562.613 APPLIED DATA STRUCTURES

Lecture 02

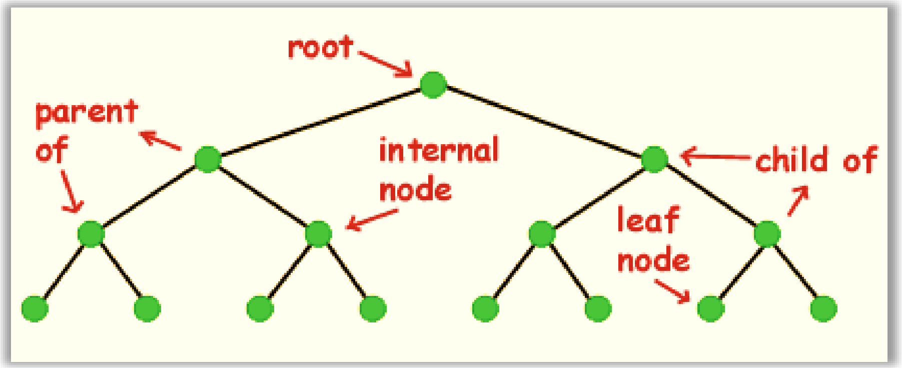
Dr. Abbas Malik

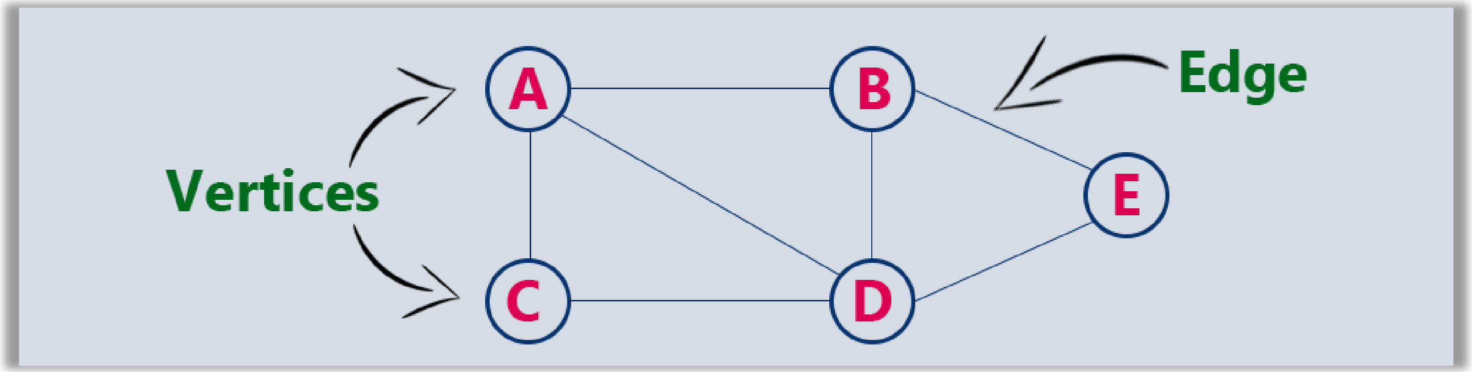
muhammad.malik@manukau.ac.nz

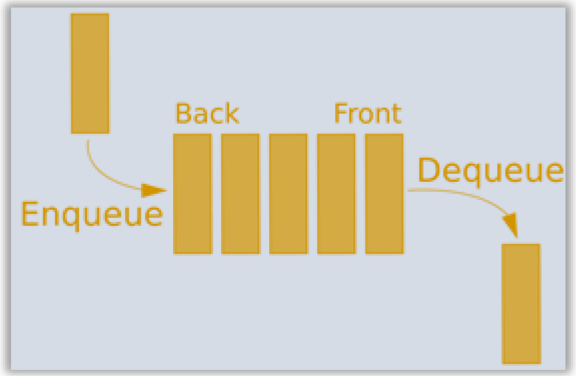
Purpose of ADS Course

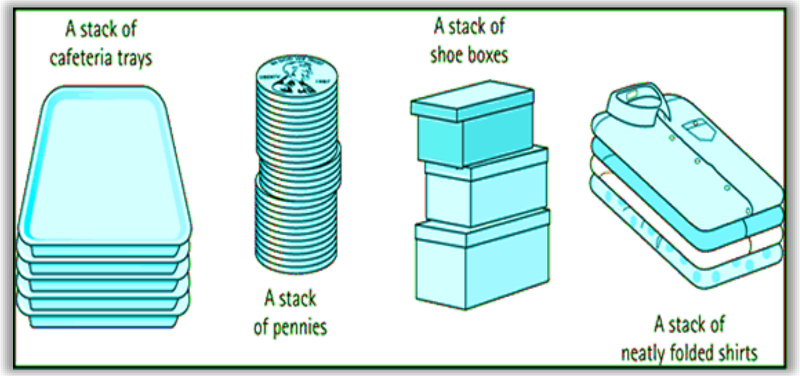
- Data Structure: a specific way to store and organize data in a computer and use it efficiently
- Algorithms: step by step instructions to solve a problem

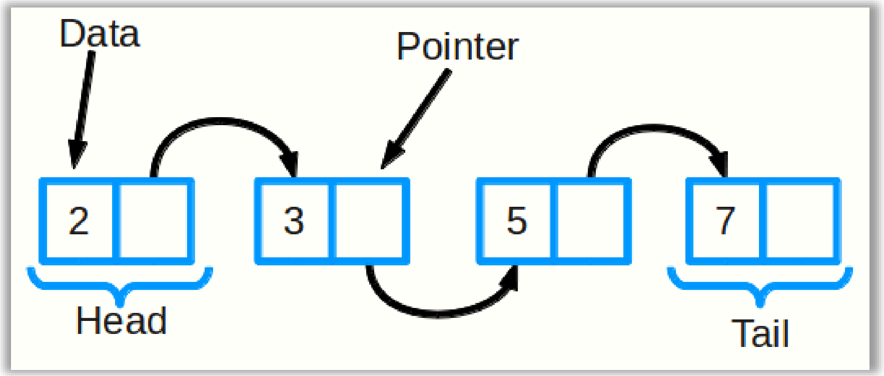
Different ADT - Data Structures











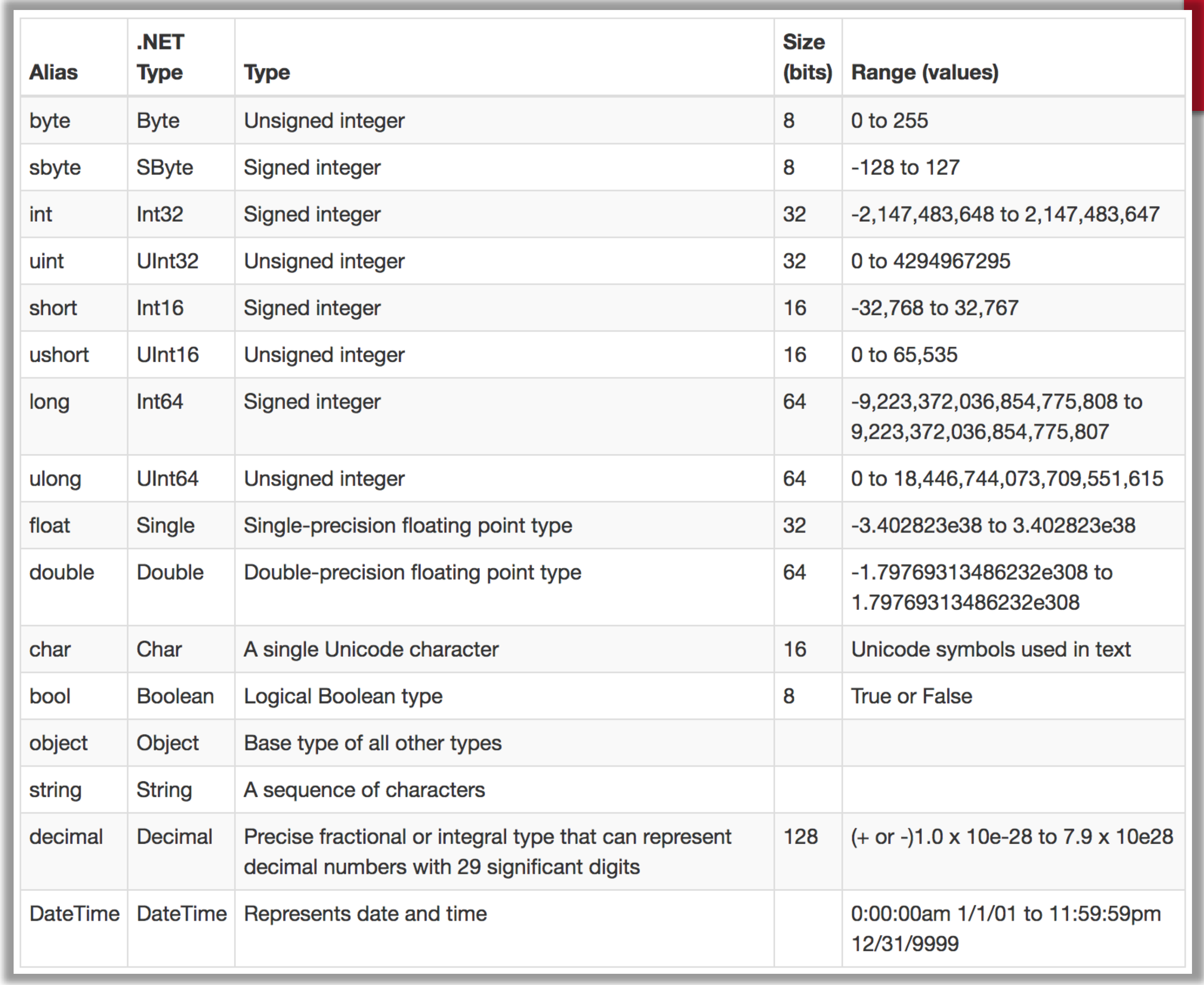
Lecture 02

- Abstract Data Types (ADT)
- Arrays
- Sorting Algorithms
- Multi-Dimensional Arrays
- Jagged Arrays

Abstract Data Type (ADT)

- **Mathematical Model** with **a collection of operations** defined on the model
- **Sets of Integers**
  - **Operations**: union, intersection, set difference, …
- **Generalization** of primitive data types
- **Encapsulation** of the data and all operations defined on ADT

Primitive Data Types

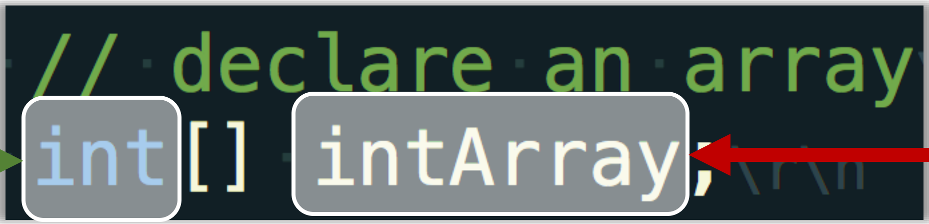


Alias	.NET Type	Type	Size (bits)	Range (values)
byte	Byte	Unsigned integer	8	0 to 255
sbyte	SByte	Signed integer	8	-128 to 127
int	Int32	Signed integer	32	-2,147,483,648 to 2,147,483,647
uint	UInt32	Unsigned integer	32	0 to 4294967295
short	Int16	Signed integer	16	-32,768 to 32,767
ushort	UInt16	Unsigned integer	16	0 to 65,535
long	Int64	Signed integer	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	UInt64	Unsigned integer	64	0 to 18,446,744,073,709,551,615
float	Single	Single-precision floating point type	32	-3.402823e38 to 3.402823e38
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
char	Char	A single Unicode character	16	Unicode symbols used in text
bool	Boolean	Logical Boolean type	8	True or False
object	Object	Base type of all other types		
string	String	A sequence of characters		
decimal	Decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	(+ or -)1.0 x 10e-28 to 7.9 x 10e28
DateTime	DateTime	Represents date and time		0:00:00am 1/1/01 to 11:59:59pm 12/31/9999

Arrays

- A collection of **fixed number** of **data values** of the **same type**
- Can define an array of any single data type

