Arrays

CS 133N/ CS 161N

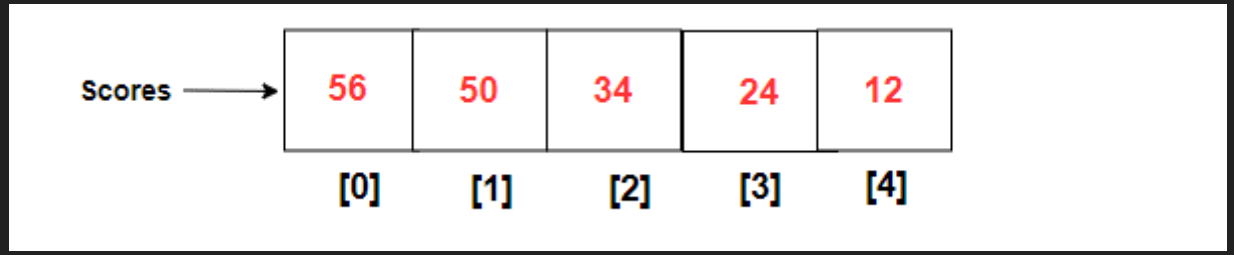
Mari Good

Objectives

- Introduce Arrays
- Introduce Value Types and Reference Types
- Introduce you to C# syntax for working with arrays
- Practice with several examples

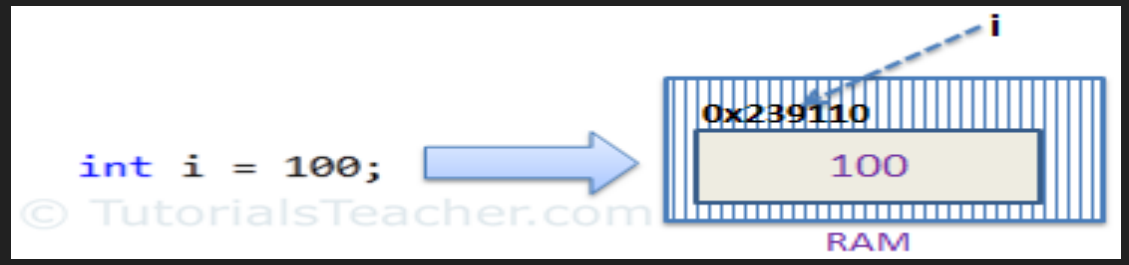
An Array

- Array is the data structure that stores fixed number of values of the same datatype.
  - Remember the Pig problem? We only had 2 dice so we used 2 variables d1 and d2. BUT what if we were playing Yahtzee and we needed 5 dice?
  - Or a set of test scores for a class?



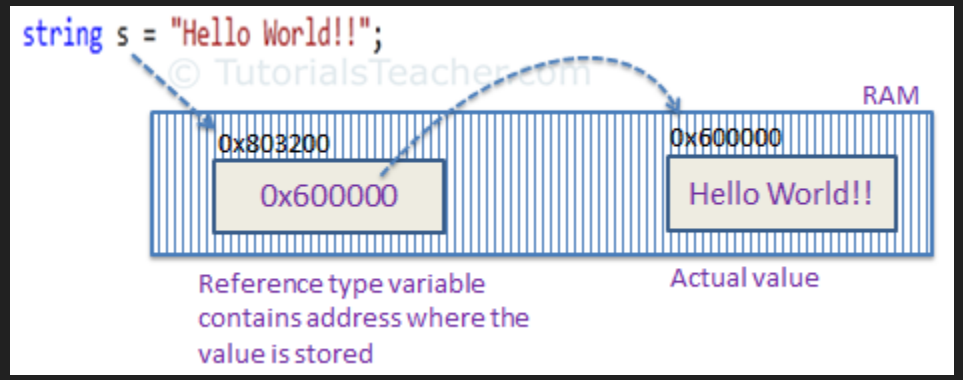
An Array is a Reference Type

- Datatypes in C# fall into 2 general categories
  - Value types – examples are int and double
    - When you declare an int, the compiler sets aside a chunk of memory that is big enough for the variable. The value is stored in that memory.



