Kiểu chuỗi ký tự (string) trong C#, lớp String và StringBuilder

Chuỗi ký tự (string) là kiểu dữ liệu phổ biến hàng đầu trong C# và các ngôn ngữ lập trình. Trong hầu hết các bài học từ đầu đến giờ bạn đều đụng chạm đến chuỗi ký tự. Trong bài học về kiểu dữ liệu cơ sở trong C# bạn đã biết sơ lược về kiểu string (System.String). Bài học này sẽ hướng dẫn chi tiết cách sử dụng kiểu string trong lập trình C#, bao gồm các hàm xử lý chuỗi, lớp StringBuilder để tạo chuỗi, định dạng chuỗi.

NỘI DUNG CỦA BÀI [Ẩn]

Kiểu string trong C# và lớp System.String của .NET

Như bạn đã biết, kiểu chuỗi ký tự trong C# – `string` – thực chất là một alias của lớp `System.String` của .NET. Alias `string` trong C# là một từ khóa và giúp đơn giản hóa làm việc với chuỗi. Nếu có lệnh using System; ở đầu file code, bạn có thể sử dụng tên ngắn gọn của class `System.String` là `String`.

> Đây cũng là câu trả lời cho vấn đề nhiều bạn thắc mắc: `String` và `string` có gì khác nhau. Chả có gì khác cả. Chúng nó là một. C# khuyên nên dùng `string` .

String (hoặc string) cho phép lưu trữ chuỗi ký tự Unicode và cung cấp một số lượng kha khá các phương thức để xử lý chuỗi. C# cũng có một số phép toán đặc biệt trên chuỗi.

> Tất cả các ví dụ trong bài này dùng C# Interactive.

Khởi tạo chuỗi

Trong bài học về kiểu dữ liệu cơ sở của C# bạn đã biết cách khởi tạo chuỗi sử dụng **string literal**: `string str = "Hello world";` Đây là cách khởi tạo chuỗi "tự nhiên" nhất. Tuy nhiên, C# cung cấp một số cách khác để khởi tạo chuỗi.

Hàm tạo của lớp String có 8 overload khác nhau giúp bạn tạo ra chuỗi theo ý muốn.

```
>//Khởi tạo string từ mảng ký tự
> char[] letters = { 'H', 'e', 'l', 'l', 'o' };
. string greetings = new string(letters);
> greetings
"Hello"

>//Khởi tạo string từ một phần của mảng ký tự
> char[] letters = { 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'};
> letters
char[11] { 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd' }
> string greeting = new string(letters, 0, 5);
> greeting
"Hello"

>//Khởi tạo string bằng cách lặp ký tự
> string aaa = new string('a', 10);
> aaa
"aaaaaaaaaa"

>//Khởi tạo chuỗi rỗng
> string empty = string.Empty;
> empty
""
```

Xin nhắc lại một số vấn đề về chuỗi ký tự trong C#.

Trong chuỗi không được có mặt ký tự \ (backslash). Lý do là ký tự này được sử dụng trong escape sequence. Ví dụ, dưới đây là một chuỗi sai (bị báo lỗi cú pháp):

```
string path = "C:\Programs\Visual Studio"; // chuỗi này bị lỗi vì chứa ký tự \.
```

Nếu muốn viết ký tự \ vào chuỗi, bạn phải viết nó hai lần:

```
string path = "C:\\Program\\Visual Studio"; // chuỗi này OK
```

hoặc thêm ký tự @ vào đầu chuỗi. Ký tự @ sẽ tắt chế độ diễn giải escape sequence.

```
string path = @"C:\Program\Visual Studio"; // chuỗi này OK vì ký tự @ sẽ tắt chế
độ nhận diện escape sequence
```

Các phép toán trên kiểu chuỗi

Phép cộng xâu

Phép toán + áp dụng trên xâu ký tự được gọi là phép cộng xâu hay phép ghép chuỗi. Nó cho phép ghép nhiều chuỗi ký tự nhỏ thành một chuỗi ký tự lớn. Ví dụ:

```
> string str1 = "Hello " + "world";
> str1
"Hello world"

> string hello = "Hello", world = "world";
. string str2 = hello + " " + world;
> str2
"Hello world"

> string str3 = "Hello";
. str3 += " ";
. str3 += "world";
> str3
"Hello world"
```

Trong 3 ví dụ trên, ví dụ đầu tiên minh họa phép cộng 2 xâu, ví dụ thứ hai – cộng nhiều xâu, ví dụ thứ ba – sử dụng phép cộng gán trên xâu.

