7SENG011W Object Oriented Programming

Arrays, More Loops

Dr Francesco Tusa

Readings

The topics we will discuss today can be found in the book

- Programming C# 10.0
  - Chapter Collections: Arrays

C# online documentation

- Arrays
- Single-Dimensional Arrays

Outline

- Summary of previous lecture
- Arrays
- More on Loops

Outline

- Summary of previous lecture
- Arrays
- More on Loops

Factorial Calculation

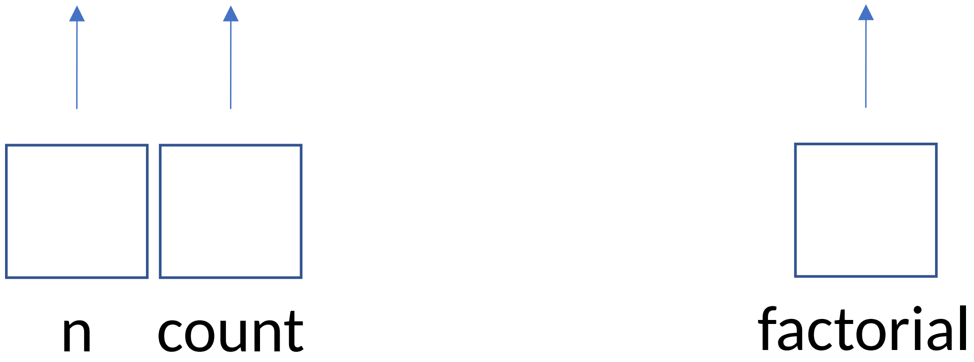
n! = n x (n-1) x (n-2) x (n-3) x … 3 x 2 x 1

For example: 5! = 5 x 4 x 3 x 2 x 1 = 120

Factorial Calculation

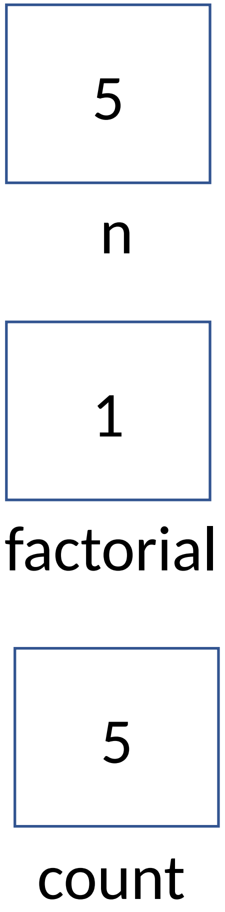
$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots 3 \times 2 \times 1$

