SIT771 – Lecture 5

Useful data structures and code evaluation

DEAKIN UNIVERSITY

Further reading

- Paul Deitel and Harvey Deitel (2018). Visual C# how to Program (6th ed). Pearson. Ebook on Deakin Library – Chapter 6, Chapter 8, and Chapter 9.

Outline

In this lecture...

- Array
- Generic collection classes
  - System.Collections.Generic
  - List
  - Dictionary
- Code evaluation
  - Hand tracing
  - Logging
  - Other techniques

ARRAY

Array.Motivation

You need to keep the names of **5** employees in a software solution, and do some processing on the names… How would you do this?

Array.Motivation

Solution 1

- Declare one variable per employee
- Will this work?
  - Yes, this will. This will work fine as the number of employees is small, i.e., only 5 employees

```
public class EmployeeInfo
{
    private string name_employee_1 = "Jane";
    private string name_employee_2 = "Jack";
    private string name_employee_3 = "Max";
    private string name_employee_4 = "Cloe";
    private string name_employee_5 = "Sarah";

    public void PrintNames()
    {
        Console.WriteLine("1st employee's name is " + name_employee_1);
        Console.WriteLine("2nd employee's name is " + name_employee_2);
        Console.WriteLine("3rd employee's name is " + name_employee_3);
        Console.WriteLine("4th employee's name is " + name_employee_4);
        Console.WriteLine("5th employee's name is " + name_employee_5);
    }

    public void ChangeNames()
    {
        name_employee_1 = name_employee_1.ToLower();
        name_employee_2 = name_employee_2.ToLower();
        name_employee_3 = name_employee_3.ToLower();
        name_employee_4 = name_employee_4.ToLower();
        name_employee_5 = name_employee_5.ToLower();
    }
}
```

Array.Motivation

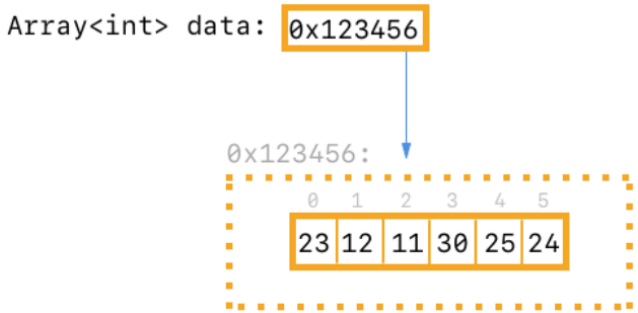
You need to keep the names of **100** employees in a software solution, and do some processing on the names… How would you do this?

Array

A collection of items

- An array is a special kind of **variable** that stores multiple values of the **same data type**:
  - E.g., [1, 30, 65, 4, 10]
  - E.g., [“jack”, “jane”, “max”, “cloe”]
  - E.g., [1.0, 2.54, 3.2, 78.8]

- An array is a **contiguous** area in memory, with the elements being next to each other, i.e., an array starts its first element at 0-th index.
- Note...
  - Arrays are **strongly typed**. This means that once you declare an array to hold a specific data type, it can only store elements of that exact type. In fact, C# is strongly typed throughout.
  - Arrays are objects (reference type).
- See the following link for more info on C# arrays: https://docs.microsoft.com/enus/dotnet/csharp/programming-guide/arrays/



Array

Declaration and access

- **datatype**[] arrayName = new **datatype**[*size*];
- **datatype**[,] arrayName = new **datatype**[*size1, size2*];
- *Random* access via indices, e.g.,
  - [i], element at index i, 1-D.
  - [i,j], element at row i and col j, 2-D.

```
17  public static void ProcessArrays()
18  {
19      int[] data;
20      //declares a variable that refers to an array
21
22      data = new int[5];
23      //creates a new array with 5 elements and stores it in data
24
25      data[0] = 40;
26      //assigns number 42 to the first element of array data
27
28      data[4] = 81;
29      //assigns number 42 to the fifth (last) element of array data
30
31      Console.WriteLine("First element: " + data[0]);
32      Console.WriteLine("Last element: " + data[4]);
33  }
34
```

```
C:\Users\bahadorreza\source\repos\S
First element: 40
Last element: 81
```

Array

Iteration

- The elements of an array can also be accessed and processed using indices and a **while** loop.

```
35  public static void ProcessArrays_v2()
36  {
37      int size;
38      int[] data;
39
40      size = Convert.ToInt32(Console.ReadLine());
41      data = new int[size];
42      Console.WriteLine("Array size set to " + size);
43
44      int i = 0;
45      while (i < size)
46      {
47          data[i] = Convert.ToInt32(Console.ReadLine());
48          i++;
49      }
50      Console.WriteLine("Array is now all set.");
51      Console.WriteLine("Here is what you have in the array...");
52
53      i = 0;
54      while (i < size)
55      {
56          Console.WriteLine(data[i]);
57          i++;
58      }
59  }
```

```
C:\Users\bahadorreza\source\repos\SIT771\SIT771\b
3
Array size set to 3
65
102
2
Array is now all set.
Here is what you have in the array...
65
102
2
```

Array

Iteration (cont.)

