Arrays and Lists

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

Table of Contents

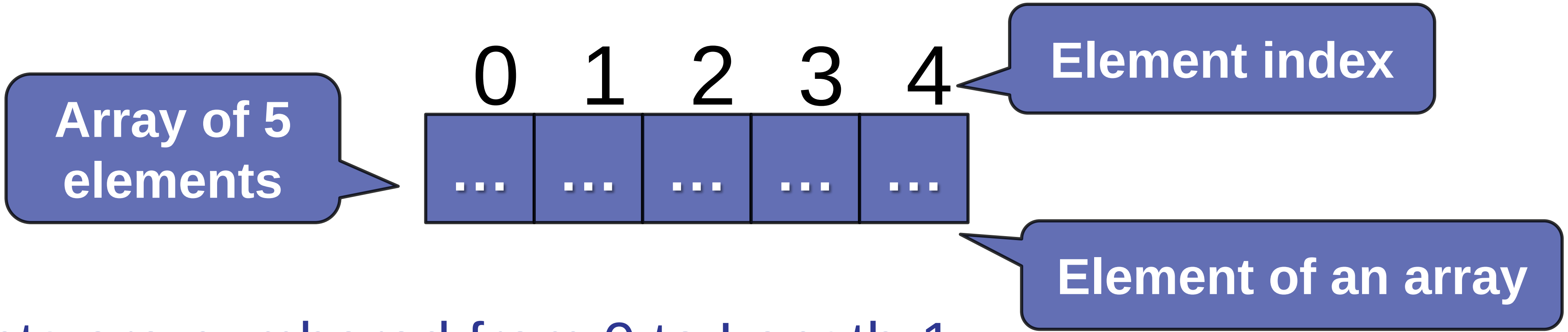
- Arrays
  - Value vs Reference Types
  - Reading Arrays from the Console
  - Foreach Loop
- Lists Overview
  - List Manipulating
  - Reading Lists from the Console
  - Sorting Lists and Arrays

ARRAYS

Working with Arrays of Elements

What are Arrays?

- In programming, an array is a sequence of elements



  - Elements are numbered from 0 to Length-1
  - Elements are of the same type (e.g. integers)
  - Arrays have fixed size (Array.Length) cannot be resized

Working with Arrays

- Allocating an array of 10 integers:

```
int[] numbers = new int[10];
```

All elements are initially == 0

- Assigning values to the array elements:

```
for (int i = 0; i < numbers.Length; i++)
  numbers[i] = 1;
```

The Length holds the number of array elements

- Accessing array elements by index:

```
numbers[5] = numbers[2] + numbers[7];
numbers[10] = 1; // IndexOutOfRangeException
```

The [] operator accesses elements by index

Days of Week – Example

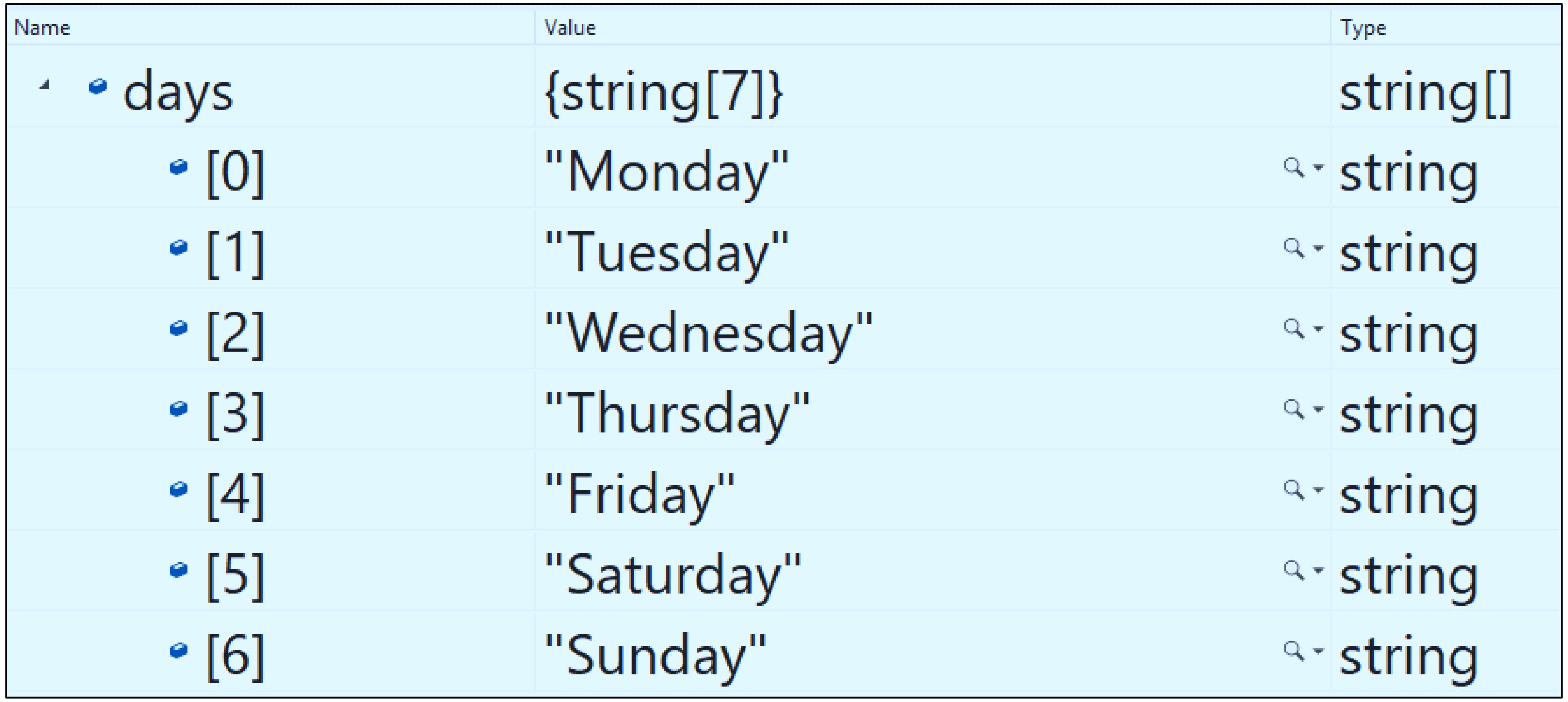
- The days of week can be stored in array of strings:

```
string[] days = {
   "Monday",
   "Tuesday",
   "Wednesday",
   "Thursday",
   "Friday",
   "Saturday",
   "Sunday"
};
```