For example: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$



While: Factorial Calculation

```
static void Main(string[] args)
{
int n = … // read from keyboard
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```


5

n

1

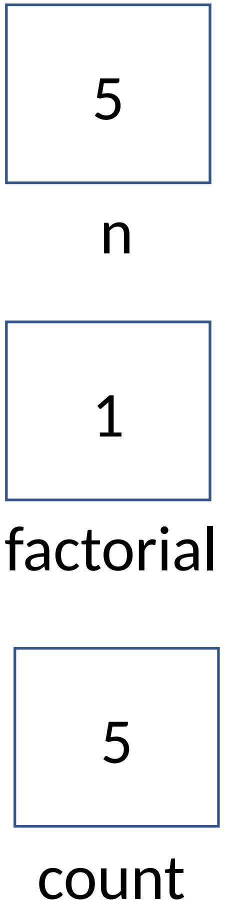
factorial

5

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```


5

n

1

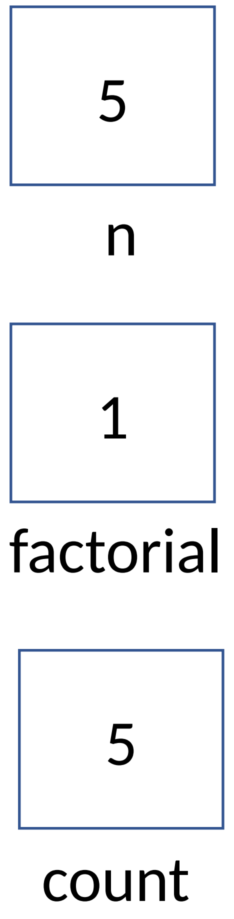
factorial

5

count

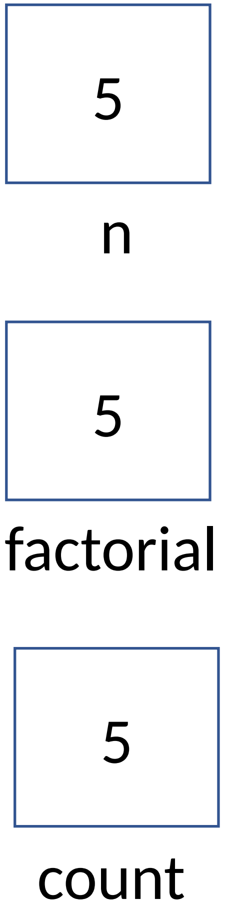
While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

5

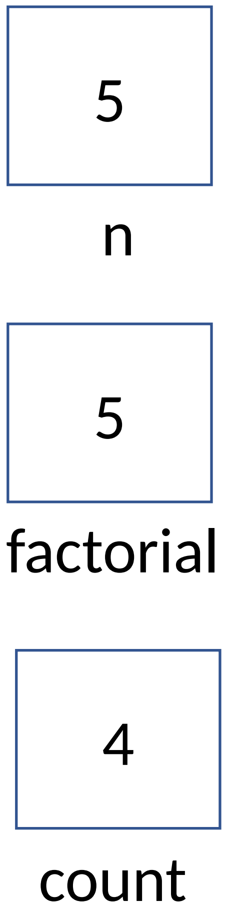
factorial

5

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

5

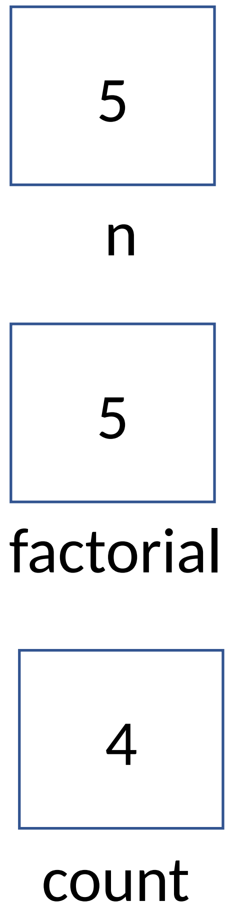
factorial

4

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

5

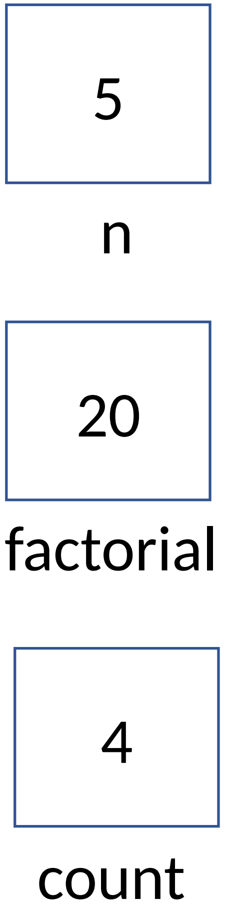
factorial

4

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

20

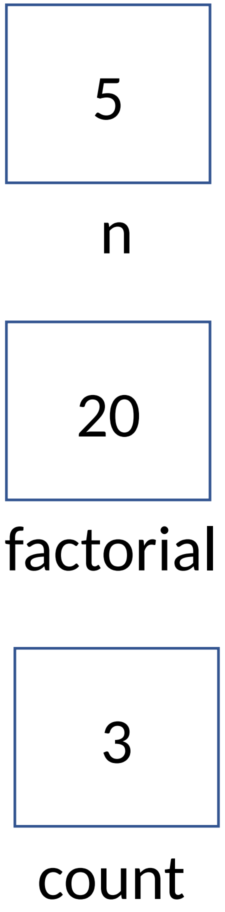
factorial

4

count

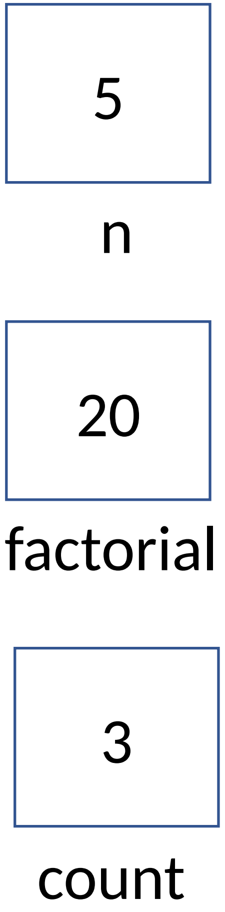
While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

20

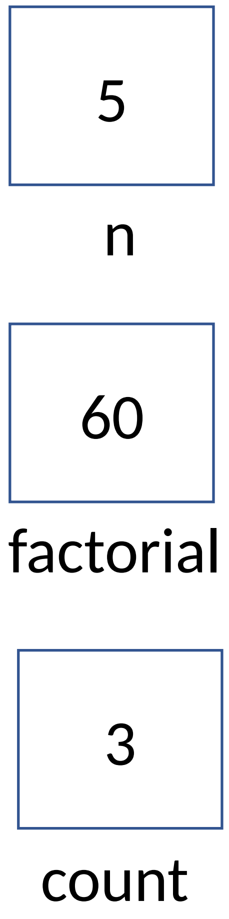
factorial

3

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

60

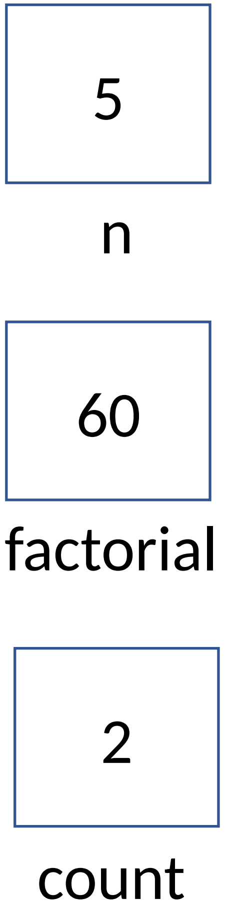
factorial

3

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

60

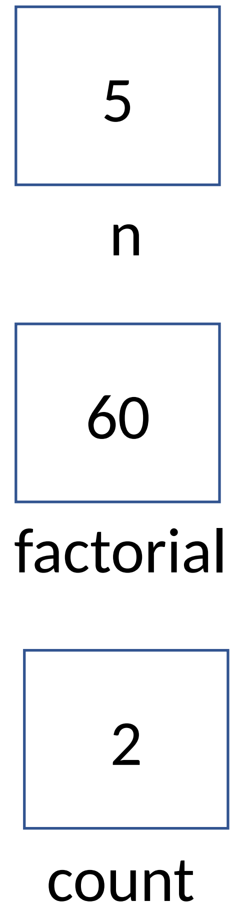
factorial

2

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

60

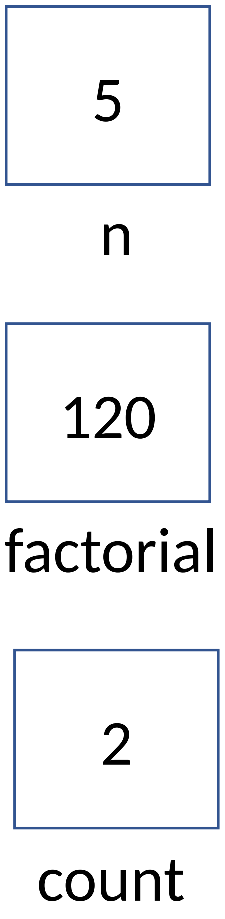
factorial

2

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

120

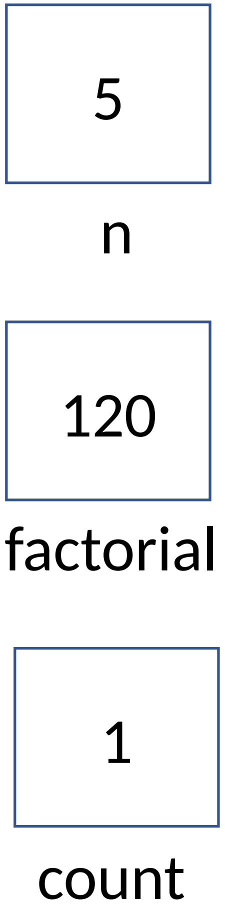
factorial

2

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)          //  false
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

120

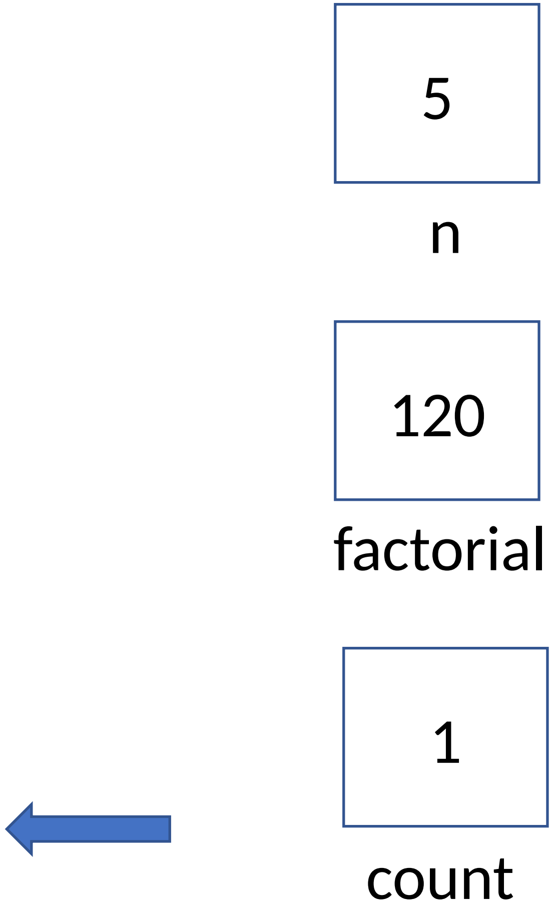
factorial

1

count

While: Factorial Calculation

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

while (count > 1)
{
        factorial = factorial * count;
        count --;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```



5

n

120

factorial

1

count

Using For Loop

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

for (int count = n; count > 1; count--)
{
        factorial *= count;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```

Using For Loop

