STRUCT

Struct trong C# là một nhóm kiểu dữ liệu rất phổ biến. Các kiểu dữ liệu cơ sở trong C# mà bạn đã biết như byte, int, long, float, double, char đều thuộc nhóm struct. Tương tự như enum, C# cũng cho phép bạn tự định nghĩa kiểu dữ liệu thuộc nhóm struct.

Trong bài học này bạn sẽ làm quen với việc định nghĩa và sử dụng kiểu dữ liệu thuộc loại struct trong C#.

Struct là gì?

Khái niệm struct

Struct là nhóm kiểu dữ liệu trong đó mỗi kiểu dữ liệu có thể chứa các **thành viên** khác. Nói cách khác, kiểu dữ liệu thuộc nhóm struct có khả năng chứa một nhóm dữ liệu thuộc nhiều kiểu khác nhau và khả năng xử lý thông tin.

Các thành viên chính của struct bao gồm:

- Biến thành viên (member variable), hay còn gọi là trường dữ liệu (data field): đây là thành phần chứa dữ liệu của struct.
- Phương thức thành viên (method): đây là thành phần xử lý thông tin.
- Đặc tính (property): đây là thành phần chịu trách nhiệm xuất nhập dữ liệu.

Ngoài ra, struct còn có thể chứa nhiều loại thành viên khác như hằng (constant), bộ đánh chỉ mục (indexer), phép toán (operator), sự kiện (event), kiểu con (nested type). Các loại thành viên này sẽ được xem xét chi tiết khi học về class. Chúng được định nghĩa và sử dụng giống hệt nhau trong class và struct.

Như đã nói nhiều lần, struct trong C# thuộc nhóm value type.

Một số struct cơ sở

Trong C#, các kiểu dữ liệu cơ sở như int, bool, char, v.v. (trừ object và string) thực tế đều là các struct (nhưng hơi đặc biệt một chút). Vì vậy các kiểu này không đơn thuần chỉ chứa dữ liệu mà còn chứa cả các phương thức để thực hiện những công việc nhất định.

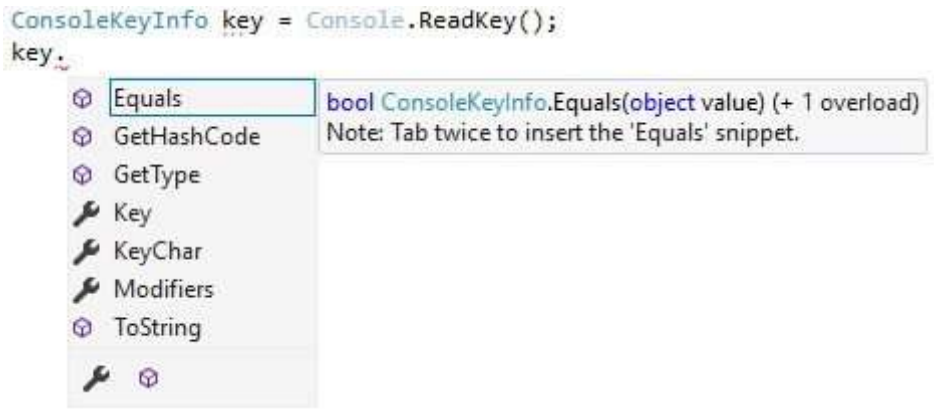
Dưới đây là một số ví dụ về sử dụng các kiểu này theo kiểu khác.

```
> var i = new int();
> var j = new Int32();
> j
0
> i
0
> i.CompareTo(j)
0
> i.Equals(j)
true
> int.MaxValue
```

```
2147483647
> int.MinValue
-2147483648
> int.Parse("20")
20
> var c = new char();
> c
'\0'
> char.MaxValue
'\uffff'
> char.MinValue
'\0'
```

Dấu chấm đặt sau tên biến hoặc sau tên kiểu là một phép toán có tên gọi là **phép toán truy xuất thành viên** (member access operator). Các phương thức của int như CompareTo, Equals được gọi là **instance method** (chỉ gọi được từ biến), trong khi Parse là một **static method** (gọi từ tên kiểu). MaxValue và MinValue là các hằng thành viên, đồng thời là **thành viên static** (truy xuất từ tên kiểu thay vì truy xuất từ biến).

