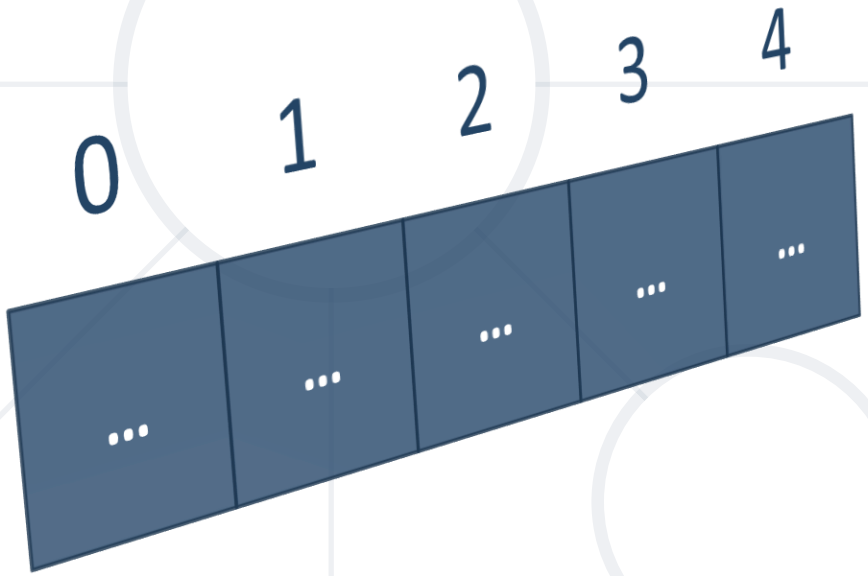
Stacks and Queues

Processing Sequences of Elements



SoftUni Team

Technical Trainers

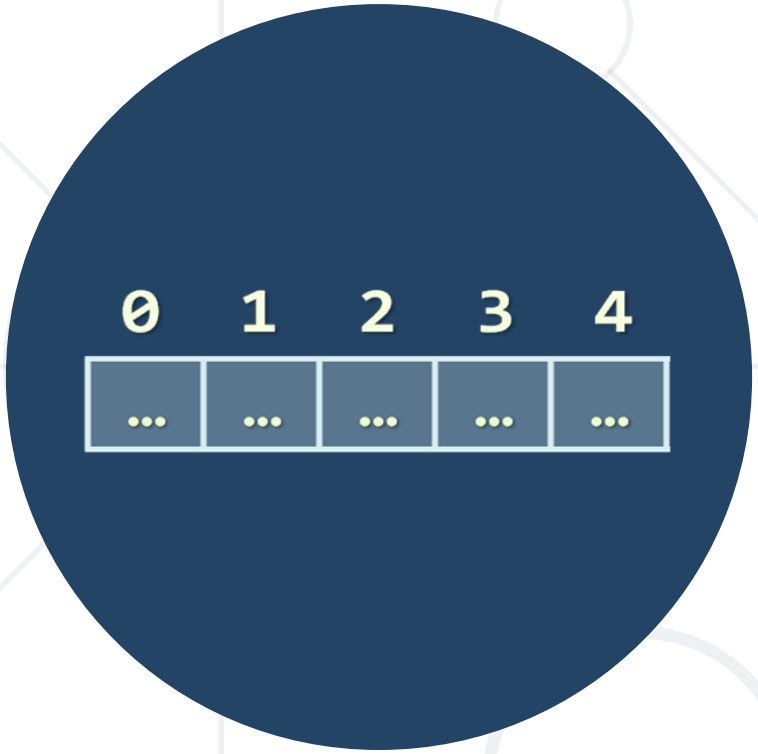
Software University

SoftUni

Software University

https://about.softuni.bg/

Table of Contents



Linear Data Structures

What is a Data Structure?

