LAB: LOOP

BME 121, Fall 2016

Rasoul Nasiri

Agenda:

- Simple loop
- Nested loop
- 1D array
- Rolling dice
- While loop
- Nested while loop
- Array
- WA3
- Array as input to method

Practice with simple loop:

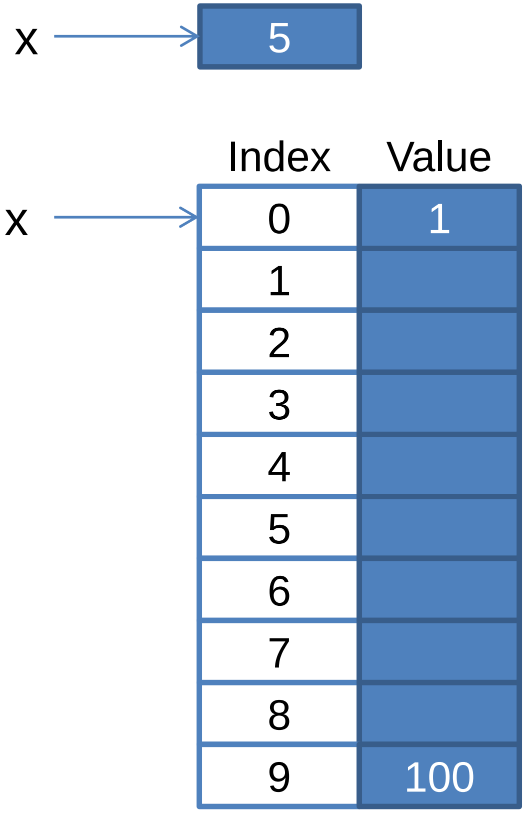
- Write a program that ask user for 10 numbers and return the min
- Write a program that generate 100 random number and print the max, min, and average

```
Random r1 = new Random();
r1.NextDouble();
```

Code: min10.cs, min10_2.cs, and minRandom.cs

1D Arrays

- Arrays are a way to have an organized collection of variables of the same type
- Sequence of memory locations reserved to keep a set of values with a specific type(double, int, bool, …).
- Each memory location is accessible by an index.



Index	Value
0	1
1	
2	
3	
4	
5	
6	
7	
8	
9	100

1D Arrays

- Definition

```
int[] x; // just saying that x is an array of integer. No creation
```

- Defintion and Creation

```
int[] x = new int[10]; // make a 1D array of 10 integers
double[] y = new double[20]; // make a 1D array of 20 doubles
```

- Assign value to an index (first element of an array is always at index 0)

```
x[0] = 65; // store value of 65 in index 0. 0 <= Index <= length-1
```

- Accessing (reading) value in an index

```
  int a = x[0];
```

- *array*.Length gives us the number of variables in the array

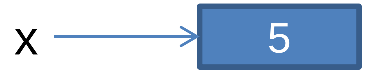
```
Console.Write(x.Length); // shows 10
```

- Last element of an array is always index (Length – 1), 9 in this case

```
x[9] = 100;
```

1D Arrays

```
int x = 5;
```

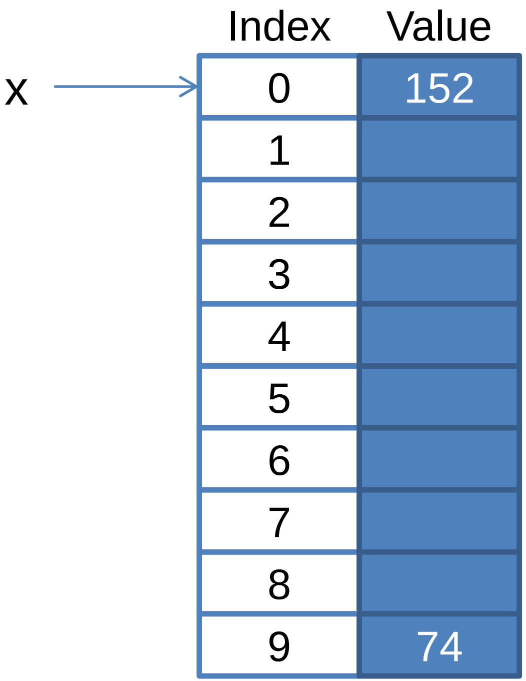


```
int[] x = new int[10];
x[0] = 152;
x[9] = 74;
```