Riêng ví dụ thứ 3 cần lưu ý: phép cộng gán này không hề thay đổi giá trị gốc của str3. Trên thực tế, mỗi lần thực hiện phép toán này, một xâu mới được tạo ra và gán trở lại cho str3. Cái này liên quan đến một đặc điểm quan trọng của string: đây là kiểu dữ liệu **immutable**. Mọi biến đổi trên string đều tạo ra một object mới chứ không thay đổi giá trị của object cũ. Do vậy, việc thực hiện quá nhiều phép biến đổi tương tự trên xâu rất không hiệu quả. C# cung cấp class StringBuilder để giải quyết vấn đề này. Bạn sẽ xem xét StringBuilder ở phần sau của bài học này.

Truy xuất ký tự trong chuỗi

Chuỗi ký tự trong C# có thể hình dung giống như một mảng của các ký tự. Do đó bạn cũng có thể sử dụng phép toán indexer để truy xuất từng ký tự trong chuỗi tương tự như truy xuất phần tử của mảng.

```
> var str4 = "Hello world";
> str4[0]
'H'
> str4[1]
'e'
>
```

Trong ví dụ trên bạn dễ dàng nhận thấy 'H' là phần tử số 0, 'e' là phần tử số 1. Dùng phép toán indexer có thể truy xuất các ký tự này giống như truy xuất một mảng của các ký tự.

Cần lưu ý rằng, phép toán indexer này cho kết quả thuộc loại read-only (chỉ đọc). Nghĩa là bạn không thể thay đổi ký tự ở vị trí đó. Trong ví dụ sau

```
> str4[2] = 'L'
(1,1): error CS0200: Property or indexer 'string.this[int]' cannot be assigned to -- it
>
```

Nếu bạn cố tình gán giá trị 'L' cho ký tự ở vị trí số 2 như trên thì sẽ dính lỗi ngay.

Property và method của lớp string

Lớp string cung cấp một số property (đặc tính) và method (phương thức) giúp làm việc với chuỗi.

> Property là loại method đặc biệt dùng để xuất nhập dữ liệu. Bạn có thể đọc thêm ở bài học về Property trong C#.

Length property

Đặc tính **Length** trả về số ký tự trong xâu. Đây là property duy nhất của lớp string.

```
> string greeting = "Hello world";
> greeting.Length
11
```

Đây là một property chỉ đọc. Bạn không thể gán giá trị cho property này.

Instance method của lớp string

Dưới đây là một số **instance method** thông dụng của lớp string.

> *Instance method* là các phương thức gọi từ object, phân biệt với *static method* là phương thức gọi trực tiếp từ tên class (không gọi từ object). Bạn sẽ học về hai loại method này sau.

bool Contains(string value): kiểm tra xem xâu có chứa xâu con value hay không

```
> string greeting = "Hello world";
> greeting.Contains("world")
true
```

---

bool EndsWith(string value): kiểm tra xem xâu con value có nằm ở cuối chuỗi hay không

```
> string greeting = "Hello world";
> greeting.EndsWith("ld")
true
```

StartsWith hoạt động tương tự nhưng kiểm tra xem xâu con value có nằm ở đầu chuỗi hay không.

---

bool Equals(string value): so sánh với xâu value

```
> string greeting = "Hello world";
> string hello = "Hello";
> greeting.Equals(hello)
false
>
```

---

string ToLower(): tạo bản sao của chuỗi nhưng mọi chữ cái hoa chuyển thành chữ cái thường

```
> string greeting = "Hello world";
> string lower = greeting.ToLower();
> lower
"hello world"
```

---

string ToUpper(): tạo bản sao của chuỗi nhưng mọi chữ cái thường chuyển thành chữ cái hoa

```
> string greeting = "Hello world";
> string upper = greeting.ToUpper();
> upper
"HELLO WORLD"
```

---

string Trim(): tạo bản sao của chuỗi nhưng cắt bỏ hết khoảng trắng ở đầu và cuối. Khoảng trắng là các ký tự như space, tab, \r, \n

```
> string greeting = "     Hello world\r\n   ";
> string trim = greeting.Trim();
> trim
"Hello world"
```

Tương tự, **TrimEnd** chỉ cắt bỏ khoảng trắng ở cuối chuỗi, **TrimStart** chỉ cắt bỏ khoảng trắng ở đầu.

Ngoài ra Trim, TrimEnd, TrimStart còn có thể cắt bỏ những ký tự khác theo yêu cầu:

```
Cắt bỏ hết các chữ H và d ở đầu và cuối chuỗi
> string greeting = "Hello world";
> string sub = greeting.Trim(new[] { 'H', 'd' });
> sub
"ello worl"
>
```

---

int IndexOf(string value): Xác định vị trí của xâu con value. Nếu xâu con value xuất hiện nhiều lần, IndexOf trả lại vị trí đầu tiên bắt gặp.

```
> string greeting = "Hello world";
> int pos = greeting.IndexOf("wor");
> pos
6
```

Tương tự **LastIndexOf** cũng dùng để xác định vị trí của xâu con. Tuy nhiên, nếu xâu con xuất hiện nhiều lần, LastIndexOf sẽ trả về vị trí bắt gặp cuối cùng. Nếu xâu con chỉ xuất hiện một lần, IndexOf và LastIndexOf trả về cùng một kết quả.