Trong bài học về console trong C#, bạn cũng gặp một struct: ConsoleKeyInfo. Đây là struct chứa kết quả thực hiện của ReadKey().



Các thành viên của struct ConsoleKeyInfo trong C#

Trong struct này chứa 3 **đặc tính** (property): Key (kiểu enum ConsoleKey), KeyChar (kiểu char), Modifier (kiểu enum ConsoleModifiers). Struct này cũng chứa 4 **phương thức** (method) kế thừa từ object: Equals, GetHashCode, GetType, ToString.

Như vậy việc sử dụng kiểu struct không hề xa lạ với bạn.

Khai báo kiểu struct

Khi bạn đã nắm qua được khái niệm struct trong C#, bây giờ chúng ta sẽ chuyển sang tự tạo kiểu struct của riêng mình.

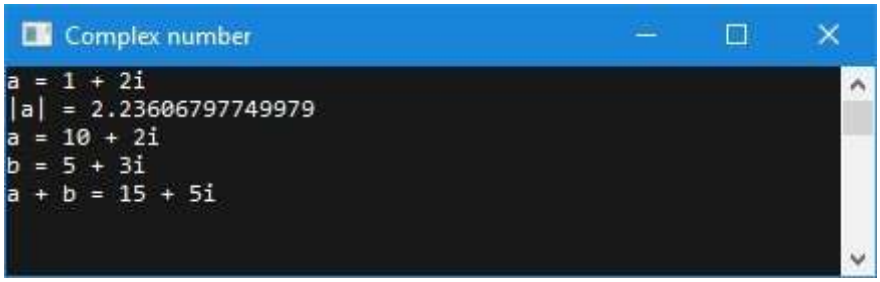
Để dễ hình dung, chúng ta cùng thực hiện một project nhỏ.

Tạo một blank solution đặt tên là S06_Struct và thêm một project ConsoleApp đặt tên là P01_StructDefinition. Viết code cho Program.cs như sau:

```
using System;
namespace P01_StructDefinition
{
    using static Console;
    /// <summary>
    /// Struct biểu diễn số phức
    /// </summary>
    struct Complex
    {
        public double Real; // trường thực
        public double Imaginary; // trường ảo
        /// <summary>
        /// Hàm tạo
        /// </summary>
        /// <param name="r">phần thực</param>
        public Complex(double r)
        {
            Real = r;
            Imaginary = 0;
        }
        /// <summary>
        /// Hàm tạo
        /// </summary>
        /// <param name="r">phần thực</param>
        /// <param name="i">phần ảo</param>
        public Complex(double r, double i)
        {
            Real = r;
            Imaginary = i;
        }
        /// <summary>
        /// Chuyển chuỗi hợp lệ thành giá trị của Real và Imaginery
        /// </summary>
        /// <param name="value"></param>
        public void Parse(string value)
        {
            var temp = value.Trim();
            if (temp.EndsWith("i") || temp.EndsWith("I"))
            {
                temp = temp.TrimEnd('i', 'I');
                var tokens = temp.Split(new[] { '+', '-' }, 2);
                Real = double.Parse(tokens[0]);
                Imaginary = double.Parse(tokens[1]);
            }
            else
            {
                Real = double.Parse(temp);
```

```
            }
        }
        /// <summary>
        /// Chuyển chuỗi hợp lệ thành giá trị của Real và Imaginery
        /// </summary>
        /// <param name="value"></param>
        /// <returns></returns>
        public static Complex FromString(string value)
        {
            var temp = new Complex();
            temp.Parse(value);
            return temp;
        }
        /// <summary>
        /// Đặc tính, trả về module của số phức
        /// </summary>
        public double Modulus => Math.Sqrt(Real * Real + Imaginary * Imaginary);
        /// <summary>
        /// Ghi đè phép toán +
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <returns></returns>
        public static Complex operator +(Complex a, Complex b)
        {
            return new Complex(a.Real + b.Real, a.Imaginary + b.Imaginary);
        }
        /// <summary>
        /// Ghi đè phép toán -
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <returns></returns>
        public static Complex operator -(Complex a, Complex b)
        {
            return new Complex(a.Real - b.Real, a.Imaginary - b.Imaginary);
        }
        /// <summary>
        /// Ghi đè phương thức ToString() của object
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            if (Imaginary == 0)
            {
                return Real.ToString();
            }
            return $"{Real} {(Imaginary > 0 ? '+' : '-')} {Math.Abs(Imaginary)}i";
        }
    }
    class Program
```

```
    {
        static void Main(string[] args)
        {
            Title = "Complex number";
            // khai báo và khởi tạo biến a thuộc kiểu Complex
            var a = new Complex(1, 2);
            WriteLine($"a = {a}");
            // sử dụng đặc tính Modulus của Complex
            WriteLine($"|a| = {a.Modulus}");
            // gọi phương thức Parse
            a.Parse("10-2i");
            WriteLine($"a = {a}");
            // gọi phương thức tĩnh FromString
            var b = Complex.FromString("5 + 3i");
            WriteLine($"b = {b}");
            // thực hiện phép cộng trên số phức
            WriteLine($"a + b = { a + b}");
            ReadKey();
        }
    }
}
```



