(T2)討論 LinqToObject 的 Aggregate，包括 Min、Max、Sum、Count、Average
CourseGUID: 5ba9a6fe-7475-4b0c-8b99-bbcf7f5e2e1c
==
(T2)討論 LinqToObject 的 Aggregate，包括 Min、Max、Sum、Count、Average
==

==

0. Summary

1.
There are 2 ways to write LINQ queries.
```
int[] intArr = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
```
1.1.
Using SQL like query expressions
```
// IEnumerable<int> greaterThanFiveV1 =
//     from intItem in intArr
//     where intItem >= 5
//     select intItem;
```
1.2.
Using Lambda Expressions.
```
// IEnumerable<int> greaterThanFiveV2 = intArr.Where(intItem => intItem >= 5);
```

==

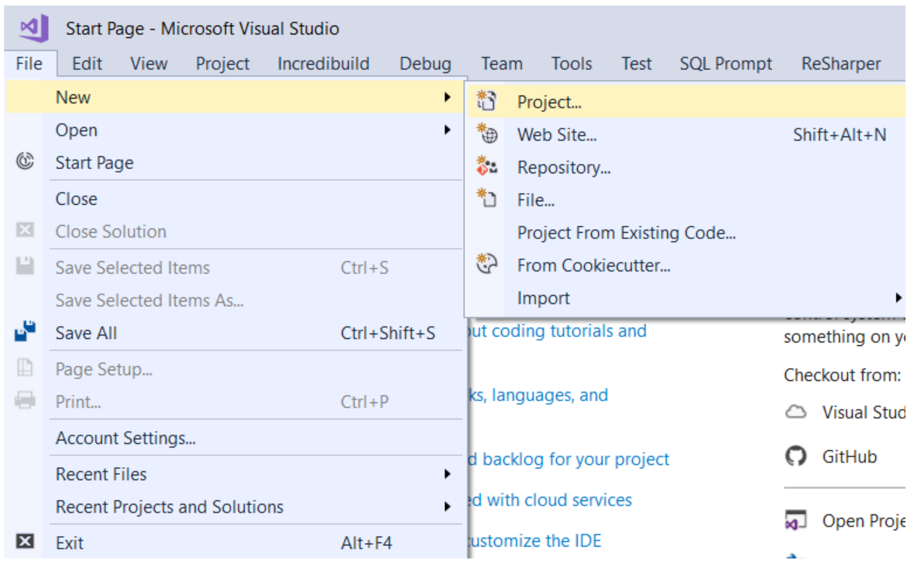
1. New Project

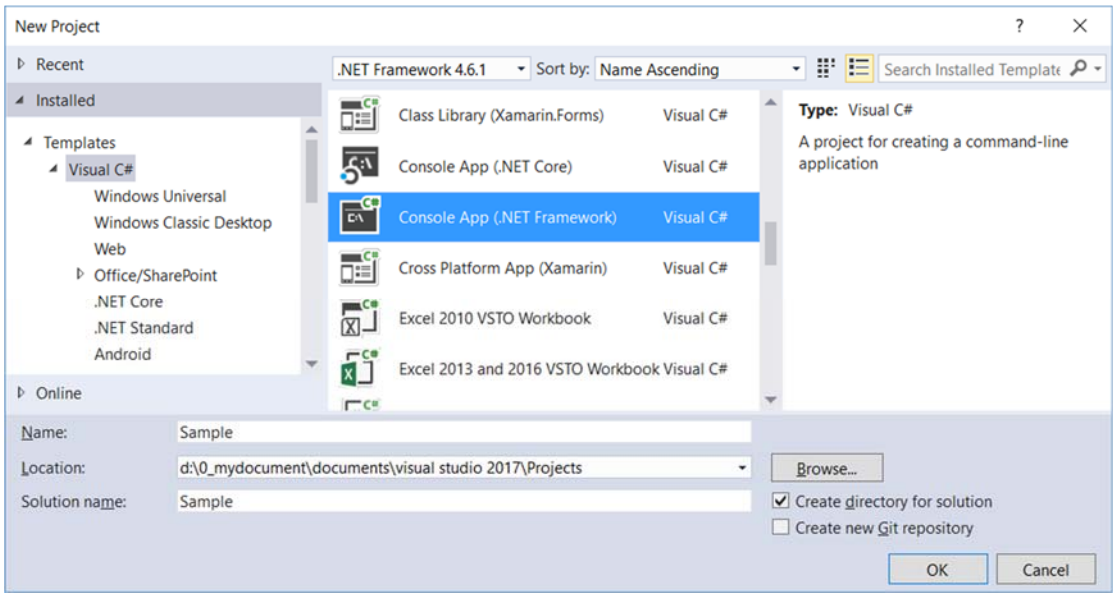
1.1. Create New Project : Sample

File --> New --> Project... -->

Visual C# --> **Console App (.Net Framework)** -->

Name: **Sample**




New Project
Recent
Installed
Templates
Visual C#
Windows Universal
Windows Classic Desktop
Web
Office/SharePoint
.NET Core
.NET Standard
Android
Online
.NET Framework 4.6.1
Sort by: Name Ascending
Search Installed Template
Class Library (Xamarin.Forms) Visual C#
Console App (.NET Core) Visual C#
Console App (.NET Framework) Visual C#
Cross Platform App (Xamarin) Visual C#
Excel 2010 VSTO Workbook Visual C#
Excel 2013 and 2016 VSTO Workbook Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for creating a command-line application

