Tutorial - .NET Containers

In this exercise you will learn more about the .NET Containers and how to use them.

Introduction:

Containers in C# are provided by the .NET framework. There are a large range of containers which are available to use and it is important to choose the one which is most appropriate for the required task. We are going to go through some of the most useful ones, but you should conduct your own research into what is available and what are its relative advantages and disadvantages. All the abstract data types we have looked at in C++, and the STL containers, have equivalents in .NET but they have different names.

Arrays:

Static arrays are the simplest container type. Refer to the lecture notes for the syntax on how to create these. They are a fixed size and do not grow or shrink dynamically.

Exercise:

Add the following function to a class in your C# project:

```
public void ArrayExample(int size)
{
    int[] array = new int[size];
    for(int index = 0; index < array.Length; index++)
    {
        Console.WriteLine(array[index]);
    }
}
```

Compile and run it.

Note how the array size is a variable, unlike in C++ where it must be a constant, and the use of the Length property to get the length of the array.

1. Replace the **for** loop with a **foreach** loop. The syntax is provided in the lecture slides.
2. Now try and use the foreach loop to set the values in the array. What happens?
3. Add a second function in the class and pass the array into that function as a parameter. Again note how Length can be used to get the length of the array from within the second function. How and why is this different from C++?
4. Create an array of a user defined class, initialize the array, instantiate the classes and iterate over the array setting and display the class values to the console.
5. Arrays are much safer to use in C# than in C++ because the bounds are checked. Attempt to access an element which is outside the array bounds and note which error you get.
6. Create a copy of the array, change some values in the copy, and display the values in both arrays to prove that you have really created a copy and not just a reference to the original!

Questions:

1. What do you notice about the initial values of the elements in the array? What would happen if you tried the same thing in C++?
2. Does .NET allow you to specify a different initial value? Research this.

Lists:

Arrays are easy to use and a good general purpose container, but have some limitations. In C++ STL, the **Vector** class is a dynamic array implementation, and in .NET, the **List** class provides the same functionality. The implementation for the List container class is contained in the **System.Collections.Generic** namespace.

Note that all of the following container types are "generic" which means that they work with any type of variable – this is similar to templates in C++. For our purposes, we'll stick with integers because of it keeps the code simple, but you can replace **int** with float or any other type you want, including your own classes. The <> brackets are used to select the type when working with generics.

Exercise:

Add the following function to your class:

```
public void ListExample()
{
    List<int> myList = new List<int>();
    myList.Add(10);
    myList.Add(1);
    foreach (int item in myList)
    {
        Console.WriteLine(item);
    }
}
```

Note how we create an empty list (just like creating an empty STL Vector), then add elements to it before iterating over the list displaying the contents.

1. Step through the code in debug mode. What is the difference between ***Size*** and ***Capacity***?
2. Change the code so that when the list is created it is initialised to a list containing the values {3, 5, 2, 9}. Step through your code again and look at the values of *size* and *capacity*
3. Use an index to access elements in the list. Deliberately try to access an element which is outside the bounds of the list and note the result.
4. Lists are more versatile than arrays but are slower in some respects. .NET provides handy functions for converting from Lists to Arrays. Convert your list into an array.
5. Elements can be removed from Lists as well as added, remove the 4th and 5th elements from you list. (Hint: there are several member functions to remove elements, ***Remove***, ***RemoveAt*** and ***RemoveRange*** - either of the second two could be used here, when would you use the first one?)
6. Lists can be sorted. Use the built in ***Sort*** method to sort your list.

Questions:

1. The sort function is great for sorting lists of intrinsic types (for example, integers), but what if you had an array of a user defined class and you wanted to sort on a particular member of the class? Research how you would do this and create code to demonstrate it.
2. Do you need have the **using System.Collections.Generic** line of code in order to use lists? What is the difference between the *using* keyword and the *include* keyword in C++?
3. When would you use lists instead of arrays and vice versa?

Stacks:

Exercise:

Add the following code to your class and run it:

```
public void StackExample()
{
    Stack<int> myStack = new Stack<int>();
    myStack.Push(3);
    myStack.Push(1);
    Console.WriteLine(myStack.Pop());
    myStack.Push(2);
    while (myStack.Count > 0)
    {
        Console.WriteLine(myStack.Pop());
    }
}
```

Questions:

1. When would you use a stack rather than an array or a list?
2. Can you use an index to access an element in a stack?
3. Can you check the value of an element in a stack without popping it off the stack?
4. What is the difference between a stack and a queue?
5. Modify the example to use a queue instead of a stack.

Dictionaries:

Exercise:

Add the following code to your class and run it:

```
public static void DictionaryExample()
{
    Dictionary<int, string> myDictionary = new Dictionary<int, string>();

    myDictionary.Add(0, "Cat");
    myDictionary.Add(1, "Dog");
    myDictionary[5] = "Bird";

    Console.WriteLine(myDictionary[5]);
}
```

Questions:

1. What does the following code output?
2. How can we check to see if a value or key exists before attempting to use it in a dictionary?
3. Try to access a key which is has not yet been added. What error message do you get?
4. How can we use a loop to iterate through a Dictionary? Research and write some code to do this.

There are many, many more Containers in .NET, each with its own specific purpose, advantages and disadvantages. There isn't a "one size fits all" container (although some are more used than others), so be sure to research and experiment with others. Choosing the wrong container for a problem can have a huge performance impact, so be careful!