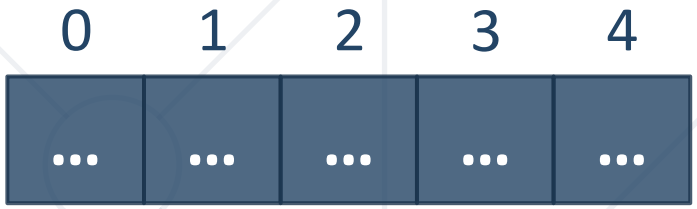
Software University

> **"In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently."** ***-- Wikipedia***

- Examples of data structures:
  - `Person` structure (first name + last name + age)
  - Array of integers – `int[]`
  - List of strings – `List<string>`
  - Queue of people – `Queue<Person>`

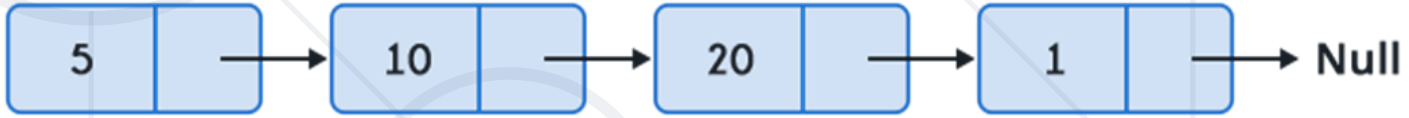
Data Structures

- **Data structures** are representations of data in the computer memory, which allow efficient access and modification
- **Linear data types**: arrays, lists, stacks, queues



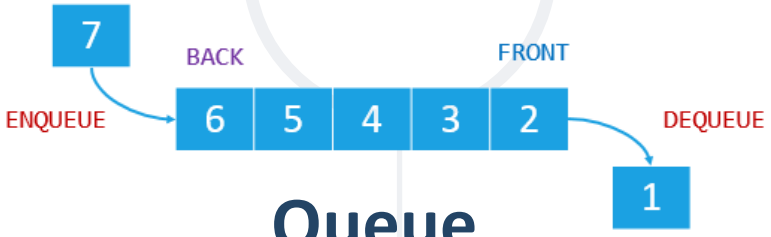
Array / list

(indexed group of elements)



Linked list

(sequence of linked elements)

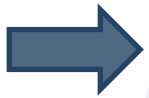


Queue

List of Numbers – Example

- **List of numbers**, representing a sequence of income amounts:

```
var incomes =
  new List<double>() {
    150, 200, 70.50, 120
  };
```



Element	Value
incomes[0]	150
incomes[1]	200
incomes[2]	70.50
incomes[3]	120
incomes[4]	300

250

- Adding a **new income**:

```
incomes.Add(300);
```

- **Modifying** an existing income:

```
incomes[1] = 250;
```

Why Are Data Structures So Important?

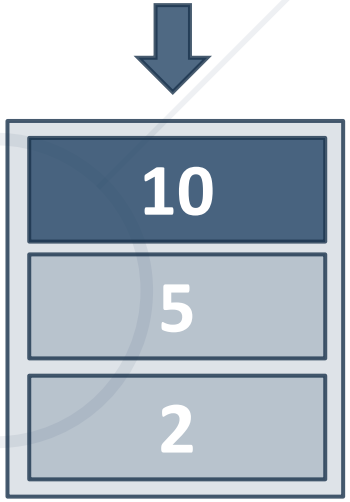
Software University

- **Data structures** and algorithms are the foundation of computer programming
- Algorithmic thinking, problem-solving and data structures are vital for software engineers
  - C# developers should know when to use `T[]`, `LinkedList<T>`, `List<T>`, `Stack<T>`, `Queue<T>`, `Dictionary<K, T>`, `HashSet<T>`, `SortedDictionary<K, T>` and `SortedSet<T>`
- **Programming == algorithms + data structures**!

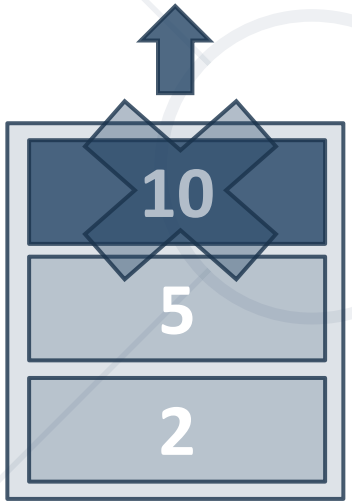
Overview and Working with Stack

Stack – Abstract Data Type

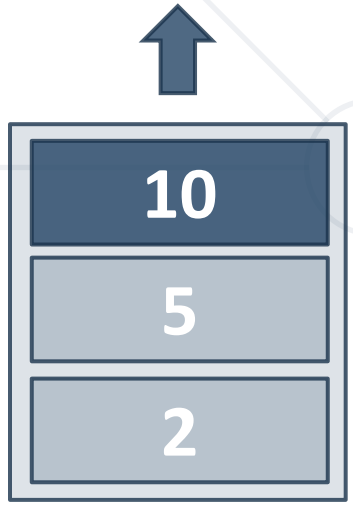
- **Stacks** provide the following functionality:
  - **Pushing** an element at the top of the stack
  - **Popping** element from the top of the stack
  - **Peeking** the topmost element without removing it