```
static void Main(string[] args)
{
int n = // 5
int factorial = 1;
int count = n;

for (int count = n; count > 1; count--)
{
        factorial *= count;
}

Console.WriteLine($"The factorial of {n} is {factorial}");
```

for use case: number of iterations is known

Switch statement: example

```
static void Main(string[] args)
    {
        int temperature = Convert.ToInt32(Console.ReadLine());

        switch (temperature)
        {
            case > 30:
                Console.WriteLine("Critical");
                break;

            case > 24:
                Console.WriteLine("Warning");
                break;

            default:
                Console.WriteLine("Normal");
                break;
        }
    }
```

Relational pattern matching; before C# 9 only constant pattern matching

Switch statement: example

```
static void Main(string[] args)
    {
      int temperature = Convert.ToInt32(Console.ReadLine());

      switch (temperature)
      {
        case > 30:
          Console.WriteLine("Critical");
          break;

        case > 24:
          Console.WriteLine("Warning");
          break;

        default:
          Console.WriteLine("Normal");
          break;
      }
    }
```

1

2

Matching is performed in text order

Switch statement: example

```
static void Main(string[] args)
     {
        int temperature = Convert.ToInt32(Console.ReadLine());

        switch (temperature)
        {
           case > 30:
              Console.WriteLine("Critical");
              break;

           case > 24:
              Console.WriteLine("Warning");
              break;

           default:
              Console.WriteLine("Normal");
              break;
        }
     }
```

statements to execute when a match expression does **NOT** match any other case pattern

Switch statement: example

```
static void Main(string[] args)
    {
        int temperature = Convert.ToInt32(Console.ReadLine());

        switch (temperature)
        {
            case > 30:
                Console.WriteLine("Critical");
                break;

            case > 24:
                Console.WriteLine("Warning");
                break;

            default:
                Console.WriteLine("Normal");
                break;
        }
    }
```

Within a switch statement, control cannot fall through from one switch section to the next

Switch statement: tutorial

```
static void Main(string[] args)
{
    int temperature = Convert.ToInt32(Console.ReadLine());
    int invalid = 0, critical = 0, warning = 0, normal = 0;
    while (invalid < 3)
    {
        switch (temperature)
        {
          case > 100:
          case < 0:
            invalid++;
                break;

          case > 30:
            critical++;
            break;

          case > 24:
                warning++;
            break;

          default:
                normal++;
            break;
        }
    }
    // print output on the console
}
```

Switch statement: tutorial

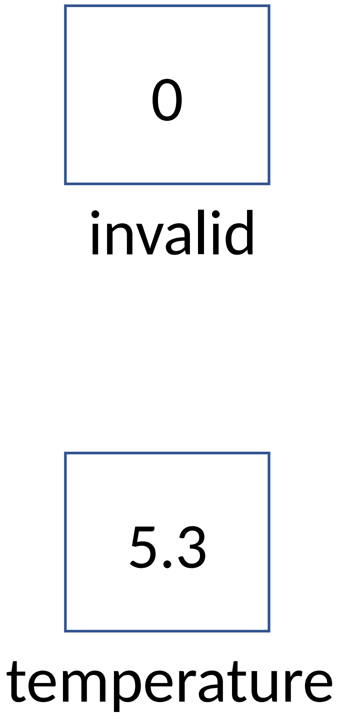
- The previous program uses **four** int variables to keep track of the different types of temperature measurements
- *invalid*, *critical*, *warning*, *normal*
- *temperature* variable contains one single measurement
- What if I asked you to keep track of **all** the measurements?

Outline

- Summary of previous lecture
- Arrays
- More on Loops

Variables

```
…
int invalid;
double temperature;
…
invalid = 0;
temperature = 5.3
```



0

invalid

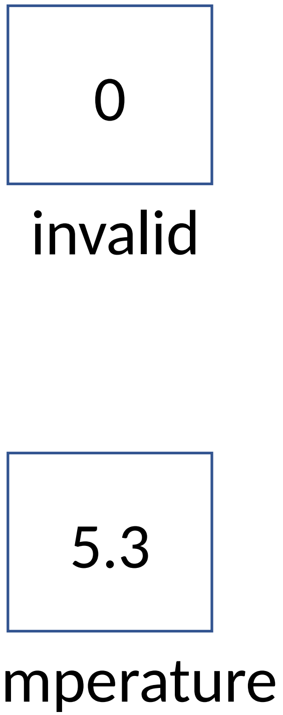
5.3

temperature

one integer or double value at the time can be stored inside the *invalid* and the *temperature* variable

Variables

```
…
int invalid;
double temperature;
…
invalid = 0;
temperature = 5.3
```



0

invalid

5.3

temperature

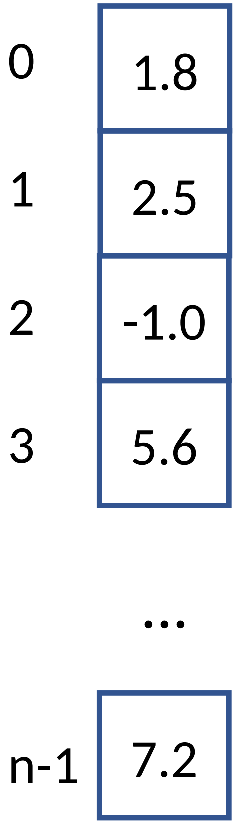
We would need *more than one* variable to store all the temperature values generated inside the *loop* – how many?

Arrays

- An array is an *indexed* list of values
- You can make an array of any *type* int, double, string, etc.
- All elements of an array must have the *same* type

Arrays

double []



0 1.8

1 2.5

2 -1.0

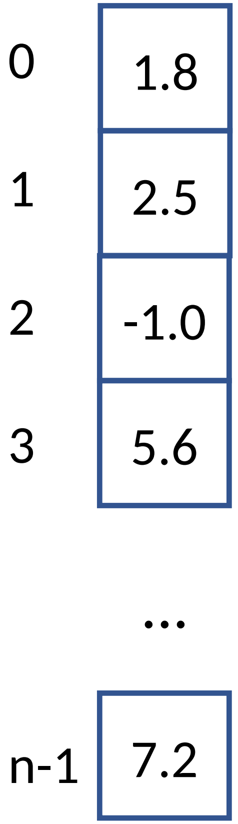
3 5.6

...

n-1 7.2

Arrays

double [] values

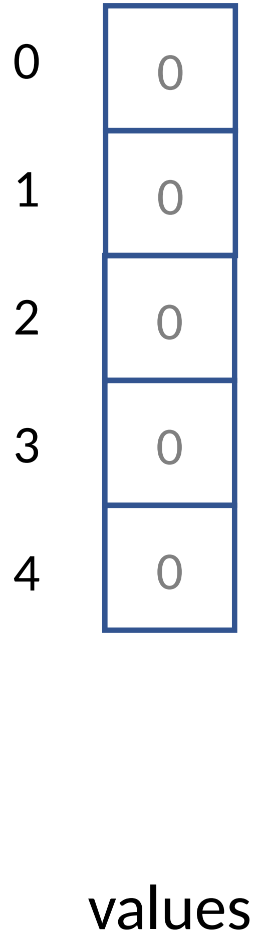


0	1.8
1	2.5
2	-1.0
3	5.6
	…
n-1	7.2

Arrays

