

عنوان مضمون

# Visual Programming-I

توسط : صفری

خزان 1397

methods and Arrays

# Arrays

- You may want to store several values of the same type at the same time, without having to use a different variable for each value.
- You could use simple string variables as follows:  

```
string friendName1 = "Todd Anthony";  
string friendName2 = "Kevin Holton";  
string friendName3 = "Shane Laigle";
```
- Arrays are indexed lists of variables stored in a single array type variable.  

```
string [3] friendNames;
```

# Arrays

- The index is simply an integer, starting with 0 for the first entry, using 1 for the second, and so on. This means that you can go through the entries using a loop:

```
int i;  
for (i = 0; i < 3; i++)  
{   WriteLine($"Name with index of {i}: {friendNames[i]}");  
}
```

# Arrays

- Declaring Arrays:

Arrays are declared in the following way:

*<baseType>[] <name>;*

Arrays must be initialized before you have access to them.

Arrays can be initialized in two ways:

1. You can either specify the complete contents of the array in a literal form  
`int[] myIntArray = { 5, 9, 10, 2, 99 };`
2. specify the size of the array and use the new keyword to initialize all array elements.  
`int[] myIntArray = new int[5];`

# Arrays

- In addition, you can combine these two methods of initialization if you want:

```
int[] myIntArray = new int[5] { 5, 9, 10, 2, 99 };
```

- With this method, the sizes *must* match. You can't, for example, write the following:

```
int[] myIntArray = new int[10] { 5, 9, 10, 2, 99 };
```

- Here, the array is defined as having 10 members, but only five are defined, so compilation will **fail**.

# Arrays

- A side effect of this is that if you define the size using a variable, then that variable must be a constant:

```
const int arraySize = 5;
```

```
int[] myIntArray = new int[arraySize] { 5, 9, 10, 2, 99 };
```

- If you omit the const keyword, this code will **fail**.
- As with other variable types, there is no need to initialize an array on the same line that you declare it. The following is perfectly legal:

```
int[] myIntArray;
```

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

# Try It Out

```
static void Main(string[] args)
{
    string[] friendNames = { "Todd Anthony", "Kevin Holton", "Shane Laigle" };
    int i;
    WriteLine($"Here are {friendNames.Length} of my friends:");
    for (i = 0; i < friendNames.Length; i++)
    {
        WriteLine(friendNames[i]);
    }
    ReadKey();
}
```



# Arrays

- If you attempt to access elements **outside of the array size**, the code will fail.
- It just so happens that there is a more resilient method of accessing all the members of an array: **using foreach loops**.
- A foreach loop enables you to address each element in an array using this simple syntax:

```
foreach (<baseType> <name> in <array>)  
{  
    // can use <name> for each element  
}
```

# Arrays

you can modify the code in the last example as follows:

```
static void Main(string[] args)
{
    string[] friendNames = { "Todd Anthony", "Kevin Holton","Shane Laigle" };
    WriteLine($"Here are {friendNames.Length} of my friends:");
    foreach (string friendName in friendNames)
    {
        WriteLine(friendName);
    }
    ReadKey();
}
```

# Pattern Matching with switch case expression

- Recall the following code, where *<testVar>* is a known type, for example an *integer*, a *string*, or a *boolean*.
- An *integer*, for example, has a numeric value, and the case would check for a specific value (1, 2, 3, and so on) and then execute some code when matched.

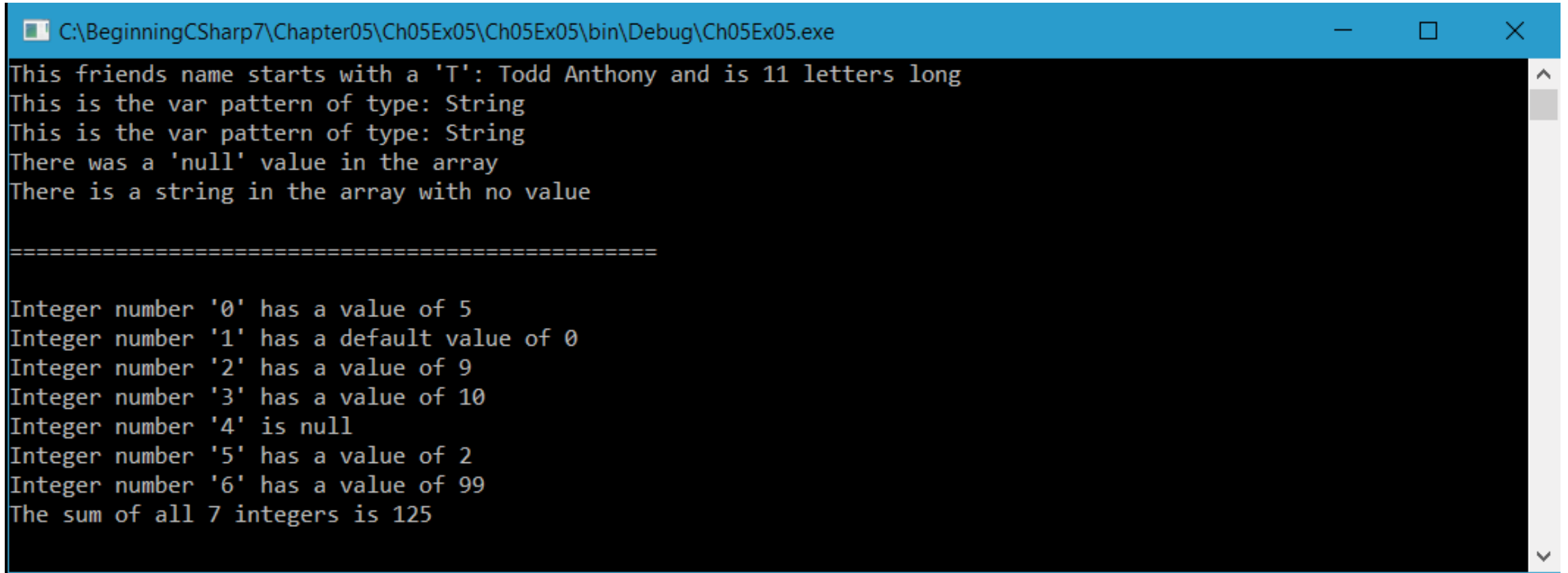
switch (*<testVar>*)

```
{    case <comparisonVal1>:  
        <code to execute if <testVar> == <comparisonVal1> >  
        break;  
    case <comparisonVal2>:  
        <code to execute if <testVar> == <comparisonVal2> >  
        break;  
    ...  
    default:  
        <code to execute if <testVar> != comparisonVals>  
        break;  
}
```

```
static void Main(string[] args)
{
    string[] friendNames = { "Todd Anthony", "Kevin Holton","Shane Laigle",null, "" };
    foreach (var friendName in friendNames)
    {
        switch (friendName)
        {
            case string t when t.StartsWith("T"):
                WriteLine("This friends name starts with a 'T': " + $"{friendName} and is {t.Length - 1}
letters long ");
                break;
            case string e when e.Length == 0:
                WriteLine("There is a string in the array with no value");
                break;
            case null:
                WriteLine("There was a 'null' value in the array");
                break;
            case var x:
                WriteLine("This is the var pattern of type: " + $"{x.GetType().Name}");
                break;
            default:
                break;
        }
    }
}
```

# Pattern Matching with switch case expression

After making sure you added using static System.Console;, execute the code. The result is shown in Figure(first part)



```
C:\BeginningCSharp7\Chapter05\Ch05Ex05\Ch05Ex05\bin\Debug\Ch05Ex05.exe
This friends name starts with a 'T': Todd Anthony and is 11 letters long
This is the var pattern of type: String
This is the var pattern of type: String
There was a 'null' value in the array
There is a string in the array with no value

=====

Integer number '0' has a value of 5
Integer number '1' has a default value of 0
Integer number '2' has a value of 9
Integer number '3' has a value of 10
Integer number '4' is null
Integer number '5' has a value of 2
Integer number '6' has a value of 99
The sum of all 7 integers is 125
```

```

static void Main(string[] args)
{
    int sum = 0, total = 0, counter = 0, intValue = 0;
    int?[] myIntArray = new int?[7] { 5, intValue, 9, 10, null, 2, 99 };
    foreach (var integer in myIntArray)
    {
        switch (integer)
        {
            case 0:
                WriteLine($"Integer number '{counter }' has a default value of 0");
                counter++;
                break;

            case intValue:
                sum += value;
                WriteLine($"Integer number '{counter }' has a value of {value}");
                counter++;
                break;

            case null:
                WriteLine($"Integer number '{counter }' is null");
                counter++;
                break;

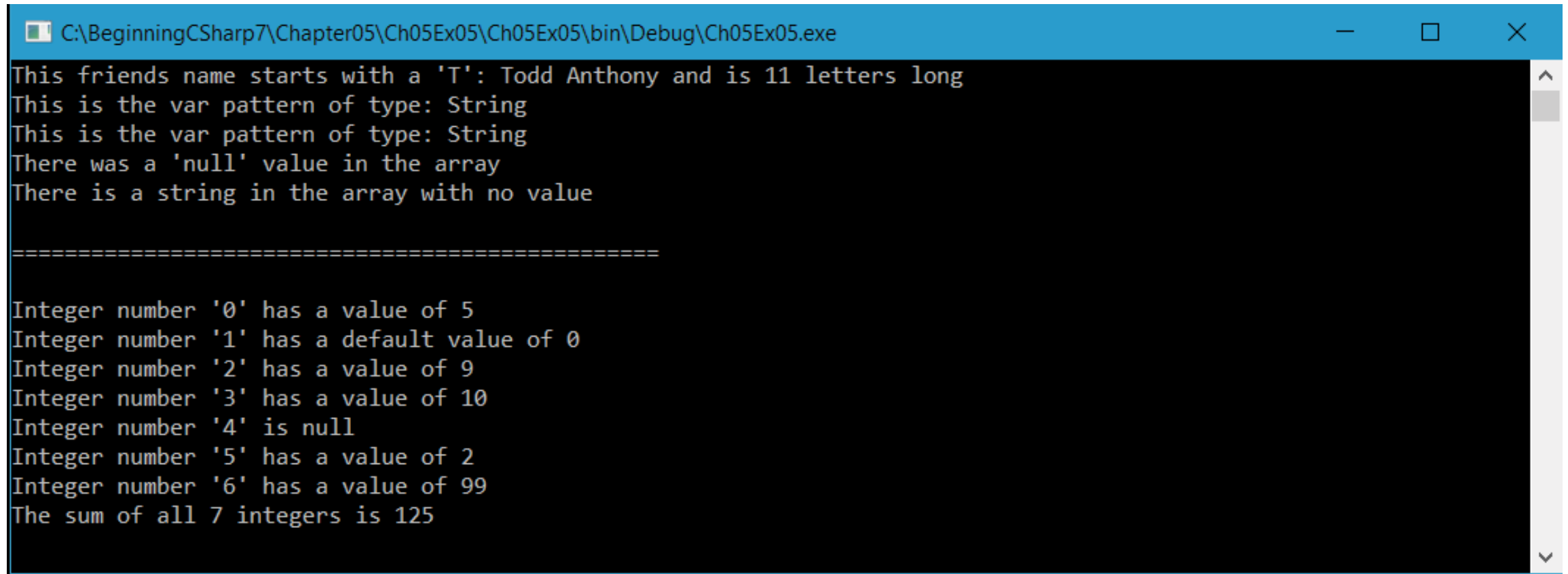
            default:
                break;
        }
    }

    ReadLine();
}

```

# Pattern Matching with switch case expression

After making sure you added using static System.Console;, execute the code. The result is shown in Figure: (last part)



```
C:\BeginningCSharp7\Chapter05\Ch05Ex05\Ch05Ex05\bin\Debug\Ch05Ex05.exe
This friends name starts with a 'T': Todd Anthony and is 11 letters long
This is the var pattern of type: String
This is the var pattern of type: String
There was a 'null' value in the array
There is a string in the array with no value

=====

Integer number '0' has a value of 5
Integer number '1' has a default value of 0
Integer number '2' has a value of 9
Integer number '3' has a value of 10
Integer number '4' is null
Integer number '5' has a value of 2
Integer number '6' has a value of 99
The sum of all 7 integers is 125
```

# Multidimensional Arrays

A multidimensional array is simply one that uses multiple indices to access its elements.

- A two-dimensional array such as this is declared as follows:

```
<baseType>[,] <name>;
```

- Arrays of more dimensions simply require more commas:

```
<baseType>[,,,] <name>;
```

- Declaring and initializing the two-dimensional array hillHeight, with a base type of double, an x size of 3, and a y size of 4 requires the following:

```
double[,] hillHeight = new double[3,4];
```



# Multidimensional Arrays

	Column 0	Column 1	Column 2	Column 3
Row 0	a[ 0 ][ 0 ]	a[ 0 ][ 1 ]	a[ 0 ][ 2 ]	a[ 0 ][ 3 ]
Row 1	a[ 1 ][ 0 ]	a[ 1 ][ 1 ]	a[ 1 ][ 2 ]	a[ 1 ][ 3 ]
Row 2	a[ 2 ][ 0 ]	a[ 2 ][ 1 ]	a[ 2 ][ 2 ]	a[ 2 ][ 3 ]

```
double[,] a = new double[3,4];
```

- Alternatively, you can use literal values for initial assignment. Here, you use nested blocks of curly braces, separated by commas:

```
double[,] a = { { 1, 2, 3, 4 }, { 2, 3, 4, 5 }, { 3, 4, 5, 6 } };
```

- To access individual elements of a multidimensional array, you simply specify the indices separated by commas:

```
a[2,1]
```

# Multidimensional Arrays

- The foreach loop gives you access to all elements in a multidimensional way, just as with singledimensional arrays:

```
double[,] hillHeight = { { 1, 2, 3, 4 }, { 2, 3, 4, 5 }, { 3, 4, 5, 6 } };  
foreach (double height in hillHeight)  
{  
    WriteLine($"{height}");  
}
```

hillHeight[0,0]

hillHeight[0,1]

hillHeight[0,2]

hillHeight[0,3]

hillHeight[1,0]

hillHeight[1,1]

hillHeight[1,2]

...

# Arrays of Arrays(*jagged* arrays)

- Multidimensional arrays, as discussed in the last section, are said to be *rectangular* because each “row” is the same size. Using the last example, you can have a y coordinate of 0 to 3 for any of the possible x coordinates.
- ***jagged* arrays** :whereby “rows” may **be varied sizes**. For this, you need an array in which each element is another array.  
You could also have arrays of arrays of arrays, or even more complex situations. However, all this is possible only if the arrays have the same base type.

# Arrays of Arrays(*jagged* arrays)

- The syntax for declaring arrays of arrays involves specifying multiple sets of square brackets in the declaration of the array, as shown here:

```
int[][] jaggedIntArray;
```

- Unfortunately, initializing arrays such as this isn't as simple as initializing multidimensional arrays. You can't, for example, follow the preceding declaration with this:

```
jaggedIntArray = new int[3][4];
```

- Even if you could do this, it wouldn't be that useful because you can achieve the same effect with simple multidimensional arrays with less effort. Nor can you use code such as this:

```
jaggedIntArray = { { 1, 2, 3 }, { 1 }, { 1, 2 } };
```

# Arrays of Arrays(*jagged* arrays)

- You have two options.
- 1. You can initialize the array that contains other arrays (let's call these subarrays for clarity) and then initialize the sub-arrays in turn:

```
int[][] jaggedIntArray = new int[2][];
```

```
jaggedIntArray[0] = new int[3];
```

```
jaggedIntArray[1] = new int[4];
```

- 2. Alternatively, you can use a modified form of the preceding literal assignment:

```
int[][] jaggedIntArray = new int[3][] { new int[] { 1, 2, 3 }, new int[] { 1 };  
new int[] { 1, 2 } };
```

- This can be simplified if the array is initialized on the same line as it is declared, as follows:

```
int[][] jaggedIntArray = { new int[] { 1, 2, 3 }, new int[] { 1 }, new int[] { 1, 2 } };
```

# Arrays of Arrays(*jagged* arrays)

- You can use foreach loops with jagged arrays, but you often need to nest these to get to the actual data.

```
int[][] divisors1To10 = { new int[] { 1 },  
                          new int[] { 1, 2 },  
                          new int[] { 1, 3 },  
                          new int[] { 1, 2, 4 },  
                          new int[] { 1, 5 },  
                          new int[] { 1, 2, 3, 6 },  
                          new int[] { 1, 7 },  
                          new int[] { 1, 2, 4, 8 },  
                          new int[] { 1, 3, 9 },  
                          new int[] { 1, 2, 5, 10 } };
```

The following code will fail:

```
foreach (int divisor in divisors1To10)  
{  
    WriteLine(divisor);  
}
```

# Arrays of Arrays(*jagged* arrays)

```
int[][] divisors1To10 = { new int[] { 1 },  
                          new int[] { 1, 2 },  
                          new int[] { 1, 3 },  
                          new int[] { 1, 2, 4 },  
                          new int[] { 1, 5 },  
                          new int[] { 1, 2, 3, 6 },  
                          new int[] { 1, 7 },  
                          new int[] { 1, 2, 4, 8 },  
                          new int[] { 1, 3, 9 },  
                          new int[] { 1, 2, 5, 10 } };
```

The following code will true:

```
foreach (int[] divisorsOfInt in divisors1To10)  
{  
    foreach(int divisor in divisorsOfInt)  
    {  
        WriteLine(divisor);  
    }  
}
```