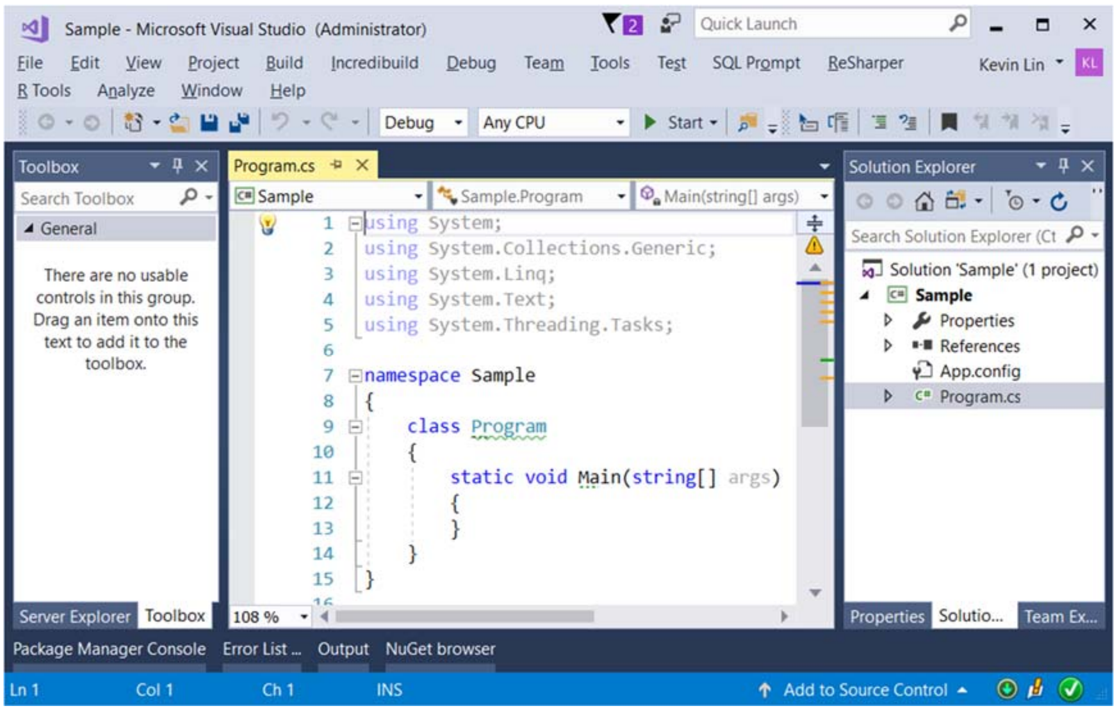
Name: Sample

Location: d:\0_mydocument\documents\visual studio 2017\Projects

Solution name: Sample

[✓] Create directory for solution

[ ] Create new Git repository



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

===

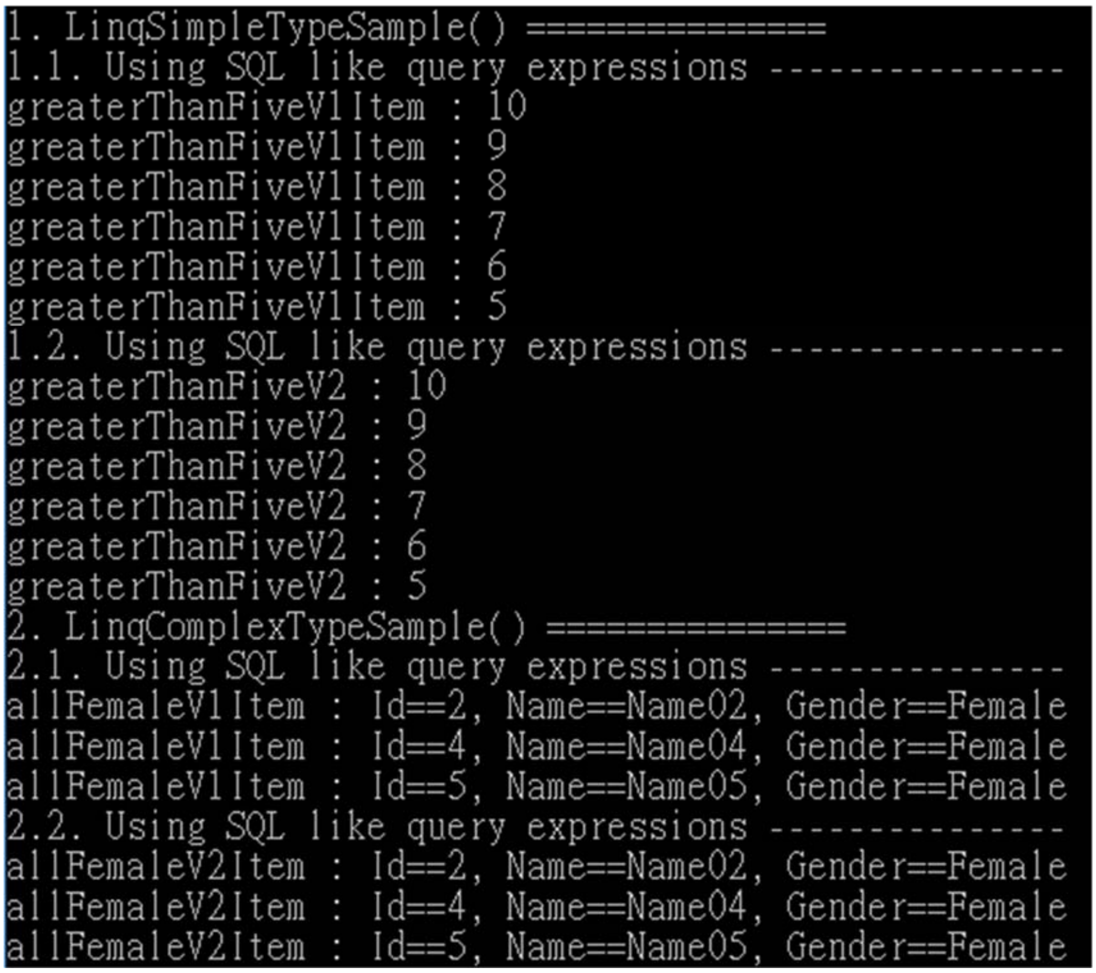
2. Sample : Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using OnLineGame;
namespace Sample
{
    class LinqSimpleTypeSample1
    {
        static void Main(string[] args)
        {
            // 1. ==========================================
            Console.WriteLine("1. LinqSimpleTypeSample() ===============");
            LinqSimpleTypeSample();
            // 2. ==========================================
            Console.WriteLine("2. LinqComplexTypeSample() ===============");
            LinqComplexTypeSample();
            // 3. ==========================================
            Console.WriteLine("3. Min_Max_Sum_Count_AverageSample() ===============");
            Min_Max_Sum_Count_AverageSample();
            // 4. ==========================================
            Console.WriteLine("4. stringMinMaxSample() ===============");
            StringMinMaxSample();
            // 5. ==========================================
            Console.WriteLine("5. AggregateSample() ===============");
            AggregateSample();
            Console.ReadLine();
        }