An Array is a Reference Type

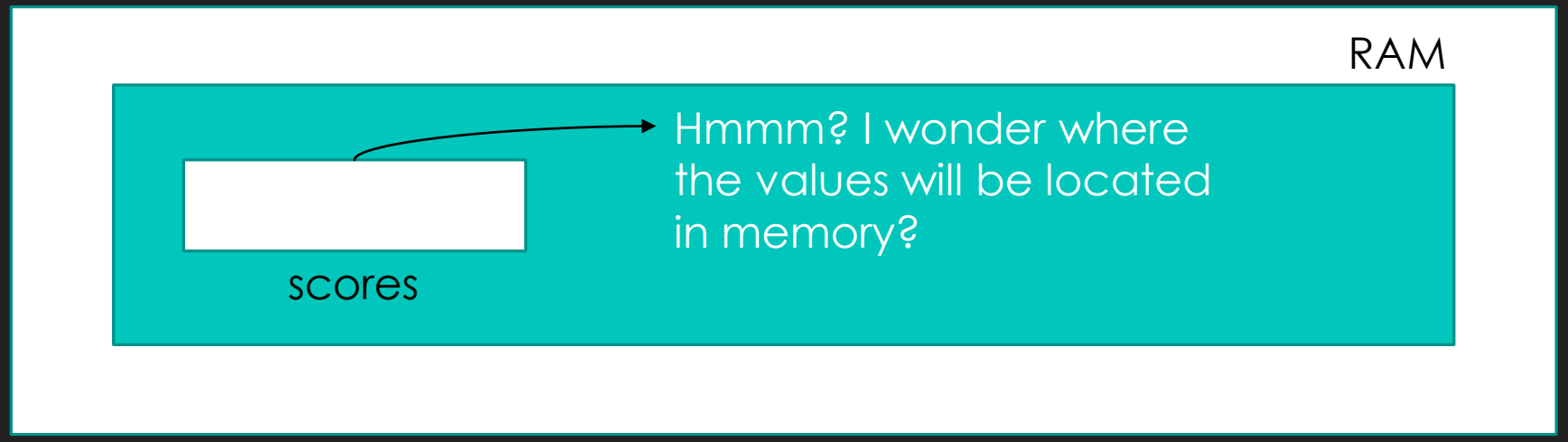
- Datatypes in C# fall into 2 general categories
  - Reference types – examples are strings and arrays
    - When you declare a string, the compiler sets aside a chunk of memory to store the ADDRESS of the string in memory. The actual string value is stored in a completely separate space in memory.



An Array is a Reference Type

- Datatypes in C# fall into 2 general categories
  - Reference types – examples are strings and arrays
    - When you declare an array, memory is allocated to hold the address of the array values in memory.

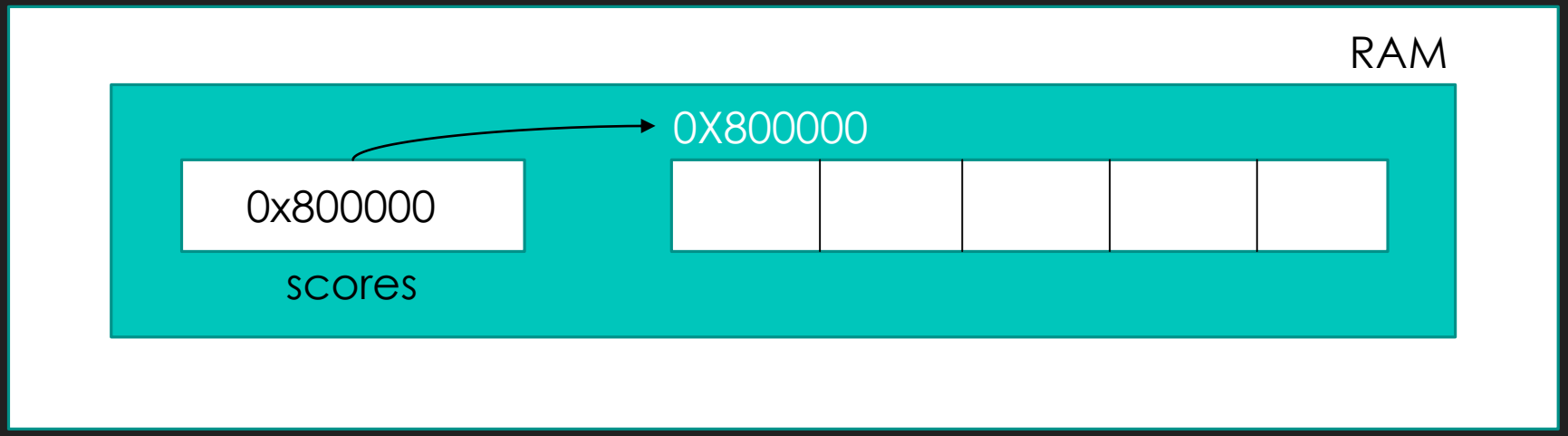
```
int[] scores;
```



An Array is a Reference Type

- Datatypes in C# fall into 2 general categories
  - Reference types – examples are strings and arrays
    - You have to "instantiate" the array in order to allocate space for the actual array values.