Operator	Notation in C#
days[0]	Monday
days[1]	Tuesday
days[2]	Wednesday
days[3]	Thursday
days[4]	Friday
days[5]	Saturday
days[6]	Sunday

Problem: Day of Week

- Enter a day number [1…7] and print the day name (in English) or "Invalid day!"



Name	Value	Type
days	{string[7]}	string[]
[0]	"Monday"	string
[1]	"Tuesday"	string
[2]	"Wednesday"	string
[3]	"Thursday"	string
[4]	"Friday"	string
[5]	"Saturday"	string
[6]	"Sunday"	string

Solution: Day of Week

```
string[] days = { "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday" };
int day = int.Parse(Console.ReadLine());


if (day >= 1 && day <= 7)
  Console.WriteLine(days[day - 1]);
else
  Console.WriteLine("Invalid day!");
```

The first day in our array is on index 0, not 1.

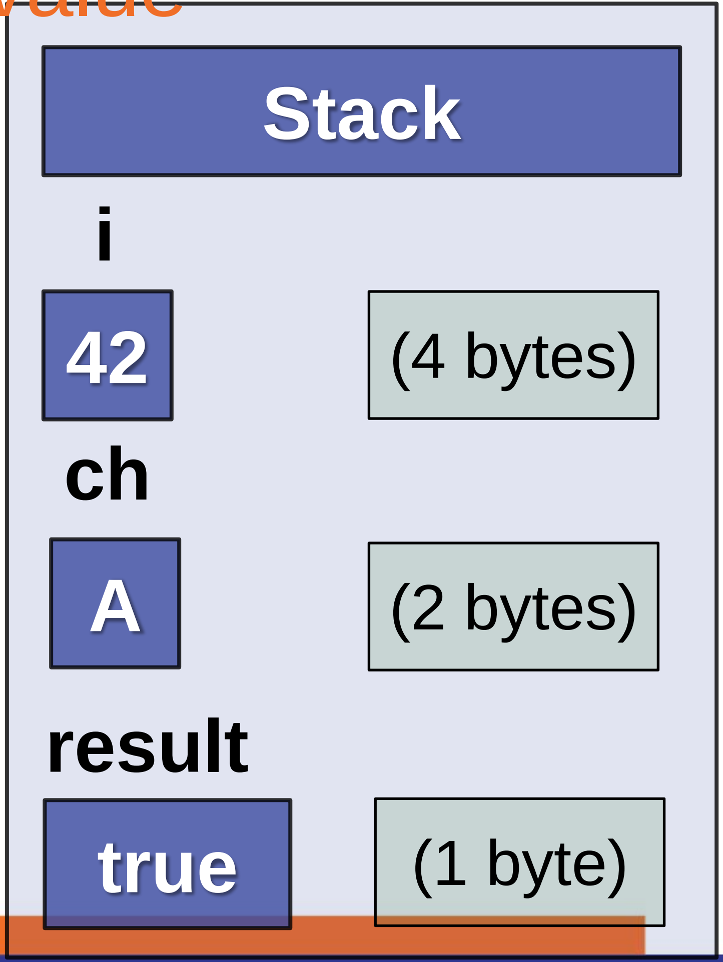
VALUE VS. REFERENCE TYPES

Memory Stack and Heap

Value Types

- Value type variables hold directly their value
  - int, float, double, bool, char, BigInteger, …
- Each variable has its own copy of the value

```
int i = 42;
char ch = 'A';
bool result = true;
```