- The elements of an array can also be accessed and processed using indices and a **for** loop.

```
61  public static void ProcessArrays_v3()
62  {
63      int size;
64      int[] data;
65
66      size = Convert.ToInt32(Console.ReadLine());
67      data = new int[size];
68      Console.WriteLine("Array size set to " + size);
69
70      for (int i = 0; i < size; i++)
71      {
72          data[i] = Convert.ToInt32(Console.ReadLine());
73      }
74      Console.WriteLine("Array is now all set.");
75      Console.WriteLine("Here is what you have in the array...");
76
77      for (int i = 0; i < size; i++)
78      {
79          Console.WriteLine(data[i]);
80      }
81  }
```

C:\Users\bahadorreza\source\repos\SIT771\SIT771\

```
3
Array size set to 3
5
12
54
Array is now all set.
Here is what you have in the array...
5
12
54
```

Array

Summary info

- E.g., calculate the **sum** of all the elements in an array of integers.

size can be replaced by data.Length

```
83      public static void SumArray()
84      {
85          int size;
86          int[] data;
87
88          size = Convert.ToInt32(Console.ReadLine());
89          data = new int[size];
90          Console.WriteLine("Array size set to " + size);
91
92          for (int i = 0; i < size; i++)
93          {
94              data[i] = Convert.ToInt32(Console.ReadLine());
95          }
96          Console.WriteLine("Array is now all set.");
97
98          int sum = 0;
99          for (int i = 0; i < data.Length; i++)
100         {
101             sum += data[i];
102         }
103         Console.WriteLine("The SUM of the elements in the array = " + sum);
104     }
```

C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Deb

```
3
Array size set to 3
-25
25
100
Array is now all set.
The SUM of the elements in the array = 100
```

Array

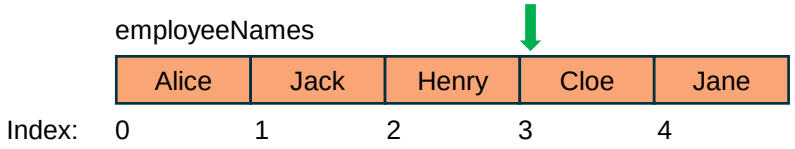
Linear search

- This is a sequential way of checking each element within an array to find a specific desired element. The implementation of linear search is **straightforward**; however, this search type can be **inefficient** for large arrays.
- There are better/alternative search algorithms…
  - Binary search
  - Jump search
  - Hash tables
  - And more!

```
106    public static void LinearSearchArray()
107    {
108        string[] employeeNames = { "Alice", "Jack", "Henry", "Cloe", "Jane"};
109        for (int i = 0; i< employeeNames.Length; i++)
110        {
111            if (employeeNames[i].ToLower().Trim() == "cloe")
112            {
113                Console.WriteLine($"Employee found at 0-based index {i}.");
114            }
115        }
116    }
117
118
119
```

C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe

```
Employee found at 0-based index 3.
```



Array.Motivation

You need to keep the names of **100** employees in a software solution, and do some processing on the names… How would you do this?

Array.Motivation

Solution 2

- Declare one array for all employees
- Will this work?
  - Yes, this will. This will work fine with any number of employees
  - Is this the best way you can have a set of employees and keep/access/manipulate their information?

```
public class EmployeeInfo_v2
{
    private static string[] _names = new string[] { "Jane", "Jack", "Max", "Cloe", "Sarah" };

    public static void PrintNames()
    {
        for (int i=0; i< _names.Length; i++)
            Console.WriteLine("Employee " + i + "'s name is " + _names[i]);
    }

    public static void ChangeNames()
    {
        for (int i = 0; i < _names.Length; i++)
            _names[i] = _names[i].ToLower();
    }
}
```

C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe

```
Employee 0's name is Jane
Employee 1's name is Jack
Employee 2's name is Max
Employee 3's name is Cloe
Employee 4's name is Sarah
```

GENERIC COLLECTION CLASSES

System.Collections.Generic

Overview

- The `System.Collections.Generic` namespace in C# provides a set of strongly typed collections that offer improved performance, type safety, and flexibility compared to their non-generic counterparts in `System.Collections`.
- Performance improvement with generic collections is due to no need for **boxing-unboxing** of items and their values. Boxing is the operation to convert any item to an object type and unboxing is the reverse action. These actions are needed with non-generic collections.
- We only cover two of these in detail…
  - List
  - Dictionary

Class	Description
Dictionary<TKey,TValue>	Represents a collection of key/value pairs that are organized based on the key.
List<T>	Represents a list of objects that can be accessed by index. Provides methods to search, sort, and modify lists.
Queue<T>	Represents a first in, first out (FIFO) collection of objects.
SortedList<TKey,TValue>	Represents a collection of key/value pairs that are sorted by key based on the associated IComparer<T> implementation.
Stack<T>	Represents a last in, first out (LIFO) collection of objects.

