

LESSON 07

I. VÒNG LẶP FOREACH TRONG LẬP TRÌNH C# CƠ BẢN

❖ Cú pháp và nguyên tắc hoạt động của foreach trong C#

➤ Cú pháp

```
foreach (<kiểu dữ liệu> <tên biến tạm> in <tên mảng hoặc tập hợp>)  
{  
    // Code xử lý  
}
```

- Các từ khoá **foreach**, **in** là từ khoá bắt buộc.
- <kiểu dữ liệu> là kiểu dữ liệu của các phần tử trong mảng hoặc tập hợp.
- <tên biến tạm> là tên 1 biến tạm đại diện cho phần tử đang xét khi duyệt mảng hoặc tập hợp.

➤ Nguyên tắc hoạt động:

Foreach cũng có nguyên tắc hoạt động tương tự như các cấu trúc lặp khác cụ thể như sau:

- Ở vòng lặp đầu tiên sẽ gán giá trị của phần tử đầu tiên trong mảng vào **biến tạm**.
- Thực hiện khối lệnh bên trong vòng lặp **foreach**.
- Qua mỗi vòng lặp tiếp theo sẽ thực hiện kiểm tra xem đã duyệt hết mảng hoặc tập hợp chưa. Nếu chưa thì tiếp gán giá trị của phần tử hiện tại vào **biến tạm** và tiếp tục thực hiện khối lệnh bên trong.
- Nếu đã duyệt qua hết các phần tử thì vòng lặp sẽ kết thúc.

➤ Qua nguyên tắc hoạt động trên ta có thể thấy:

- **Biến tạm** trong vòng lặp **foreach** sẽ tương đương với phần tử *i* trong cách duyệt của vòng lặp **for** (đã trình bày)
- Qua mỗi bước lặp ta chỉ có thể thao tác với giá trị của phần tử đang xét mà không thể tương tác với các phần tử đứng trước nó hay đứng sau nó.
- Bằng cách duyệt của **foreach** ta không thể thay đổi giá trị của các phần tử vì lúc này giá trị của nó đã được sao chép ra một 1 biến tạm và ta chỉ có thể thao tác với **biến tạm**.

- Thậm chí việc thay đổi **giá trị** của **biến tạm** cũng không được phép. Nếu ta cố làm điều đó thì sẽ gặp lỗi sau:

```
0 references
static void Main(string[] args)
{
    int[] collection = new int[5];
    foreach(int item in collection)
    {
        item = 2;
    }
}
```

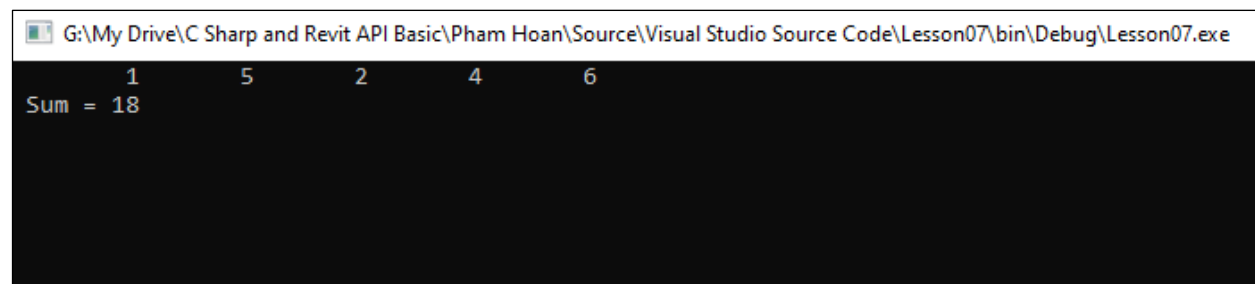
❖ Sử dụng foreach trong C#

Trong trường truy xuất các phần tử trong danh sách như Mảng, List, Collection, Generic, để duyệt các danh sách, tập hợp có tính chất như trên thì **foreach** là lựa chọn tốt nhất.

Chúng ta sẽ tìm hiểu sức mạnh của **foreach** qua các bài học sau. Còn trong bài học này mình chỉ ví dụ đơn giản để các bạn có thể nắm cú pháp cũng như cách sử dụng **foreach**.