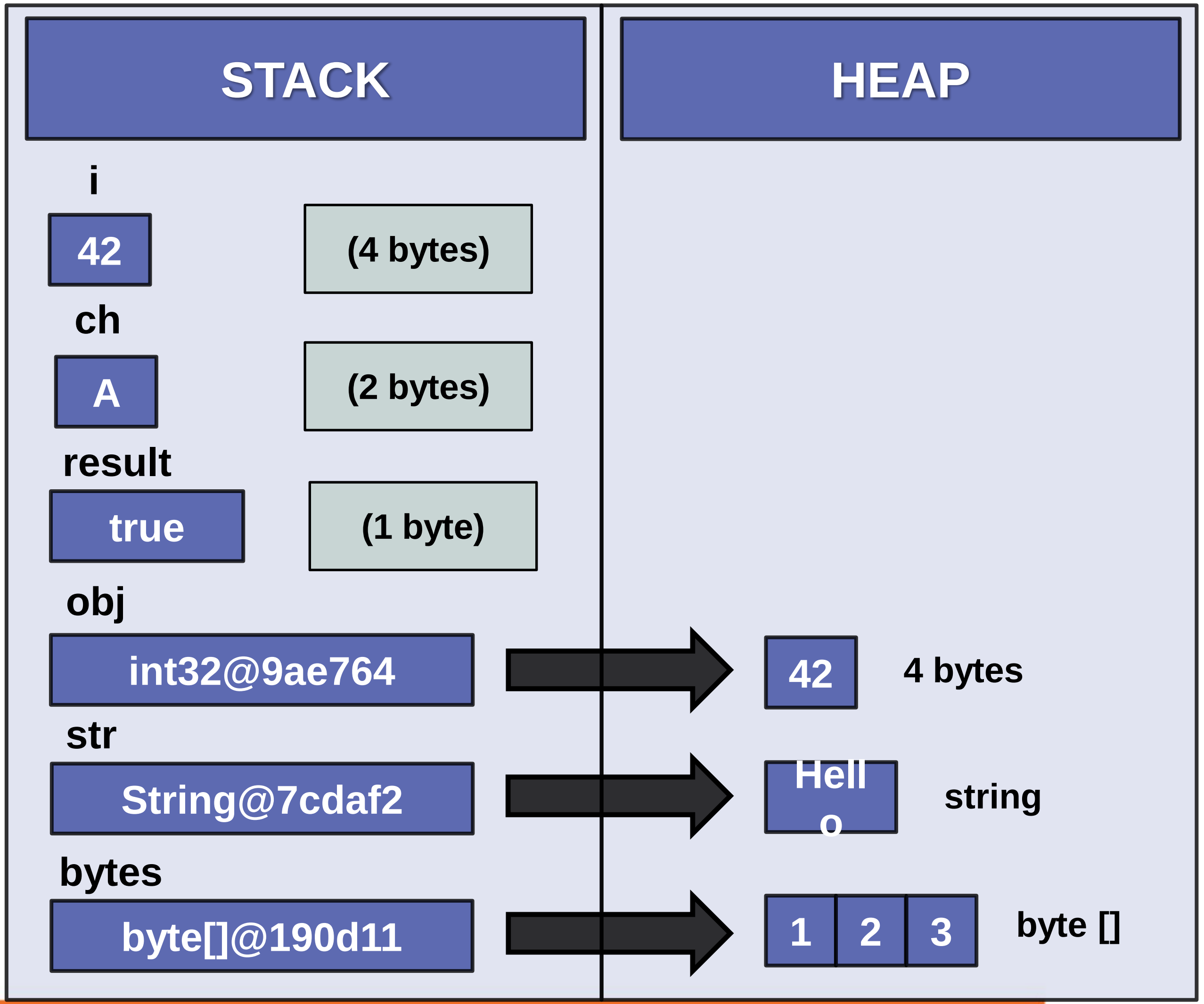


Reference Types

- Reference type variables hold a reference (pointer / memory address) of the value itself
  - string, int[], char[], string[], Random
- Two reference type variables can reference the same object
  - Operations on both variables access / modify the same data

Value Types vs. Reference Types

```
int i = 42;
char ch = 'A';
bool result = true;
object obj = 42;
string str = "Hello";
byte[] bytes ={ 1, 2, 3 };
```



Example: Value Types

```
public static void Main() {
   int num = 5;
   Increment(number, 15);
   Console.WriteLine(number);
}


public static void Increment(int num, int value) {
   num += value;
}
```