Kết quả chạy chương trình với struct Complex

Trong project này bạn đã định nghĩa kiểu số phức Complex thuộc nhóm struct với các thành phần:

Real, Imaginary: các **trường dữ liệu** (biến thành viên) của Complex. Hai trường dữ liệu này được khai báo với từ khóa điều khiển truy cập `public`. Từ khóa này cho phép code bên ngoài struct sử dụng được biến thành viên.

Phương thức Parse: có tác dụng chuyển xâu ký tự về giá trị của trường Real và Imaginary. Phương thức này thuộc loại instance method, nghĩa là chỉ gọi từ object của Complex.

Hai phương thức Complex (trùng tên với struct) không có tên kiểu trả về là hai **hàm tạo** (constructor) của struct. Hàm tạo được sử dụng với phép toán `new` để khởi tạo object của Complex.

Phương thức tĩnh (static method) FromString: có cùng tác dụng như Parse nhưng gọi từ tên struct thay vì gọi từ object của struct.

Phương thức ToString: **ghi đè phương thức** ToString của lớp tổ tông Object, có tác dụng chuyển object của Complex thành chuỗi ký tự. Phương thức này tự động được gọi khi in ra console bằng lệnh Write/WriteLine hoặc trong phương thức Format của lớp string.

Operator+, Operator- : **ghi đè phép toán** + và – cho object của Complex. Nhờ hai phép toán ghi đè này bạn có thể viết biểu thức chứa phép toán + và – tương tự như với các kiểu số thông thường.

Modulus: là một **đặc tính** (property) của Complex, trả về giá trị module của số phức. Đây là đặc tính chỉ đọc.

Cú pháp khai báo struct

C# cho phép **khai báo kiểu** dữ liệu mới thuộc nhóm struct sử dụng từ khóa struct theo cú pháp sau:

```
[access_modifier] struct <type_name>
{
// Khai báo các thành viên
}
```

Access modifier là thành phần không bắt buộc và là một trong các từ khóa public hoặc internal có tác dụng điều chỉnh phạm vi sử dụng của kiểu. Giá trị mặc định là internal. Nếu dùng internal, kiểu chỉ có thể sử dụng nội bộ trong phạm vi project. Nếu để public, kiểu có thể được sử dụng bởi project khác. Nếu bạn xây dựng thư viện kiểu cho người khác sử dụng thì cần đặt từ khóa public. Nếu không chỉ rõ từ khóa, C# sẽ coi là internal.