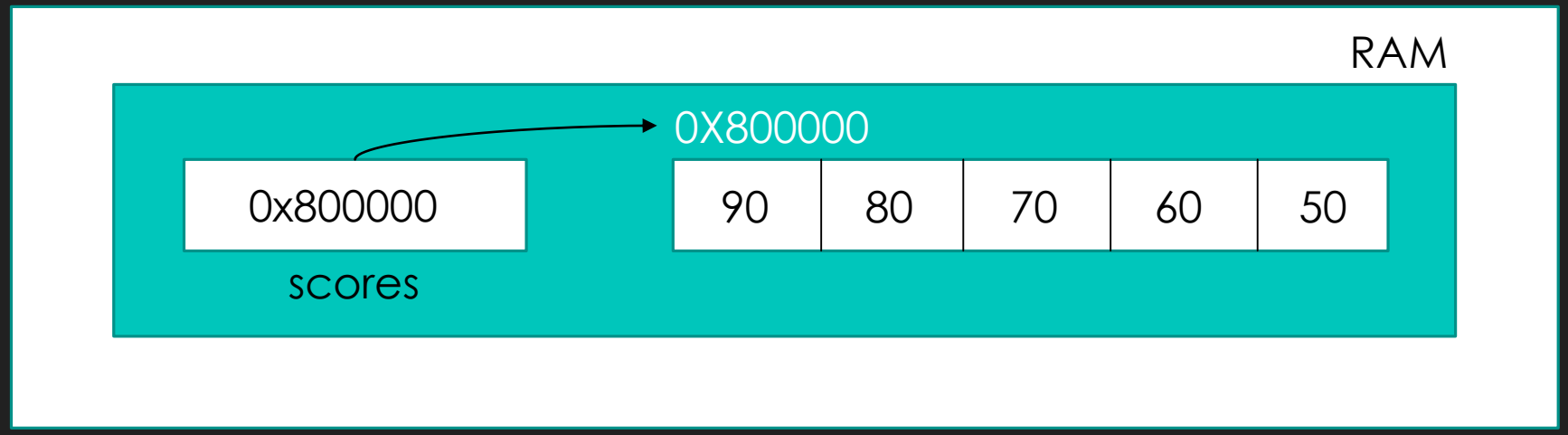
```
int[] scores = new int[5];
```



An Array is a Reference Type

- Datatypes in C# fall into 2 general categories
  - Reference types – examples are strings and arrays
    - You can do this with an initializer list when you know the array values at design time.