```
double [] values = new double[5];
```

Fixed number of elements

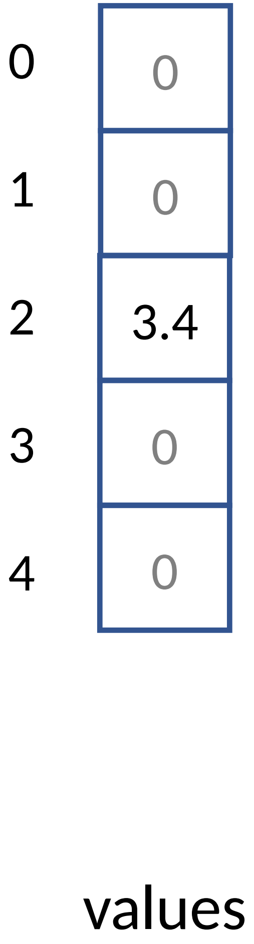


0 0
1 0
2 0
3 0
4 0

values

Arrays

```
double [] values = new double[5];

values[2] = 3.4;
```

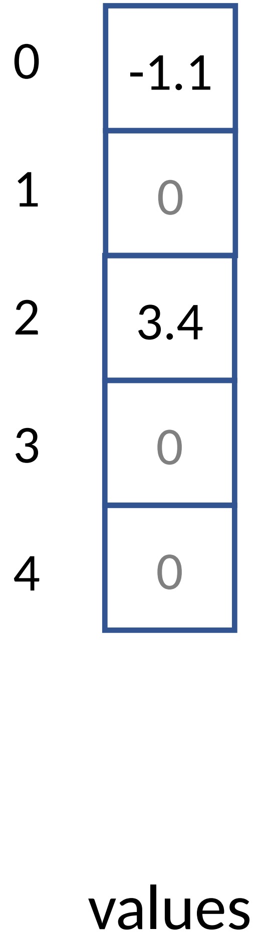


0	0
1	0
2	3.4
3	0
4	0

values

Arrays

```
double [] values = new double[5];

values[2] = 3.4;
values[0] = -1.1;
```

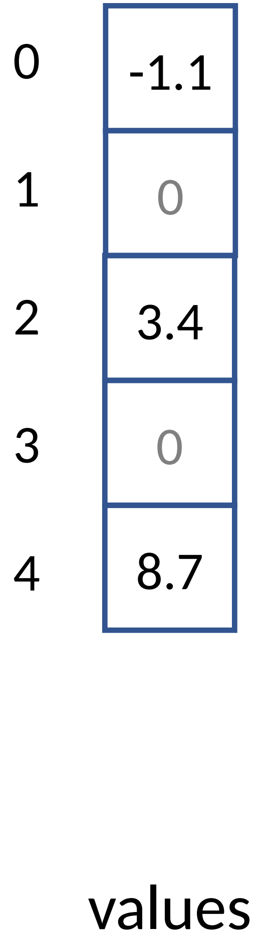


0	-1.1
1	0
2	3.4
3	0
4	0

values

Arrays

```
double [] values = new double[5];

values[2] = 3.4;
values[0] = -1.1;
values[4] = 8.7;
```



0	-1.1
1	0
2	3.4
3	0
4	8.7

values

Arrays

```
double [] values = new double[5];

values[2] = 3.4;
values[0] = -1.1;
values[4] = 8.7;
```

The index starts at 0

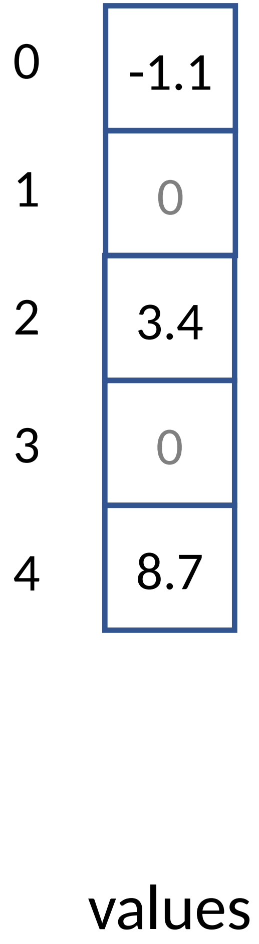
ends at n-1

0	-1.1
1	0
2	3.4
3	0
4	8.7

values

Arrays

```
double [] values = new double[5];

values[2] = 3.4;
values[0] = -1.1;
values[4] = 8.7;

values[5] = 2.3; // ???
```



0	-1.1
1	0
2	3.4
3	0
4	8.7

values

Arrays: size

```
int size = Convert.ToInt32(Console.ReadLine());
double [] values = new double[size]; // size instead of constant 5

values[2] = 3.4;
values[0] = -1.1;
values[4] = 8.7;

values[5] = 2.3; // ???
```

Arrays: let’s try it

- Let’s try the previous code in Visual Studio
- The code can be **compiled** without errors – *dotnet build*
- An error is generated when the program is **executed** – *dotnet run*

Unhandled exception. ***System.IndexOutOfRangeException****: Index was outside the bounds of the array.*

Arrays: let’s try it

If we did this in *C program (native code)*

- **Undefined behaviour:** the program might continue to run with **unpredictable** results
  - It can *crash*
  - It can *produce incorrect results*
  - It can *work fine until some point* in the future when the consequences of the error become apparent

Arrays: let’s try it

In a *C# program (managed code)*

- The program runs in the **.NET managed environment**
  - Runtime *checking* for *array bounds*
  - Accessing an out-of-bounds element *generates* a runtime *IndexOutOfRangeException*
  - *Exceptions* can be *caught* and *handled* preventing the program from crashing

Arrays: initialisation

```
double [] values = {0.1, 3.76, -1.23, 0.45}
```

- Curly braces can be used to initialize an array
- This can ONLY be done when you declare the variable

