CptS 422

Junhao Zhang “Freddie”

ID# 11356533

HW 14

1. The ***int j*** does not help looping and sorting the elements in the data structure.

   Solution:

```
private static void Q1()
     {
         // Generate an array of random values
         int[] nums = new int[50];
         for (int i = 0; i < nums.Length; i++)
         {
             nums[i] = sr_rand.Next(100);
         }

         // Sort using insertion sort
         for (int i = 1; i < nums.Length; i++)
         {
             int j = i;
             while (j > 0 && nums[j] < nums[j - 1])
             {
                 int temp = nums[j];
                 nums[j] = nums[j - 1];
                 nums[j - 1] = temp;

                 --j;
             }
         }

         // Display the results
         Console.Write("Q1: ");
         foreach (int num in nums) { Console.Write(num + " "); }
         Console.WriteLine();
     }
```

2. “MethodInfo mi” is null because 'GetMethod' only takes 'public' method with no further attributes required

3. `float f1 = 4.25f;`

   The right value is a 'float' and 'float' type which is 4 bytes has enough memory for it, hence this assignment will not have data loss occur.

```
double d1 = f1;
```

Double is 8 bytes, and float is 4 bytes, there will be enough space for it, since base Ten number 4.25 can be converted to 01000000100010000000000000000000 binary base based on IEEE 754, which will not have data loss.

```
float f2 = 1.4f;
```

The right value is a 'float' and 'float' type which is 4 bytes has enought memory for it, hence this assignment will not have data loss occur

```
double d2 = f2;
```

Base Ten number 1.4f can be converted to 0011111110110011001100 1100110011 in binary based on the IEEE 754, which has data loss

4. 'object' is just a block of memory. When we created a reference type from an integer value, it means that the memory that represents '41' had been assigned to the object.However, when we are doing num++, it mean the new value had be assigned to num after the calculation.

5. Based on the description on MSDN of "string.StartsWith", for the string such as: firstLine="GETS xxxx" can also be valid, but in fact it is not.

6.
```
int w = a + b;
```

Since a and b can be any integer from int.MinValue to int.MaxValue, this operation can cause data loss. One of the Example is w = int.MaxValue + int.MaxValue.

```
int x = a - b;
```

Since a and b can be any integer from int.MinValue to int.MaxValue, this operation can cause data loss. One of the Example is x = int.MinValue - int.MinValue.

```
int y = a * b;
```

*Since a and b can be any integer from int.MinValue to int.MaxValue, this operation can cause data loss. One of the Example is y = int.MaxValue * int.MaxValue which needs 8 bytes to store.*

```
int z = a / b;
```

Since a and b can be any integer from int.MinValue to int.MaxValue, this operation can cause data loss. One of the Example is z = int.MinValue / int.MinValue which needs 8 bytes to store.

7. The method1 is better than others, because both method2 and method3 opens the file to read, and did not close it in time, which could cause the potential problems such as but not limited to:
   1. if some process is reading/writing/appending the file, or
   2. after checked the existence of the file some other process needs to read or write it but it was not closed.

8. Based on the definition on MSDN, dynamic is not a type, even compiler does not know what type it should be, which means the "node" can be replaced by any objects of any types, and potentially those "node" could become object in other types also. Since we are building a link-list,we should use the "Node " or certain class type for each node in the link-list, but using 'dynamic' will have the potential risk of losing the nodes.

9.
```
Interlocked.Increment(ref s_sharedValue);
```

   It is thread safe because when multiple threads want to access one shared object, the 'Interlocked' can lock the shared object with the thread that is accessing this object, and let the other threads wait until done.

10.
```
string stringValue = "\0";
```

   It can let Console.Write throw exception since its code is not a valid symbol. I believe there are more like that.