.Score);
Console.WriteLine($"Min score is {minScore}");

// Get the max score
var maxScore = studentsWithScores.Max(st => st.Score);
Console.WriteLine($"Max score is {maxScore}");
```

LINQ – extension methods



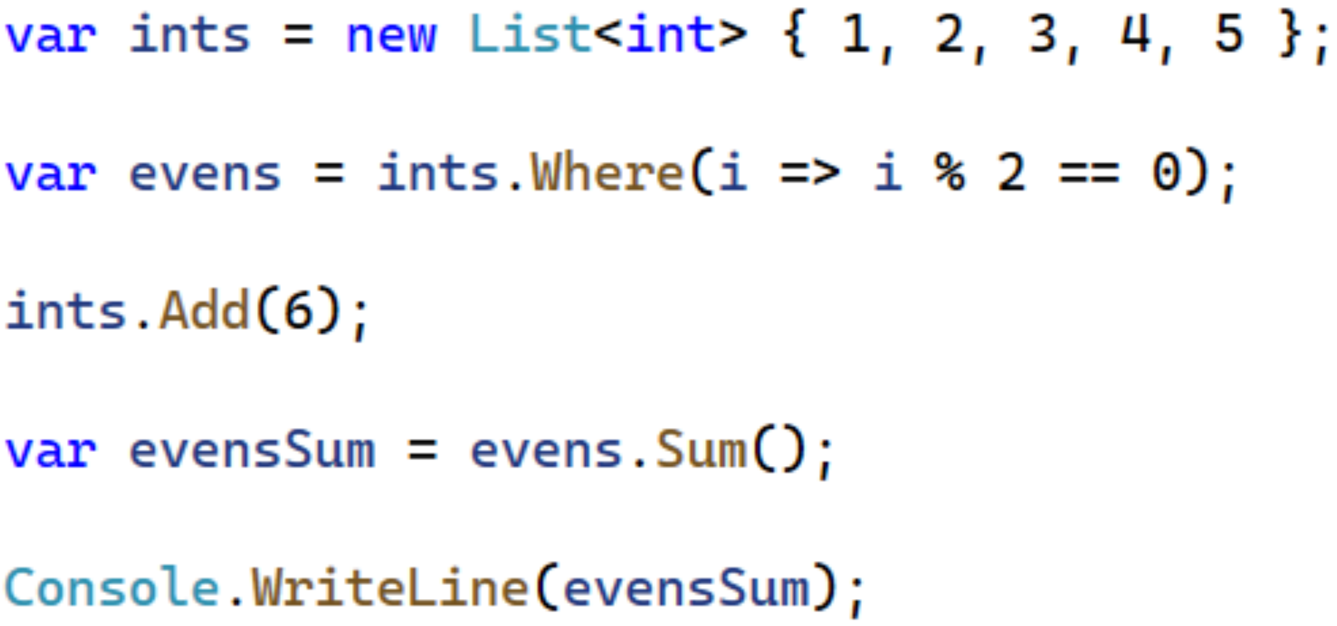
```
var arrayOne = new[] { 0, 1, 2, 3 };
var arrayTwo = new[] { 2, 3, 4, 5 };

// Intersect two collections
var intersection = arrayOne.Intersect(arrayTwo);

// Get only unique elements from collection one
var exception = arrayOne.Except(arrayTwo);

// "Pair" two collections
var zip = arrayOne.Zip(arrayTwo);

// Concatenate two collections
var concatenation = arrayOne.Concat(arrayTwo);