Push



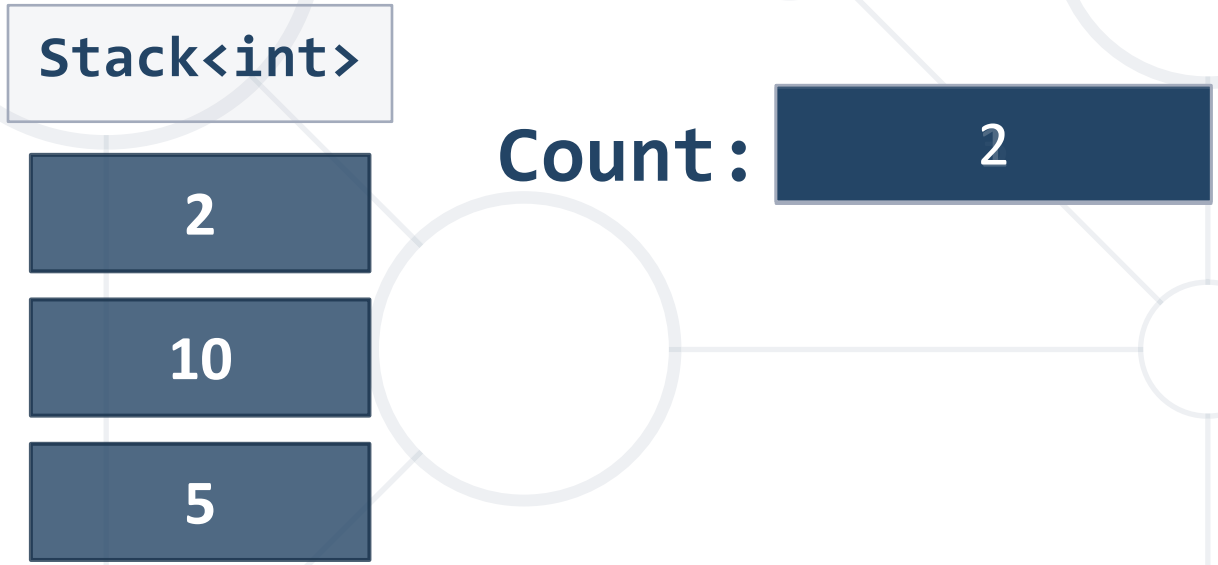
Pop



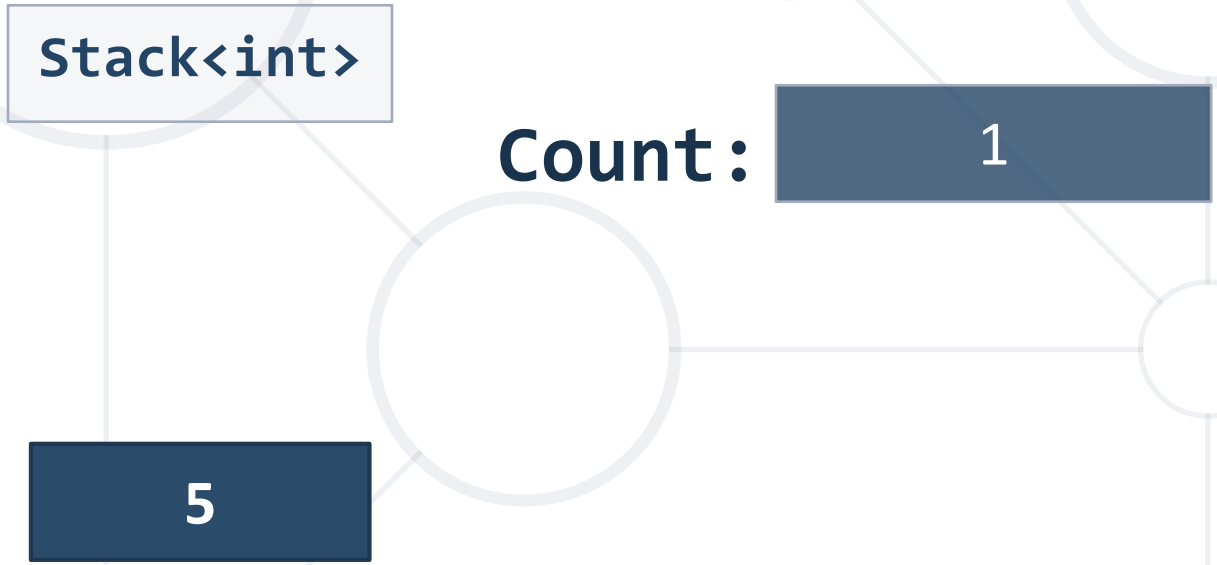
Peek

Push() – Adds an Element On Top of the Stack

Pop() – Returns and Removes the Top Element



Peek() – Returns the Element at the Top



Problem: Reverse Strings

- Create a program that:
  - Reads an **input string**
  - **Reverses** it using a **stack**

```
I Love C#  ➡  #C evoL I
```

```
Stacks and Queues  ➡  seueuQ dna skcatS
```

Solution: Reverse Strings

```
var input = Console.ReadLine();
var stack = new Stack<char>();
foreach (var ch in input)
{
  stack.Push(ch);
}
while (stack.Count != 0)
{
  Console.Write(stack.Pop());
}
Console.WriteLine();
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/3174#0

Stack – Methods

```
Stack<int> stack = new Stack<int>();

int count = stack.Count;
bool exists = stack.Contains(2);
int[] array = stack.ToArray();
stack.Clear();
stack.TrimExcess();
```

Retains the order of elements

Remove all elements

Resize the internal array

Problem: Simple Calculator

- Implement a **simple calculator** that can evaluate simple expressions (only addition and subtraction)

`2 + 5 + 10 - 2 - 1`	⇒	`14`
`2 - 2 + 5`	⇒	`5`
`2 - 1 + 5`	⇒	`6`
`2 - 0 + 5`	⇒	`7`

Solution: Simple Calculator (1)

```
var input = Console.ReadLine();
var values = input.Split(' ');
var stack = new Stack<string>(values.Reverse());
while (stack.Count > 1)
{
  int first = int.Parse(stack.Pop());
  string operator = stack.Pop();
  int second = int.Parse(stack.Pop());
  // TODO: Add switch for operation
}
Console.WriteLine(stack.Pop());