Source: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections#BKMK_Generic

List

A more flexible array

- A **List** object works like an array with some extra possibilities
  - You can **add** new elements to the list
  - You can **insert** elements into the list
  - You can **delete** elements from the list
  - You can **sort** the elements of the list (using a method)

```
C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe
This is the first element: 18.8
18.8, 0.25, 0.5,
```

```
public static void ListProcessing_v1()
{
    List<double> numbers;
    //declares a list of double values

    numbers = new List<double>();
    //creates a new empty list of double values

    numbers.Add(100.1);
    //adds 100.1 to the list

    numbers.Remove(100.1);
    //removes 100.1 from the list

    numbers.Add(21.03);
    //adds 21.03 to the list

    numbers.RemoveAt(0);
    //removes the first number in the list

    numbers.Insert(0, 18.8);
    //inserts 18.8 into the first position of the list

    Console.WriteLine("This is the first element: " + numbers[0]);

    for (int i = 1; i < 3; i++)
    {
        numbers.Add(i * 0.25);
    }

    foreach (double v in numbers)
    {
        Console.Write(v + ", ");
    }
}
```

Dictionary

A more complex list

- Dictionaries represent collections of **key-value** pairs. Keys can be used to access the elements.
- For more details, see: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-5.0

```
C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe
Selected student's name is student 1
```

```
142         public class Student
143         {
144             private string _name;
145             public string Name { get { return this._name; } }
146
147             public Student(string name)
148             {
149                 this._name = name;
150             }
151         }
152
153         public static void DictionaryProcessing()
154         {
155             Dictionary<string, Student> students;
156             //declares a variable (students) to refer to the dictionary object
157
158             students = new Dictionary<string, Student>();
159             //creates the dictionary object
160
161             students.Add("id123", new Student("student 1"));
162             //adds a new student object to the dictionary with a string id or key of id123
163
164             students["id456"] = new Student("student 2");
165             //adds a new student object to the dictionary with a string id or key of id456
166
167             Student _student = students["id123"];
168             //fetches the student object with the key of id123
169
170             Console.WriteLine("Selected student's name is " + _student.Name);
171         }
172     }
```

Dictionary

More examples…

- Note how dictionaries can be initialized, and their values can be accessed.

```
C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe
In mathematics, PI = 3.1415
In mathematics, Golden Ratio = 1.618
In mathematics, Euler's Number = 2.7182
The animal class of a dog is mammal
The animal class of a snake is reptile
```

```
277  public class SomeDictionaries
278  {
279      private static Dictionary<string, string> _animalClasses = new Dictionary<string, string>()
280      {
281          {"dog", "mammal" },
282          {"frog", "amphibian" },
283          {"snake", "reptile" },
284      };
285
286      private static Dictionary<string, double> _mathFamousNumbers = new Dictionary<string, double>()
287      {
288          {"PI", 3.1415 },
289          {"Euler's Number", 2.7182 },
290          {"Golden Ratio", 1.6180 },
291      };
292
293      public static void AccessDictionaryItems()
294      {
295          Console.WriteLine("In mathematics, PI" + " = " + _mathFamousNumbers["PI"]);
296          Console.WriteLine("In mathematics, Golden Ratio" + " = " + _mathFamousNumbers["Golden Ratio"]);
297          Console.WriteLine("In mathematics, Euler's Number" + " = " + _mathFamousNumbers["Euler's Number"]);
298
299          Console.WriteLine("The animal class of a dog is " + _animalClasses["dog"]);
300          Console.WriteLine("The animal class of a snake is " + _animalClasses["snake"]);
301      }
302  }
```

CODE EVALUATION

Hand tracing

AKA desk checking

- A manual (non-computerized) checking of the logic of the program/algorithm
  - To check if the logic is correct
  - To check if expected results can be achieved

- The person acts like the computer to run through the lines of code. It is best to have a workable and ready set of answers to check the program against. The procedure can be implemented using a table…
  - Have line numbers so you can identify each line of code
  - Have a condition column to keep track of different conditions
  - Have a variable column to track the values assigned to each variable under each condition
  - Have an input/output column to track the user inputs and program outputs

Hand tracing

Example...