```
double [] values = new double[4];

values[0] = 0.1;
values[1] = 3.76;
values[2] = -1.23;
values[3] = 0.45;
```

Arrays: accessing elements

- Elements of an array can be accessed via the [] operator

```
int[] values = { 1, 7, 3, 5, 4 };
values[3] = 18; // { 1, 7, 3, 18, 4 }
int x = values[1] + 3; // { 1, 7, 3, 18, 4 }, x = 10
```

Arrays: *Length* variable

- Arrays have a *Length* variable built-in that contains the length of the array

```
int[] values = new int[12];
int size = values.Length; // 12

int[] values2 = { 1, 2, 3, 4, 5}
int size2 = values2.Length; // 5
```

Initial Problem

- The previous program uses **four** int variables to keep track of the different types of temperature measurements
- *invalid*, *critical*, *warning*, *normal*
- *temperature* variable contains one single measurement
- What if I asked you to keep track of **all** the measurements?

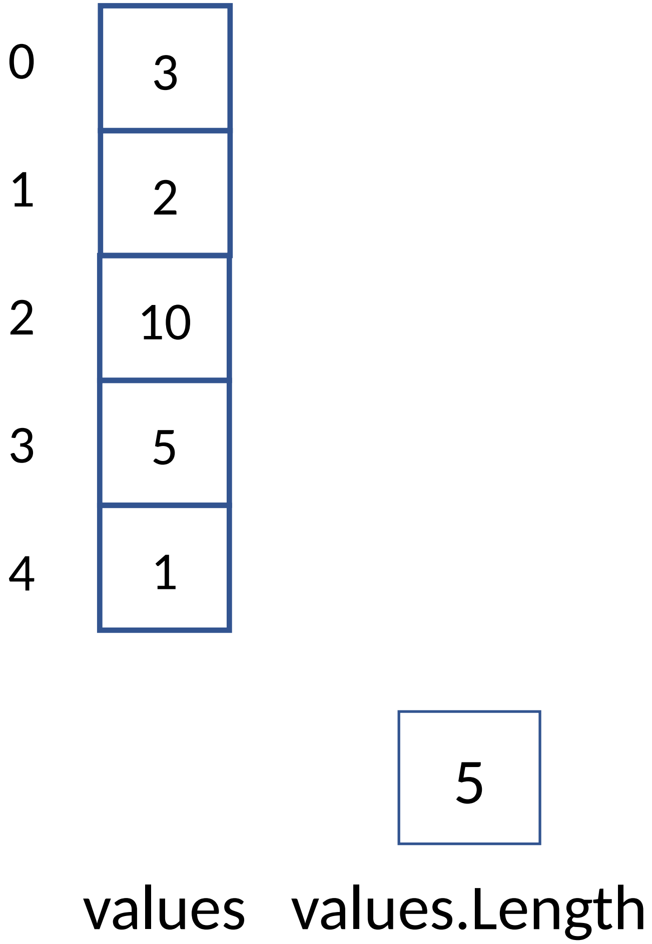
- **Has the array solved our problem?**

Outline

- Summary of previous lecture
- Arrays
- More on Loops

Arrays: for loop

```
int[] values = { 3, 2, 10, 5, 1 };

for (int i = 0; i < values.Length; i++)
{
    Console.WriteLine(values[i]);
}
```



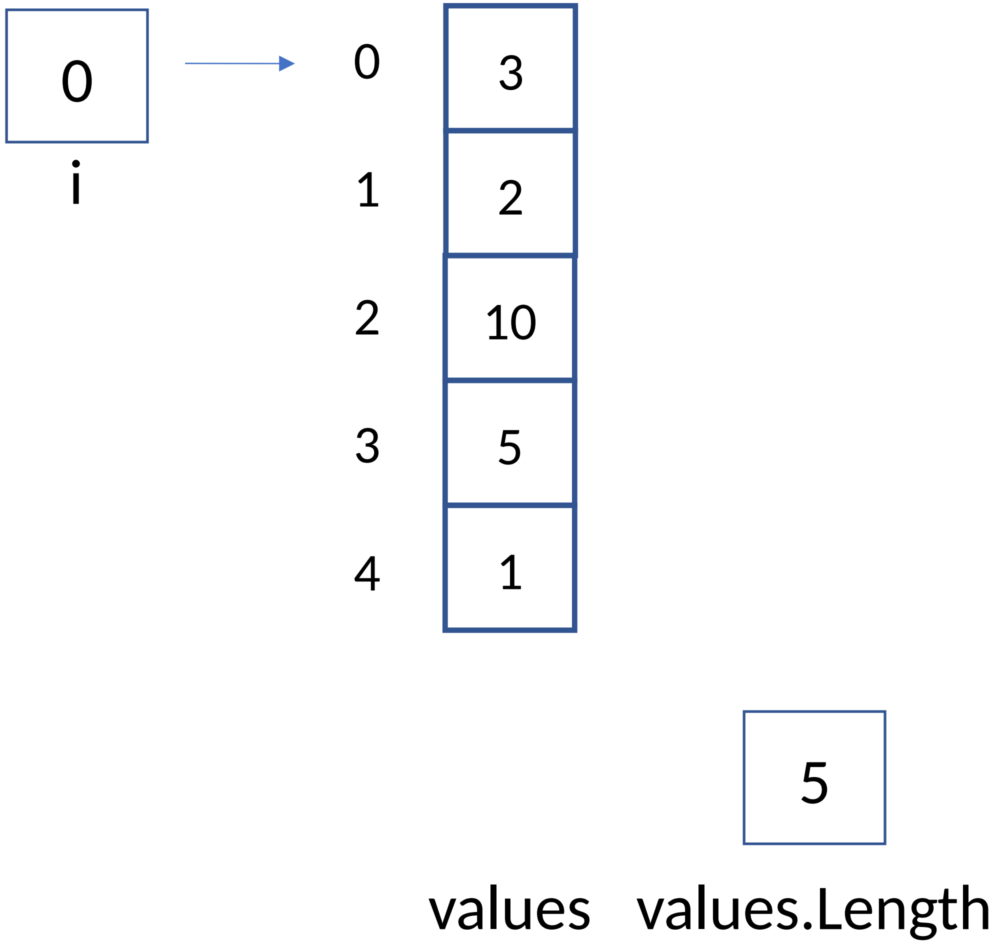
0	3
1	2
2	10
3	5
4	1

5

values values.Length

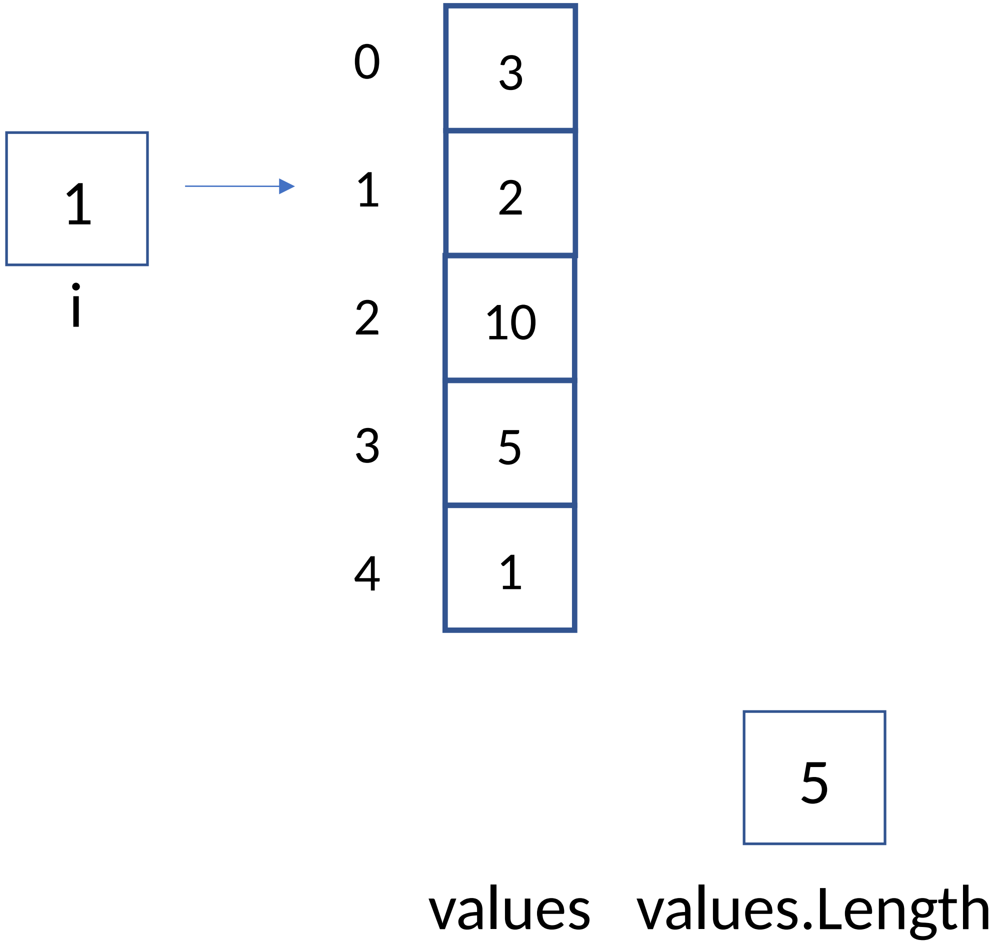
Arrays: for loop

```
int[] values = { 3, 2, 10, 5, 1 };

for (int i = 0; i < values.Length; i++)
{
    Console.WriteLine(values[i]);
}
```



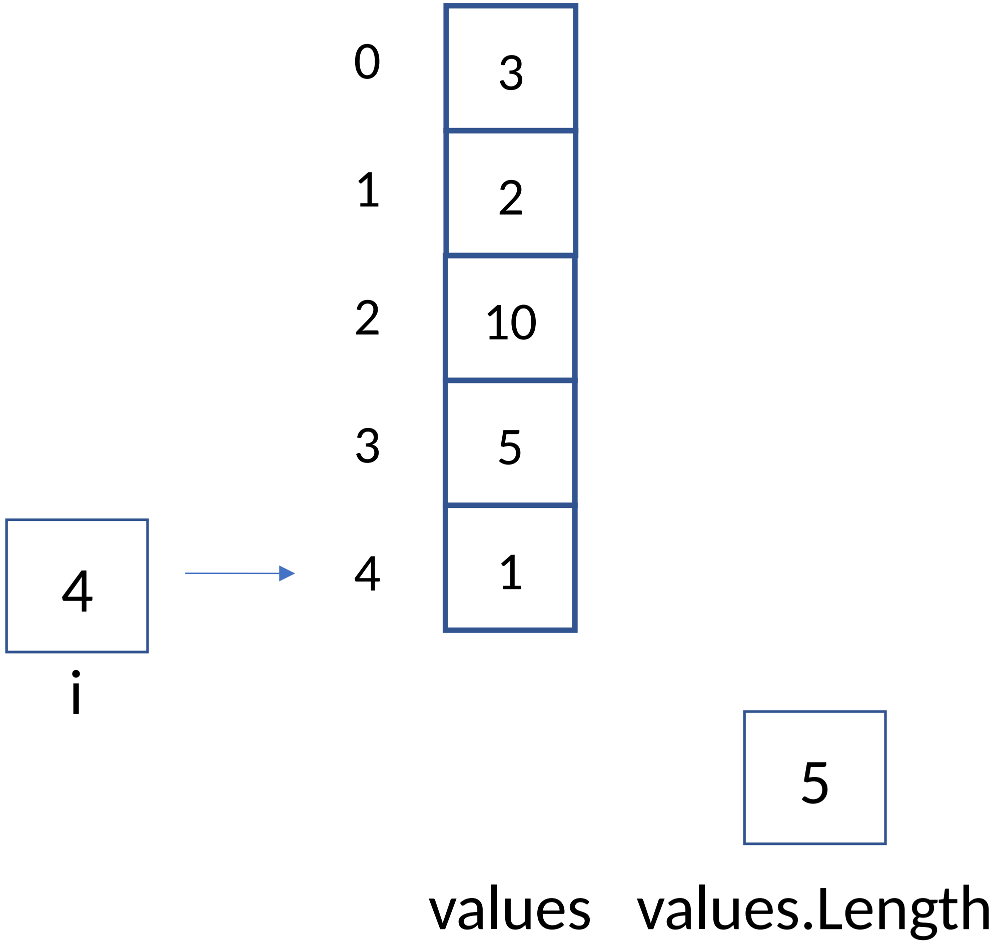
Arrays: for loop

```
int[] values = { 3, 2, 10, 5, 1 };

for (int i = 0; i < values.Length; i++)
{
    Console.WriteLine(values[i]);
}
```



Arrays: for loop