Xét chương trình sau:

```
//VD Tính tổng
int[] IntArray = { 1, 5, 2, 4, 6 };
int Sum = 0;
/*
 * Sử dụng foreach để duyệt mảng và in giá trị của các phần tử trong mảng.
 * Đồng thời tân dụng vòng lặp để tính tổng các phần tử trong mảng.
 */
foreach (int item in IntArray)
{
    Console.WriteLine("\t" + item);
    Sum += item;
}
Console.WriteLine("\n Sum = " + Sum);
Console.ReadKey();
```



```
G:\My Drive\C Sharp and Revit API Basic\Pham Hoan\Source\Visual Studio Source Code\Lesson07\bin\Debug\Lesson07.exe
1
5
2
4
6
Sum = 18
```

➤ **Nhận xét:**

- ✓ Ta có thể thấy cách duyệt foreach ngắn gọn hơn nhiều so với cách duyệt bằng vòng lặp for thông thường.
- ✓ Ta cũng chẳng quan tâm đến việc phải xử lý độ dài mảng hay chỉ số phần tử để truy xuất 1 phần tử nào đó.

➤ **Hiệu suất xử lý:**

- Đối với mảng danh sách hoặc tập hợp có khả năng truy xuất ngẫu nhiên thì **for** chiếm ưu thế.
- Đối với mảng danh sách hoặc tập hợp không có khả năng truy xuất ngẫu nhiên thì **foreach** chiếm ưu thế.
- Nhìn chung hiệu suất của **for** và **foreach** còn phụ thuộc vào cấu trúc dữ liệu đang xét cho nên việc so sánh này chỉ mang tính chất tham khảo.

Sau đây là 2 đoạn chương trình kiểm tra tốc độ của **for** và **foreach** đối với 2 cấu trúc dữ liệu là **mang 1 chiều** (có khả năng truy xuất ngẫu nhiên) và danh sách liên kết **LinkedList** (không có khả năng truy xuất ngẫu nhiên):

🚩 Đầu tiên là mảng 1 chiều:

```
0 references
static void Main(string[] args)
{
    /* Kiểm tra tốc độ của for */
    // using System.Diagnostics

    Stopwatch start = new Stopwatch();
    start.Start();

    int[] IntArray = new int[Int32.MaxValue / 100]; //(20tr ptu)
    int s = 0;
    int Length = IntArray.Length;
    for (int i = 0; i < Length; i++)
    {
        s += IntArray[i];
    }
    start.Stop();
    Console.WriteLine(" Thời gian chạy của for: {0} giây {1} mili giây",
        start.Elapsed.Seconds, start.Elapsed.Milliseconds);

    /* Kiểm tra tốc độ của foreach */
    Stopwatch start2 = new Stopwatch();
    start2.Start();
    int[] IntArray2 = new int[Int32.MaxValue / 100];
    int s2 = 0;
    foreach (int item in IntArray2)
    {
        s2 += item;
    }
    start2.Stop();
    Console.WriteLine(" Thời gian chạy của foreach: {0} giây {1} mili giây",
        start2.Elapsed.Seconds, start2.Elapsed.Milliseconds);
}
```

➤ Đoạn chương trình mình thực hiện:

- Khai báo 1 mảng 1 chiều có 20 triệu phần tử (khai báo số phần tử lớn để có thể thấy được sự chênh lệch về tốc độ)
- Lần lượt dùng for, foreach để duyệt mảng đó và thực hiện 1 câu lệnh nào đó.
- Cuối cùng là xuất ra thời gian thực thi của từng trường hợp dưới dạng giây và mili giây.

➤ Kết quả khi chạy đoạn chương trình trên:

```
G:\My Drive\C Sharp and Revit API Basic\Pham Hoan\Source\Visual Studio Source Code\Lesson07\bin\Debug\Lesson07.exe
Thời gian chạy của for: 0 giây 73 mili giây
Thời gian chạy của foreach: 0 giây 76 mili giây
```

Dựa vào kết quả ta có thể thấy được sự chênh lệch nhỏ về tốc độ, nếu kiểm tra với số phần tử lớn hơn hoặc cấu trúc dữ liệu phức tạp hơn thì chênh lệch này càng lớn.