        // 1. ==========================================
        static void LinqSimpleTypeSample()
        {
            //There are 2 ways to write LINQ queries.
            int[] intArr = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
            //1.Using SQL like query expressions
            Console.WriteLine("1.1. Using SQL like query expressions ---------------");
            IEnumerable<int> greaterThanFiveV1 = from intItem in intArr
                                                 where intItem >= 5
                                                 select intItem;
            foreach (int greaterThanFiveV1Item in greaterThanFiveV1)
            {
                Console.WriteLine($"greaterThanFiveV1Item : {greaterThanFiveV1Item}");
            }
            //2.Using Lambda Expressions.
            Console.WriteLine("1.2. Using Lambda Expressions ---------------");
            IEnumerable<int> greaterThanFiveV2 = intArr.Where(intItem => intItem >= 5);
            foreach (int greaterThanFiveV2Item in greaterThanFiveV2)
            {
                Console.WriteLine($"greaterThanFiveV2 : {greaterThanFiveV2Item}");
            }
        }
        // 2. ==========================================
static void LinqComplexTypeSample()
{
    //There are 2 ways to write LINQ queries.
    List<GamerA> listGamerA = new List<GamerA>
    {
        new GamerA{Id = 1,Name="Name01",Gender = "Male"},
        new GamerA{Id = 2,Name="Name02",Gender = "Female"},
        new GamerA{Id = 3,Name="Name03",Gender = "Male"},
        new GamerA{Id = 4,Name="Name04",Gender = "Female"},
        new GamerA{Id = 5,Name="Name05",Gender = "Female"}
    };
    //1.Using SQL like query expressions
    Console.WriteLine("2.1. Using SQL like query expressions ----------------");
    IEnumerable<GamerA> allFemaleV1 = from gamer in listGamerA
                                      where gamer.Gender == "Female"
                                      select gamer;
    foreach (GamerA allFemaleV1Item in allFemaleV1)
    {
        Console.WriteLine($"allFemaleV1Item : {allFemaleV1Item}");
    }
    //2.Using Lambda Expressions.
   Console.WriteLine("2.2. Using Lambda Expressions ----------------");
    IEnumerable<GamerA> allFemaleV2 = listGamerA.Where(gamer => gamer.Gender == "Female");
    foreach (GamerA allFemaleV2Item in allFemaleV2)
    {
        Console.WriteLine($"allFemaleV2Item : {allFemaleV2Item}");
    }
}