Array Data Type

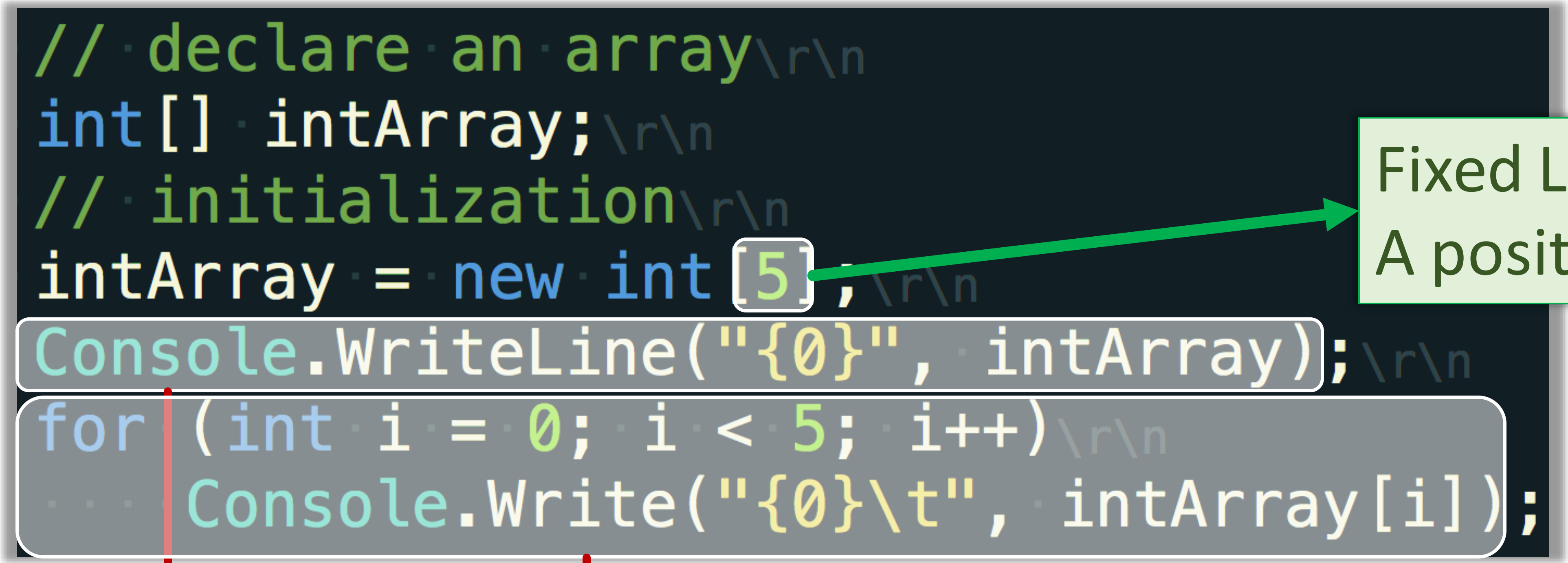


```
// declare an array
int[] intArray;
```

Array Name

- Using intArray => unassigned variable error

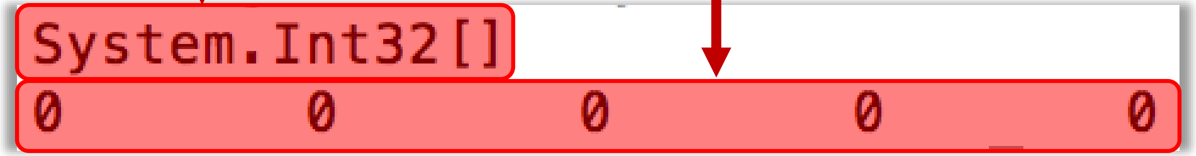
Array Initialization



```
// declare an array
int[] intArray;
// initialization
intArray = new int[5];
Console.WriteLine("{0}", intArray);
for (int i = 0; i < 5; i++)
    Console.Write("{0}\t", intArray[i]);
```

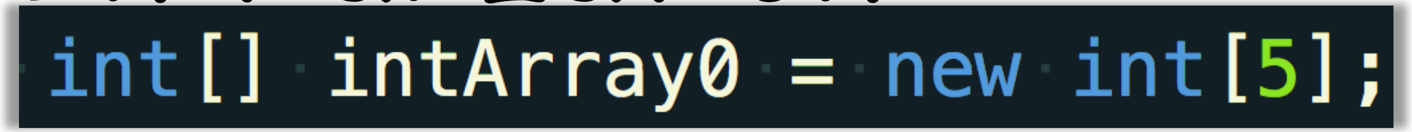
Fixed Length

A positive No

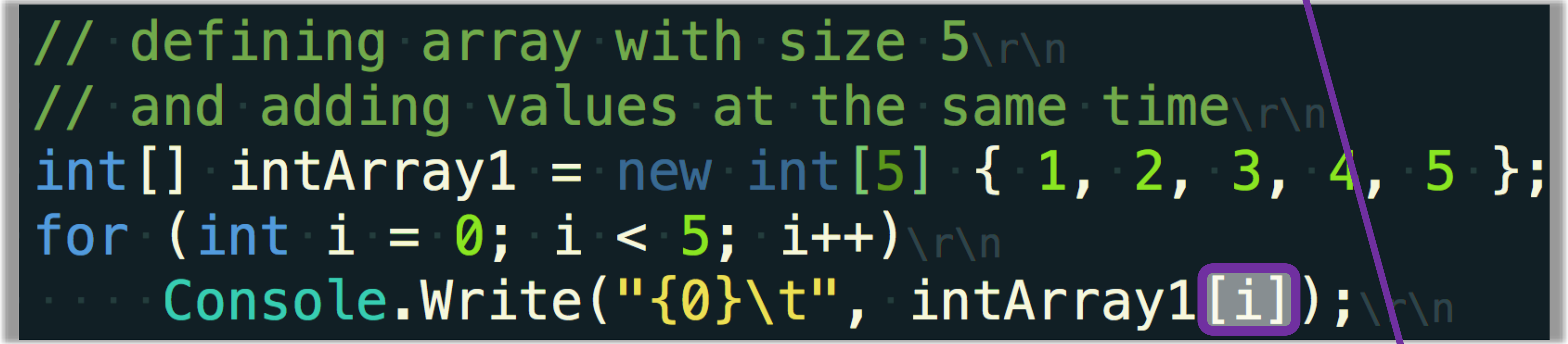


```
System.Int32[]
0       0       0       0       0
```

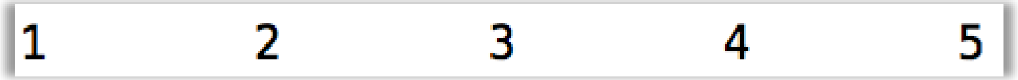
Array Declaration & Initialization



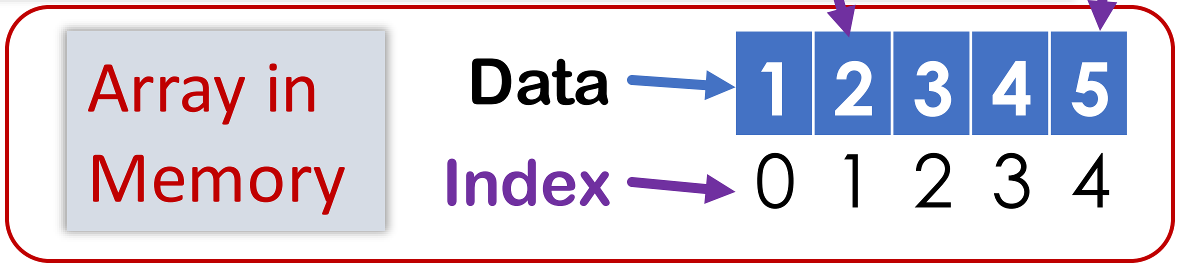
```
int[] intArray0 = new int[5];
```



```
// defining array with size 5
// and adding values at the same time
int[] intArray1 = new int[5] { 1, 2, 3, 4, 5 };
for (int i = 0; i < 5; i++)
    Console.Write("{0}\t", intArray1[i]);
```

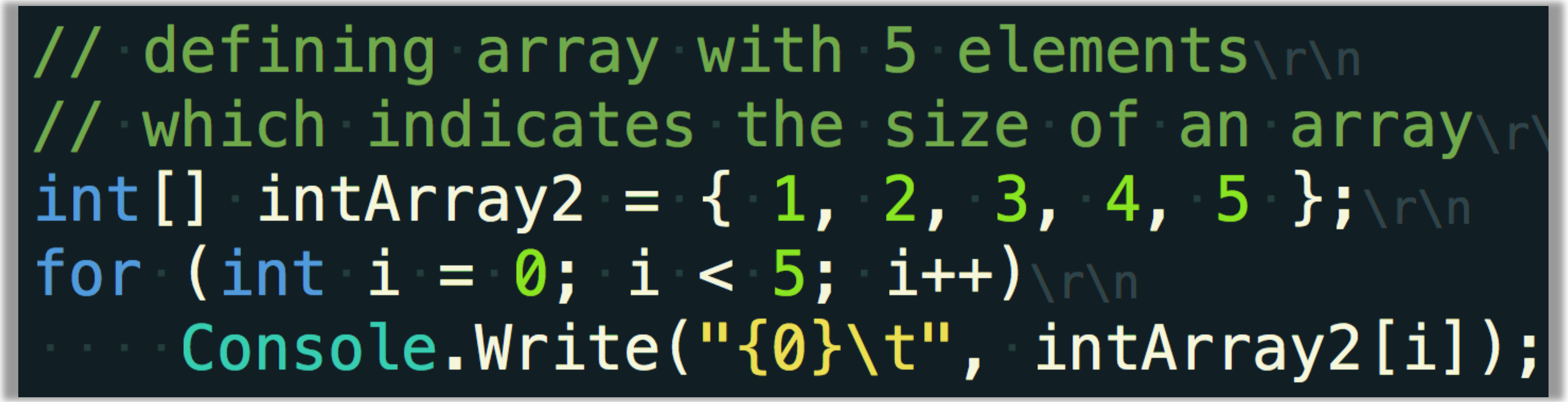


```
1       2       3       4       5
```

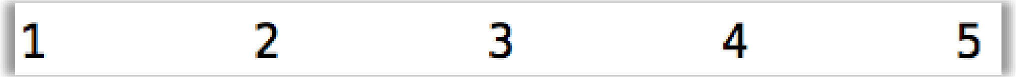


Array Declaration & Initialization

- Accessing Array Elements - Index always starts with ZERO

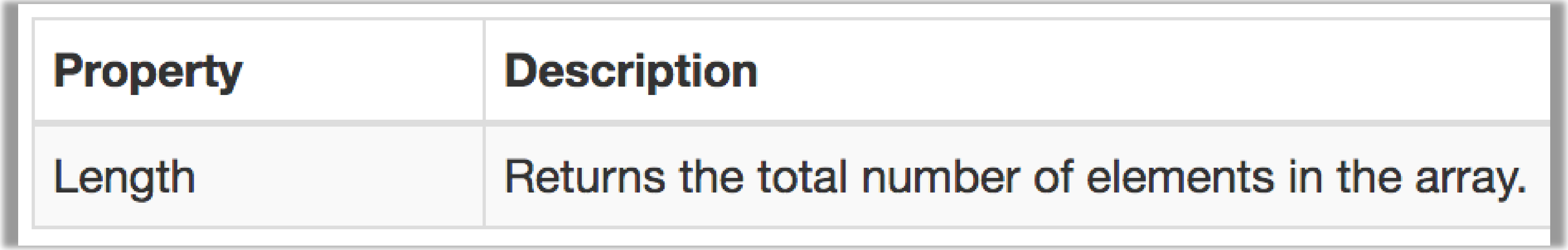


```
// defining array with 5 elements
// which indicates the size of an array
int[] intArray2 = { 1, 2, 3, 4, 5 };
for (int i = 0; i < 5; i++)
    Console.Write("{0}\t", intArray2[i]);
```