```
int[] values = { 3, 2, 10, 5, 1 };

for (int i = 0; i < values.Length; i++)
{
    Console.WriteLine(values[i]);
}
```



0 3
1 2
2 10
3 5
4 1

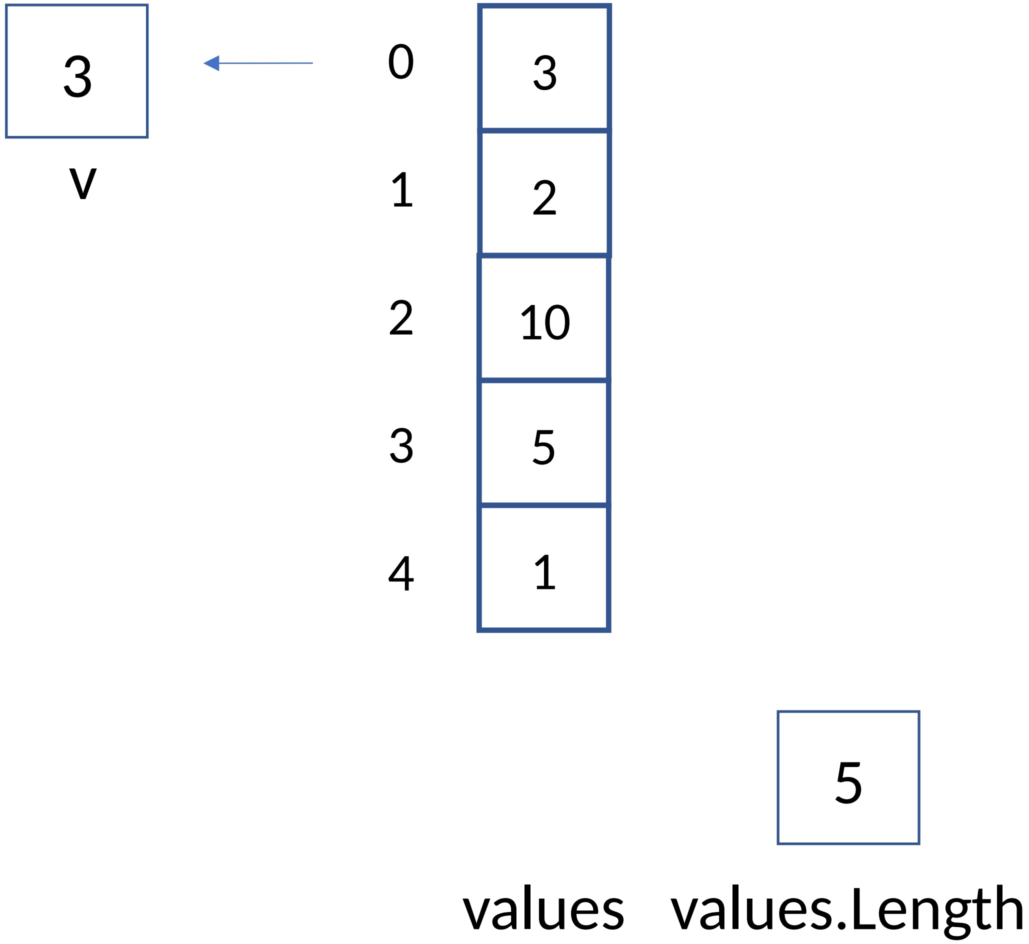
4
i

5

values values.Length

Arrays: foreach loop

```
int[] values = { 3, 2, 10, 5, 1 };

foreach (int v in values)
{
    Console.WriteLine(v);
}
```



3

v

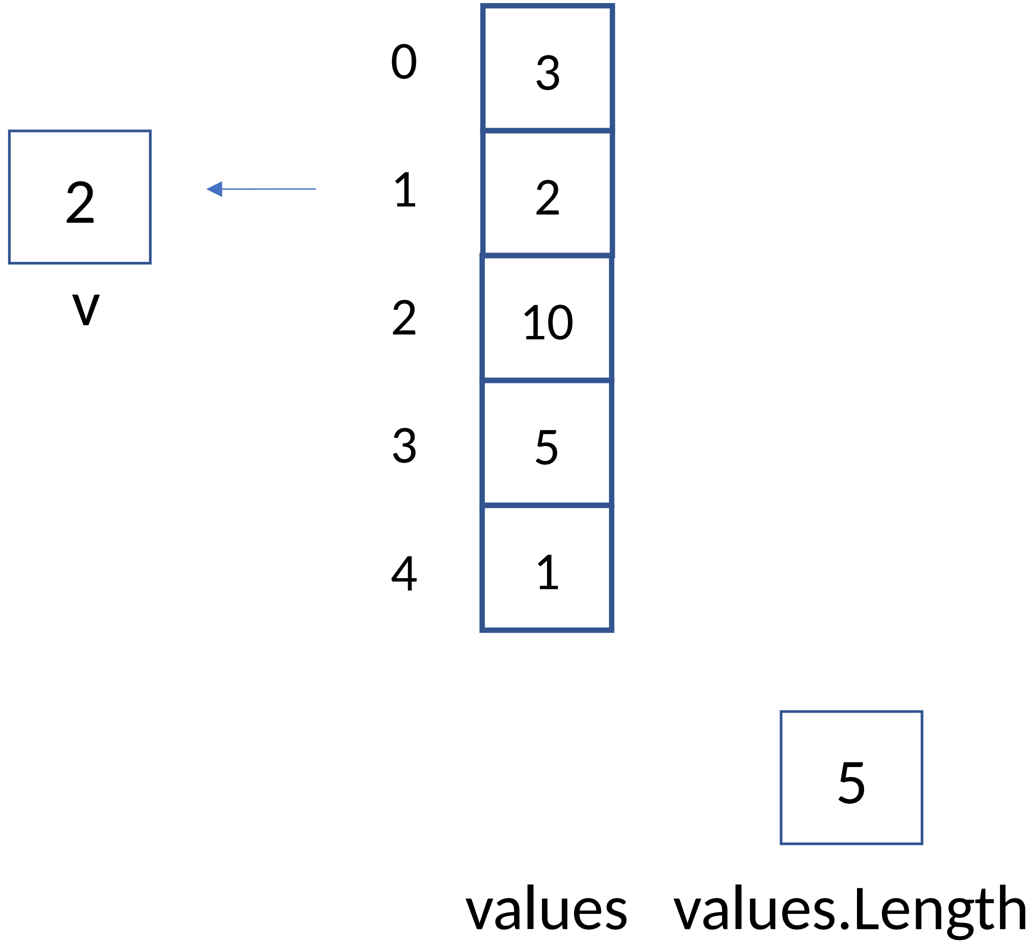
0	3
1	2
2	10
3	5
4	1

5

values values.Length

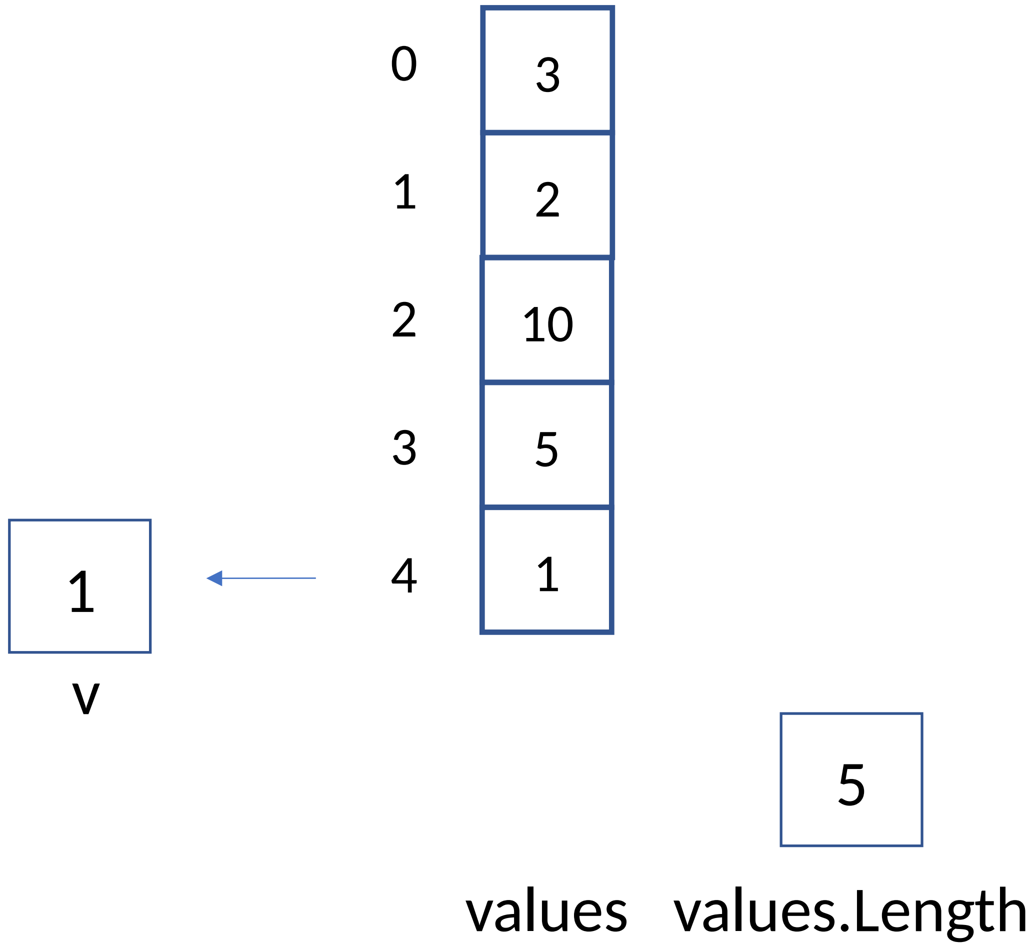
Arrays: foreach loop