// 3. ==========================================
static void Min_Max_Sum_Count_AverageSample()
{
    int[] intArr = { 10, 9, 8, 5, 4, 3, 7, 6, 2, 1 };
    Console.WriteLine("3.1. intArr ---------- ");
    foreach (int intArrItem in intArr)
    {
        Console.WriteLine($"intArrItem:{intArrItem}");
    }
    Console.WriteLine("3.2. intArr.Where(n => n % 2 == 0) ---------- ");
    foreach (int intArrItem in intArr.Where(n => n % 2 == 0))
    {
        Console.WriteLine($"intArr.Where(n => n % 2 == 0) Item :{intArrItem}");
    }
    Console.WriteLine("3.3. NoAggregateSample ---------- ");
    int? smallestItem = null;
    foreach (int intArrItem in intArr)
    {
        if (!smallestItem.HasValue || intArrItem < smallestItem)
        {
            smallestItem = intArrItem;
        }
    }
    Console.WriteLine($"smallestItem:{smallestItem}");
    int? largestItem = null;
    foreach (int intArrItem in intArr)
    {
        if (!largestItem.HasValue || intArrItem > largestItem)
```

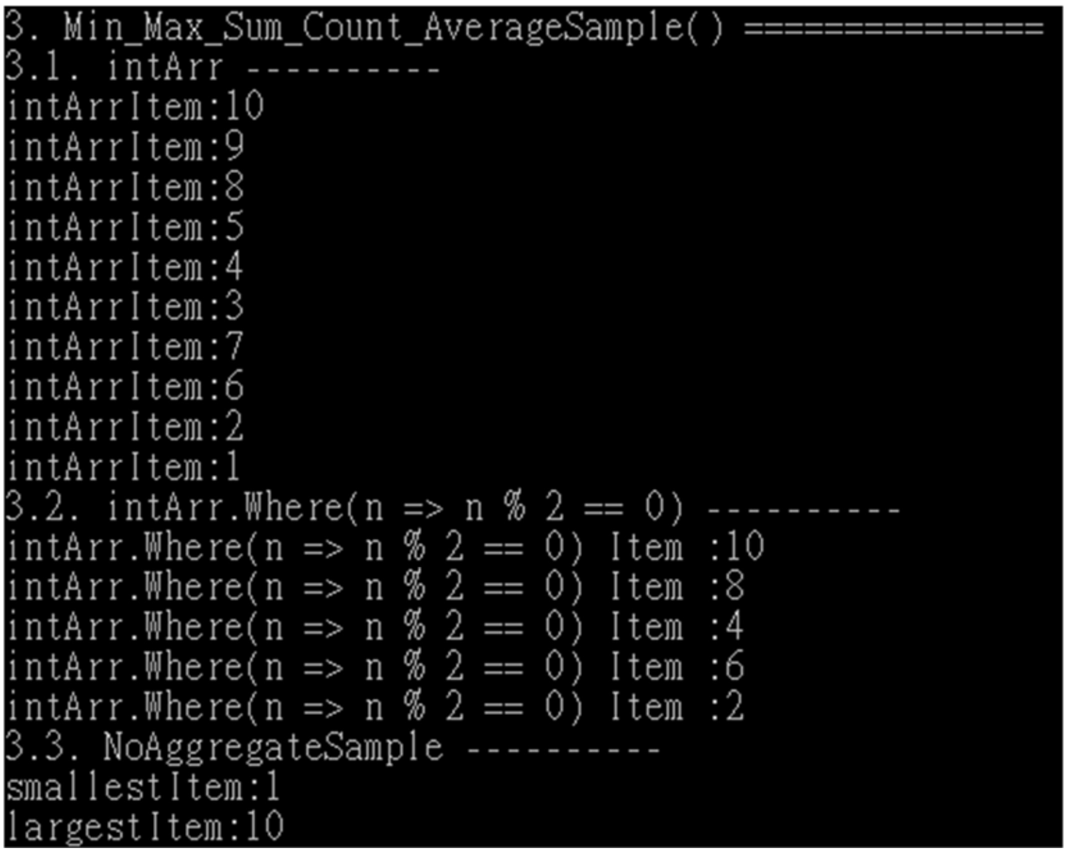
```
                {
                    largestItem = intArrItem;
                }
            }
            Console.WriteLine($"largestItem:{largestItem}");
            Console.WriteLine("3.4. Min_Max_Sum_Count_Average ---------- ");
            int smallestNumber = intArr.Min();
            int smallestEvenNumber = intArr.Where(n => n % 2 == 0).Min();
            Console.WriteLine($"intArr.Min()=={intArr.Min()}");
            Console.WriteLine($"intArr.Where(n => n % 2 == 0).Min()=={intArr.Where(n => n % 2 ==
0).Min()}");
            int largestNumber = intArr.Max();
            int largestEvenNumber = intArr.Where(n => n % 2 == 0).Max();
            Console.WriteLine($"intArr.Max()=={intArr.Max()}");
            Console.WriteLine($"intArr.Where(n => n % 2 == 0).Max()=={intArr.Where(n => n % 2 ==
0).Max()}");
            int sumOfAllNumbers = intArr.Sum();
            int sumOfAllEvenNumbers = intArr.Where(n => n % 2 == 0).Sum();
            Console.WriteLine($"intArr.Sum()=={intArr.Sum()}");
            Console.WriteLine($"intArr.Where(n => n % 2 == 0).Sum()=={intArr.Where(n => n % 2 ==
0).Sum()}");
            int countOfAllNumbers = intArr.Length;
            int countOfAllEvenNumbers = intArr.Where(n => n % 2 == 0).Count();
            Console.WriteLine($"intArr.Length=={intArr.Length}");
            Console.WriteLine($"intArr.Where(n => n % 2 == 0).Count()=={intArr.Where(n => n % 2 ==
0).Count()}");
            double averageOfAllNumbers = intArr.Average();
            double averageOfAllEvenNumbers = intArr.Where(n => n % 2 == 0).Average();
            Console.WriteLine($"intArr.Average()=={intArr.Average()}");
            Console.WriteLine($"intArr.Where(n => n % 2 == 0).Average()=={intArr.Where(n => n % 2 ==
0).Average()}");
        }
        // 4. ==========================================
        static void StringMinMaxSample()
        {
            string[] gamerName = { "Name00001", "Name02", "Name123456789" };
            Console.WriteLine($"GamerName.Min(x => x.Length):{gamerName.Min(x => x.Length)}");
            Console.WriteLine($"GamerName.Max(x => x.Length):{gamerName.Max(x => x.Length)}");
        }