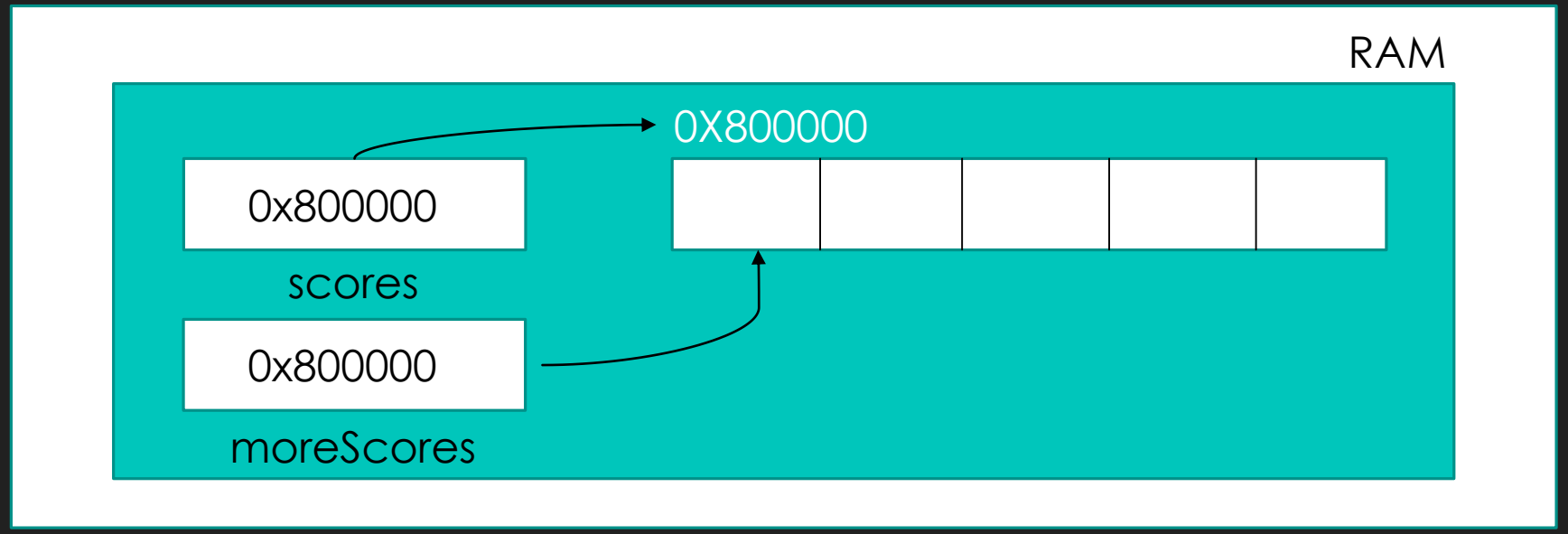
```
int[] scores = {90, 80, 70, 60, 50};
```



An Array is a Reference Type

- Because an array is a reference type, ASSIGNING an array name to another array name DOES NOT MAKE A COPY of the array.

```
int[] scores = new int[5];
int[] moreScores = scores;
```

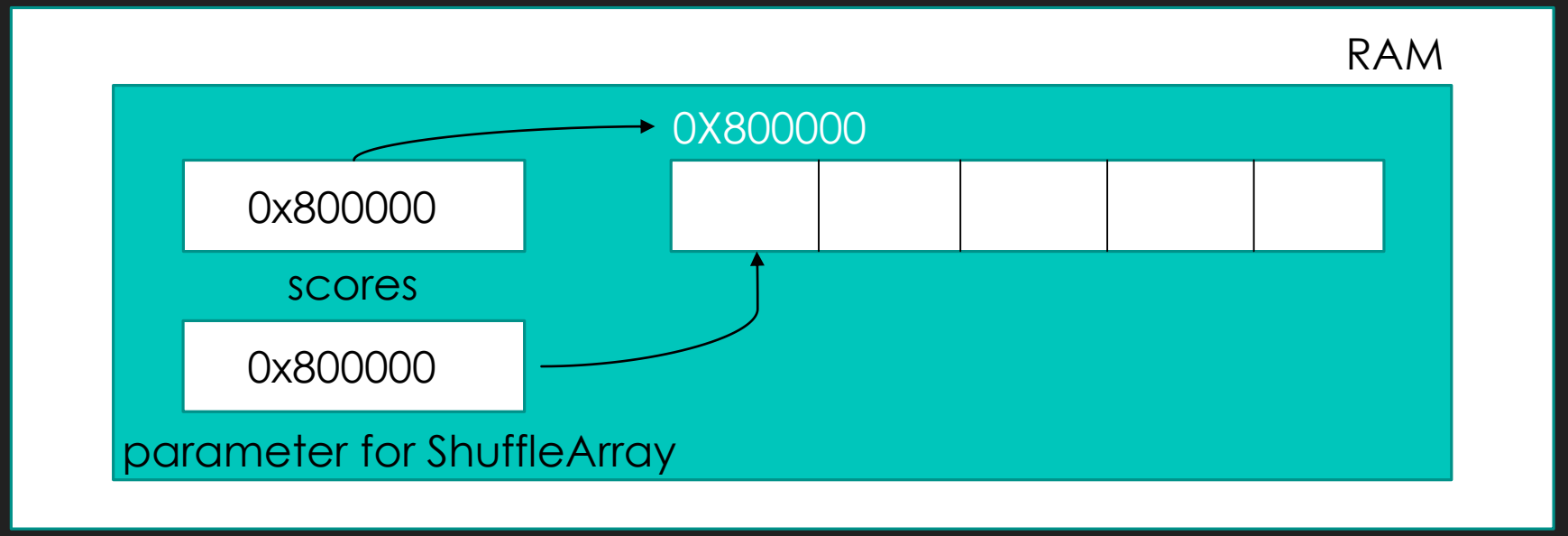


An Array is a Reference Type

- Because an array is a reference type, arrays are passed into a method BY REFERENCE. When the method changes the array it is changed in the calling code.