Note:

x refers to the entire array

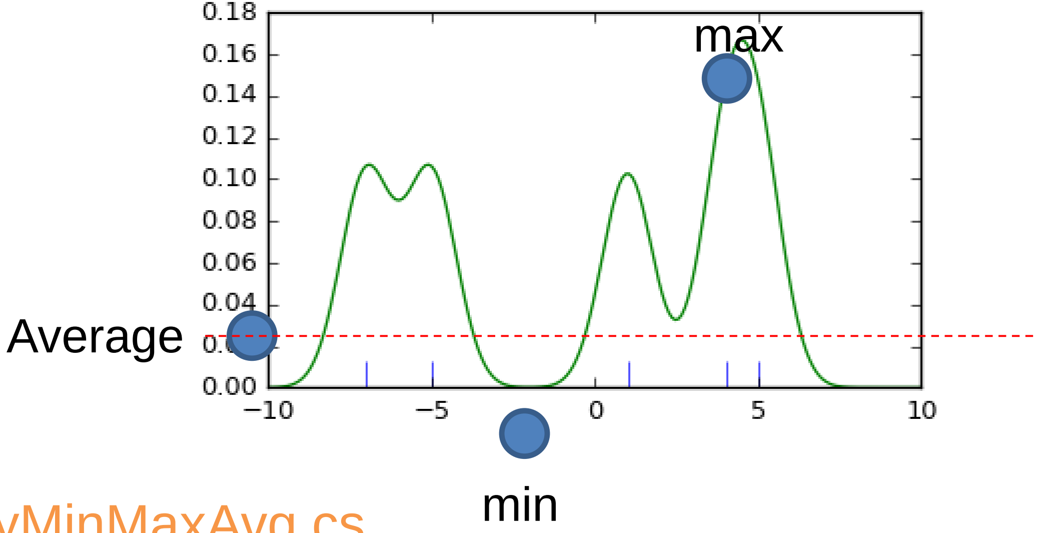
x[index] refers to one particular element in the array



Index	Value
0	152
1	
2	
3	
4	
5	
6	
7	
8	
9	74

For Loop and Array – Find min and max

- Create an array of double values with length 100.
- Fill it with random values in one loop
- Find min, max, and average using another loop



Code: arrayMinMaxAvg.cs

For Loop and Array – rolling a dice

- Simulate rolling a dice
- The output of rolling could be between 1 and 6
- Let's roll 100 times and count the number of times you see each number in a 1D array of length 6. (Number of 1's would be in index 0, 2's in index 1, 3 index 2, and so on.)
- Calculate the percentage of observations for each of 6 values
- What would happen if we increase 100 to 1000, or 1000000?

rollingDice.cs

For Loop : controlling input

- Ask user to enter an even number and continue until you get an even number

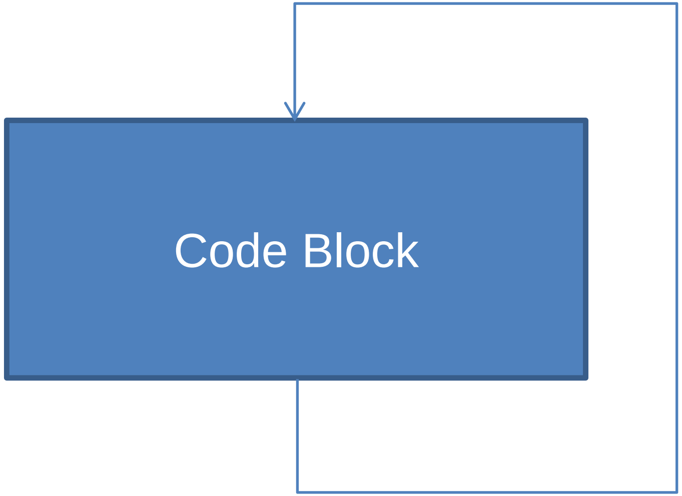
```
for(int i = 0; i < 100; i++)
{
    Write(“Please enter an even number : ”);
    if(int.Parse(ReadLine()) % 2 == 0)
    {
      Write("We have an even");
      i = 100; // break
    }else
    {
       Write(“The number is not correct.");
    }
}
```

How many times we have to read? 1, 10, 100, …?

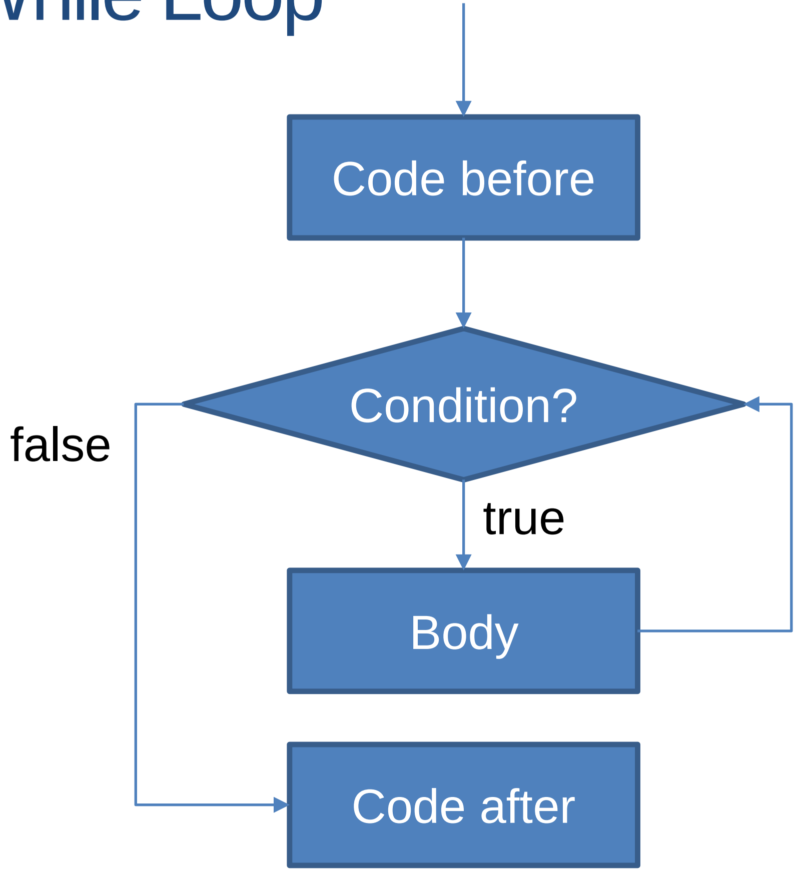
Code: userInputFor.cs

Programming Loops

- A language feature that allows a programmer to tell the computer to **perform a certain (set of) instruction(s) over and over again**
- Machines are very efficient at repetitive labour!



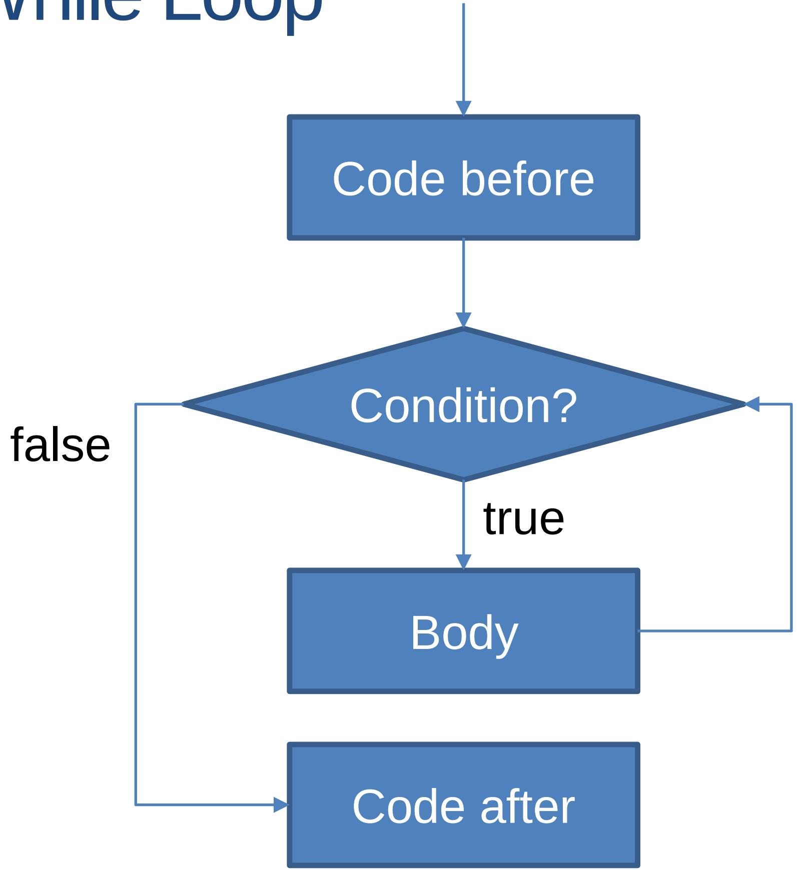
While Loop



- While Condition is true, keep executing the code in Body

```
// Code before while loop
while(Condition)
{
  // Body
}
// Code after while loop
```

While Loop



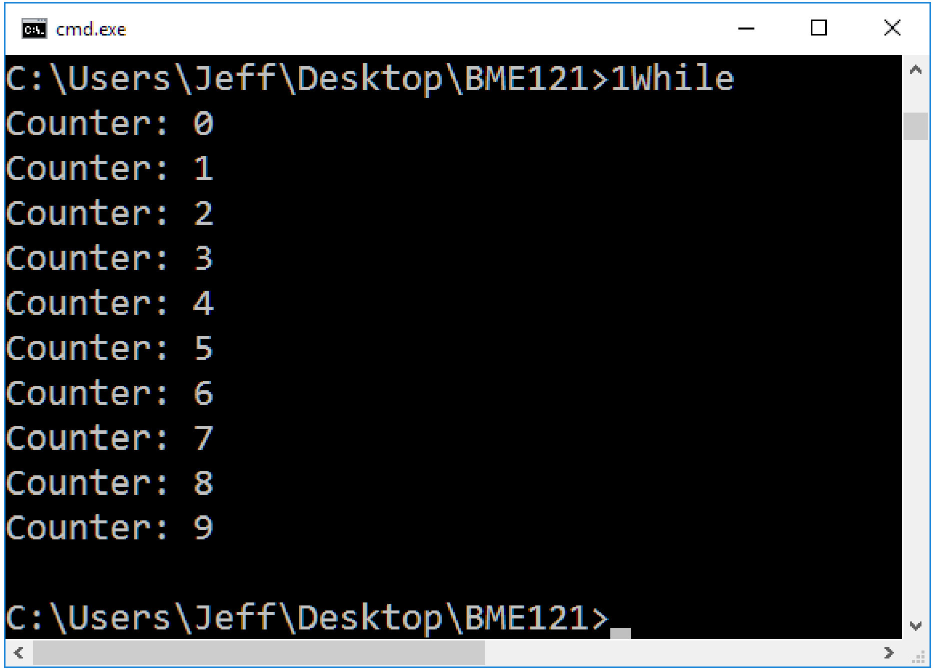
- Also known as a Pretest Loop: condition is checked **before** each iteration of the loop
- If condition == false the first time it is tested, the instructions in the loop body will never execute
- If the condition is **true** all the time, then the loop will never stop executing
    - Make sure it becomes **false** at some point

While Loop – Example: Count from 0 to 9

```
// Declare a counter, with initial value 0
int n = 0;

// Declare the while loop, with end
condition
while(n < 10)
{
   // Display current value of counter
   Console.WriteLine(“Counter: {0}", n);

   // Increment counter
   n++;     // same as n = n + 1;
}
```



```
C:\Users\Jeff\Desktop\BME121>1While
Counter: 0
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
Counter: 6
Counter: 7
Counter: 8
Counter: 9

C:\Users\Jeff\Desktop\BME121>
```

BREAK ….