number == 5

num == 20

Example: Reference Types

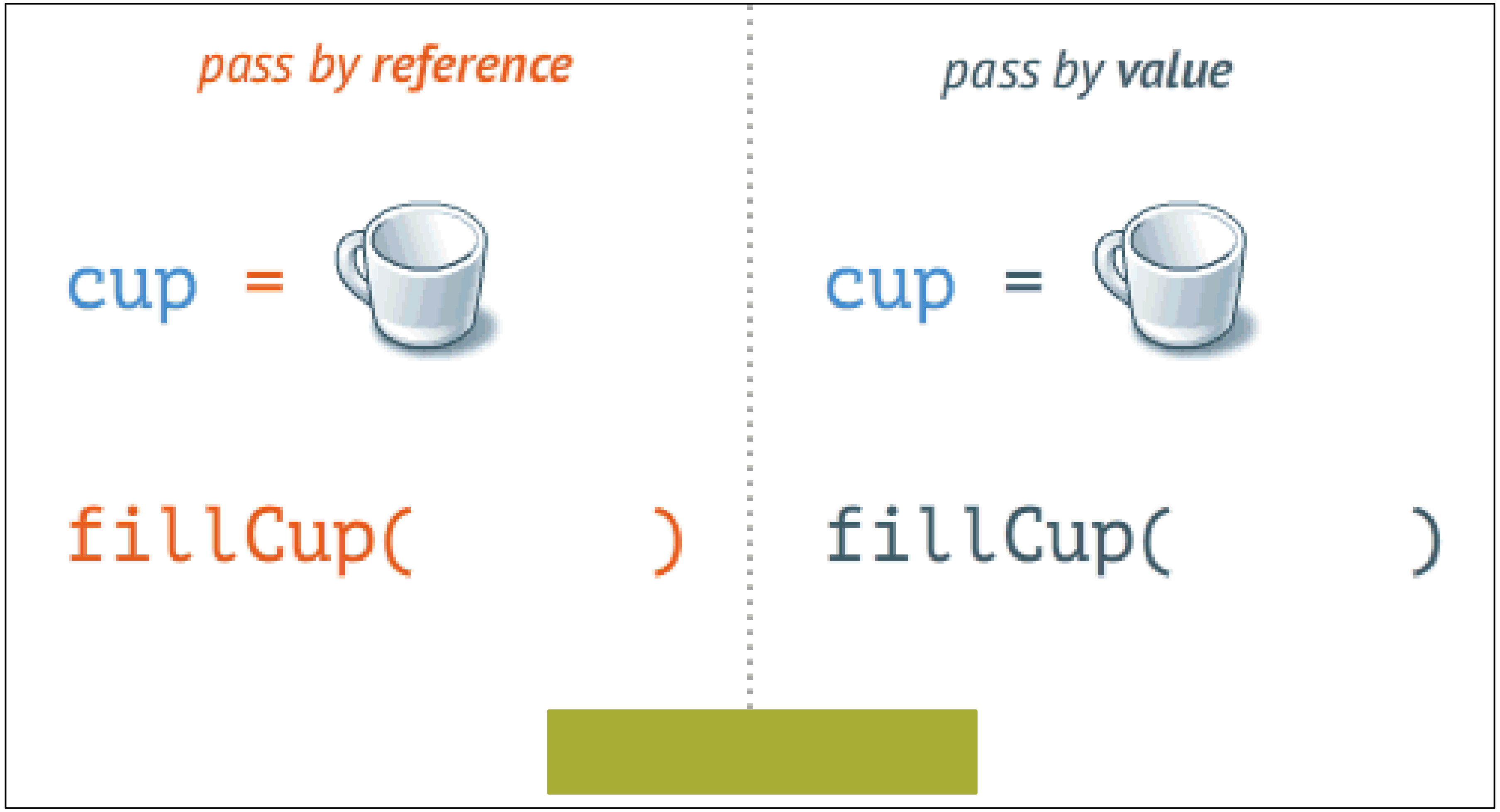
```
public static void Main() {
  int[] nums = { 5 };
  Increment(nums, 15);
  Console.WriteLine(nums[0]);
}

public static void Increment(int[] nums, int value) {
  nums[0] += value;
}
```

nums[0] == 20

nums[0] == 20

Value vs. Reference Types



UNIVERSITY of GREENWICH

Alliance with FPT Education

READING ARRAY

Using a `for` Loop or `String.Split()`

Reading Arrays From the Console

- First, read from the console the array length:

```
int n = int.Parse(Console.ReadLine());
```

- Next, create an array of given size n and read its elements:

```
int[] arr = new int[n];
for (int i = 0; i < n; i++) {
  arr[i] = int.Parse(Console.ReadLine());
}
```

Reading Array Values from a Single Line

- Arrays can be read from a single line of separated values

```
2 8 30 25 40 72 -2 44 56
```

```
string values = Console.ReadLine();
string[] items = values.Split();
int[] arr = new int[items.Length];

for (int i = 0; i < items.Length; i++)
  arr[i] = int.Parse(items[i]);
```

Split( ) splits by space into string[]

Shorter: Reading Array from a Single Line

- Read an arrays of integers using functional programming:

```
var inputLine = Console.ReadLine();
string[] items = inputLine.Split(' ');
int[] arr = items.Select(int.Parse).ToArray();
```

using System.LINQ;

```
int[] arr = Console.ReadLine().Split(' ')
                    .Select(int.Parse).ToArray();
```

Or shorter

Printing Arrays on the Console

- To print all array elements, a for-loop can be used
  - Separate elements with white space or a new line

```
string[] arr = {"one", "two"};
// == new string [2] {"one", "two"};
// Process all array elements
for (int index = 0; index < arr.Length; index++) {
// Print each element on a separate line
  Console.WriteLine("arr[{0}] = {1}", index, arr[index]);
}
```