```

Solution: Simple Calculator (2)

```
switch (operator)
{
   case "+":
      stack.Push((first + second).ToString());
      break;
   case "-":
      stack.Push((first - second).ToString());
      break;
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/3174#2

Problem: Stack Sum

- Calculate the **sum in the stack**
  - Before that you will receive commands
    - **Add:** adds the two numbers
    - **Remove**: removes count numbers

```
1 2 3 4
adD 5 6
REmove 3
eNd
```

```
Sum: 6
```

```
3 5 8 4 1 9
add 19 32
remove 10
add 89 22
end
```

```
Sum: 192
```

Solution: Stack Sum (1)

```
var input =
     Console.ReadLine().Split().Select(int.Parse).ToArray();
Stack<int> stack = new Stack<int>(input);
var commandInfo = Console.ReadLine().ToLower();

while (commandInfo != "end")
{
  var tokens = commandInfo.Split();
  var command = tokens[0].ToLower();
  if (command == "add")
    // TODO: Parse the numbers and add them
  else if(…)
```

Solution: Stack Sum (2)

```
    else if(command == "remove") {
        var countOfRemovedNums = int.Parse(tokens[1]);
        if (stack.Count < countOfRemovedNums)
            continue;
        for (int i = 0; i < countOfRemovedNums; i++)
            stack.Pop();
    }
    commandInfo = Console.ReadLine().ToLower();
}

var sum = stack.Sum();
Console.WriteLine($"Sum: {sum}");
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/3174#1

Problem: Matching Brackets

- We are **given an arithmetic expression** with brackets (**with nesting**)
- **Extract all sub-expressions** in brackets