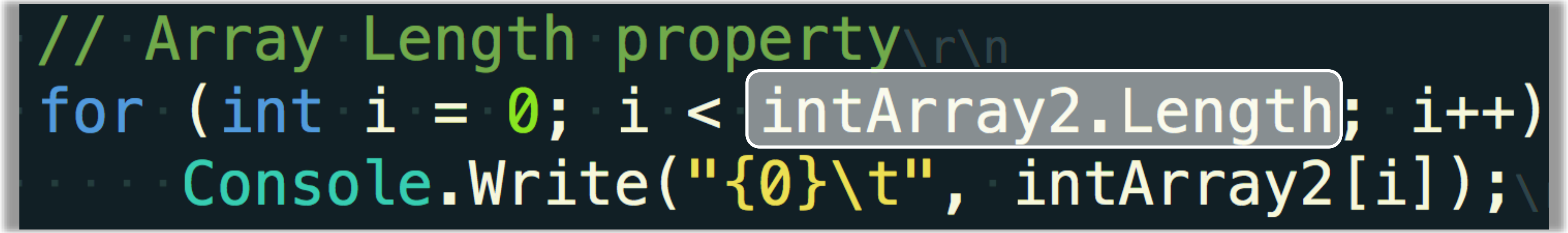


```
1         2         3         4         5
```

Array Property



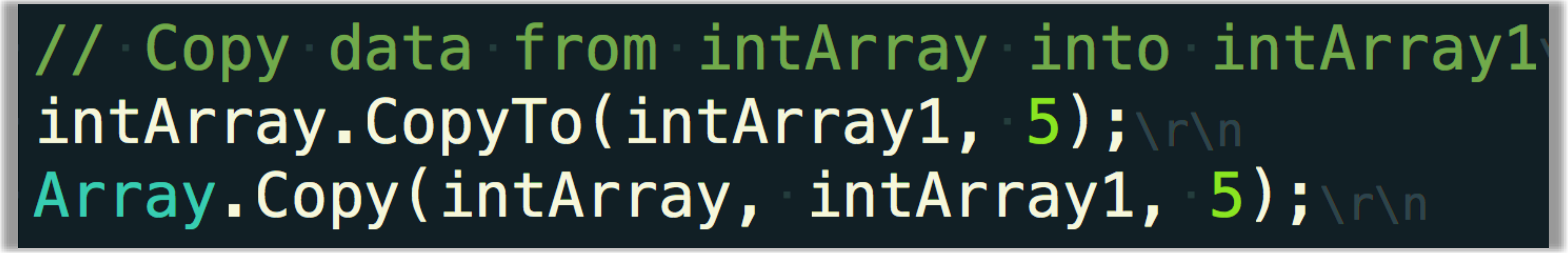
Property	Description
Length	Returns the total number of elements in the array.



```
// Array Length property
for (int i = 0; i < intArray2.Length; i++)
    Console.Write("{0}\t", intArray2[i]);
```

Array Helper Class

- CopyTo & Array.Copy



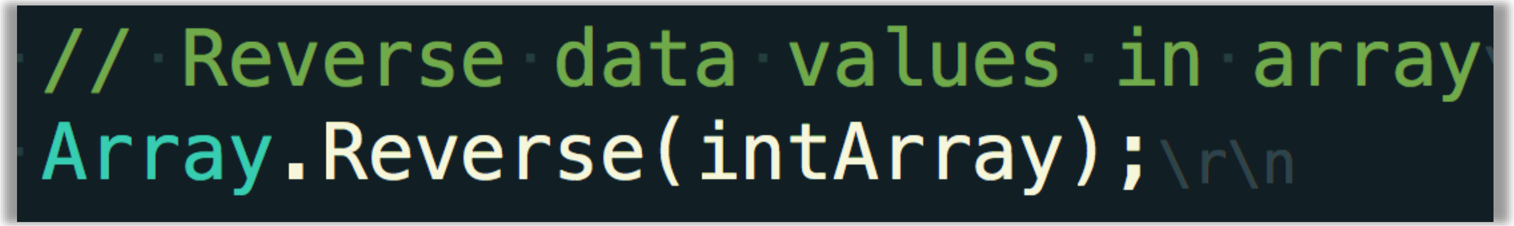
```
// Copy data from intArray into intArray1
intArray.CopyTo(intArray1, 5);
Array.Copy(intArray, intArray1, 5);
```

intArray	5	3	7	2	4
intArray1	1	2	3	4	5

After Copy Operation intArray1	5	3	7	2	4

Array Helper Class

- Reverse Data in an array – Array.Reverse



```
// Reverse data values in array
Array.Reverse(intArray);
```

intArray	5	3	7	2	4
intArray	4	2	7	3	5

Array Helper Class

- Array.GetLength
- Array.Clear
- Array.GetUpperBound
- Array.GetLowerBound
- Array.GetValue
- Array.IndexOf
- Array.LastIndexOf
- Array.SetValue
- Array.Sort

Sorting Algorithms

- Sorting data in ***ASCENDING*** or **DESCENDING** order
- **Dictionary Words** are usually in ***ASCENDING*** order
- Searching the **best price online** - want to sort data according to price in ***ASCENDING*** order
- **Best Student in course** - **DESCENDING** order

Bubble Sort

intArray	5	3	7	2	4

End of Pass ONE

Compare intArray[0] with intArray[1],
if (intArray[i] < intArray[i-1]) then swap values and move next
else move next

3	5	2	4	7

End of Pass TWO

3	2	4	5	7

End of Pass THREE

2	3	4	5	7

Bubble Sort

$\textbf{Input}: an\ integer\ array$
$\textbf{Output}: Sorted\ array$

```
1.  $temp \leftarrow 0$
2.  $i \leftarrow array.Length - 1$
3.  $j \leftarrow 0$
4.  $if\,(\boldsymbol{array[j] > array[j+1]})$
5.      $temp \leftarrow array[j+1]$
6.      $array[j+1] \leftarrow array[j]$
7.      $array[j] \leftarrow temp$
8.  $j \leftarrow j + 1$
9.  $if\,(j < i)$
10.     Go to step 4
11. $i \leftarrow i - 1$
12. $if\,(i > 0)$
13.     Go to step 3
14. Return array
```

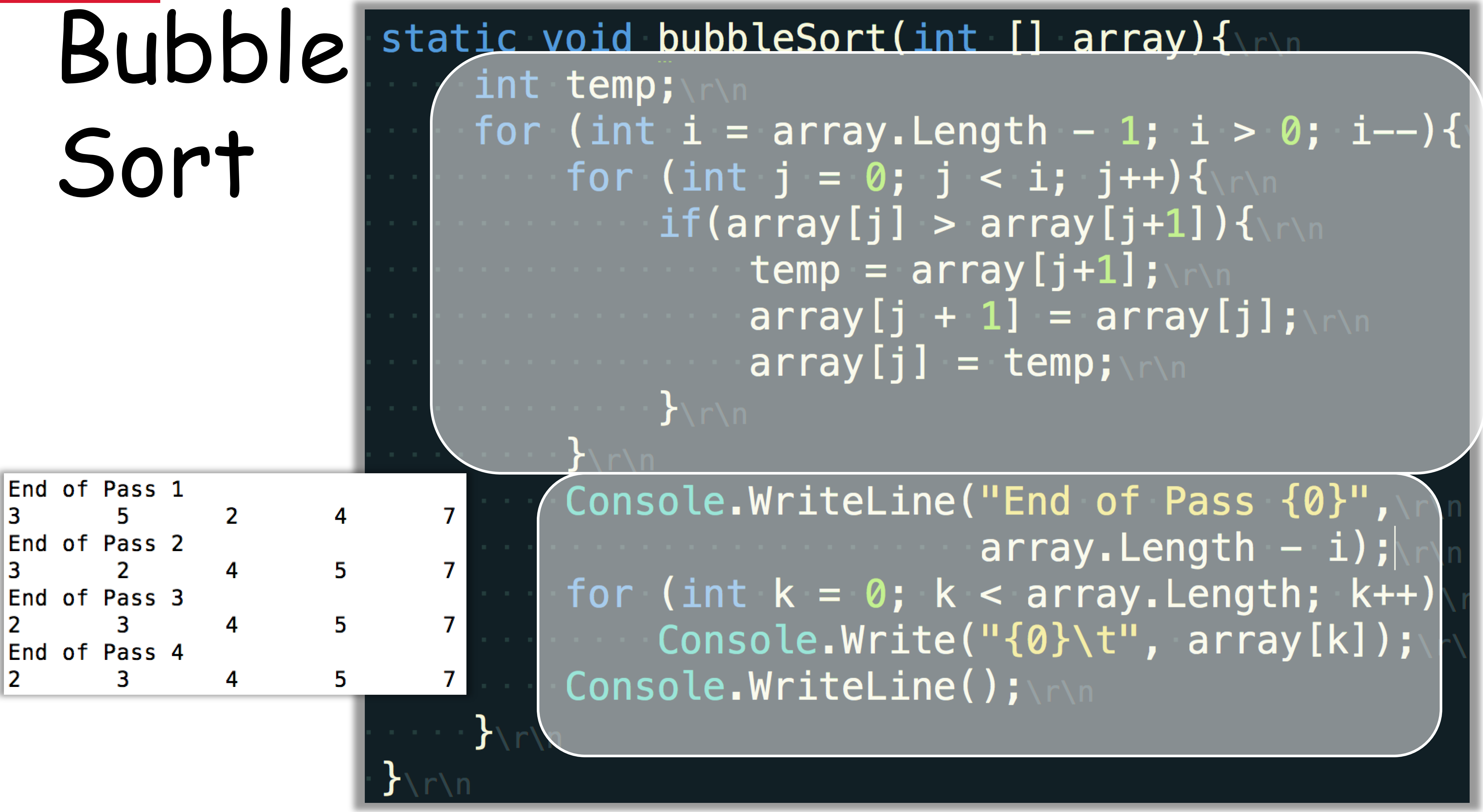
Bubble Sort

$\boldsymbol{Input}: an\ integer\ array$

$\boldsymbol{Output}: Sorted\ array$

```
$temp \leftarrow 0$
$for\ (i \leftarrow array.Length\ - 1; \boldsymbol{i > 0}; i - -)$
    $for\ (j \leftarrow 0;\ \boldsymbol{j < i};\ j + +)$
        $if(\boldsymbol{array[j] > array[j + 1]})$
            $temp \leftarrow array[j + 1]$
            $array[j + 1] \leftarrow array[j]$
            $array[j] \leftarrow temp$
```

Bubble Sort



```
static void bubbleSort(int [] array){
    int temp;
    for (int i = array.Length - 1; i > 0; i--){
        for (int j = 0; j < i; j++){
            if(array[j] > array[j+1]){
                temp = array[j+1];
                array[j + 1] = array[j];
                array[j] = temp;
            }
        }
        Console.WriteLine("End of Pass {0}",
                          array.Length - i);
        for (int k = 0; k < array.Length; k++)
            Console.Write("{0}\t", array[k]);
        Console.WriteLine();
    }
}
```

```
End of Pass 1
3       5       2       4       7
End of Pass 2
3       2       4       5       7
End of Pass 3
2       3       4       5       7
End of Pass 4
2       3       4       5       7
```

Insertion Sort

- One element list is always sorted
- Start with 2nd element and sort the first two elements and so on

```
$if\ (\boldsymbol{array[j] > array[j+1]})$
    Swap $array[j+1]\ and\ array[j]$
```

6	2	5	10	1	8	22	14

Insert the element at the right position in the left sorted array

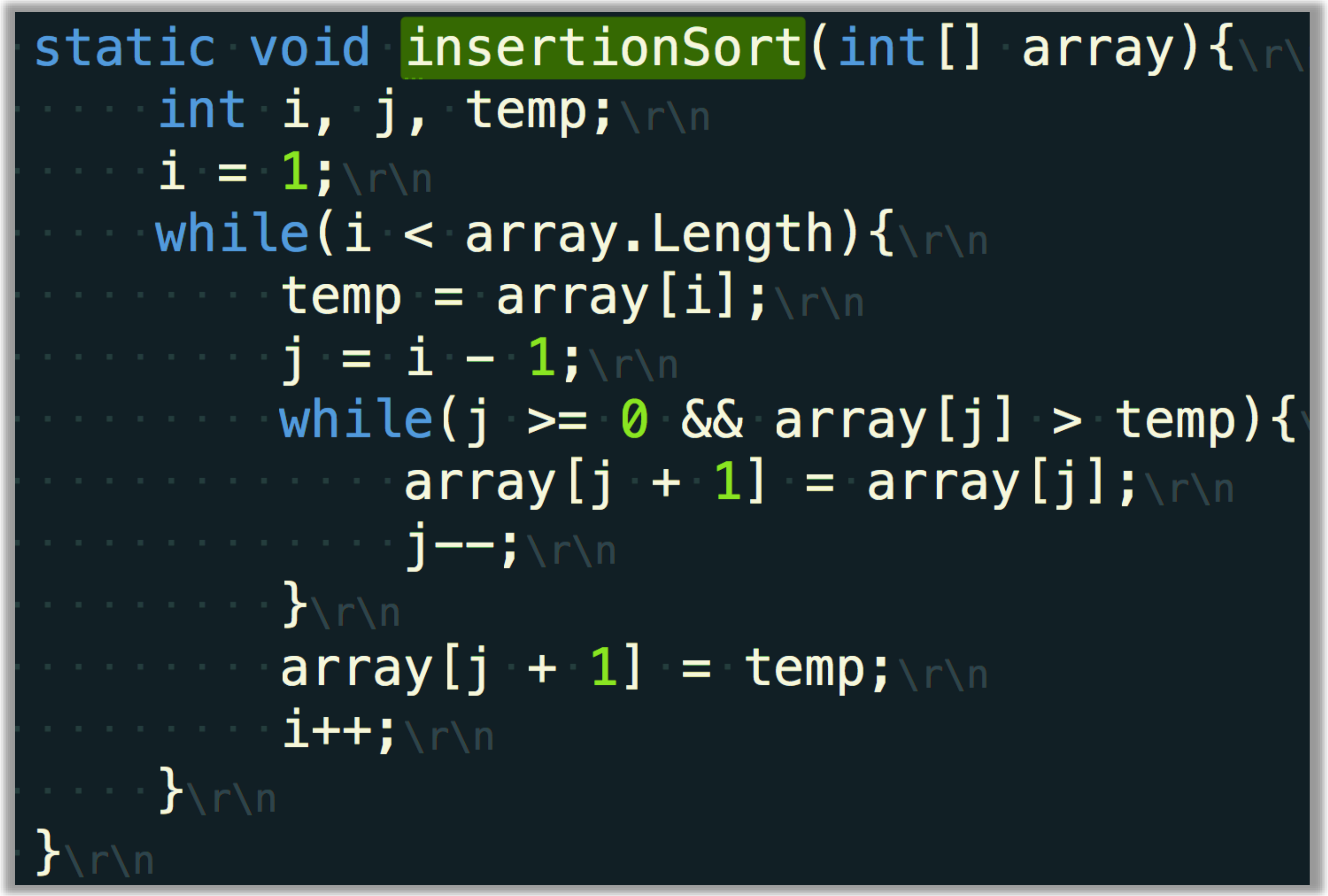
Insertion Sort

$\boldsymbol{Input}: an\ integer\ array$

$\boldsymbol{Output}: Sorted\ array$

```
$i \leftarrow 1$
while $(\boldsymbol{i < array.Length})$
    $temp \leftarrow array[i]$
    $j \leftarrow i - 1$
    while $(\boldsymbol{j \geq 0\ and\ array[j] > temp})$
        $array[j+1] \leftarrow array[j]$
        $j \leftarrow j - 1$
    $array[j+1] \leftarrow temp$
    $i \leftarrow i + 1$
```

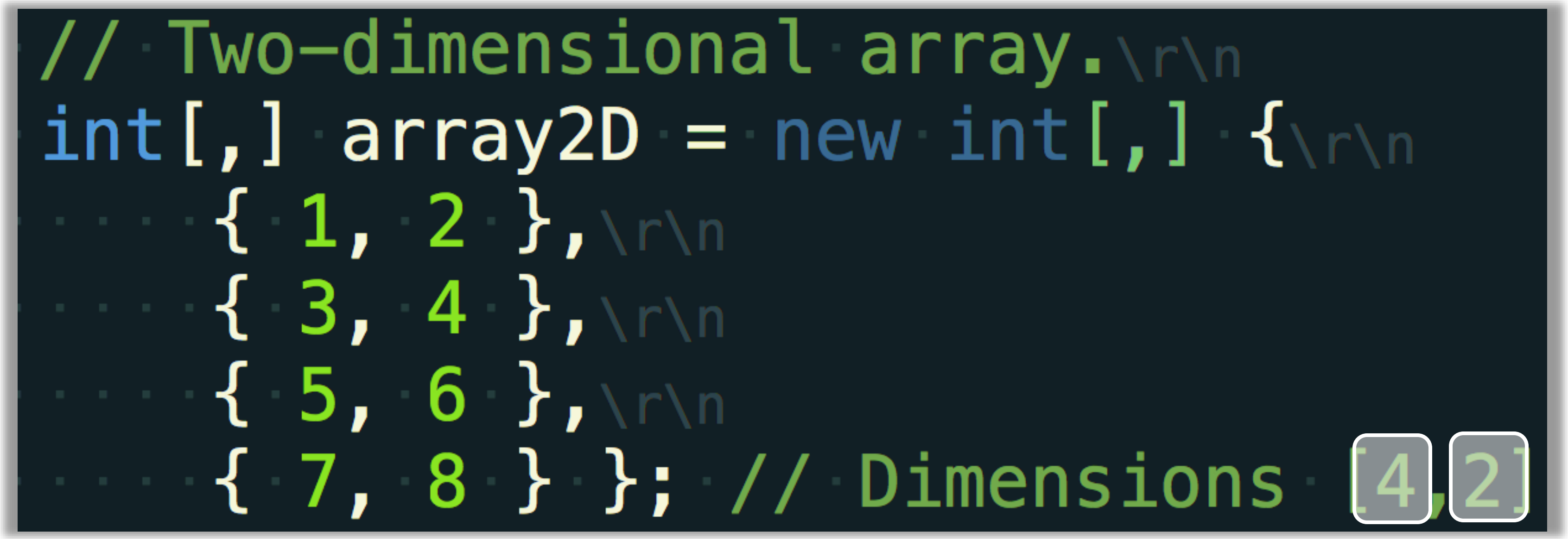
Insertion Sort



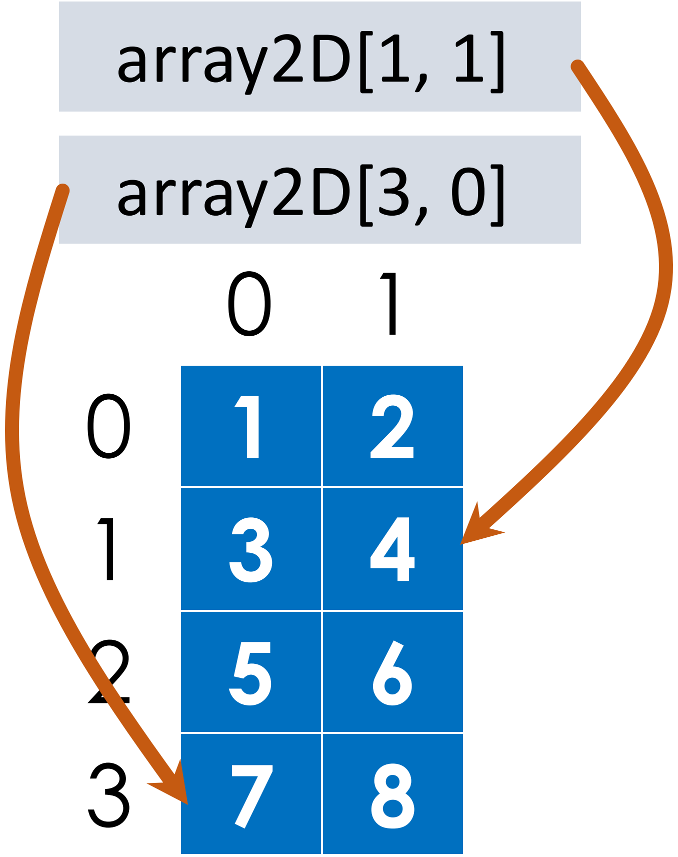
```
static void insertionSort(int[] array){
    int i, j, temp;
    i = 1;
    while(i < array.Length){
        temp = array[i];
        j = i - 1;
        while(j >= 0 && array[j] > temp){
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = temp;
        i++;
    }
}
```

Multi-Dimensional Arrays

- Arrays with more than one dimension
- Theoretically *n* dimensions



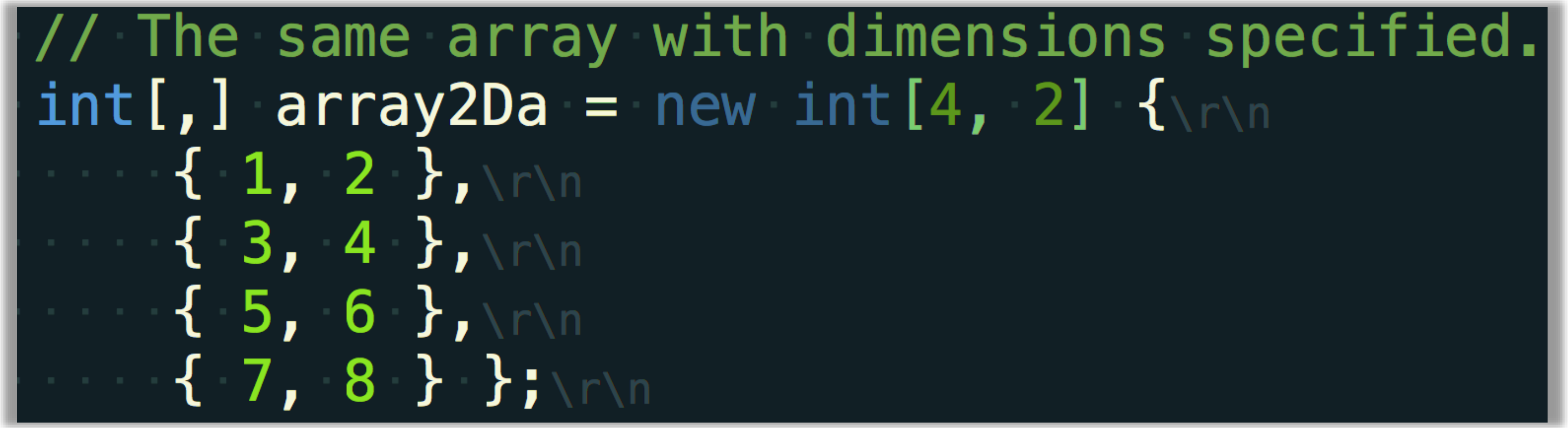
```
// Two-dimensional array.
int[,] array2D = new int[,] {
    { 1, 2 },
    { 3, 4 },
    { 5, 6 },
    { 7, 8 } }; // Dimensions [4,2]
```



	0	1
0	1	2
1	3	4
2	5	6
3	7	8

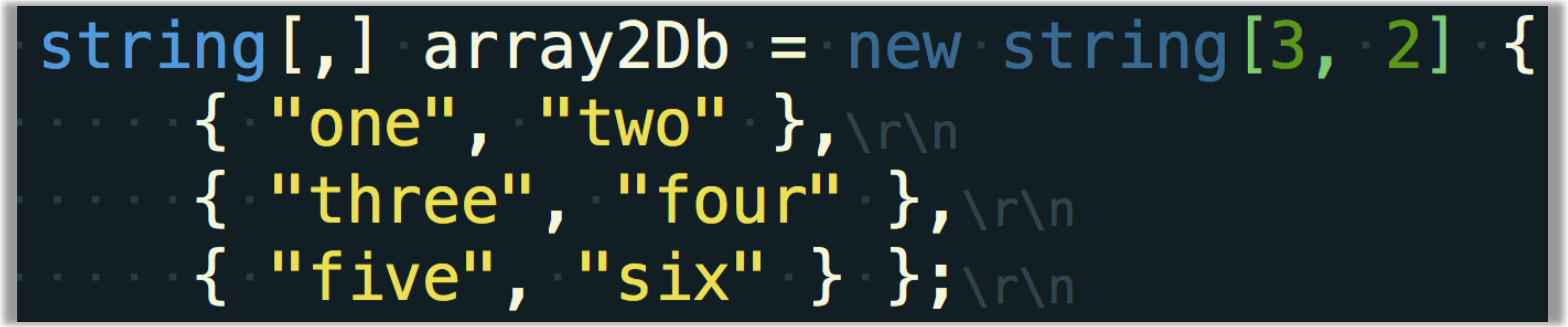
4 arrays with 2 elements in each

Multi-Dimensional Arrays



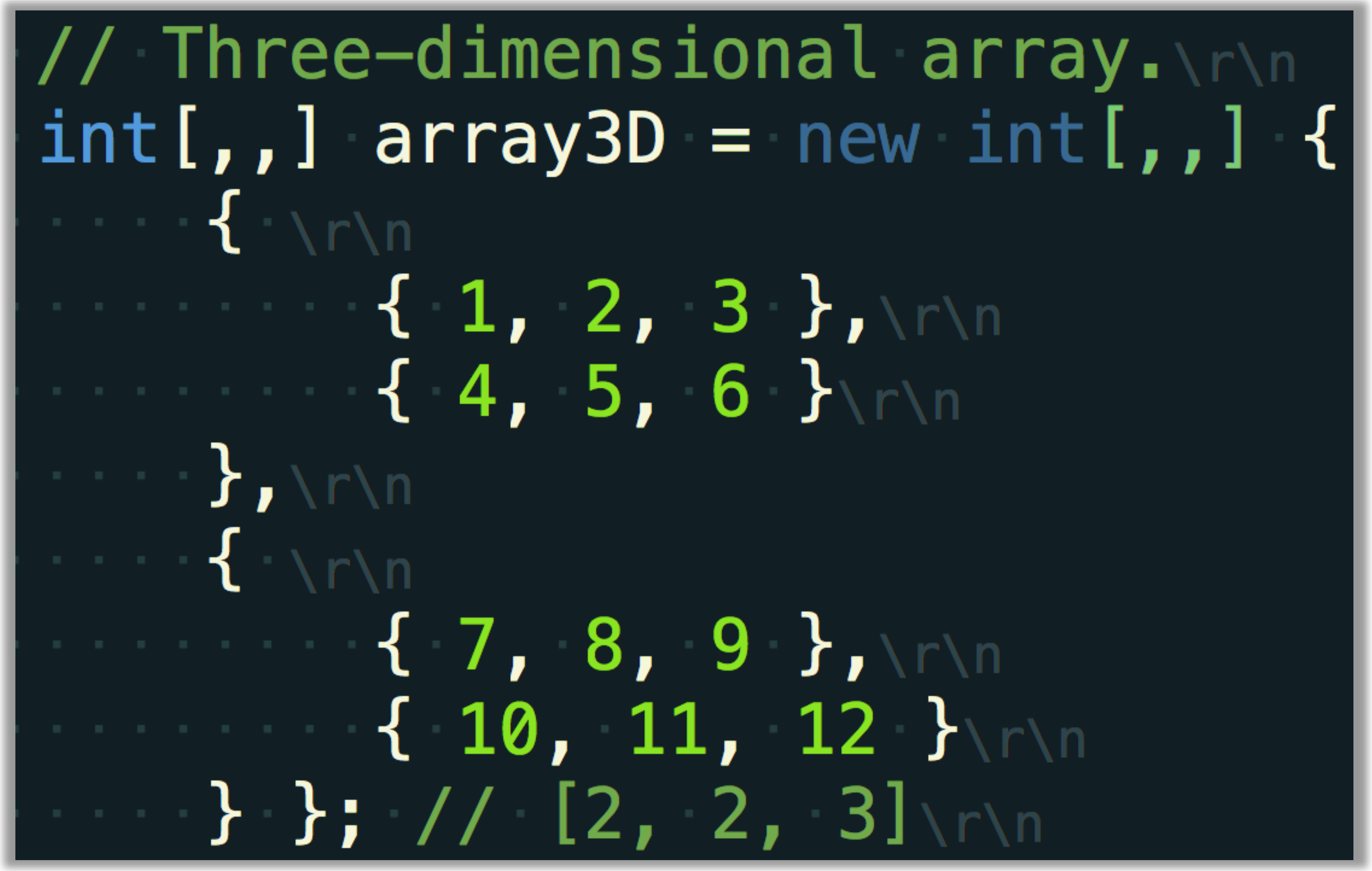
```
// The same array with dimensions specified.
int[,] array2Da = new int[4, 2] {
    { 1, 2 },
    { 3, 4 },
    { 5, 6 },
    { 7, 8 } };
```

	0	1
0	one	two
1	three	four
2	five	six



```
string[,] array2Db = new string[3, 2] {
    { "one", "two" },
    { "three", "four" },
    { "five", "six" } };
```

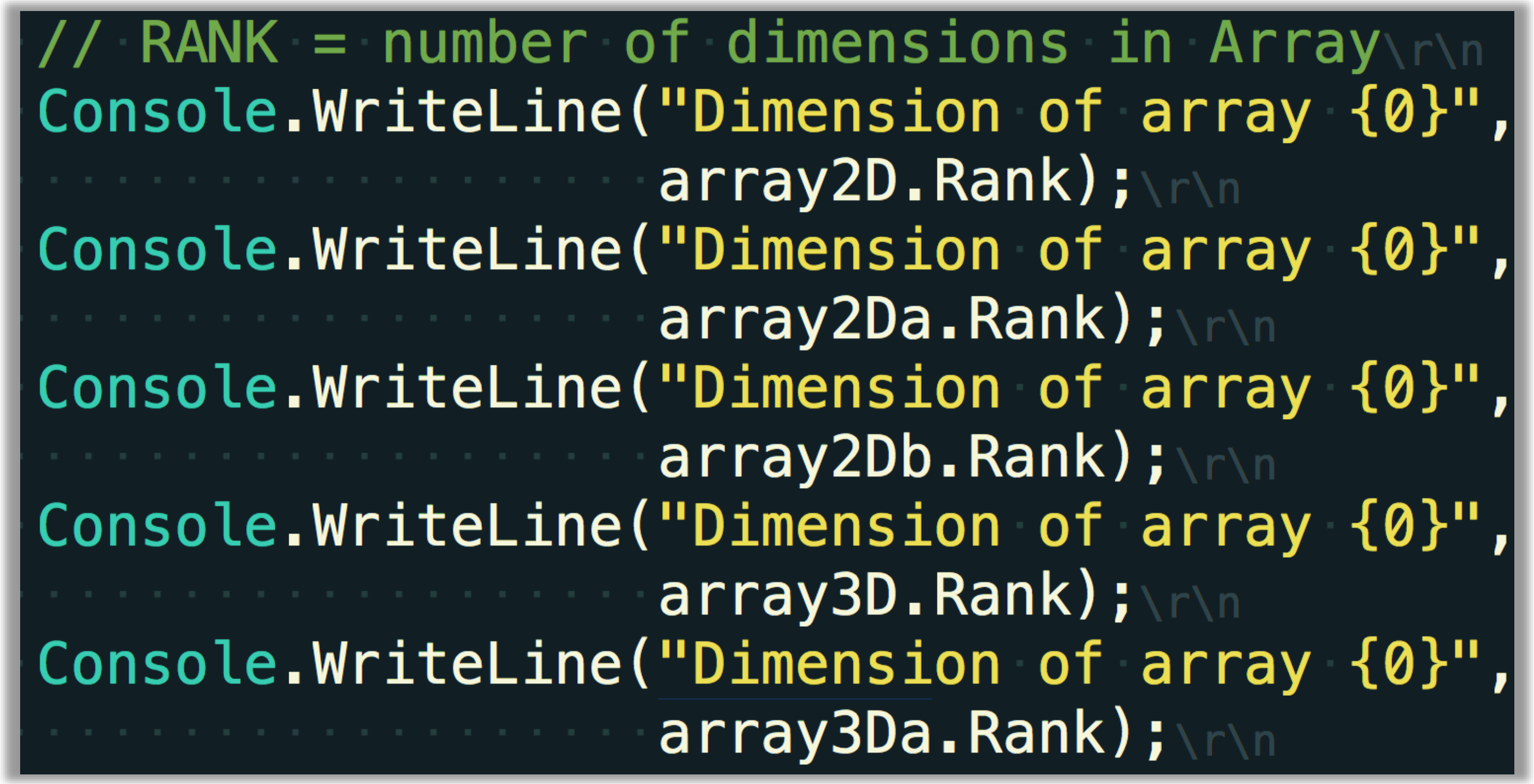
Multi-Dimensional Arrays



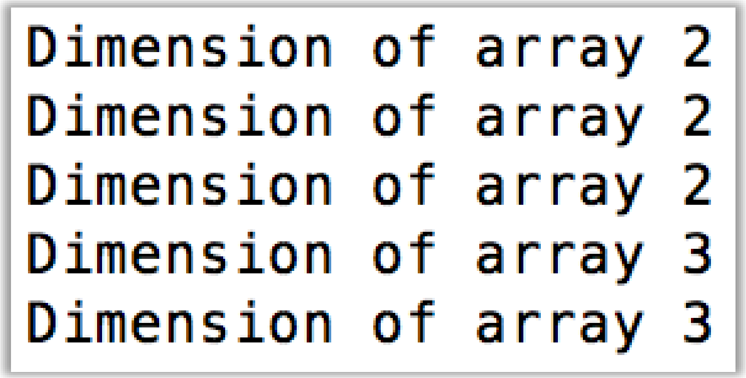
```
// Three-dimensional array.
int[,,] array3D = new int[,,] {
    {
        { 1, 2, 3 },
        { 4, 5, 6 }
    },
    {
        { 7, 8, 9 },
        { 10, 11, 12 }
    } }; // [2, 2, 3]
```

Array Rank – Property of Array

- arrayName.Rank



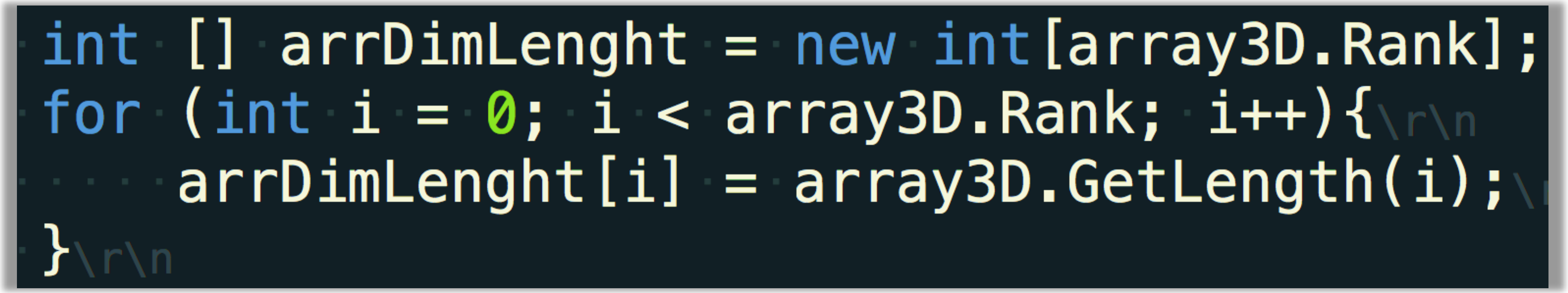
```
// RANK = number of dimensions in Array
Console.WriteLine("Dimension of array {0}",
                  array2D.Rank);
Console.WriteLine("Dimension of array {0}",
                  array2Da.Rank);
Console.WriteLine("Dimension of array {0}",
                  array2Db.Rank);
Console.WriteLine("Dimension of array {0}",
                  array3D.Rank);
Console.WriteLine("Dimension of array {0}",
                  array3Da.Rank);
```



```
Dimension of array 2
Dimension of array 2
Dimension of array 2
Dimension of array 3
Dimension of array 3
```

Printing Multidimensional Array

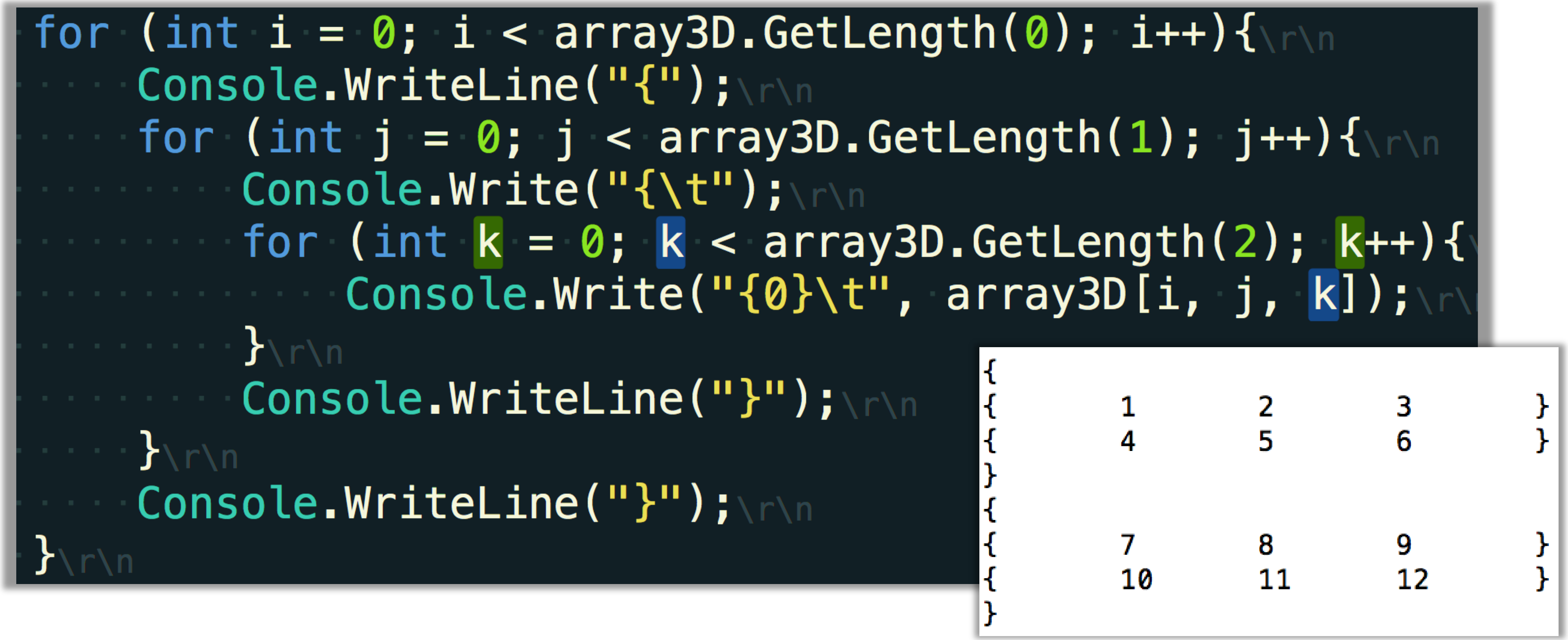
- **arrayName.GetLength(int index)**
    - Gives length of each dimension



```
int [] arrDimLenght = new int[array3D.Rank];
for (int i = 0; i < array3D.Rank; i++){\r\n
    arrDimLenght[i] = array3D.GetLength(i);
}\r\n
```

2	2	3
0	1	2

Printing Multidimensional Array



```
for (int i = 0; i < array3D.GetLength(0); i++){
    Console.WriteLine("{");
    for (int j = 0; j < array3D.GetLength(1); j++){
        Console.Write("{\t");
        for (int k = 0; k < array3D.GetLength(2); k++){
            Console.Write("{0}\t", array3D[i, j, k]);
        }
        Console.WriteLine("}");
    }
    Console.WriteLine("}");
}
```

```
{
{        1        2        3        }
{        4        5        6        }
}
{
{        7        8        9        }
{        10       11       12       }
}
```

Matrix - 2 Dimensional Array

- A **rectangular arrangements** of elements in **rows** and **columns**
- Rank of a Matrix: $ROWS \times COLUMNS$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & . & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad n \times n$$

Matrix - Operations

- Addition
- Subtraction
- Multiplication
- Scalar Multiplication - Multiplication with a number
- Transposition
- Inverse
- Row Operations

Matrix - Types

- **Rectangular** - $ROWS \neq COLUMNS$
- **Square** - $ROWS = COLUMNS$
    - Diagonal Matrix
    - Triangular Matrix
    - Identity Matrix
    - Symmetric Matrix

Matrix - Addition & Subtraction

- Two matrix can be added or subtracted only when they have the **same RANK**

$$\begin{bmatrix} 1 & 3 & 1 \\ 1 & -2 & 8 \end{bmatrix} + \begin{bmatrix} 5 & 6 & 4 \\ 4 & 5 & 5 \end{bmatrix} = \begin{bmatrix} 1+5 & 3+6 & 1+4 \\ 1+4 & -2+5 & 8+5 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & 9 & 5 \\ 5 & 3 & 13 \end{bmatrix}$$

Matrix – Addition & Subtraction

- A Matrix – a 2-Dimensional Array

Matrix1

Row 1	0	1	2	3
Row 2	1	4	5	6
		0	1	2
		Col 1	Col 2	Col 3

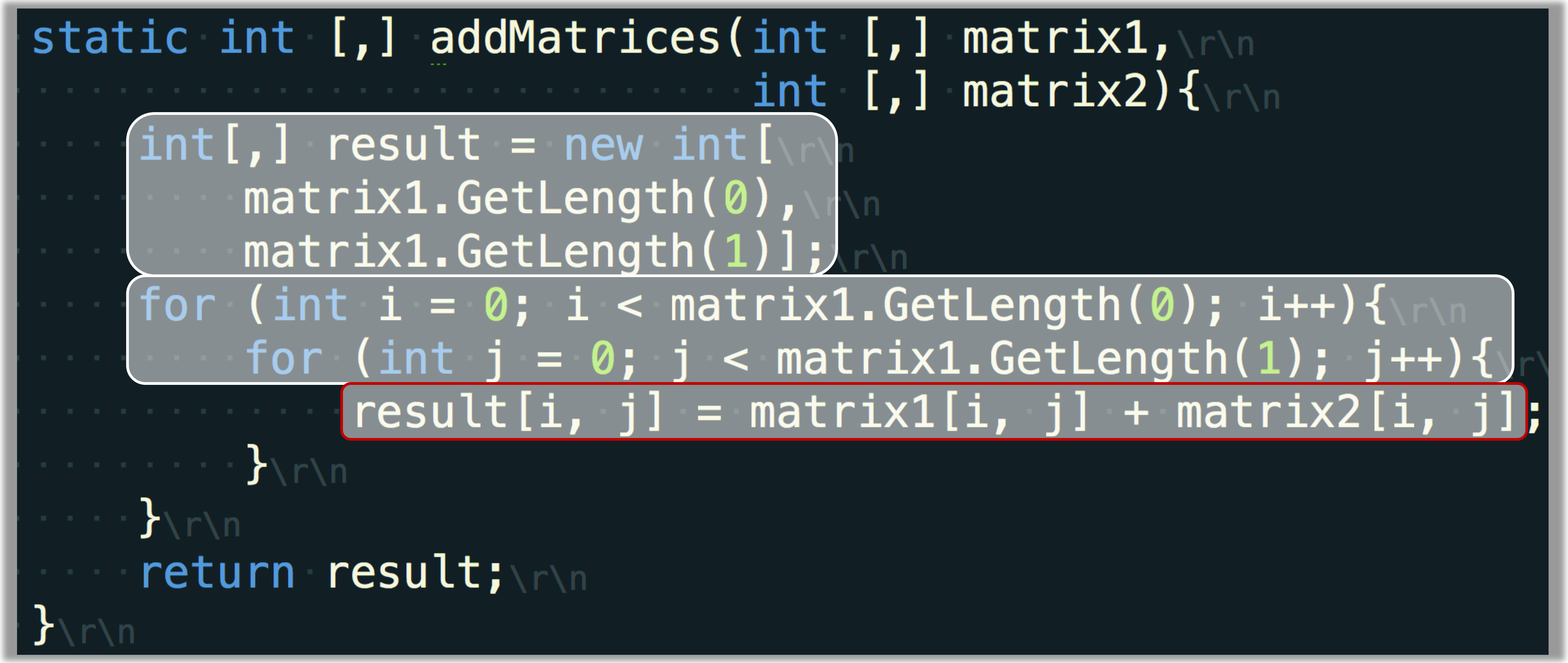
Matrix2

0	6	7	2
1	6	3	0
	0	1	2
	Col 1	Col 2	Col 3

Result

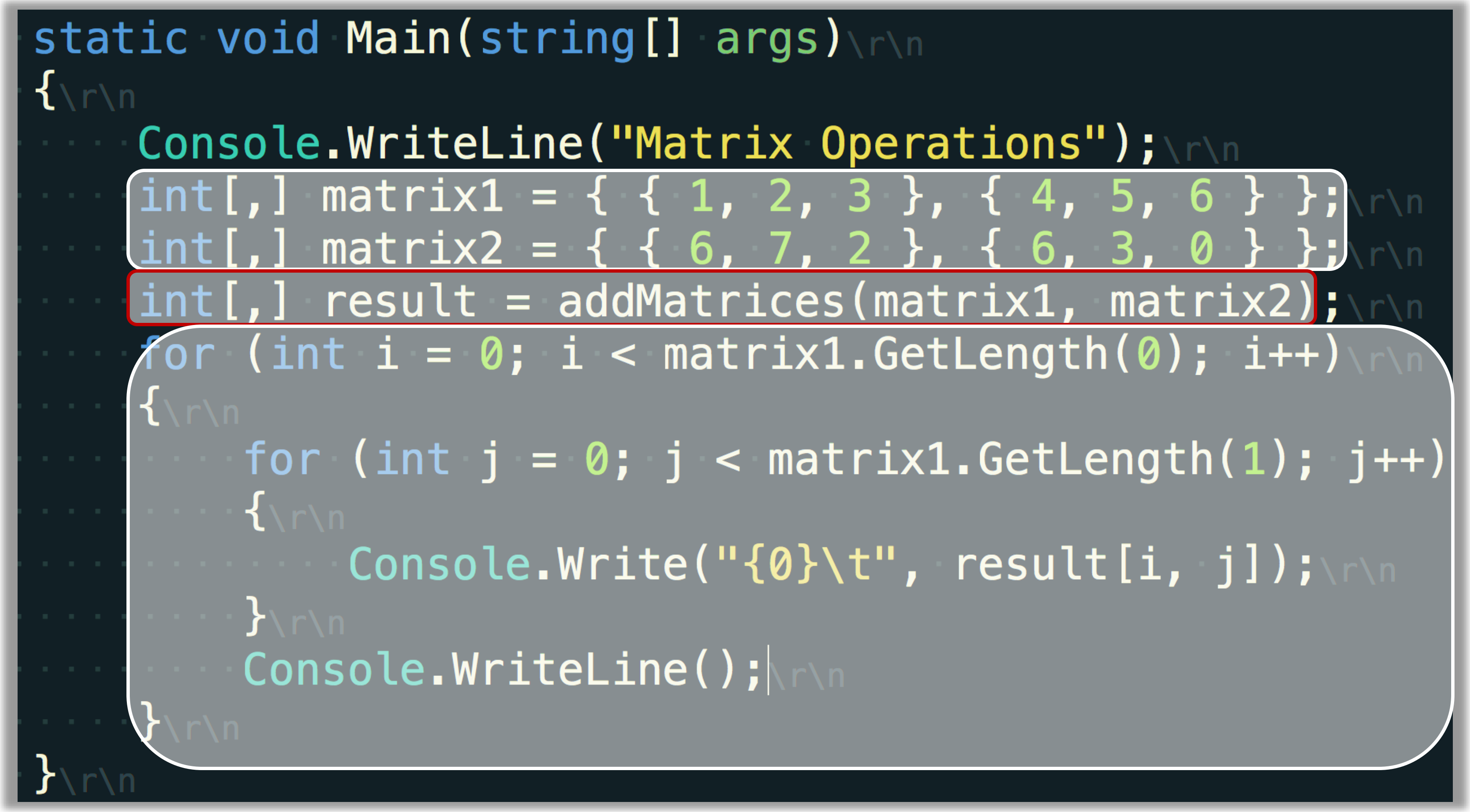
0	7	9	5
1	10	8	6
	0	1	2
	Col 1	Col 2	Col 3

Matrix - Addition & Subtraction

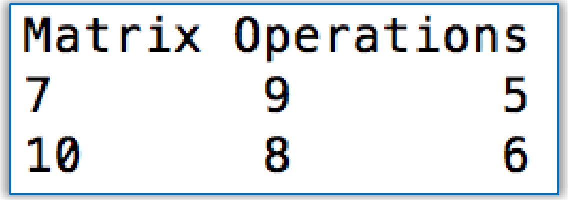


```
static int [,] addMatrices(int [,] matrix1,
                           int [,] matrix2){
    int[,] result = new int[
        matrix1.GetLength(0),
        matrix1.GetLength(1)];
    for (int i = 0; i < matrix1.GetLength(0); i++){
        for (int j = 0; j < matrix1.GetLength(1); j++){
            result[i, j] = matrix1[i, j] + matrix2[i, j];
        }
    }
    return result;
}
```

Matrix - Addition & Subtraction

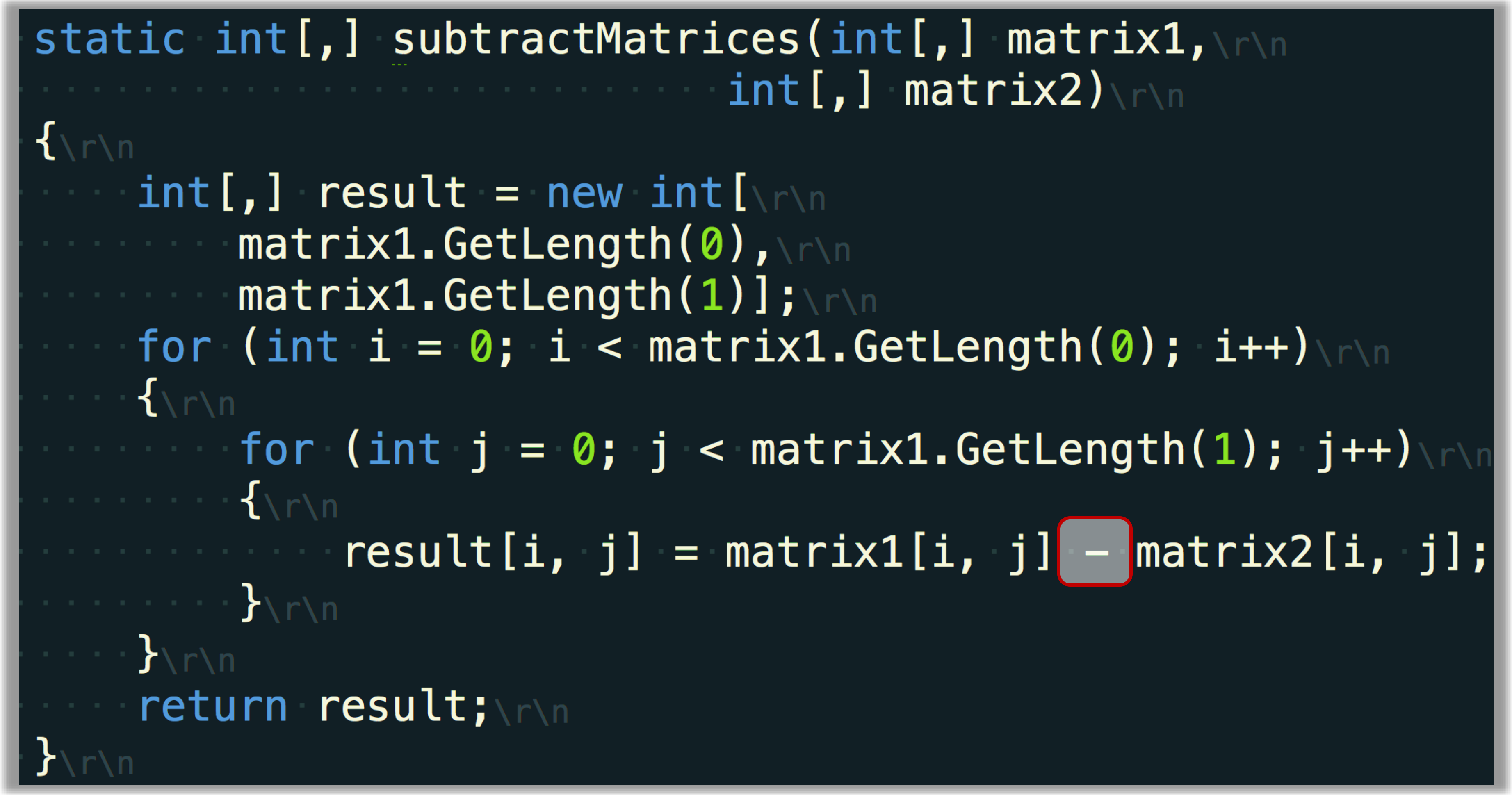


```
static void Main(string[] args)
{
    Console.WriteLine("Matrix Operations");
    int[,] matrix1 = { { 1, 2, 3 }, { 4, 5, 6 } };
    int[,] matrix2 = { { 6, 7, 2 }, { 6, 3, 0 } };
    int[,] result = addMatrices(matrix1, matrix2);
    for (int i = 0; i < matrix1.GetLength(0); i++)
    {
        for (int j = 0; j < matrix1.GetLength(1); j++)
        {
            Console.Write("{0}\t", result[i, j]);
        }
        Console.WriteLine();
    }
}
```



```
Matrix Operations
7               9               5
10              8               6
```

Matrix - Addition & Subtraction



```
static int[,] subtractMatrices(int[,] matrix1,
                               int[,] matrix2)
{
    int[,] result = new int[
        matrix1.GetLength(0),
        matrix1.GetLength(1)];
    for (int i = 0; i < matrix1.GetLength(0); i++)
    {
        for (int j = 0; j < matrix1.GetLength(1); j++)
        {
            result[i, j] = matrix1[i, j] - matrix2[i, j];
        }
    }
    return result;
}
```

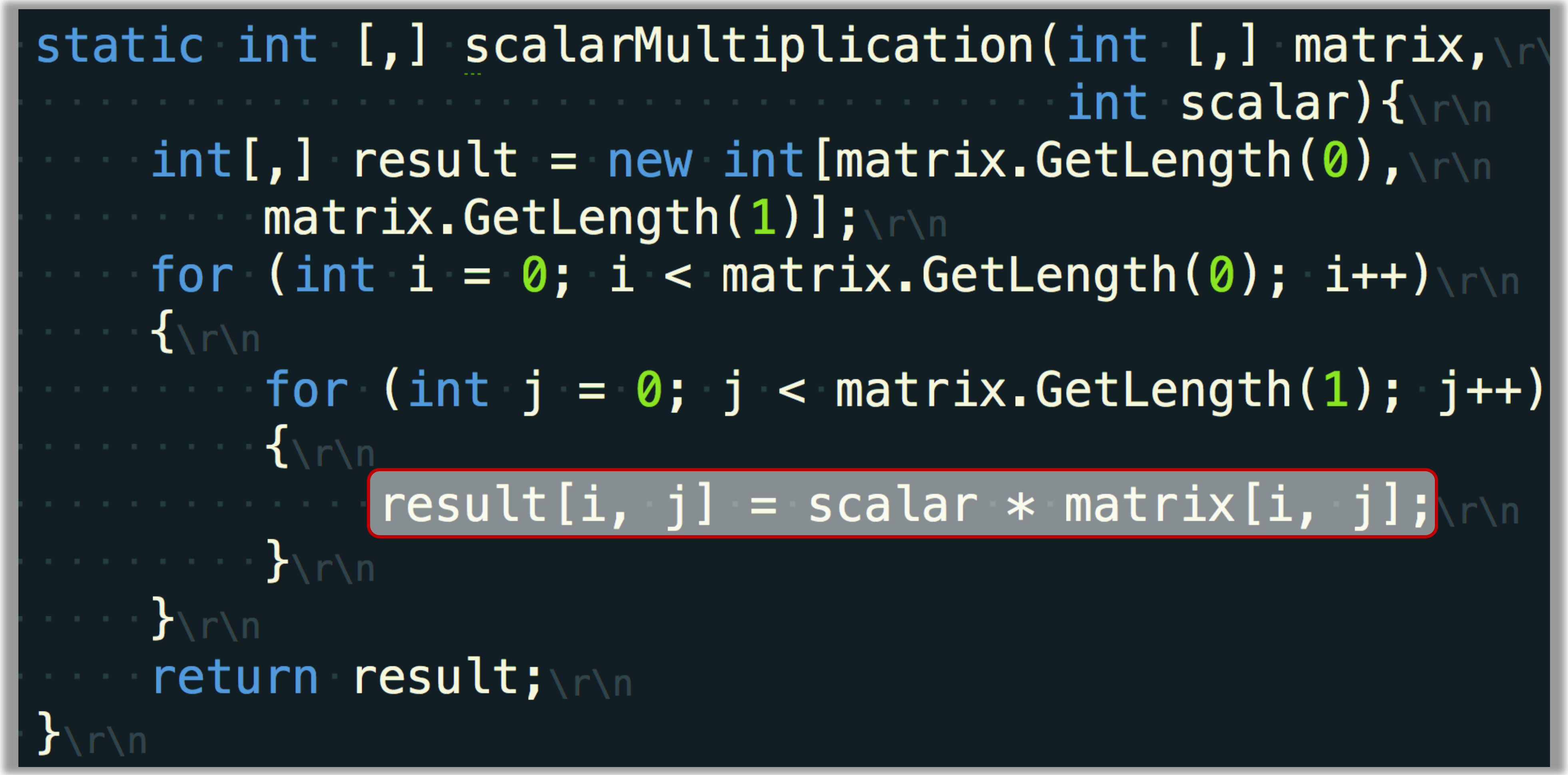
Matrix - Scalar Multiplication

- Matrix is multiplied with a number - multiply all entries in the matrix with the number

$$3 \times \begin{bmatrix} 1 & 3 & 1 \\ 1 & -2 & 8 \end{bmatrix} = \begin{bmatrix} 3 \times 1 & 3 \times 3 & 3 \times 1 \\ 3 \times 1 & 3 \times -2 & 3 \times 8 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 9 & 3 \\ 3 & -6 & 24 \end{bmatrix}$$

Matrix – Scalar Multiplication



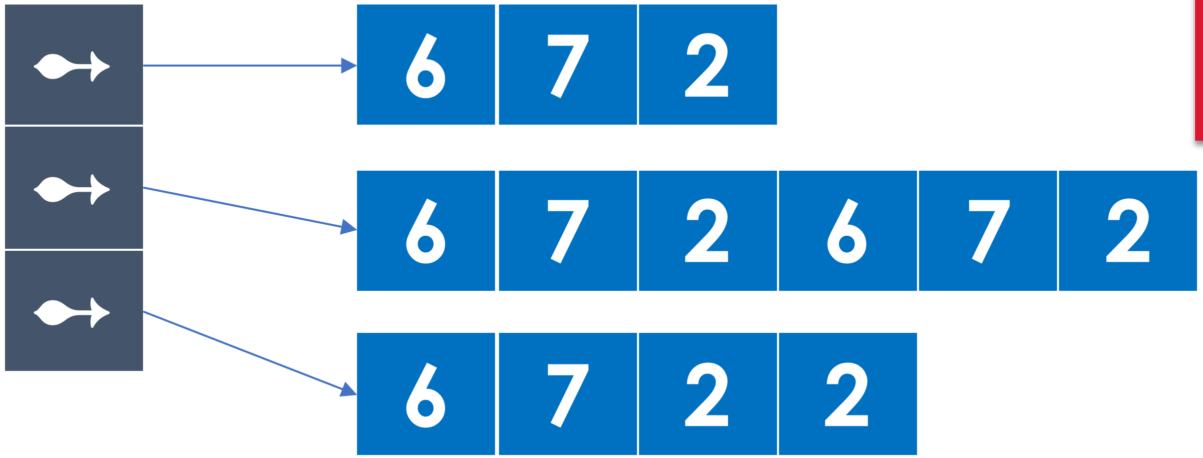
```
static int [,] scalarMultiplication(int [,] matrix,
                                    int scalar){
    int[,] result = new int[matrix.GetLength(0),
        matrix.GetLength(1)];
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        for (int j = 0; j < matrix.GetLength(1); j++)
        {
            result[i, j] = scalar * matrix[i, j];
        }
    }
    return result;
}
```

Matrix - Operations

- Addition
- Subtraction
- Multiplication
- Scalar Multiplication - Multiplication with a number
- Transposition
- Inverse
- Row Operations

Jagged Arrays

- Array of Arrays
  - Elements are arrays

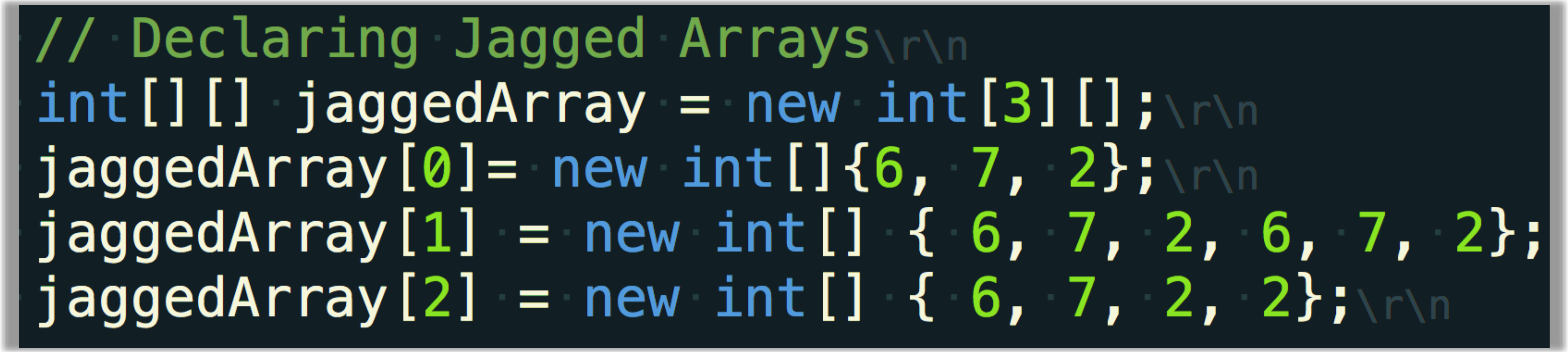


6 7 2

6 7 2 6 7 2

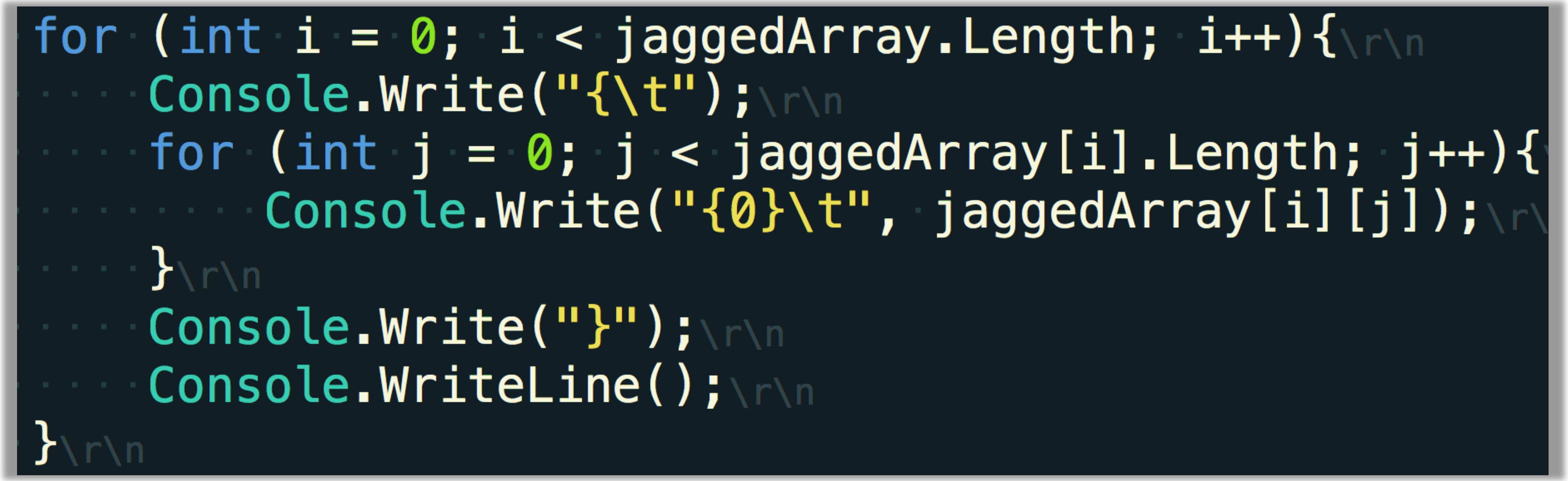
6 7 2 2

Jagged Array of 1-Dimentional Arrays

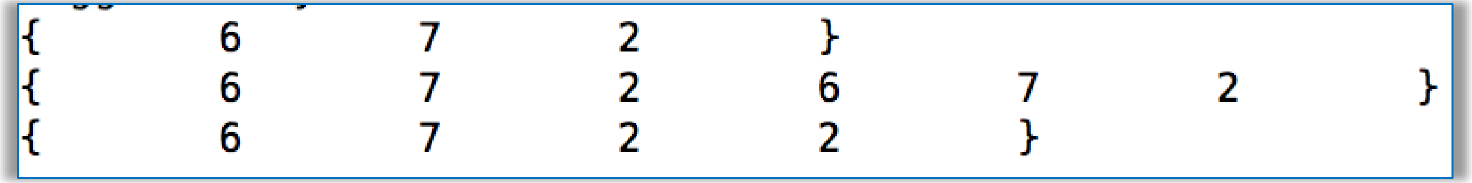


```
// Declaring Jagged Arrays\r\n
int[][] jaggedArray = new int[3][];\r\n
jaggedArray[0]= new int[]{6, 7, 2};\r\n
jaggedArray[1] = new int[] { 6, 7, 2, 6, 7, 2};
jaggedArray[2] = new int[] { 6, 7, 2, 2};\r\n
```

Jagged Arrays - Printing

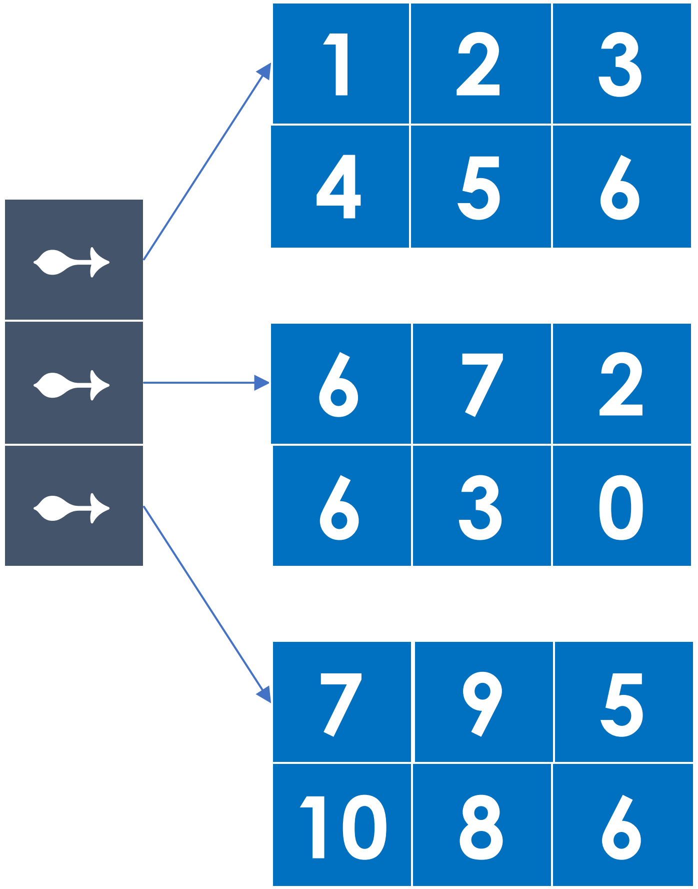


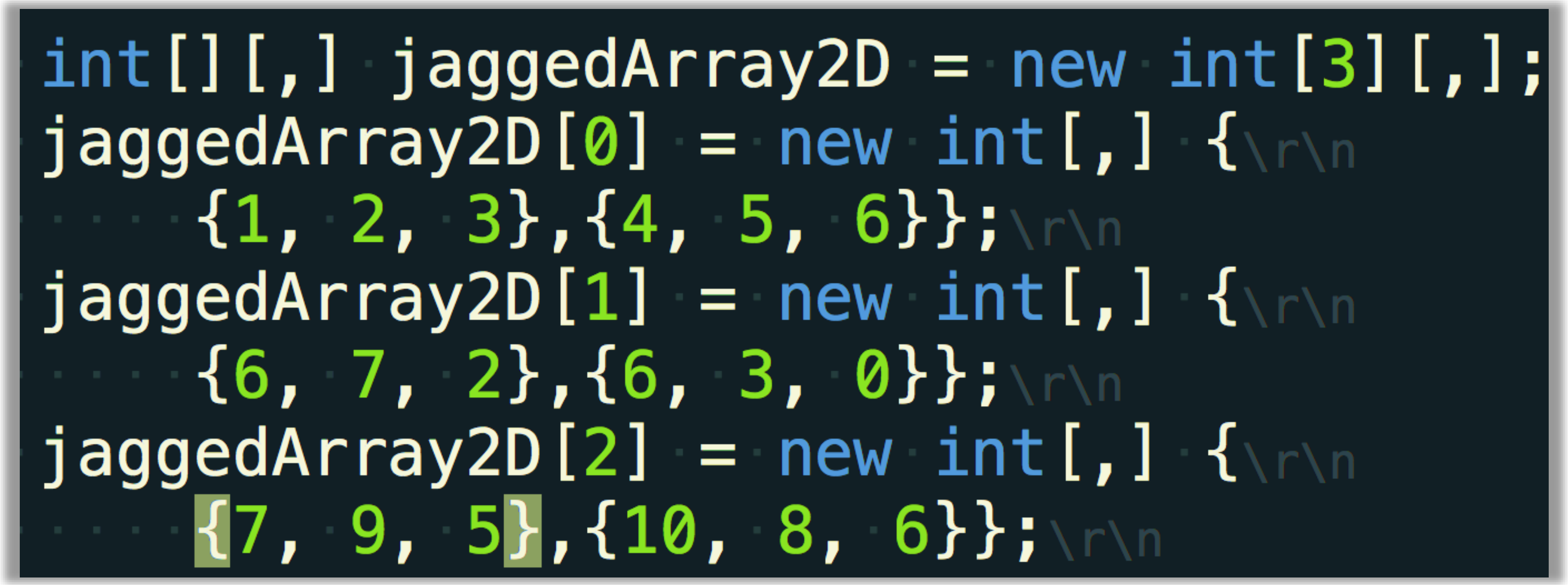
```
for (int i = 0; i < jaggedArray.Length; i++){
    Console.Write("{\t");
    for (int j = 0; j < jaggedArray[i].Length; j++){
        Console.Write("{0}\t", jaggedArray[i][j]);
    }
    Console.Write("}");
    Console.WriteLine();
}
```



```
{       6       7       2       }
{       6       7       2       6       7       2       }
{       6       7       2       2       }
```

Jagged Arrays





```
int[][,] jaggedArray2D = new int[3][,];
jaggedArray2D[0] = new int[,] {
    {1, 2, 3},{4, 5, 6}};
jaggedArray2D[1] = new int[,] {
    {6, 7, 2},{6, 3, 0}};
jaggedArray2D[2] = new int[,] {
    {7, 9, 5},{10, 8, 6}};
```

Reference and Reading Material

- C# Tutorial: Link
- Arrays: Link
- Bubble Sort: Link
- Insertion Sort: Link
- Multidimensional Arrays – Link
- Matrix: Link