🚦 **Tiếp theo là đên danh sách liên kết `LinkedList`:**

```
static void Main(string[] args)
{
    /*Khai báo 1 LinkedList chứa các số nguyên int và khởi tạo giá trị cho nó.*/

    LinkedList<int> list = new LinkedList<int>();
    for (int i = 0; i < 10000; i++)
    {
        list.AddLast(i);
    }

    /* Kiểm tra tốc độ của for */
    Stopwatch st = new Stopwatch();
    int s1 = 0, length = list.Count;
    st.Start();
    for (int i = 0; i < length; i++)
    {
        s1 += list.ElementAt(i);
    }
    st.Stop();

    /* Kiểm tra tốc độ của foreach */
    Stopwatch st2 = new Stopwatch();
    int s2 = 0;
    st2.Start();
    foreach (int item in list)
    {
        s2 += item;
    }
    st2.Stop();

    /* In ra giá trị tính tổng giá trị các phần tử khi duyệt bằng for và foreach để
    chắc chắn rằng cả 2 đều chạy đúng */
    Console.WriteLine(" s1 = {0} s2 = {1}", s1, s2);
    Console.WriteLine(" Thời gian chạy của for = {0} giây {1} mili giây",
        st.Elapsed.Seconds, st.Elapsed.Milliseconds);
    Console.WriteLine(" Thời gian chạy của foreach = {0} giây {1} mini giây",
        st2.Elapsed.Seconds, st2.Elapsed.Milliseconds);
    Console.ReadKey();
}
```

```
G:\My Drive\C Sharp and Revit API Basic\Pham Hoan\Source\Visual Studio Source Code\Lesson07\bin\Debug\Lesson07.exe
s1 = 49995000 s2 = 49995000
Thời gian chạy của for = 0 giây 307 mili giây
Thời gian chạy của foreach = 0 giây 0 mini giây
```

*Tùy vào từng trường hợp mà ta nên dùng **for** hay **foreach**. Không nên lạm dụng 1 thứ quá nhiều.*

Bài tập:

1. Viết chương trình C# để nhập vào 1 mảng và đếm số lần xuất hiện của từng phần tử trong mảng và in kết quả trên màn hình.
2. Chia mảng trên thành mảng chẵn, mảng lẻ trong C#

II. LỚP STRING TRONG LẬP TRÌNH C# CĂN BẢN

String là một kiểu dữ liệu tham chiếu được dùng để lưu trữ chuỗi ký tự. Vì là một kiểu dữ liệu nên cách khai báo và sử dụng hoàn toàn tương tự các kiểu dữ liệu khác (đã trình bày ở bài trước):

Hôm nay mình sẽ không bàn đến khai báo của nó nữa mà đi sâu vào các thuộc tính và phương thức mà lớp **String** hỗ trợ.

❖ **Thuộc tính** trong lớp **String**:

STT	Thuộc tính
1	Chars Lấy đối tượng <i>Char</i> tại một vị trí cụ thể trong đối tượng <i>String</i> hiện tại
2	Length Lấy số ký tự của đối tượng <i>String</i> hiện tại

❖ **Phương thức** trong lớp **String**:

Tên phương thức	Ý nghĩa	Ghi chú
<code>String.Compare(string strA, string strB)</code>	So sánh 2 chuỗi <code>strA</code> và <code>strB</code> có bằng nhau hay không. Nếu bằng nhau thì trả về 0, nếu <code>strA > strB</code> thì trả về 1 trường hợp còn lại trả về -1	Chúng ta có thể gọi phương thức so sánh này thông qua tên biến bằng cách: <code><tên biến>.CompareTo(string strB)</code>
<code>String.Concat(string strA, string strB)</code>	Nối 2 chuỗi <code>strA</code> và <code>strB</code> thành một chuỗi	Phương thức này tương tự như phép cộng chuỗi bằng toán tử cộng.
<code>IndexOf(char value)</code>	Trả về một số nguyên kiểu <code>int</code> là vị trí xuất hiện đầu tiên của ký tự <code>value</code> trong chuỗi.	Nếu như không tìm thấy thì phương thức sẽ trả về -1.
<code>Insert(int startIndex, string value)</code>	Trả về một chuỗi mới trong đó bao gồm chuỗi cũ đã chèn thêm chuỗi <code>value</code> tại vị trí <code>startIndex</code> .	
<code>String.IsNullOrEmpty(string value)</code>	Kiểm tra chuỗi <code>value</code> có phải là chuỗi <code>null</code> hoặc rỗng hay không. Nếu đúng thì trả về	
<code>LastIndexOf(char value)</code>	https://drive.google.com/uc?id=1xUngZYYcus_BZ5xOD1hKGal8vjJmitoD là vị trí xuất hiện cuối cùng của ký tự <code>value</code> trong chuỗi.	