`struct` là từ khóa bắt buộc dùng để chỉ định rằng kiểu đang khai báo thuộc nhóm struct.

Type_name (bắt buộc) là tên của kiểu đang được định nghĩa. Tên của kiểu cũng phải tuân thủ quy tắc đặt định danh của C#. Ngoài ra, tên kiểu nên tuân thủ quy ước viết PascalCase. Trong đó, ký tự đầu tiên của định danh luôn là chữ cái in hoa. Nếu định danh bao gồm nhiều từ ghép lại thì chữ cái đầu của mỗi từ cũng được viết hoa.

Như trong ví dụ trên:

```
struct Complex
{
// thân struct
}
```

Complex là tên struct, access modifier là `internal` (vì không viết gì)

Visual Studio cho phép thiết lập các quy ước này trong code style (Tools ➡️ Options ➡️ Text Editor ➡️ C# ➡️ Code Style). Nếu vi phạm quy ước, Visual Studio sẽ biểu thị chỗ lỗi bằng cách gạch chân.

Toàn bộ phần thân của khai báo kiểu struct phải là một khối code (đặt trong cặp dấu {}). Trong thân của struct chứa khai báo các thành viên.

Có một quy ước khác về nơi khai báo kiểu struct. Do struct là một cấu trúc phức tạp, bạn nên khai báo mỗi struct trong một file code riêng với tên file trùng với tên struct. Nếu có nhiều struct thuộc cùng nhóm, bạn nên đặt các file code vào cùng một folder. Đồng thời đặt tên namespace theo đúng cấu trúc folder của project. Cách làm này giúp đồng bộ giữa cấu trúc vật lý (file) và cấu trúc logic (namespace), giúp dễ dàng tìm và điều chỉnh code về sau.

Sử dụng struct

Một khi đã định nghĩa xong struct, bạn có thể sử dụng nó như bất kỳ kiểu dữ liệu nào trong C#.

Để khởi tạo object của struct, bạn cần dùng từ khóa new và gọi một trong số các hàm tạo.

```
// khai báo và khởi tạo biến a thuộc kiểu Complex
var a = new Complex(1, 2);
```

Từ object của struct bạn có thể truy xuất các thành viên public như đọc/gán giá trị cho biến/đặc tính, gọi phương thức, thực hiện các biểu thức.

```
// sử dụng đặc tính Modulus của Complex
WriteLine($"|a| = {a.Modulus}");
// gọi phương thức Parse
a.Parse("10-2i");
// thực hiện phép cộng trên số phức
WriteLine($"a + b = { a + b}");
```

Nếu trong struct có khai báo thành viên tĩnh (static member), bạn không cần khởi tạo object mà có thể gọi trực tiếp thành viên đó qua tên struct.

```
var b = Complex.FromString("5 + 3i");
```

Khai báo các thành viên cơ bản của struct

Biến thành viên

Biến thành viên (member variable) là thành phần chứa dữ liệu của struct trong C#. Nó được khai báo trực tiếp trong thân của struct (phải nằm trực tiếp trong khối code của thân struct).

Dữ liệu này có thể được sử dụng bởi bất kỳ thành phần nào khác của struct. Nói cách khác, biến thành viên có phạm vi tác dụng (scope) là toàn bộ thân struct, bất kể vị trí khai báo của biến. Tùy vào thiết lập, dữ liệu này cũng có thể được sử dụng bởi thành phần bên ngoài struct.

Biến thành viên được khai báo với cú pháp sau:

```
[access_modifier] <type> <variable_name>;
```

Tức là cú pháp khai báo biến thành viên giống hệt cú pháp khai báo biến cục bộ trong C#, ngoại trừ access_modifier.

Access modifier là hai từ khóa `public` hoặc `private` dùng để điều khiển truy cập vào biến thành viên. Biến public cho phép code bên ngoài struct sử dụng; Biến private chỉ cho phép sử dụng trong nội bộ struct.