While Loop – controlling input

- Ask user to enter an even number and continue until you get an even number

```
for(int i = 0; i < 100; i++)
{
    Write(“Please enter an even number : ”);
    if(int.Parse(ReadLine()) % 2 == 0)
    {
      Write("We have an even");
      i = 100; // break
    }else
    {
       Write(“The number is not correct.");
    }
}
```

Code : 7.userInputWhile.cs

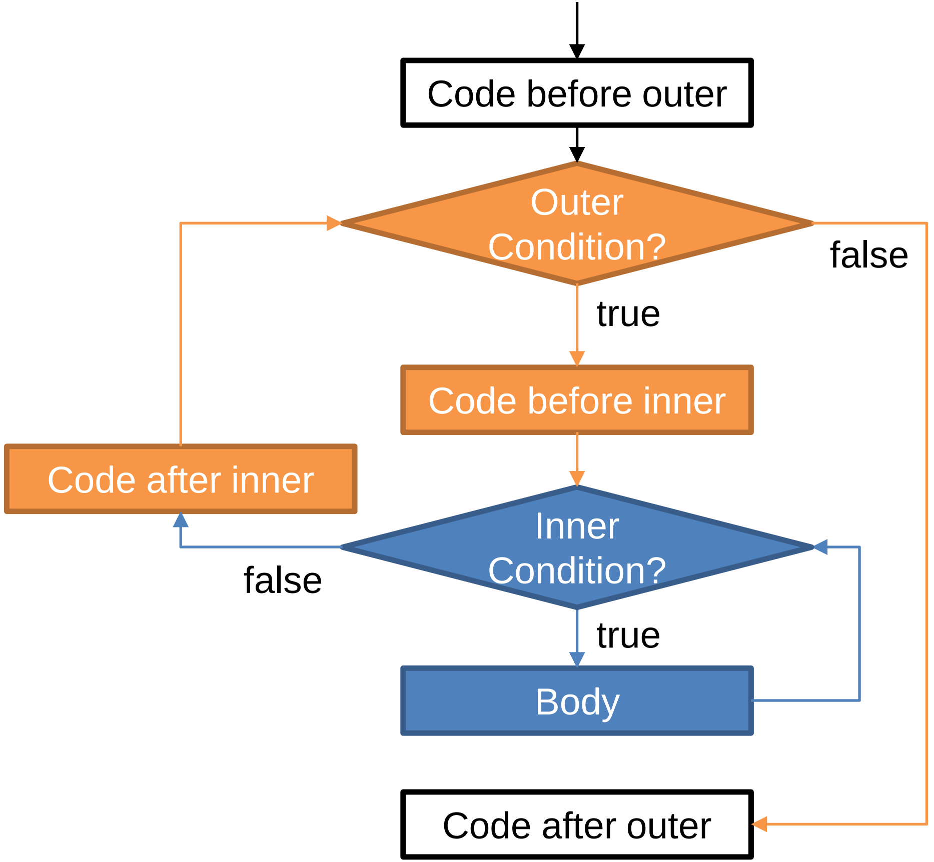
- While Condition is true, keep executing the code in Body

```
// Code before while loop
while(Condition)
{
  // Body
}
// Code after while loop
```

Which Loop to Use?

- For
  - Iterate for an exact number of times
  - Used when we know ahead of time how many repetitions we need
- While
  - If Condition is False from the beginning, do not iterate the loop
  - Used when we don’t know how many repetitions we need, but we know when to begin or stop repeating

Nested While Loops



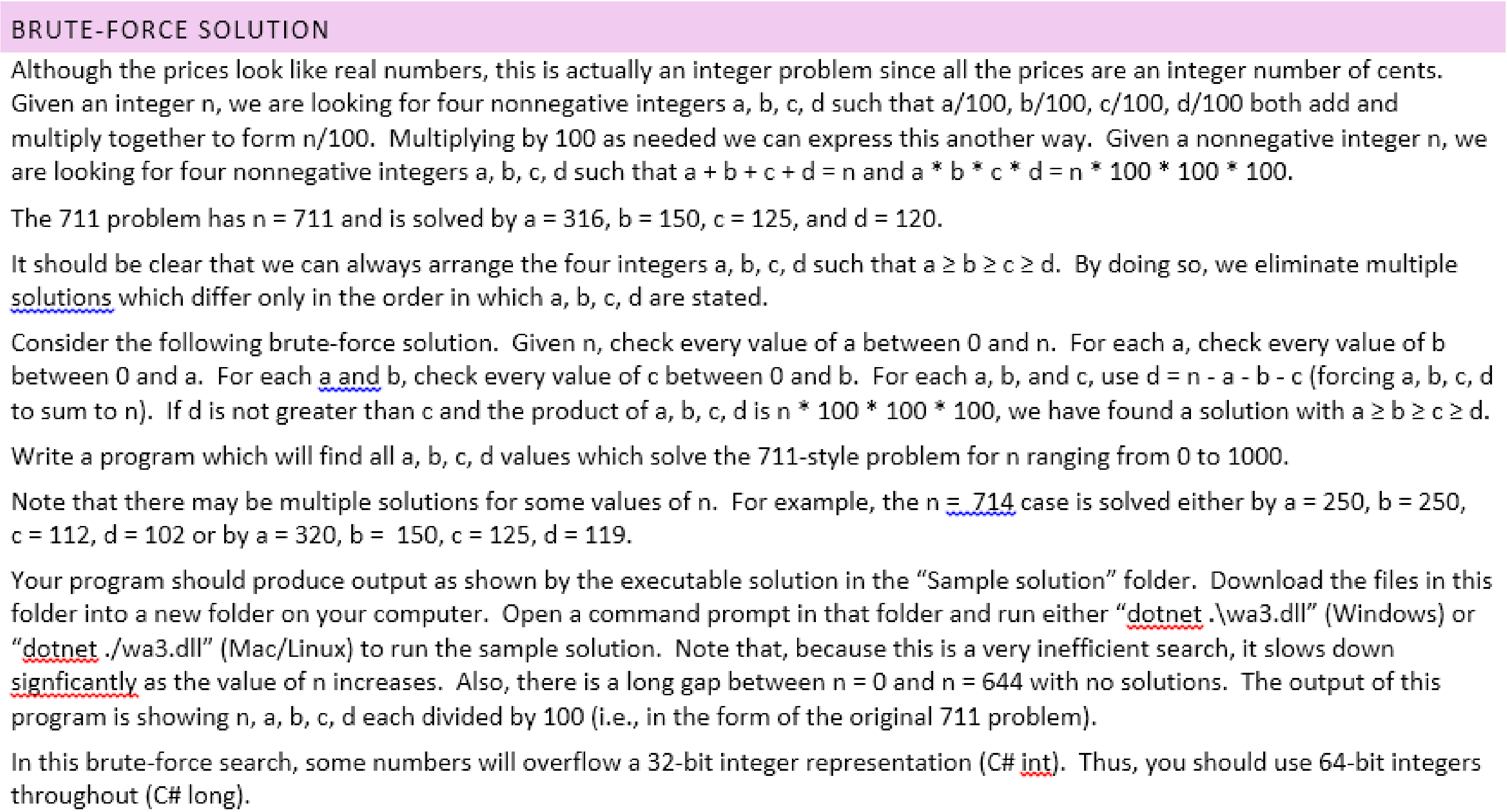
```
// Code before outer loop
while(OuterCondition)
{
    // Code before inner loop
    while(InnerCondition)
    {
        // Body
    }
    // Code after inner loop
}
// Code after outer loop
```

Practice – Cashier’s Tilt

- Using two nested while loops, create a software simulating a cashier’s tilt:
  - Inner loop: repeatedly ask for item prices until any letter is entered, then calculate the subtotal, tax, and total and show them on the screen
    - Hint:
      - `int n;`
      - `bool isNumeric = int.TryParse("123", out n);`
    - This will attempt to convert “123” into an integer, the bool stores whether it was successful, where true = success. If it is successful, n stores the converted value.
  - Outer loop:
    - Ask cashier if he/she want to continue.
    - if the cashier enters letter Q, quit the program;
    - Else go to the next customer
  - Bonus Challenge: also calculate and display the total sales by the cashier just before quitting the program

Code: cashier.cs

WA3



BRUTE-FORCE SOLUTION

Although the prices look like real numbers, this is actually an integer problem since all the prices are an integer number of cents. Given an integer n, we are looking for four nonnegative integers a, b, c, d such that a/100, b/100, c/100, d/100 both add and multiply together to form n/100. Multiplying by 100 as needed we can express this another way. Given a nonnegative integer n, we are looking for four nonnegative integers a, b, c, d such that a + b + c + d = n and a * b * c * d = n * 100 * 100 * 100.

The 711 problem has n = 711 and is solved by a = 316, b = 150, c = 125, and d = 120.

It should be clear that we can always arrange the four integers a, b, c, d such that a ≥ b ≥ c ≥ d. By doing so, we eliminate multiple solutions which differ only in the order in which a, b, c, d are stated.

Consider the following brute-force solution. Given n, check every value of a between 0 and n. For each a, check every value of b between 0 and a. For each a and b, check every value of c between 0 and b. For each a, b, and c, use d = n - a - b - c (forcing a, b, c, d to sum to n). If d is not greater than c and the product of a, b, c, d is n * 100 * 100 * 100, we have found a solution with a ≥ b ≥ c ≥ d.

Write a program which will find all a, b, c, d values which solve the 711-style problem for n ranging from 0 to 1000.

Note that there may be multiple solutions for some values of n. For example, the n = 714 case is solved either by a = 250, b = 250, c = 112, d = 102 or by a = 320, b = 150, c = 125, d = 119.

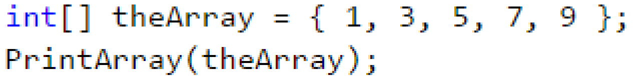
Your program should produce output as shown by the executable solution in the “Sample solution” folder. Download the files in this folder into a new folder on your computer. Open a command prompt in that folder and run either “dotnet .\wa3.dll” (Windows) or “dotnet ./wa3.dll” (Mac/Linux) to run the sample solution. Note that, because this is a very inefficient search, it slows down signficantly as the value of n increases. Also, there is a long gap between n = 0 and n = 644 with no solutions. The output of this program is showing n, a, b, c, d each divided by 100 (i.e., in the form of the original 711 problem).

In this brute-force search, some numbers will overflow a 32-bit integer representation (C# int). Thus, you should use 64-bit integers throughout (C# long).

arrayMethod.cs

Array and method

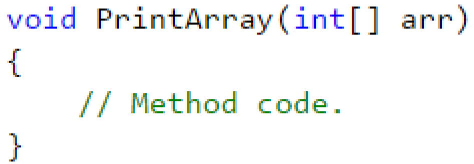
- Method working with array



```
int[] theArray = { 1, 3, 5, 7, 9 };
PrintArray(theArray);
```

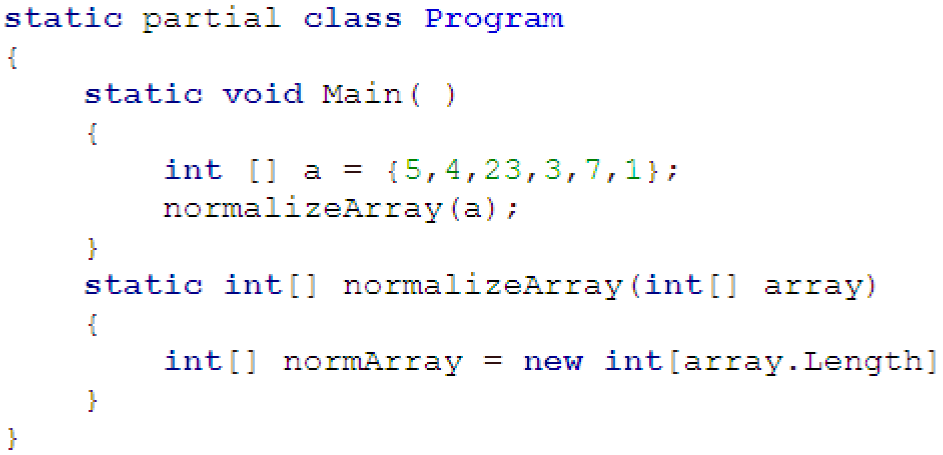
⋮

- Array as an input to method



```
void PrintArray(int[] arr)
{
    // Method code.
}
```

- Array as output of a method



```
static partial class Program
{
    static void Main( )
    {
        int [] a = {5,4,23,3,7,1};
        normalizeArray(a);
    }
    static int[] normalizeArray(int[] array)
    {
        int[] normArray = new int[array.Length]
    }
}
```

Code : arrayMethod.cs

Array and method

- Write three methods that receive and array of integers and return :
  - Print
  - Min
  - Average
  - Normalized array

THE END …