// Get the unique values
var distinct = concatenation.Distinct();
```

LINQ – SQL-like syntax



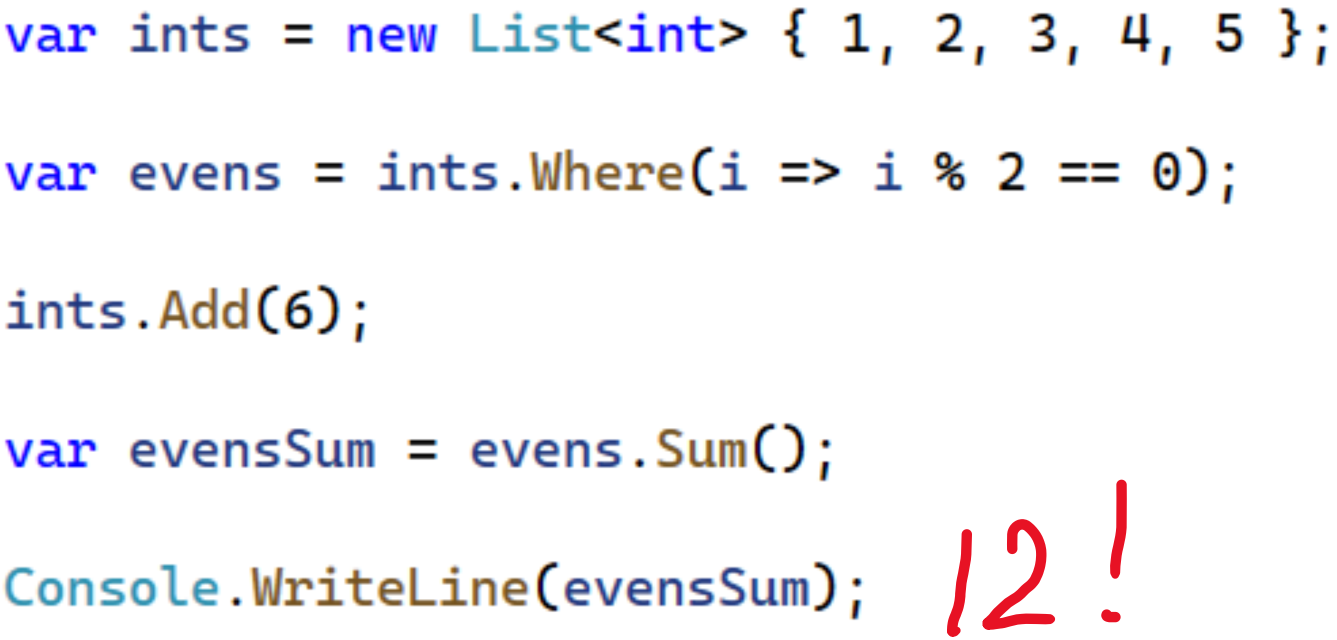
```
var students = GetStudentsList();
var scores = GetAverageScores();

var studentsWithScores = from student in students
                         join scoreInfo in scores.OrderByDescending(sc => sc.Score)
                         on student.Id equals scoreInfo.StudentId
                         select new { Student = student, Score = scoreInfo.Score };
```

LINQ – lazy calculation



```
var ints = new List<int> { 1, 2, 3, 4, 5 };

var evens = ints.Where(i => i % 2 == 0);

ints.Add(6);

var evensSum = evens.Sum();

Console.WriteLine(evensSum);
```

LINQ – lazy calculation



```
var ints = new List<int> { 1, 2, 3, 4, 5 };

var evens = ints.Where(i => i % 2 == 0);

ints.Add(6);

var evensSum = evens.Sum();

Console.WriteLine(evensSum);
```

12!

LINQ demo

Visual Studio

Books of the day

Aditiya Y. Bhargava – Grokking Algorithms

T. H. Cormen – Introducing To Algorithms

Links of the day

Dictionaries in C# Under the Hood - Habr

LINQ explained with sketches (steven-giesel.com)

LINQ Basics (dotnetpattern.com)

101 LINQ Samples

LINQPad - The .NET Programmer's Playground

Hometask

Create extension method Clear for System.String class. Method should remove all non-letter and non-numeric symbols (e.g. “\n”, “\t”, start and end whitespaces etc). Result of this method should be cleared string that contains only letters (a-z, A-Z), numbers (0-9) and whitespaces between the words.
Example: input string ” \tThis is \n a test string 123 ” should be converted to ”This is a test string 123”:

```
var testString = ” \tThis is \n a test string ”;
var clearedString = testString.Clear();
```

That’s all for this time!