      // 5. ==========================================
        static void AggregateSample()
        {
            //5.1. -------------------------------------------------
            Console.WriteLine("5.1. NoAggregateSample ---------- ");
            string[] gamerNames = { "Name01", "Name02", "Name03", "Name04", "Name05" };
            string gamerNamesStr1 = string.Empty;
            foreach (string gamerNamesItem in gamerNames)
            {
                if (gamerNamesItem.Equals(gamerNames.Last()))
                {
                    gamerNamesStr1 += gamerNamesItem;
                }
                else
                {
                    gamerNamesStr1 += $"{gamerNamesItem} , ";
                }
```

```
 }
Console.WriteLine(gamerNamesStr1);
// Return "Name01 , Name02 , Name03 , Name04, Name05"
//5.2. ------------------------------------------------
Console.WriteLine("5.2. AggregateSample ---------- ");
string gamerNamesStr2 = gamerNames.Aggregate((a, b) => $"{a} , {b}");
Console.WriteLine(gamerNamesStr2);
// Return "Name01 , Name02 , Name03 , Name04, Name05"
//Step1: a=="Name01", b=="Name02",
//so return "Name01 , Name02";
//Step2: a=="Name01 , Name02" , b=="Name03",
//so return "Name01 , Name02 , Name03";
//Step3: a=="Name01 , Name02 , Name03" , b=="Name04",
//so return "Name01 , Name02 , Name03 , Name04";
//Step4: a=="Name01 , Name02 , Name03 , Name04" , b=="Name05",
//so return "Name01 , Name02 , Name03 , Name04, Name05";
//5.3. ------------------------------------------------
//product of all numbers
Console.WriteLine("5.3. NoAggregateSample : product of all numbers ---------- ");
int[] intArr = { 10, 9, 8, 7, 6 };
int intArrProduct1 = 1;
foreach (int i in intArr)
 {
     intArrProduct1 = intArrProduct1 * i;
 }
Console.WriteLine(intArrProduct1);
// Return 30240 , because 10*9*8*7*6
//5.4. ------------------------------------------------
//product of all numbers
Console.WriteLine("5.4. AggregateSample : product of all numbers ---------- ");
int intArrProduct2 = intArr.Aggregate((a, b) => a * b);
Console.WriteLine(intArrProduct2);
// Return 30240 , because 10*9*8*7*6
//Step1: a==10, b==9,
//so return 10*9;
//Step2: a==10*9" , b==8,
//so return 10*9*8;
//Step3: a==10*9*8 , b==7,
//so return 10*9*8*7;
//Step4: a==10*9*8*7 , b==6,
//so return 10*9*8*7*6
//5.5. ------------------------------------------------
//product of all numbers
Console.WriteLine("5.5. AggregateSample : product of all numbers ---------- ");
int intArrProduct3 = intArr.Aggregate(5, (a, b) => a * b);
Console.WriteLine(intArrProduct3);
//Return 151200, because 5*10*9*8*7*6
//1.
//Enumerable.Aggregate<TSource, TAccumulate>
//(IEnumerable<TSource> source, TAccumulate seed,
//Func<TAccumulate, TSource, TAccumulate> func)
//Reference:
//https://msdn.microsoft.com/en-us/library/bb549218(v=vs.110).aspx
//Applies an accumulator function over a sequence.
```

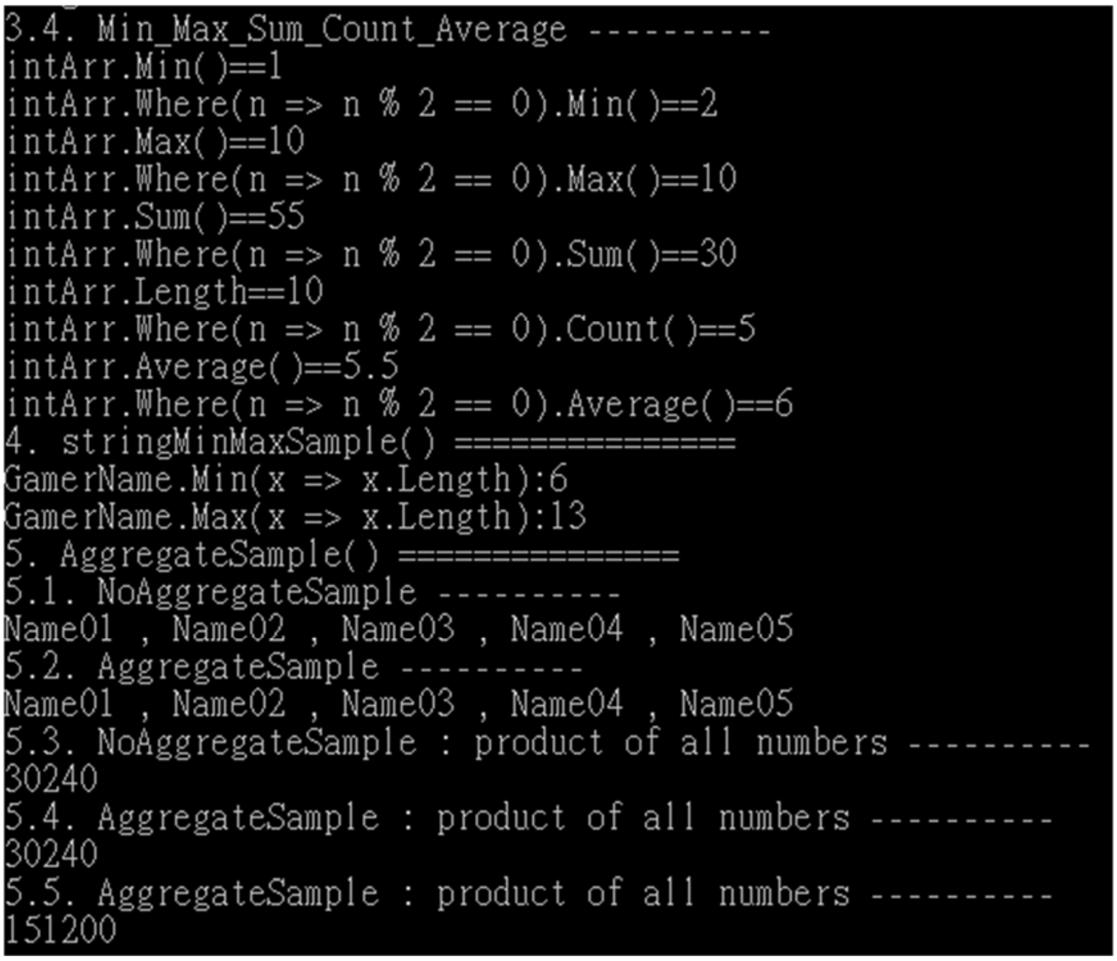
```
            //The specified seed value is used as the initial accumulator value.
            //---------------
            //2.
            //Step1: a==5, b==10,
            //so return 5*10;
            //Step2: a==5*10" , b==9,
            //so return 5*10*9;
            //Step3: a==5*10*9 , b==8,
            //so return 5*10*9*8;
            //Step4: a==5*10*9*8 , b==7,
            //so return 5*10*9*8*7
            //Step4: a==5*10*9*8*7 , b==6,
            //so return 5*10*9*8*7*6
        }
    }
}
namespace OnLineGame
{
    public class GamerA
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Gender { get; set; }
        public override string ToString()
        {
            return $"Id=={Id}, Name=={Name}, Gender=={Gender}";
        }
    }
}
```



```
1. LinqSimpleTypeSample() ================
1.1. Using SQL like query expressions ----------------
greaterThanFiveV1Item : 10
greaterThanFiveV1Item : 9
greaterThanFiveV1Item : 8
greaterThanFiveV1Item : 7
greaterThanFiveV1Item : 6
greaterThanFiveV1Item : 5
1.2. Using SQL like query expressions ----------------
greaterThanFiveV2 : 10
greaterThanFiveV2 : 9
greaterThanFiveV2 : 8
greaterThanFiveV2 : 7
greaterThanFiveV2 : 6
greaterThanFiveV2 : 5
2. LinqComplexTypeSample() ================
2.1. Using SQL like query expressions ----------------
allFemaleV1Item : Id==2, Name==Name02, Gender==Female
allFemaleV1Item : Id==4, Name==Name04, Gender==Female
allFemaleV1Item : Id==5, Name==Name05, Gender==Female
2.2. Using SQL like query expressions ----------------
allFemaleV2Item : Id==2, Name==Name02, Gender==Female
allFemaleV2Item : Id==4, Name==Name04, Gender==Female
allFemaleV2Item : Id==5, Name==Name05, Gender==Female
```



```
3. Min_Max_Sum_Count_AverageSample() ===============
3.1. intArr ----------
intArrItem:10
intArrItem:9
intArrItem:8
intArrItem:5
intArrItem:4
intArrItem:3
intArrItem:7
intArrItem:6
intArrItem:2
intArrItem:1
3.2. intArr.Where(n => n % 2 == 0) ----------
intArr.Where(n => n % 2 == 0) Item :10
intArr.Where(n => n % 2 == 0) Item :8
intArr.Where(n => n % 2 == 0) Item :4
intArr.Where(n => n % 2 == 0) Item :6
intArr.Where(n => n % 2 == 0) Item :2
3.3. NoAggregateSample ----------
smallestItem:1
largestItem:10
```



```
3.4. Min_Max_Sum_Count_Average ----------
intArr.Min()==1
intArr.Where(n => n % 2 == 0).Min()==2
intArr.Max()==10
intArr.Where(n => n % 2 == 0).Max()==10
intArr.Sum()==55
intArr.Where(n => n % 2 == 0).Sum()==30
intArr.Length==10
intArr.Where(n => n % 2 == 0).Count()==5
intArr.Average()==5.5
intArr.Where(n => n % 2 == 0).Average()==6
4. stringMinMaxSample() ===============
GamerName.Min(x => x.Length):6
GamerName.Max(x => x.Length):13
5. AggregateSample() ===============
5.1. NoAggregateSample ----------
Name01 , Name02 , Name03 , Name04 , Name05
5.2. AggregateSample ----------
Name01 , Name02 , Name03 , Name04 , Name05
5.3. NoAggregateSample : product of all numbers ----------
30240
5.4. AggregateSample : product of all numbers ----------
30240
5.5. AggregateSample : product of all numbers ----------
151200
```