Như trong ví dụ trên, Real và Imaginary là hai biến thành viên public thuộc kiểu double:

```
public double Real; // trường thực
public double Imaginary; // trường ảo
```

Tên biến bên cạnh tuân thủ quy tắc đặt định danh thì nên tuân theo một số quy ước khác. Tên biến thành viên public nên tuân theo cách viết PascalCase (giống như tên struct). Tên biến thành viên private nên bắt đầu bằng ký tự gạch chân và tiếp theo là chữ cái thường.

Phương thức thành viên (method)

Phương thức (method) là một thành viên của struct chịu trách nhiệm xử lý thông tin. Phương thức trong C# cho phép tái sử dụng code mà không phải viết lặp đi lặp lại nhiều lần. Do đó dễ hình dung phương thức là một khối code được đặt tên và chứa các lệnh để cùng thực hiện một nhiệm vụ cụ thể.

Phương thức của C# tương tự như hàm (function) và thủ tục (procedure) của Pascal, chương trình con Sub của visual basic, v.v.. Khác biệt lớn nhất là phương thức của C# bắt buộc phải là thành viên của một cấu trúc dữ liệu như struct hoặc class. Trong C# không có phương thức "tự do" hay "toàn cục".

Thực tế là bạn đã sử dụng (gọi) khá nhiều phương thức trong các bài học trước nhưng chưa học cách định nghĩa (khai báo) phương thức mới. Phương thức thành viên được khai báo với cú pháp sau:

```
[access_modifier] <return_type> <method_name>([parameters])
{
/* thân phương thức */
}
```

Như trong ví dụ trên chúng ta đã khai báo một số phương thức:

```
public void Parse(string value)
{
    var temp = value.Trim();
    if (temp.EndsWith("i") || temp.EndsWith("I"))
    {
        temp = temp.TrimEnd('i', 'I');
        var tokens = temp.Split(new[] { '+', '-' }, 2);
        Real = double.Parse(tokens[0]);
        Imaginary = double.Parse(tokens[1]);
    }
    else
    {
        Real = double.Parse(temp);
    }
}
public static Complex FromString(string value)
{
    var temp = new Complex();
    temp.Parse(value);
    return temp;
}
```

Access modifier của phương thức giống hệt như đối với biến thành viên và có cùng ý nghĩa.

Tên phương thức phải tuân thủ quy tắc đặt định danh, đồng thời cũng tuân theo quy ước viết PascalCase giống như đặt tên biến thành viên public.

Return type là kiểu của kết quả trả về của phương thức. Return type có thể là bất kỳ kiểu dữ liệu nào của C# và .NET. Nếu phương thức không trả về kết quả gì thì return type là từ khóa `void`. Nếu return type khác `void` thì trong thân phương thức bắt buộc phải có lệnh `return <value>` để trả giá trị lại cho nơi gọi. Nếu thiếu lệnh return C# sẽ báo lỗi và không biên dịch tiếp.

Parameters (danh sách tham số, còn gọi là danh sách tham số hình thức) là danh sách biến mà chúng ta có thể sử dụng trong phương thức. Danh sách tham số được định nghĩa theo cách sau:

```
(<kiểu_1>  <tham_số_1>, <kiểu_2>  <tham_số_2, …)
```

Hình dung một cách đơn giản, danh sách tham số chính là một chuỗi khai báo biến cục bộ viết tách nhau bởi dấu phẩy. Do đó, mỗi tham số đều tuân thủ quy tắc khai báo:

```
<kiểu_dữ_liệu> <tên_biến>
```

Danh sách tham số không bắt buộc phải có trong khai báo phương thức. Nếu danh sách tham số trống, bạn chỉ cần viết cặp dấu ngoặc tròn sau tên phương thức.

Bạn sẽ học kỹ hơn nữa về phương thức khi học khai báo class.