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

ShuffleArray(scores);
```



Let's start with an example

- Design and implement a program that processes a set of test scores.The user will enter scores in a range of 0 - 100 (invalid input will be ignored) from the keyboard and will enter -1 for the score when data entry is complete. The application will then calculate the average score, the lowest score and the highest score and display all 3 pieces of output on the screen.
- I'm going to start by writing code …

Declaring an Array

- I'll need to declare a variable
  - to store the scores. I'll use an array of ints or int[]
  - to store how many scores I think I need
  - to store the actual number of scores in the array

```
int numScores = 0;
const int SIZE = 5;
int[] scores = new int[SIZE];
```

datatype is int[]

Allocates space for 5 integers. The address of the first of those integers is stored in scores.

Accessing an Array Element

- Here's a loop that fills the array with values

```
int score = GetInt("score", -1, 100);
while (score != -1 && numScores < scores.Length)
{
    scores[numScores] = score;
    numScores++;
    if (numScores != scores.Length)
        score = GetInt("score", -1, 100);
}
```

Length is a property that returns the number of elements allocated for the array.

scores[0] is the first element in the array. Each time the loop iterates numScores will increase and the next value will get stored in the next spot in the array.

Accessing an Array Element

- Here's a loop that displays all of the scores

0 is the first array index.

```
for (int i = 0; i < numScores; i++)
    Console.WriteLine("Score {0} is: {1}", i + 1, scores[i]);
```

scores[i] access each array element starting at 0 and ending at (numScores – 1)

Accessing an Array Element

- Here's a loop that calculates the average

0 is the first array index.

scores[i] access each array element starting at 0 and ending at (numScores – 1)

```
for (int i = 0; i < numScores; i++)
      total += scores[i];
average = (double)total / numScores;
Console.WriteLine("The average is " + average.ToString("n2"));
```

Foreach Loop

- When an array is COMPLETELY full, you can use a foreach loop

Declare a variable to store array element.

The array name goes after the keyword in.

```
foreach (int aScore in scores)
    total += aScore;
```

The loop repeats from the beginning to the end of the array. With each iteration, the array element is placed in aScore.

```
average = (double)total / numScores;
Console.WriteLine("The average is " + average.ToString("n2"));
```

Passing an Array into a Method

- Here's a method to display the scores

datatype is int[]

```
static void DisplayScores(int[] scores, int numScores)
{
    for (int i = 0; i < numScores; i++)
        Console.WriteLine("Score {0} is: {1}", i + 1, scores[i]);
}

DisplayScores(scores, numScores);
```

Use the array in the body of the loop as usual. If you change an array element in the method, the change is reflected in main. An array is a reference type!

Just the array name, no datatype and no []!

Returning an Array from a Method

- Here's a method that generates the scores randomly

Return datatype is int[]

```
static int[] GenerateRandomScores(int size, int min, int max) {
    Random gen = new Random();
    int[] scores = new int[size];
    for (int i = 0; i < scores.Length; i++)
        scores[i] = gen.Next(min, max + 1);
    return scores;
}
```

Declare and allocate the array inside the method

Use the array in the usual way inside the method.

Return the array name

Declare the array but don't allocate space.

```
int[] scores = GenerateRandomScores(5, 1, 100);
```

Let's Try Some Together

- I'm going to do these in Visual Studio because I want to be able to use the debugger and random numbers.
- The lab description gives you a link to a Visual Studio solution with the code from the slides. The same code is available in dotnetfiddle.net.
- Let's look at what's there and then do all of the TODOs.

You Try The Rest

- Now that you've got the hang of it, try the rest of the problems from the lab yourself or with a partner.
- In the description of each problem in moodle, I've given you help in the form of one or more of these design tools
  - An illustration
  - An algorithm
  - A structure chart
- You'll have lots of questions. It's ok to ask me or a tutor or a classmate. I'll record screencasts as we discover what you need more help with.

What's Next

- Strings
- Don't forget
  - Reading Quiz 6
  - Programming Quiz 6
  - Lab 6 – 6 problems