Problem: Print Numbers in Reverse Order

- Read an array of integers (n lines of integers), reverse it and print its elements on a single line, space-separated:

```
3
10
20
30
```

→

```
30 20 10
```

```
4
-1
20
99
5
```

→

```
5 99 20 -1
```

Solution: Print Numbers in Reverse Order

```
// Read the array (n lines of integers)
var n = int.Parse(Console.ReadLine());
var arr = new int[n];
for (int i = 0; i < n; i++)
  arr[i] = int.Parse(Console.ReadLine());
// Print the elements from the last to the first
for (int i = n-1; i >= 0; i--)
  Console.Write(arr[i] + " ");
Console.WriteLine();
```

Problem: Rounding Numbers

- Read an array of real numbers (space separated), round them in "away from 0" style and print the output as in the examples:

```
0.9 1.5 2.4 2.5 3.14
```

```
0.9 => 1
1.5 => 2
2.4 => 2
2.5 => 3
3.14 => 3
```

```
-5.01 -1.599 -2.5 -1.50 0
```

```
-5.01 => -5
-1.599 => -2
-2.5 => -3
-1.50 => -2
0 => 0
```

Solution: Rounding Numbers

- Rounding turns each value to the nearest integer

```
double[] nums = ReadNumbers(); // write your method
int[] roundedNums = new int[nums.Length];
for (int i = 0; i < nums.Length; i++) {
  roundedNums[i] = (int)Math
               .Round(nums[i], MidpointRounding.AwayFromZero);
}
// TODO: Print each number
```

2.5 => 3

Printing Arrays with for / String.Join(...)

- Use for-loop:

```
int[] arr = { 10, 20, 30, 40, 50};
for (int i = 0; i < arr.Length; i++)
        Console.WriteLine(arr[i]);
```

- Use :

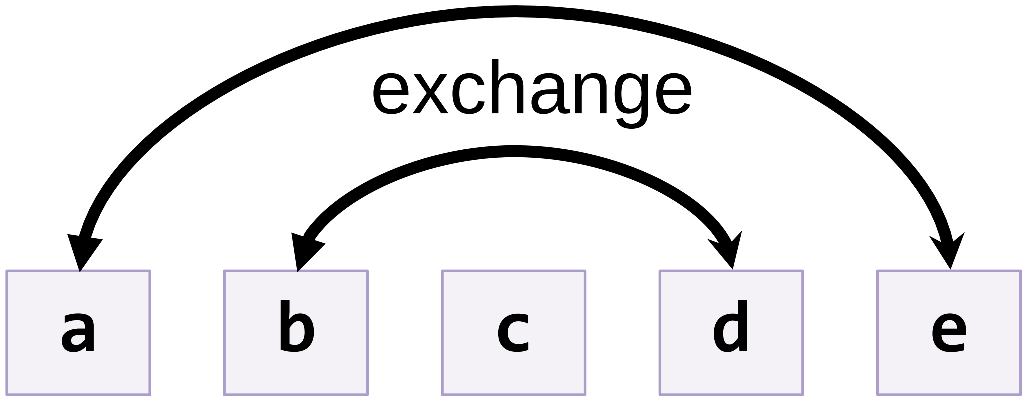
```
int[] arr = { 1, 2, 3 };
Console.WriteLine(string.Join(", ", arr)); // 1, 2, 3
string[] strings = { "one", "two" };
Console.WriteLine(string.Join(" - ", strings)); // one - two
```

Problem: Reverse Array of Strings

- Read an array of strings (space separated values), reverse it and print its elements:

```
a b c d e  ➡  e d c b a        -1 hi ho w  ➡  w ho hi -1
```

- Reversing array elements:



Solution: Reverse Array of Strings

```
var nums = Console.ReadLine().Split(' ').ToArray();
for (int i = 0; i < nums.Length / 2; i++)
  SwapElements(nums, i, nums.Length - 1 - i);
Console.WriteLine(string.Join(" ", nums));

static void SwapElements(string[] arr, int i, int j) {
  var oldElement = arr[i];
  arr[i] = arr[j];
  arr[j] = oldElement;
}
```

FOREACH LOOP

Iterate through Collections

Foreach Loop

- Iterates through all elements in a collection
- Cannot access the current index
- Read-only

```
foreach (var item in collection)
{
    // Process the value here
}
```

Print an Array with Foreach

```
int[] numbers = { 1, 2, 3, 4, 5 };
foreach (int number in numbers)
{
    Console.Write($"{number} ");
}
```

```
1 2 3 4 5
```

Summary

- Arrays hold a sequence of elements
  - Elements are numbered from 0 to length-1
- Creating (allocating) an array
- Accessing array elements by index
- Printing array elements

LISTS

List<T> – Overview

- List<T> holds a list of elements of any type

```
List<string> names = new List<string>();
//Create a list of strings

names.Add("Peter");

names.Add("Maria");

names.Add("George");

foreach (var name in names)

  Console.WriteLine(name);

names.Remove("Maria");

Console.WriteLine(string.Join(", ", names));
```