```
1 + (2 - (2 + 3) * 4 / (3 + 1)) * 5
```

```
(2 + 3)
(3 + 1)
(2 - (2 + 3) * 4 / (3 + 1))
```

Solution: Matching Brackets

```
var input = Console.ReadLine();
var stack = new Stack<int>();
for (int i = 0; i < input.Length; i++) {
  char ch = input[i];
  if (ch == '(') {
    stack.Push(i);
  } else if (ch == ')') {
    int startIndex = stack.Pop();
    string contents = input.Substring(
                      startIndex, i - startIndex + 1);
    Console.WriteLine(contents);
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/3174#3

Overview and Working with Queue

Queue – Abstract Data Type

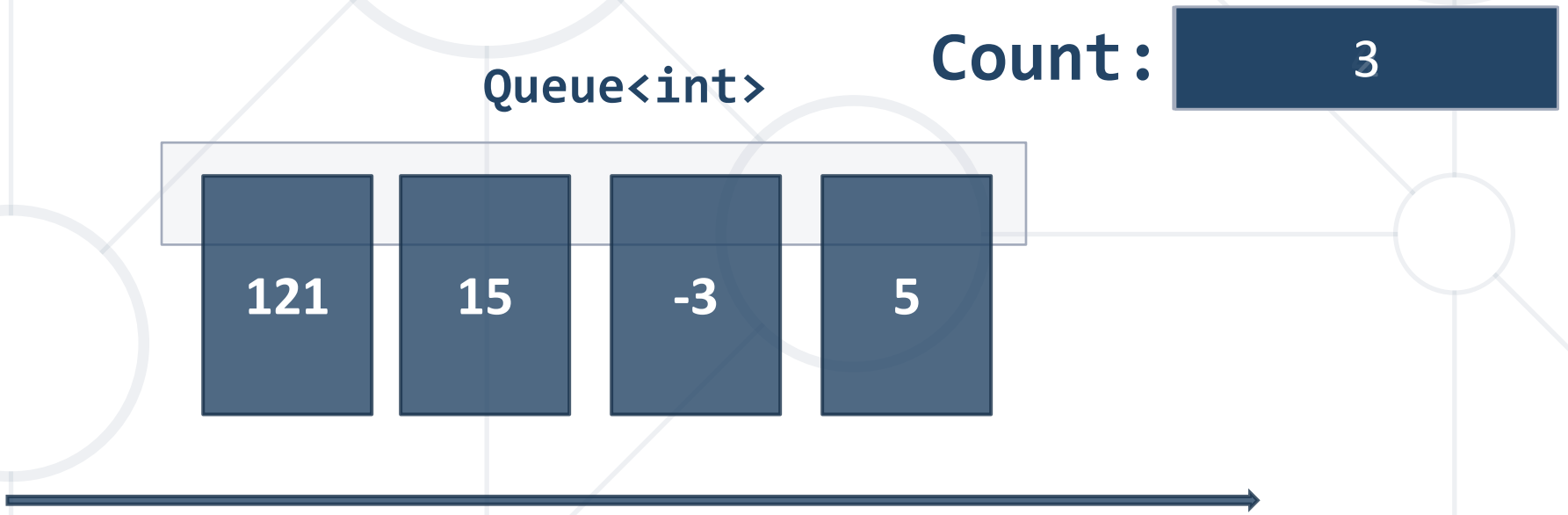
- **Queues** provide the **following functionality**:
  - Adding an element at the end of the queue