Hàm tạo (constructor)

Hàm tạo (constructor) là một loại phương thức đặc biệt giúp khởi tạo giá trị cho các thành viên của struct. Về hình thức, hàm tạo có tên trùng với struct và không chỉ định kiểu trả về. Tất cả những vấn đề khác, hàm tạo giống hệt như đối với phương thức.

```
public Complex(double r)
{
    Real = r;
    Imaginary = 0;
}
public Complex(double r, double i)
{
    Real = r;
    Imaginary = i;
}
```

Hàm tạo của struct bắt buộc phải có danh sách tham số. Bạn không thể viết hàm tạo không tham số cho struct.

Trong hàm tạo của struct bạn bắt buộc phải khởi tạo giá trị cho tất cả các trường của struct.

Bạn có thể không viết hàm tạo nào cho struct. Khi đó compiler sẽ tự động sinh cho bạn một hàm tạo không tham số. Trong trường hợp đó, khi khởi tạo object của struct, tất cả các biến thành viên sẽ nhận giá trị mặc định của kiểu.

Đặc tính (property)

Trong struct Complex bạn đã khai báo đặc tính Modulus như sau:

```
public double Modulus {
    get {
        return Math.Sqrt(Real * Real + Imaginary * Imaginary);
    }
}
```

Đặc tính (property) là một loại thành viên đặc biệt dùng để kiểm soát nhập/xuất dữ liệu cho struct. Property được sử dụng đặc biệt phổ biến trong struct và class của C#.

Cấu trúc chung nhất để khai báo thuộc tính như sau:

```
 [access_modifier] <type_name> <property_name>
 {
     [access_modifier] get { /* get method body */ };
     [access_modifier] set { /* set method body */ };
 } [= <value>];
```

Trong đó, tên thuộc tính (property name) được đặt theo quy tắc đặt định danh và quy ước giống như biến thành viên public.

Access modifier giống hệt như của biến và phương thức.

Hai phương thức get và set được gọi chung là accessor. Mỗi phương thức get hoặc set có thể sử dụng từ khóa điều khiển truy cập của riêng mình, giúp property đó biến thành loại:

- chỉ đọc (read-only): public get, private set;
- chỉ gán (assign-only): private get, public set;
- truy cập tự do (full access): public get, public set (mặc định).

Về bản chất, get và set là hai phương thức nhưng không có danh sách tham số. Trong phương thức set có thể sử dụng từ khóa `value` để đại diện cho dữ liệu gán vào cho property. Trong phương thức get phải có lệnh `return` để trả giá trị. Trong property có thể vắng mặt một trong hai phương thức get hoặc set.

Khi sử dụng, property được truy cập giống hệt như biến thành viên.

```
var a = new Complex(1, 2);
// sử dụng đặc tính Modulus của Complex
WriteLine($"|a| = {a.Modulus}");
```

Sau khi học khai báo class bạn sẽ học chi tiết hơn nữa về đặc tính trong C#.

Các thành phần khác

Trong struct bạn có thể **ghi đè phương thức** (method override) kế thừa từ lớp Object của .NET. Trong ví dụ trên bạn đã ghi đè phương thức ToString của Object, dùng để chuyển giá trị sang chuỗi ký tự.

```
public override string ToString()
{
    if (Imaginary == 0)
    {
        return Real.ToString();
    }
    return $"{Real} {(Imaginary > 0 ? '+' : '-')} {Math.Abs(Imaginary)}i";
}
```

Bạn cũng có thể nạp chồng các toán tử (operator overloading) cho struct để sử dụng toán tử tương ứng cho object của struct. Trong ví dụ trên, bạn đã nạp chồng hai phép toán cộng (+) và trừ (-) để có thể thực hiện các phép toán tương ứng trên object của Complex.

```
public static Complex operator +(Complex a, Complex b)
{
    return new Complex(a.Real + b.Real, a.Imaginary + b.Imaginary);
}
public static Complex operator -(Complex a, Complex b)
{
    return new Complex(a.Real - b.Real, a.Imaginary - b.Imaginary);
}
```

Các thành phần này (và một số thành phần khác chưa được nhắc tới) sẽ được xem xét rất chi tiết trong các nội dung của phần lập trình hướng đối tượng và xây dựng class.