List<T> – Overview (2)

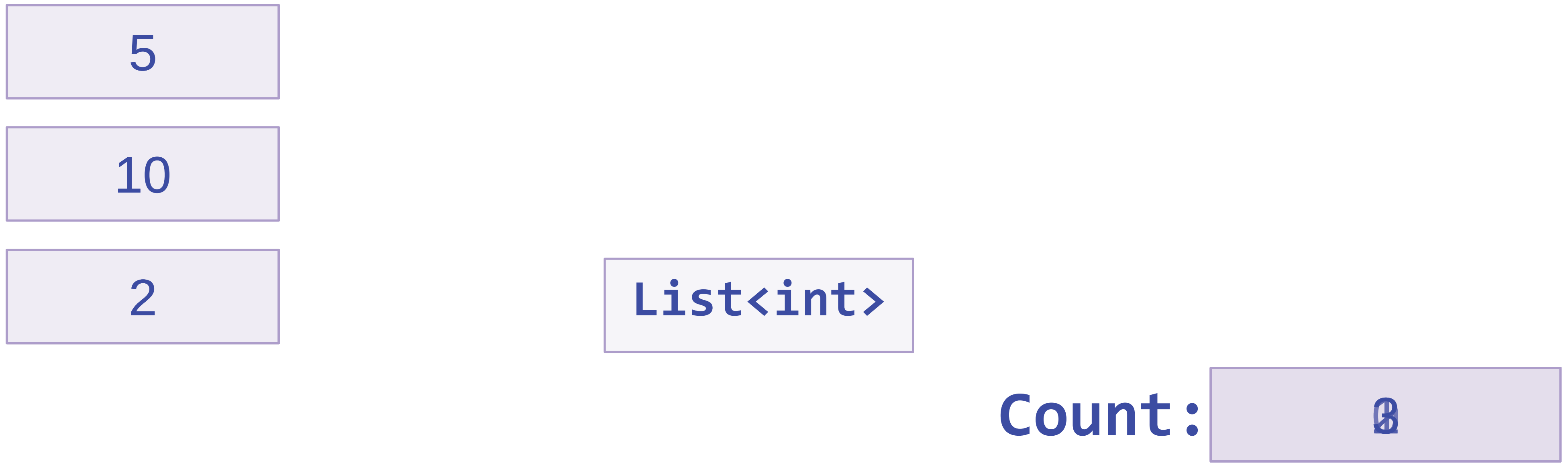
```
List<int> nums = new List<int>
                      { 10, 20, 30, 40, 50, 60 };
nums.RemoveAt(2);
nums.Add(100);
nums.Insert(0, -100);
Console.WriteLine(string.Join(", ", nums));
```

```
-100, 10, 20, 40, 50, 60, 100
```

List<T> – Data Structure

- List<T> holds a list of elements (like array, but extendable)
- Provides operations to add / insert / remove / find elements:
  - – Add(element) – adds an element to the List<T>
  - – Count – number of elements in the List<T>
  - – Remove(element) – removes an element (returns true / false)
  - – RemoveAt(index) – removes element at index
  - – Insert(index, element) – inserts an element to given position
  - – Contains(element) – determines whether an element is in the list
  - – Sort() – sorts the array/list in ascending order

Add() – Appends an Element



Add() – Appends an Element

10

20

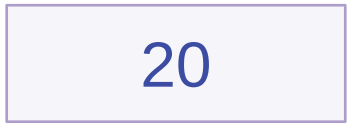
30

- We create an empty List and start adding elements.
- The Count increases each time we add an element.

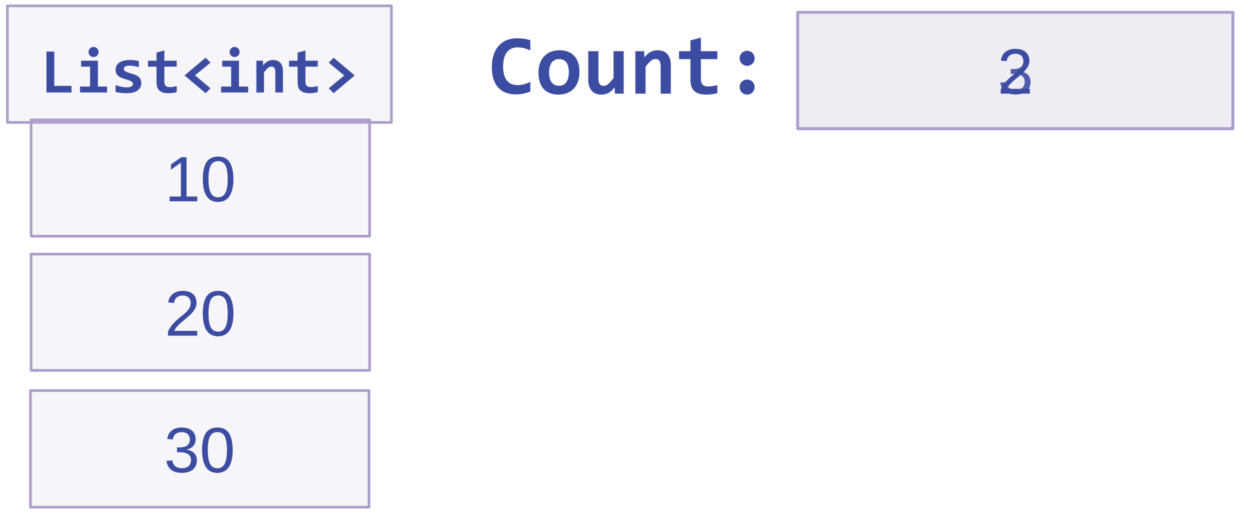
List<int>

Count: 3

Remove() – Deletes an Element



- We remove an element from the List.
- The Count decreases each time we remove an element.