```
int[] values = { 3, 2, 10, 5, 1 };

foreach (int v in values)
{
    Console.WriteLine(v);
}
```



Arrays: foreach loop

```
int[] values = { 3, 2, 10, 5, 1 };

foreach (int v in values)
{
    Console.WriteLine(v);
}
```



0 3
1 2
2 10
3 5
4 1

1
v

5

values values.Length

Program arguments

```
$ program arg0 arg1 arg2 … argn
```

Program arguments

$ program arg0 arg1 arg2 … argn

Name of the program
to be executed

Program arguments

$ program arg0 arg1 arg2 … argn

Arguments (values) provided
as input to that program

Program arguments

```
$ program arg0 arg1 arg2 … argn
```

Example:

```
$ dotnet run Args.csproj 100 30.5 120.0
```

program name

Arrays: program arguments

$ program arg0 arg1 arg2 … argn

Example:

$ dotnet run Args.csproj 100 30.5 120.0

Arguments provided as input to that program – will be strings

Arrays: program arguments

```
static void Main(string[] args)
{
    Console.WriteLine("The length of args is: " + args.Length);
    Console.WriteLine("The elements of args are: ");
    foreach (string arg in args)
    {
        Console.WriteLine(arg);
    }

}
```

an array of strings is provided as input to our program

Arrays: program arguments

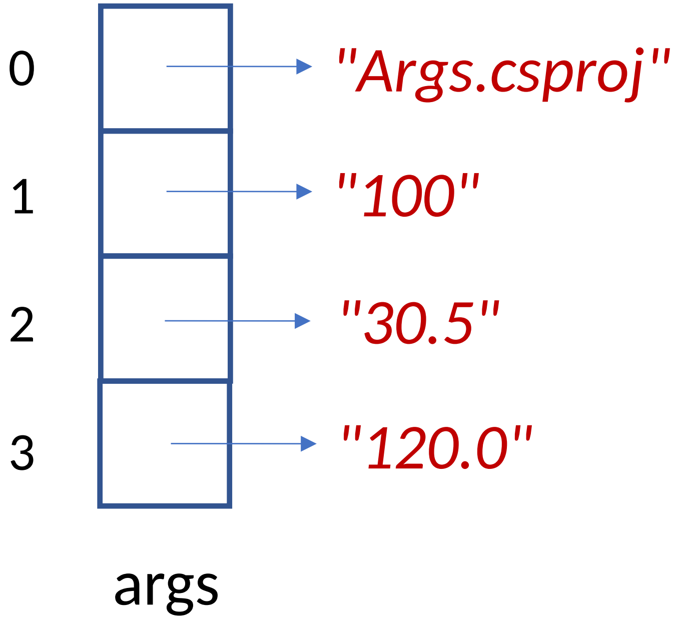
```
$ program arg0 arg1 arg2 … argn
```

Example:

```
$ dotnet.exe run Args.csproj 100 30.5 120.0
```

(behind the scenes…)

```
string[] args = {"Args.csproj", "100", "30.5", "120.0"};
```



0 "Args.csproj"

1 "100"

2 "30.5"

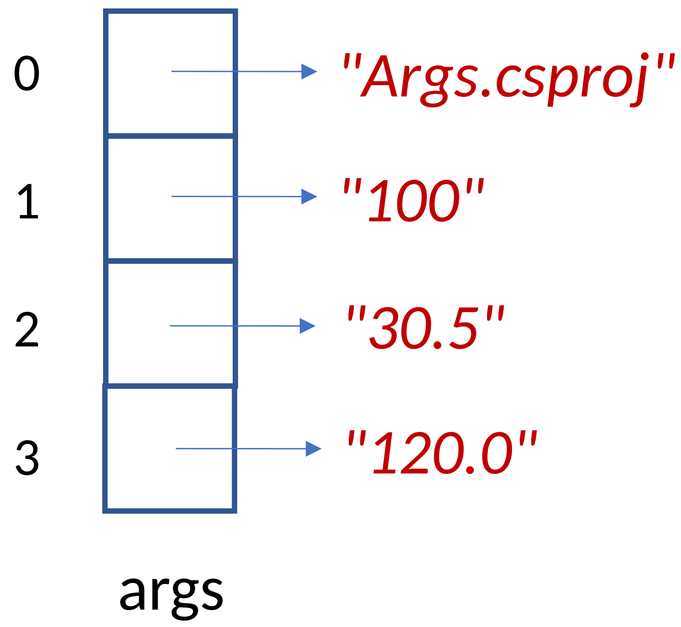
3 "120.0"

args

Arrays: program arguments

```
static void Main(string[] args)
{
        ...
        int size = Convert.ToInt32(args[1]);
        double[] values = new double[size];
        ...

}
```

an array of strings is provided as input to our program



Let’s try the code

Arrays: loops

```
int[] values = new int[5];
int i = 0;

while (i < values.Length)
{
    values[i] = i * i;
    if (values[i] > 5)
    {
      Console.Write(values[i] + " ");
    }
    i++;
}
```

Questions