- Calculate price/discount (**note:** the code is not written in C#)

```
1 calcPrice()
2     Input price
3     IF price > 100 THEN
4         discount = price * 15 / 100
5         price = price - discount
6     ENDIF
7     Display price
8 STOP
```

Source of images: https://sites.google.com/a/campioncollege.com/it_eveningschoool/problem-solving-and-programming/desk-check-guide

Desk Check

Inputs: price = $200
Correct results: price = $170.

Line Number	discount	price	Conditions	Input/Output
1				
2		200		price ? 200
3			200 > 100 ? is T	
4	200 * 15 / 100 = 30			
5		200 - 30 = 170		
6				
7				price = 170
8				

Inputs: price = $50
Correct results: price = $50.

Line Number	discount	price	Conditions	Input/Output
1				
2		50		price ? 50
3			50 > 100 ? is F	
6				
7				price = 50
8				

Logging

AKA printf debugging

- In many cases, you need to locate a bug or an issue without knowing exactly what causes the issue or where the bug is located in the code…
  - Logging is the process of placing additional code within your program to log the content of specific variables and monitor the output progress.
  - **Example:** You want to compute the factorial of a number. Recall that factorial(0) = 1 and factorial(n>0) = n x (n-1) x ... x 1.

The code below is not complete (i.e., will not stop) as it does not consider the base case of *n*=0. What exception do you expect?

```
public static int Factorial(int n)
{
    return n * Factorial(n - 1);
}
```

Adding this line aims to trace the value of *n* and you will find the program keeps calculating even when *n*=0 and negative.

```
public static int Factorial(int n)
{
    Console.WriteLine(n);
    return n * Factorial(n - 1);
}
```

C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe

```
-6046
-6047
-6048
-6049
-6050
-6051
-6052
-6053
-6054
-6055
```

Other techniques

Common (in addition to HT and Logging)

- Debugger
  - To make use of the debugging tools within the specific programming toolkit
  - Strategies such as breakpoints, step-in, and step-over
- Visualization method
  - To use flowcharts or diagrams for tracing and debugging code
  - Visualize control flow
  - Explain visually to other team members
- Mental tracing
  - To visualize code execution in mind!
  - For experienced developers and simple code
  - Error-prone for the less experienced and complex code logic and structures

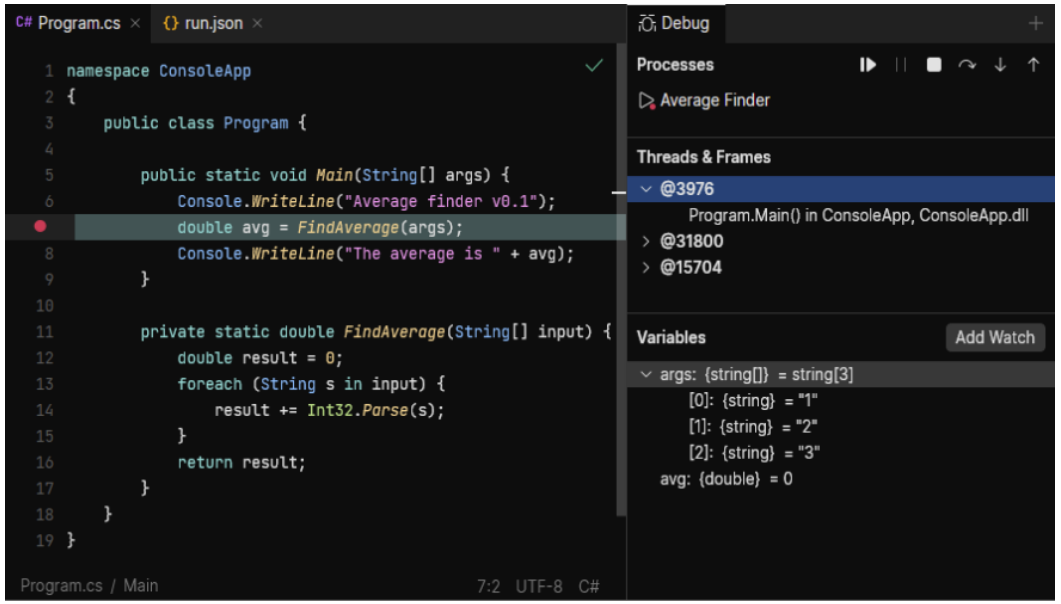


Image source: C# debugging walkthrough | JetBrains Fleet Documentation

Epilogue

BAD PROGRAMMERS WORRY ABOUT THE CODE. GOOD PROGRAMMERS WORRY ABOUT DATA STRUCTURES AND THEIR RELATIONSHIPS...

LINUS TORVALDS