Insert() – Inserts an Element at Position

- We Insert an element at index 1.
- Other Elements indices are changed upon insertion.

-10

```
List<int>
```

10

30

Count: 3

READING LISTS FROM THE CONSOLE

Using for Loop or String.Split()

Reading Lists from the Console

- First, read from the console the list length:

```
int n = int.Parse(Console.ReadLine());
```

- Next, create a list of given size n and read its elements:

```
List<int> list = new List<int>();
for (int i = 0; i < n; i++)
{
  int number = int.Parse(Console.ReadLine());
  list.Add(number));
}
```

Reading List Values from a Single Line

- Lists can be read from a single line of space separated values:

```
2 8 30 25 40 72 -2 44 56
```

```
string values = Console.ReadLine();
List<string> items = values.Split(' ').ToList();
List<int> nums = new List<int>();
for (int i = 0; i < items.Count; i++)
  nums.Add(int.Parse(items[i]));
```

Convert a collection into List

```
List<int> items = Console.ReadLine()
  .Split(' ').Select(int.Parse).ToList();
```

Printing Lists on the Console

- Printing a list using a for-loop:

```
List<string> list = new List<string>() {
  "one", "two", "three", "four", "five", "six"};
for (int index = 0; index < list.Count; index++)
  Console.WriteLine("arr[{0}] = {1}", index, list[index]);
```

- Printing a list using a string.Join(…):

```
List<string> list = new List<string>() {
  "one", "two", "three", "four", "five", "six"};
Console.WriteLine(string.Join("; ", list));
```

Problem: Sum Adjacent Equal Numbers

- Write a program to sum all adjacent equal numbers in a list of decimal numbers, starting from left to right.
- Examples:

3 3 6 1 $\rightarrow$ 12 1

8 2 2 4 8 16 $\rightarrow$ 16 8 16

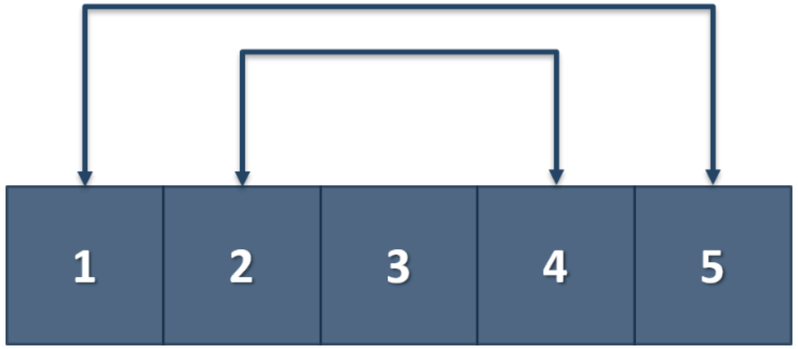
5 4 2 1 1 4 $\rightarrow$ 5 8 4

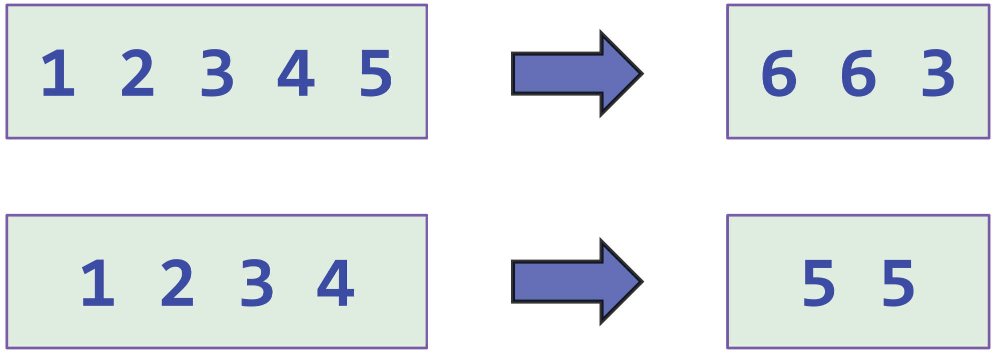
Solution: Sum Adjacent Equal Numbers

```
List<double> numbers = Console.ReadLine()
         .Split().Select(double.Parse).ToList();
for (int i = 0; i < numbers.Count - 1; i++)
  if (numbers[i] == numbers[i + 1])
  {
     numbers[i] += numbers[i + 1];
     numbers.RemoveAt(i + 1);
     i = -1;
  }
Console.WriteLine(string.Join(" ", numbers));
```