<code>ToCharArray()</code>	Trả về một mảng các ký tự trong chuỗi ban đầu.	
<code>ToLower()</code>	Trả về một chuỗi mới đã viết thường tất cả các ký tự trong chuỗi ban đầu.	
<code>ToUpper()</code>	Trả về một chuỗi mới đã viết hoa tất cả các ký tự trong chuỗi ban đầu	
<code>Trim()</code>	Trả về một chuỗi mới đã bỏ tất cả các khoảng trắng ở đầu và cuối chuỗi ban	

<code>Remove(int startIndex, int count)</code>	Trả về một chuỗi mới đã gỡ bỏ count ký tự từ vị trí <code>startIndex</code> trong chuỗi ban đầu.	
<code>Replace(char oldValue, char newValue)</code>	Trả về một chuỗi mới đã thay thế các ký tự <code>oldValue</code> bằng ký tự <code>newValue</code> trong chuỗi ban đầu.	
<code>Split(char value)</code>	Trả về một mảng các chuỗi được cắt ra từ chuỗi ban đầu tại những nơi có ký tự <code>value</code>	Phương thức này ta có thể truyền vào nhiều ký tự khác nhau. Khi đó phương thức sẽ thực hiện cắt theo từng ký tự đã truyền vào.
<code>Substring(int startIndex, int length)</code>	Trả về một chuỗi mới được cắt từ vị trí <code>startIndex</code> với số ký tự cắt là <code>length</code> từ chuỗi ban đầu.	Nếu như bạn gọi <code>Substring</code> mà chỉ truyền vào giá trị <code>startIndex</code> thì mặc định phương thức sẽ cắt từ vị trí <code>startIndex</code> đến cuối chuỗi.

Lưu ý:

- Các phương thức mà mình có ghi **String** phía trước là các phương thức gọi thông qua tên lớp. Các phương thức còn lại được gọi thông qua đối tượng.
- Các phương thức khi gọi sẽ tạo ra đối tượng mới rồi thao tác trên đối tượng đó chứ không thao tác trực tiếp với đối tượng đang xét.

Xét ví dụ sau:

```
0 references
static void Main(string[] args)
{
    string a = "TTDesignCompany";
    a.Replace(oldValue: "TT", newValue: "T&T");

    Console.WriteLine(a);
    Console.ReadKey();

    a = a.Replace(oldValue: "TT", newValue: "T&T");
    Console.WriteLine(a);
    Console.ReadKey();
}
```

❖ Ứng dụng lớp String vào việc xử lý chuỗi:

Để hiểu rõ cách sử dụng các phương thức trên. Chúng ta cùng thực hiện chuẩn hoá một chuỗi họ tên của người dùng với các yêu cầu:

- Cắt bỏ hết các khoảng trắng dư ở đầu cuối chuỗi. Các từ cách nhau một khoảng trắng nếu phát hiện có nhiều hơn 1 khoảng trắng thì thực hiện cắt bỏ.
- Viết hoa chữ cái đầu tiên của mỗi từ, các chữ cái tiếp theo thì viết thường.

Ý tưởng:

- Cắt khoảng trắng dư ở đầu và cuối chuỗi thì ta có thể sử dụng phương thức **Trim**.
- Khoảng trắng ở giữa thì ta sẽ duyệt cả chuỗi nếu phát hiện có 2 khoảng trắng thì thay thế nó bằng 1 một khoảng trắng. Để làm điều này ta có thể dùng:
 - ✓ **IndexOf** để phát hiện khoảng trắng.
 - ✓ **Replace** để thay thế 2 khoảng trắng thành 1 khoảng trắng.
- Viết hoa chữ cái đầu và viết thường các chữ cái còn lại thì ta có thể cắt chuỗi họ tên ra thành các từ và ứng với mỗi từ ta thực hiện như yêu cầu đề bài. Để làm điều này ta có thể sử dụng:
 - ✓ **Split** để cắt ra các từ.

- ✓ **Substring** để cắt ra các chữ cái mong muốn.
- ✓ **ToUpper** để viết hoa và **ToLower** để viết thường

(Mở ví dụ + chữa bài)

III. STRUCT TRONG LẬP TRÌNH C# CĂN BẢN.

❖ **Struct là gì? Đặc điểm của Struct.**

Struct là một kiểu dữ liệu có cấu trúc, được kết hợp từ các kiểu dữ liệu nguyên thủy do người lập trình định nghĩa để thuận tiện trong việc quản lý dữ liệu và lập trình.

Xét bài toán sau: Ta cần lưu trữ thông tin của 10 sinh viên với mỗi sinh viên gồm có các thông tin như

- ✓ Mã số.
- ✓ Họ tên.
- ✓ Nơi sinh.
- ✓ CMND.

Khi đó, để lưu thông tin của 1 sinh viên ta cần 4 biến chứa 4 thông tin trên. Nếu muốn lưu thông tin 10 sinh viên thì cần 40 biến. Chắc không quá nhiều, nhưng nếu muốn lưu thông tin của 1000, 10000 sinh viên thì sao?

Từ đó người ta mới đưa ra khái niệm **kiểu dữ liệu có cấu trúc** để giải quyết vấn đề trên.

Ý tưởng là **đóng gói** các thông tin đó vào **1 đối tượng duy nhất**. Như vậy thay vì phải khai báo 40 biến thì ta chỉ cần khai báo 1 mảng 10 phần tử mà mỗi phần tử có kiểu dữ liệu ta đã định nghĩa.

❖ **Đặc điểm của struct:**

- Là một kiểu dữ liệu tham trị (**kiểu dữ liệu tham trị**)
- Dùng để đóng gói các trường dữ liệu khác nhau nhưng có liên quan đến nhau.
- Bên trong **struct** ngoài các biến có kiểu dữ liệu cơ bản còn có các phương thức, các struct khác.
- Muốn sử dụng phải khởi tạo cấp phát vùng nhớ cho đối tượng thông qua toán tử **new**.
- **Struct** không được phép kế thừa (Trình bày sau)

❖ Khai báo và sử dụng struct

➤ Cú pháp

```
struct <tên struct>
{
    public <danh sách các biến>;
}
```

Trong đó:

- <tên struct> là tên kiểu dữ liệu do mình tự đặt và tuân thủ theo quy tắc đặt tên (đã trình bày)
- <danh sách các biến> là danh sách các biến thành phần được khai báo như khai báo biến bình thường (đã trình bày)
- Từ khoá **public** là từ khoá chỉ định phạm vi truy cập (sẽ trình bày sau). Trong ngữ cảnh hiện tại thì có thể hiểu từ khoá này giúp cho người khác có thể truy xuất được để sử dụng

➤ Ví dụ:

```
struct SinhVien
{
    public int MSSV;
    public string HoTen;
    public string NoiSinh;
    public int CMND;
}
```

Lưu ý: bên trong vẫn còn 2 khai báo chưa được nhắc đến đó là:

- Constructor (hàm khởi tạo).
- Các phương thức mà mình muốn cung cấp để hỗ trợ người dùng khi thao tác với dữ liệu bên trong **struct**.
- Hai phần này sẽ được trình bày trong các bài sau.

❖ Sử dụng

Ta có thể truy xuất đến từng thành phần dữ liệu của struct thông qua toán tử “.”
Kèm theo tên thành phần muốn truy xuất.

Xét bài toán sau:

Viết chương trình lưu trữ thông tin của sinh viên bao gồm: mã số, họ tên, điểm toán, điểm lý, điểm văn. Thực hiện nhập thông tin cho 1 sinh viên và tính điểm trung bình theo công thức $(\text{toán} + \text{lý} + \text{văn})/3$ và xếp loại học lực của Sv đó.

(Mở ví dụ + chữa bài)

```
G:\My Drive\C Sharp and Revit API Basic\Pham Hoan\Source\Visual Studio Source Code\Lesson07\bin\Debug\Lesson07.exe
Nhap thong tin sinh vien:
Ma so: 12345
Ho ten: Pham Cong Hoan
Diem toan: 8
Diem ly: 9
Diem van: 8
*****
Danh gia ket qua sinh vien vua nhap la:
Diem TB cua sinh vien Pham Cong Hoan la: 8.33
Xep loai sinh vien Pham Cong Hoan la: Gioi
```

❖ Kết luận:

- Struct là gì? Đặc điểm của struct.
- Khai báo và sử dụng struct.

IV. ENUM TRONG LẬP TRÌNH C# CĂN BẢN.

❖ Enum là gì? Đặc điểm của Enum .

Enum là từ khoá dùng để khai báo một kiểu liệt kê (Enumeration). Kiểu liệt kê là một tập hợp các hằng số do người dùng tự định nghĩa.

Nói cách khác, enum là cách mà C# hỗ trợ người dùng gom nhóm các hằng số lại với nhau và có chung một tên gọi (thường các hằng số này sẽ có liên quan với nhau ví dụ như các trạng thái của 1 sự vật, các tính chất của 1 sự vật, . . .)

❖ Đặc điểm của struct:

Đặc điểm của **enum**:

- Là một kiểu dữ liệu tham trị
- **Enum** không được phép kế thừa (sẽ trình bày sau)

❖ Khai báo và sử dụng struct

➤ Cú pháp

```
enum <tên enum>
{
    < biểu tượng hằng 1>,
    < biểu tượng hằng 2>,
    < biểu tượng hằng 3>,
    ...
}
```

Trong đó:

- <tên enum> là tên kiểu liệt kê do mình tự đặt và tuân thủ theo quy tắc đặt tên (đã trình bày)
- <danh sách các biểu tượng hằng> là danh sách các biểu tượng hằng thành phần mỗi biểu tượng hằng cách nhau bằng dấu “,”.

➤ Ví dụ:

```
0 references
enum XepLoai
{
    TrungBinh = 0,
    Kha = 1,
    Gioi = 2,
    XuatSac = 3
}
```

Lưu ý:

- Ta hoàn toàn có thể quy định giá trị cho từng biểu tượng hằng bằng cách trực tiếp khi khai báo. Ví dụ :

```
0 references
enum XepLoai
{
    TrungBinh = 4,
    Kha = 7,
    Gioi = 9,
    XuatSac = 10
}
```

- Khi đó các biểu tượng hằng TrungBinh, Kha, Gioi, XuatSac sẽ đại diện cho các số nguyên lần lượt là 4, 7, 9, 10.
- Nếu ta không quy định giá trị cho các biểu tượng hằng thì giá trị của biểu tượng hằng đầu tiên sẽ mặc định là 0 và tăng dần cho các biểu tượng hằng tiếp theo.

❖ Sử dụng

Ta có thể truy xuất đến từng thành phần dữ liệu của enum thông qua toán tử “.”
Kèm theo tên thành phần muốn truy xuất.

Ví dụ:

```
//VD01: Chơi game có 4 levels
2 references
enum Level
{
    Easy = 0,
    Normal = 1,
    Hard = 2,
    VeryHard = 3
}
//Sử dụng
0 references
static void Main(string[] args)
{
    Level startLevel = Level.Normal;
}
```

Lưu ý:

Mặc dù bản chất các biểu tượng hằng là đại diện cho các số nguyên nhưng bạn không thể so sánh trực tiếp chúng với các số nguyên được mà phải ép kiểu.

Ví dụ:

```
//Chọn level chơi game.
Console.WriteLine("Mời bạn chọn level chơi game (0~3) :");
int userChoose = int.Parse(Console.ReadLine());

if (userChoose == Level.Easy)
{
    LevelName = "Easy level";
}
```

Để chương trình không báo lỗi ta có thể ép kiểu biểu tượng hằng Easy về kiểu `int`.

```
if (userChoose == (int)Level.Easy)
{
    LevelName = "You chose Easy level";
}
```

Chúng ta cũng có thể ép kiểu ngược lại từ số nguyên sang kiểu liệt kê:

```
Level normal = (Level) 1;
```

❖ Ý nghĩa:

- Khi khai báo 1 biến nào đó, các lập trình viên thường cố gắng xây dựng 1 tập các giá trị của biến đó (nếu có thể) và gom nhóm chúng bằng enum.
- Chính vì được sử dụng với mục đích gom nhóm các hằng có liên quan với nhau thành 1 tên duy nhất nên khi sử dụng bạn không cần phải nhớ chính xác tên hằng mà chỉ cần nhớ tên `enum` chứa nó là đủ việc còn lại đã có visual studio hỗ trợ

BÀI TẬP

1. Tạo 1 kiểu dữ liệu struct lưu trữ thông tin học sinh gồm:

- Họ tên;
- Giới tính;
- Điểm Toán, Văn, Anh,

Tính điểm trung bình và xếp loại (TB-K-G) cho học viên này theo kiểu enum.

2. Lưu trữ 1 danh sách các học viên trong lớp học và phân loại các học viên thành 3 nhóm (TB-K-G) và in ra số lượng học viên trong từng nhóm và tên HV đó.