  - Removing the first element from the queue

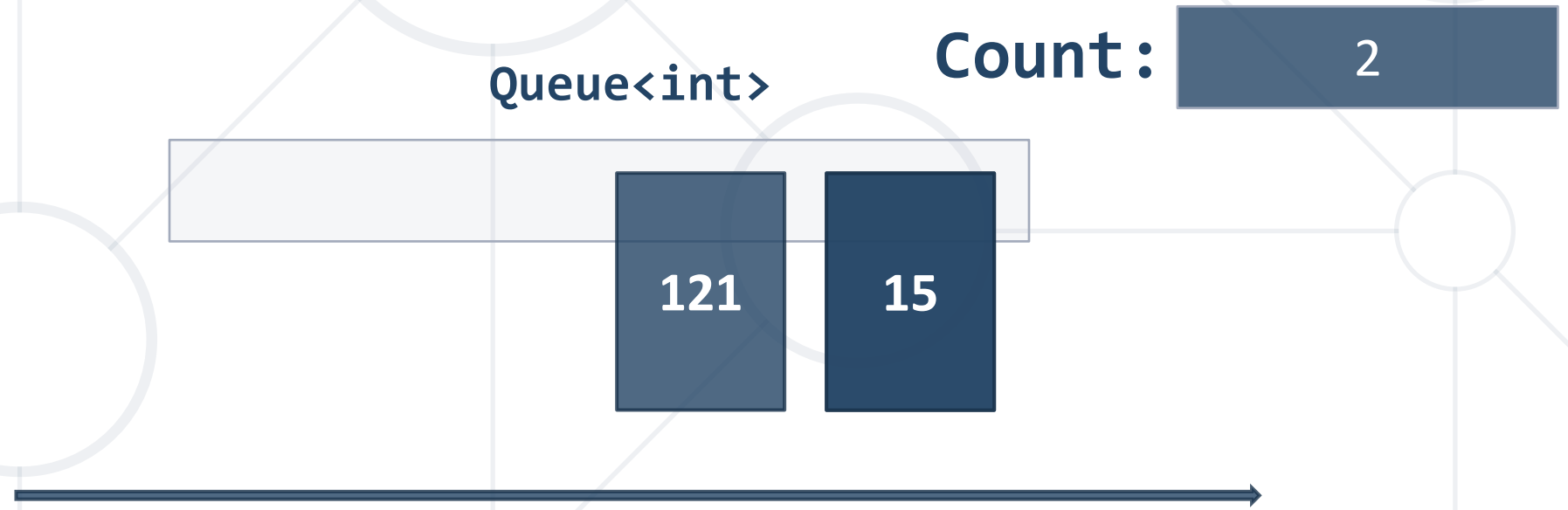
  - Getting the first element of the queue without removing it

Enqueue() – Adds an Element to the Front

Dequeue() – Returns and Removes the First Element



Peek() – Returns the First Element



Problem: Hot Potato

- Children **form a circle** and pass a hot potato **clockwise**
- Every n-th toss **a child is removed** until **only one remains**
- **Upon removal** the potato is passed **along**
- Print the child that remains last

```
Alva James William
2
```

```
Removed James
Removed Alva
Last is William
```

Solution: Hot Potato

```
var children = Console.ReadLine().Split(' ');
var number = int.Parse(Console.ReadLine());
Queue<string> queue = new Queue<string>(children);
while (queue.Count != 1)
{
  for (int i = 1; i < number; i++)
  {
    queue.Enqueue(queue.Dequeue());
  }
  Console.WriteLine($"Removed {queue.Dequeue()}");
}
Console.WriteLine($"Last in {queue.Dequeue()}");
```

Copies elements from the specified collection and keeps their order

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/3174#6

Queue – Methods

```
Queue<int> queue = new Queue<int>();
int count = queue.Count;
bool exists = queue.Contains(2);
int[] array = queue.ToArray();
queue.Clear();
queue.TrimExcess();
```

Retains the order of elements

Remove all elements

Resize the internal array

Problem: Traffic Jam

- Cars are **queuing up** at a **traffic light**
- Every **green light** n cars **pass** the crossroads
- After the **end command**, print **how many cars** have **passed**

```
3
Enzo's car
Jade's car
Mercedes CLS
Audi
green
BMW X5
green
end
```

```
Enzo's car passed!
Jade's car passed!
Mercedes CLS passed!
Audi passed!
BMW X5 passed!
5 cars passed the crossroads.
```

Solution: Traffic Jam

```
int n = int.Parse(Console.ReadLine());
var queue = new Queue<string>();
int count = 0;
string command;
while ((command = Console.ReadLine()) != "end")
{
  if (command == "green")
    // TODO: Add green light logic
  else
    queue.Enqueue(command);
}
Console.WriteLine($"{count} cars passed the crossroads.");
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/3174#7

Summary

- Linear data structures hold sequences of elements
- **Stack<T>**
  - **LIFO** data structure
- **Queue<T>**
  - **FIFO** data structure
- Working with **built-in methods**

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg