Unit 2 – Understanding Arrays and Lists in C#

Using Arrays in C#

The Concept of an Array

- An ***array*** is a structure that allows you to use ***a single name*** to refer to a ***group of two or more variables***.
- To distinguish one variable in the group from another, a number, called the ***index*** or ***subscript***, is used.
- This concept is similar to the street address of a house. Each house on a given street is identified by the ***same street name***. However, each house also is identified by a ***unique number***, which makes it possible to locate any given house.
- For example, shown at the right is an overhead view of a portion of Centre Street North in Brampton. Since each house on this street is identified by a unique number, there is never any confusion distinguishing one house from another.
- Arrays are used whenever a program needs to process a group (usually a large group) of related data.
- Arrays help you to create shorter and simpler code in many situations because ***loops*** can be used to process the array elements efficiently, regardless of the size of the array.

Important Details about Arrays in C#

- All the elements in an array have the same data type.
- Because C# must allocate memory for each element of an array, avoid creating very large arrays.
- Arrays have both ***upper*** and ***lower*** bounds and the elements of the array are contiguous within those bounds. In C, C++, C# and a host of other languages derived from C, the ***lowest index is always zero***.
- If a program attempts to access an element of an array using an index that is either negative or greater than the upper bound, an "ArgumentOutOfRangeException" is thrown.
- Arrays can be thought of as ***fixed-size lists***. Once an array has been declared and initialized, the number of elements in the array remains ***fixed***.
- C# also provides support for ***Lists***, which can be thought of as ***variable-size arrays*** or ***dynamic arrays***. Lists are essentially arrays that can grow and shrink in size while a program is executing. Lists in C# are covered later in this unit.

Several Examples of Array Declarations

Number of elements in the array.

```
//Create a one-dimensional, empty array of "double"
//values. The elements of the array exist but
//they have not yet been assigned any values.

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

Index	0	1	2	3
Data	–	–	–	–

In this example, a variable of array type is ***declared***, an array object is created and storage space is allocated for the ***elements*** (also called ***components***) of the array. However, the elements of the array do not yet have values.

```
//Create and initialize an array of "double" values.
//Initial values are given in an initializer list.
//An initializer list is a set of values, separated
//by commas and enclosed in braces.

double[] temperature = new double[4] {0,2,4,6};
```

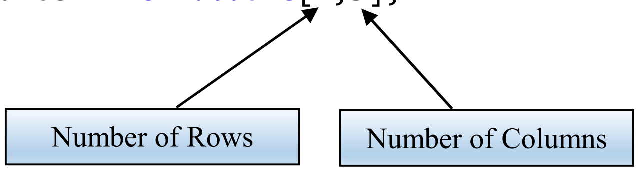
Index	0	1	2	3
Data	0	2	4	6

In C#, arrays are implemented as ***objects***. Therefore, the **new** keyword must be used in the declaration of an array to create a new array object. Note that array ***indices*** (singular ***index***, also called ***subscripts***) in C# always start at zero.

```
//Create an array of "string" values. No values
//have been assigned yet to the elements of the array.
string[] name = new string[4];
```

Index	0	1	2	3
Data	–	–	–	–

```
//The following declares a two-dimensional array called
//'distance.' It consists of two rows (horizontal) and
//3 columns (vertical). Its purpose is to store distances
//between points. As with other similar examples, the
//array elements have not yet been assigned values.
double[,] distance = new double[2,3];
```



Number of Rows

Number of Columns

	0	1	2
0	–	–	–
1	–	–	–

The statements shown at the left can be used to declare and create a ***two-dimensional*** array of `double` values. The row indices run from 0 to 1 and the column indices run from 0 to 2. Without any assignment statements, however, the two-dimensional array is empty (i.e. the elements have not yet been assigned any values).

```
//'distance[i,j]' stores the distance from point 'i' to
//point 'j.' For example, the distance from point 0 to
//point 1 is 10.7.
distance[0,0] = 0;
distance[0,1] = 10.7;
distance[0,2] = 25.3;
distance[1,0] = 10.7;
distance[1,1] = 0;
distance[1,2] = 16.3;
```

	0	1	2
0	0	10.7	25.3
1	10.7	0	16.3

Once the assignment statements at the left are executed, the two-dimensional array (also known as a ***matrix***) will contain the values shown above.

```
//Use an initializer list of initializer lists to initialize the
//two-dimensional array 'distance.'
double[,] distance = new double[2,3] { { 0, 10.7, 25.3 },
                                       { 10.7, 0, 16.3 } };
```

This statement is an alternative (and preferable) method of declaring, creating and initializing the two-dimensional array shown above. Each row of the matrix is enclosed in braces and listed in the desired order.

```
//A two-dimensional array used as a height map for an algorithm
//such as the "diamond-square" algorithm. For the sake of
//simplicity, the array is only 5x5. In reality, it would be
//much larger.
double[,] height = new double[5,5] { { 10, 0, 0, 0, 10 },
                                     { 0, 0, 0, 0, 0 },
                                     { 0, 0, 0, 0, 0 },
                                     { 0, 0, 0, 0, 0 },
                                     { 10, 0, 0, 0, 10 } };
```

We shall study the diamond-square algorithm in detail later in this unit.

Exercises involving Arrays

1. Create a memory map for each code segment. In addition, determine the problem that is solved in each case. (Some variables have intentionally been given silly names to disguise their purpose.)

Code Segment	*Memory Map (Trace Chart)*	*Problem Solved?*

`int[] a = { -1, 5, 3, -6, 3 };`
`int moe = a[0];`
 	
`for (int x = 1; x < a.Length; x++)`
`{`
`    if (a[x] < moe)`
`       moe = a[x];`
`}`
		By the time the loop has finished executing, the variable "moe" stores
		______________
		______________
		______________
		______________
		______________

`Random randomGenerator = new Random();`
`int[] a = new int[6];`
 	
`for (int i = 0; i < a.Length; i++)`
`{`
`    bool rep = false;`
`    int r;`
 	
`    do`
`    {`
`       r = randomGenerator.Next(1, 70);`
`       rep = false;`
 	
`       for (int j = 0; j < i; j++)`
`       {`
`          if (a[j] == r)`
`          {`
`             rep = true;`
`             break; //exit 'for' loop`
`          }`
`       }//end inner for`
 	
`    } while (rep);`
 	
`    a[i] = r;`
 	
`}//end outer for`
		By the time the outer for loop has finished executing, the array "a" stores
		______________
		______________
		______________
		______________
		______________
		______________
		______________
		______________
		______________
		______________
		______________
		______________
		______________
		______________
		______________

2. The following table lists answers to question 1. Check your answers to ensure that they are correct.

Code Segment	***Memory Map (Trace Chart)***	***Problem Solved?***

```
int[] a = { -1, 5, 3, -6, 3 };
int moe = a[0];

for (int x = 1; x < a.Length; x++)
{
    if (a[x] < moe)
        moe = a[x];
}
```

Data stored in the array "a."

Index	0	1	2	3	4
Data	–1	5	3	–6	3

x	moe
-	-1
1	-1
2	-1
3	-6
4	-6
-	-6

By the time the loop has finished executing, the variable "moe" stores **the smallest value stored in the array.**

```
Random randomGenerator = new Random();
int[] a = new int[6];

for (int i = 0; i < a.Length; i++)
{
    bool rep = false;
    int r;

    do
    {
        r = randomGenerator.Next(1, 70);
        rep = false;

        for (int j = 0; j < i; j++)
        {
            if (a[j] == r)
            {
                rep = true;
                break; //exit 'for' loop
            }
        }//end inner for

    } while (rep);

    a[i] = r;

}//end outer for
```

Since the given code produces ***random*** integers, it is not possible to predict exactly what will occur when the code is executed. The following is an example of what ***could happen***.

Array Index / i	0	1	2	3	4	5	r
-	-	-	-	-	-	-	-
0	27	-	-	-	-	-	27
1	27	3	-	-	-	-	3
2	27	3	51	-	-	-	51
3	27	3	51	-	-	-	3
3	27	3	51	-	-	-	16
4	27	3	51	16	-	-	27
4	27	3	51	16	-	-	51
4	27	3	51	16	42		42
5	27	3	51	16	42	9	9

Notice the numbers displayed in **red**. Since each of these numbers already occurred for a previous value of "i," a new value of "r" needs to be generated.

By the time the outer for loop has finished executing, the array "a" stores **six random integers ranging from 1 to 69, without repetition (i.e. each random integer is different from all the others).**

3. ***On paper***, write C# code to perform each of the following tasks. Do not use a computer for this question except for verifying that your code is correct.

(a) Find the ***largest value*** stored in an array.

(b) Find the ***average*** of the values stored in an array.

(c) Find the ***median*** of the values stored in an array.

(d) Copies the values stored in an array to another array. (Avoid this in practice because it uses a great deal of memory.)

(e) Fill an array of 52 elements with random integers ranging from 0 to 51 without repetition. (This is equivalent to shuffling a deck of 52 cards. Use a diagram to illustrate this.) See question 1 for a hint.

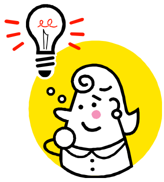
4. Write a C# program for a word "jumble" game (also known as word scramble). The user is given a word in "jumbled" form (the letters are randomly rearranged) and the user is given a limited number of guesses and/or a time limit to figure out the word. For example, if the user is given the string "bmejul," the correct answer would be "jumble."

ICS4U0 – Roman Converter Project

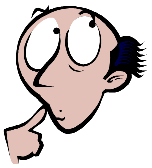
Roman to Hindu-Arabic Converter

Write a program that can convert a number expressed in Roman form to a number expressed in Hindu-Arabic form and vice versa. Your program must

- be able to convert any value from 1 to 3999999 from Hindu-Arabic to Roman or vice versa
- respond ***intelligently*** to ***any*** user input
- conform to the usual conventions of good coding



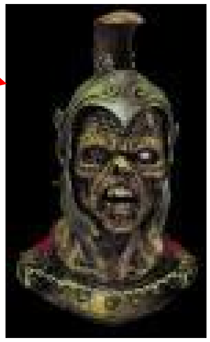
Before setting out to write Code, Consider this…

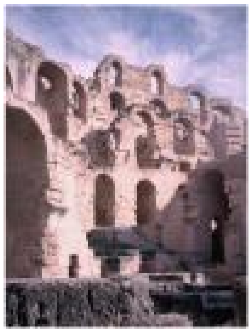


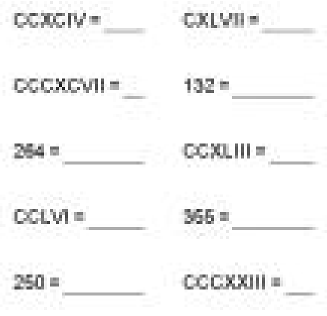
1. What are the rules for writing numbers using Roman numerals?
2. How can you design an *algorithm* that converts from Hindu-Arabic to Roman?
3. How can you design an *algorithm* that converts from Roman to Hindu-Arabic?
4. How are numbers greater than 3999 represented using Roman numerals?
5. How can you make your program recognize invalid values such as "XXMMMM?"

The **look**
on Mr. Nolfi's face
whenever…

1. …students install software or change computer settings without asking for permission!
2. …students try to write programs to solve problems that they do not know how to solve!







CCXCIV = ____

CCCXCVII = ____

284 = ____

CCLVI = ____

250 = ____

CXLVII = ____

132 = ____

CCXLIII = ____

365 = ____

CCCXXIII = ____

STOP! DO NOT WRITE ANY CODE YET! First we need to TRY SPECIFIC EXAMPLES and develop A PLAN!

The table below shows the basic “building blocks” of Roman numbers less than 4000 and their respective values. That is, any Hindu-Arabic number less than 4000 can be written as a Roman number that uses some combination of the symbols listed below. The best way to store the Roman symbols and their values is to use two arrays. (Keep in mind that in C, C++ and C#, ***array indices always begin at zero***. This is not the case in VB, where indices can range from any **Integer** value to any other **Integer** value.)

Index (Subscript) **Array Name**	0	1	2	3	4	5	6	7	8	9	10	11	12
romanSymbol	"M"	"CM"	"D"	"CD"	"C"	"XC"	"L"	"XL"	"X"	"IX"	"V"	"IV"	"I"
romanSymbolValue	1000	900	500	400	100	90	50	40	10	9	5	4	1

Hindu-Arabic to Roman Algorithm Example

Convert 1642 to Roman form.

Operation	*Remainder*	*Quotient*	*Roman String*
	1642	-	“”
÷1000	642	1	“M”
÷900	642	0	“M”
÷500	142	1	“MD”
÷400	142	0	“MD”
÷100	42	1	“MDC”
÷90	42	0	“MDC”
÷50	42	0	“MDC”
÷40	2	1	“MDCXL”
÷10	2	0	“MDCXL”
÷9	2	0	“MDCXL”
÷5	2	0	“MDCXL”
÷4	2	0	“MDCXL”
÷1	0	2	“MDCXLII”

Roman to Hindu-Arabic Algorithm Example

Convert “MCMXLIV” to Hindu-Arabic form.

i	*Character at Index i*	*Character at Index i+1*	*Operation*	*Hindu-Arabic Form*
-	-	-	-	0
0	“M”	“C”	+1000	1000
1	“C”	“M”	–100	900
2	“M”	X	+1000	1900
3	“X”	“L”	–10	1890
4	“L”	“I”	+50	1940
5	“I”	“V”	–1	1939
6	“V”	-	+5	1944

Hindu-Arabic to Roman Algorithm Pseudo-Code

```
store all possible one character and two
   character Roman symbol combinations in
   descending order in an array
store Hindu-Arabic values of above in descending
   order in another array
set roman to null string
set remainder to value of Hindu-Arabic number
for (i=0; i<number elements of array; i++)
{
  set quotient to quotient of remainder divided
    by element "i" of the array storing divisors
  set remainder to remainder of remainder
    divided by element "i" of the same array
  concatenate quotient Roman symbols (of type
    found at element "i" of Roman symbol array)
    to roman
}
```

Roman to Hindu-Arabic Algorithm Pseudo-Code

```
set len to length of the Roman number string
for (i=0; i<len; i++)
{
  set char to character at position "i"
  set value to Hindu-Arabic value of char
  if (i<len-1)
  {
    set nextChar to character at position "i+1"
    set valueNext to Hindu-Arabic value of nextChar
  }
  if (valueNext<=value)
    set HinduArabic to HinduArabic + value
  else
    set HinduArabic to HinduArabic - value
}
```

Exercises

Convert 2007 to Roman form.

Operation	*Remainder*	*Quotient*	*Roman String*

Convert “MCMXCVIII” to Hindu-Arabic form.

i	*Character at Index i*	*Character at Index i+1*	*Operation*	*Hindu-Arabic Form*

Convert 3999 to Roman form.

Operation	*Remainder*	*Quotient*	*Roman String*

Convert MMMCDXLIV” to Hindu-Arabic form.

i	*Character at Index i*	*Character at Index i+1*	*Operation*	*Hindu-Arabic Form*

Roman Converter Evaluation Guide

Victim: ______________________________

Categories	***Criteria***	***Descriptors***					***Mark***
		Level 4	Level 3	Level 2	Level 1	Level 0	
Knowledge and Understanding (KU)	**Degree of Completeness** □ be able to convert any value from 1 to 3999999 from Hindu-Arabic to Roman or vice versa	Very High (All features implemented)	High (Most features implemented)	Moderate (Some important features implemented)	Minimal (A few features implemented)	Insufficient (Little to nothing implemented)	20
Application (APP)	**Correctness** To what degree does the program produce correct output?	Very High	High	Moderate	Minimal	Insufficient	20
	Avoidance of Code Duplication To what degree has the student used methods (i.e functions) to avoid duplication of code? (i.e. to avoid copy & paste coding)	Very High	High	Moderate	Minimal	Insufficient	
	Data Validation and Exception Handling To what degree are exceptions caught and handled? To what degree can the program detect invalid input?	Very High	High	Moderate	Minimal	Insufficient	
Thinking, Inquiry and Problem Solving (TIPS)	**Independence** To what degree has the student been able to implement the solution ***without*** asking for assistance?	Very High	High	Moderate	Minimal	Insufficient	30
	Research When problems are encountered during the design, implementation and validation phases, to what degree has the student consulted resources ***before*** asking for help?	Very High	High	Moderate	Minimal	Insufficient	
	Algorithm/Implementation Efficiency □ To what level does the algorithm use resources (memory, processor time, etc) efficiently? □ To what degree are appropriate data types used?	Very High	High	Moderate	Minimal	Insufficient	
Communication (COM)	**Indentation of Code** **Insertion of Blank Lines in Strategic Places** (to make code easier to read)	Very Few or no Errors	A Few Minor Errors	Moderate Number of Errors	Large Number of Errors	Very Large Number of Errors	30
	Comments (Internal Documentation) □ Effectiveness of explaining abstruse (difficult-to-understand) code □ Effectiveness of introducing major blocks of code □ Avoidance of comments for self-explanatory code	Very High	High	Moderate	Minimal	Insufficient	
	Descriptiveness of Identifier Names Variables, Constants, Objects, Methods, Classes, etc **Method and Class Design** □ Methods are self-contained (can be used in other programs without modification) □ Parameters and return types are logical □ Class structure is logical and efficient **Clarity of Code** How easy is it to understand, modify and debug code? **Adherence to Naming Conventions** □ lowerCamelCase used for variable, object, methods □ UpperCamelCase used for classes and constructors □ ALL_UPPER_CASE used for constants	Masterful	Good	Adequate	Passable	Insufficient	