Problem: Gauss' Trick

- Write a program that sum all numbers in a list in the following order:
  - first + last, first + 1 + last - 1, first + 2 + last - 2, … first + n, last – n
- Examples:



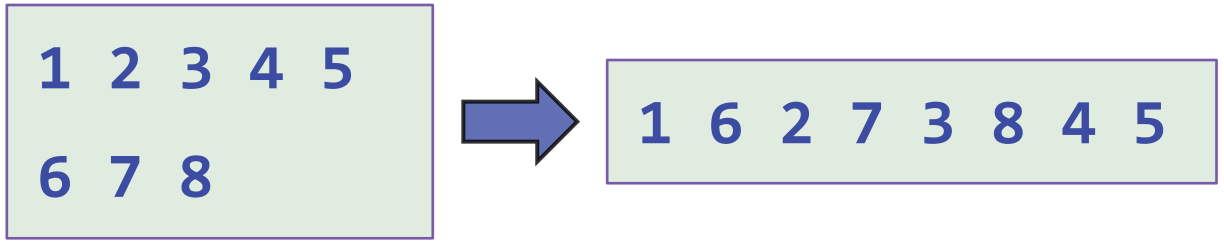


Solution: Gauss' Trick

```
List<int> numbers = Console.ReadLine()
            .Split().Select(int.Parse).ToList();
int originalLength = numbers.Count;
for (int i = 0; i < originalLength / 2; i++)
{
  numbers[i] += numbers[numbers.Count - 1];
  numbers.RemoveAt(numbers.Count - 1);
}
Console.WriteLine(string.Join(" ", numbers));
```

Problem: Merging Lists

- You receive two lists with numbers. Print a result list which contains the numbers from both of the lists.
  - If the length of the two lists are not equal, just add the remaining elements at the end of the list:
  - list1[0], list2[0], list1[1], list2[1], …



```
1 2 3 4 5
6 7 8
```

```
1 6 2 7 3 8 4 5
```

Solution: Merging Lists (1)

```
//TODO: Read the input
List<int> resultNums = new List<int>();
for (int i = 0; i < Math.Min(nums1.Count, nums2.Count); i++)
  //TODO: Add numbers in resultNums
if (nums1.Count > nums2.Count)
  resultNums.AddRange(GetRemainingElements(nums1, nums2));
else if (nums2.Count > nums1.Count)
  resultNums.AddRange(GetRemainingElements(nums2, nums1));
Console.WriteLine(string.Join(" ", resultNums));
```

Solution: Merging Lists (2)

```
static List<int> GetRemainingElements(List<int> longerList,
List<int> shorterList)
{
   List<int> nums = new List<int>();
   for (int i = shorterList.Count; i < longerList.Count; i++)
      nums.Add(longerList[i]);
   return nums;
}
```

SORTING LISTS AND ARRAYS

Sorting Lists

- Sorting a list == reorder its elements incrementally: Sort()
  - List items should be comparable, e.g. numbers, strings, dates, …

```
List<string> names = new List<string>()
 {"Peter", "Michael", "George", "Victor", "John" };
names.Sort();

Console.WriteLine(string.Join(", ", names));
// George, John, Michael, Peter, Victor
names.Sort();
names.Reverse();
Console.WriteLine(string.Join(", ", names));
// Victor, Peter, Michael, John, George
```

Sort in natural (ascending) order

Reverse the sorted result

Problem: List of Products

- Read a number n and n lines of products. Print a numbered list of all the products ordered by name.
- Examples:

```
4
Potatoes
Tomatoes
Onions
Apples
```

```
1.Apples
2.Onions
3.Potatoes
4.Tomatoes
```

Solution: List of Products

```
int n = int.Parse(Console.ReadLine());
List<string> products = new List<string>();
for (int i = 0; i < n; i++)
{
  string currentProduct = Console.ReadLine();
  products.Add(currentProduct);
}
products.Sort();
for (int i = 0; i < products.Count; i++)
  Console.WriteLine($"{i + 1}.{products[i]}");
```

Problem: Remove Negatives and Reverse

- Read a list of integers, remove all negative numbers from it.
  - Print the remaining elements in reversed order.
  - In case of no elements left in the list, print "empty".

`10 -5 7 9 -33 50`	→	`50 9 7 10`
`7 -2 -10 1`	→	`1 7`
`-1 -2 -3`	→	`empty`

Solution: Remove Negatives and Reverse

```
List<int> nums = // TODO: Read the List from the console.
for (int i = 0; i < nums.Count; i++)
  if (nums[i] < 0) { nums.RemoveAt(i--); }

nums.Reverse();
if (nums.Count == 0)
 Console.WriteLine("empty");
else
 Console.WriteLine(string.Join(" ", nums));
```

Summary

- Lists hold a sequence of elements (variable-length)
- Can add / remove / insert elements at runtime
- Creating (allocating) a list: new List<T>()
- Accessing list elements by index
- Printing list elements: string.Join(…)