Ngoài ra, IndexOf và LastIndexOf cũng có thể tìm kiếm vị trí của một ký tự trong xâu, có thể yêu cầu tìm kiếm từ một vị trí xác định (thay vì tìm từ đầu chuỗi).

---

int IndexOfAny(char[] anyOf): xác định vị trí bắt gặp đầu tiên của một ký tự trong nhóm

```
> string hello = "Hello world";
> int pos = hello.IndexOfAny(new[] { 'l', 'o' });
> pos
2
> // 2 là số thứ tự của ký tự 'l' đầu tiên gặp trong xâu "Hello world"
```

Tương tự, **IndexOfAny** cũng cho phép giới hạn vị trí bắt đầu tìm kiếm, thay vì tìm từ đầu chuỗi.

---

string Insert(int startIndex, string value): tạo một bản sao của chuỗi với một xâu con value chèn vào vị trí startIndex

```
> string hello = "Hello world";
> var insert = hello.Insert(6, "there ");
> insert
"Hello there world"
> //"there " được chèn vào vị trí số 6 của xâu "Hello world"
```

---

string Remove(int startIndex, int count): tạo ra bản sao của chuỗi nhưng bỏ đi count ký tự từ vị trí startIndex

```
> string hello = "Hello world";
> string remove = hello.Remove(5, 6);
> remove
"Hello"
> //cắt bỏ 6 ký tự từ vị trí số 5
```

Nếu bỏ tham số count thì sẽ xóa đi tất cả các ký tự từ vị trí startIndex.

---

string Replace(string oldValue, string newValue): tạo bản sao của chuỗi trong đó thay thế một chuỗi con bằng chuỗi con khác

```
> string replaced = "Hello world".Replace("world", "baby!");
> replaced
"Hello baby!"
>
```

Overload **string Replace(char oldChar, char newChar)** hoạt động tương tự nhưng thay thế ký tự này bằng ký tự khác.

---

string[] Split(params char[] separator): cắt chuỗi thành nhiều chuỗi con sử dụng một nhóm ký tự đánh dấu

```
> // cắt chuỗi thành các chuỗi con sử dụng dấu cách làm ký tự đánh dấu
> string[] result = "Hello world from C#".Split(new[] { ' ' });
> result
string[4] { "Hello", "world", "from", "C#" }
> // kết quả thu được là 4 từ riêng rẽ
```

Tương tự cũng có thể sử dụng một nhóm xâu đánh dấu thay cho ký tự, chỉ định số lượng chuỗi con tối đa.

---

char[] ToCharArray(): chuyển đổi toàn bộ chuỗi thành mảng char[]

char[] ToCharArray(int startIndex, int length): chuyển đổi một phần chuỗi (length ký tự, tính từ vị trí startIndex) thành mảng char[]

void CopyTo(int sourceIndex, char[] destination, int destinationIndex, int count): Sao chép một số ký tự sang một mảng char[]

Static method của lớp string

Dưới đây là danh sách các phương thức static – phương thức gọi trực tiếp từ class string, không gọi từ object.

static int Compare(string strA, string strB): so sánh hai xâu strA và strB. Nếu strA bằng strB trả về 0; strA > strB – trả về 1; strA < strB – trả về -1.

```
> string.Compare("Hello", "Hello")
0
> string.Compare("Hello", "hello")
1
> string.Compare("hello", "Hello")
-1
> string.Compare("hello world", "Hello")
1
>
```

Để dễ hình dung, việc so sánh này giống như sắp xếp tên theo thứ tự abc. Kết quả +1 biểu thị chuỗi thứ nhất sẽ đứng sau chuỗi thứ hai; Kết quả -1 biểu thị chuỗi thứ nhất sẽ đứng trước chuỗi thứ hai.

---

static bool Equals(string a, string b): xác định xem chuỗi a và b có giá trị bằng nhau hay không

```
> string.Equals("Hello", "Hello")
true
> var str1 = "Hello";
> var str2 = "Hello";
> string.Equals(str1, str2)
true
>
```

Cùng là so sánh chuỗi nhưng Equals chỉ xác định "bằng" hay "khác", không giống như Compare ở bên trên.

---

static string Concat(params string[] values): ghép nối nhiều chuỗi con thành một chuỗi lớn. Số lượng chuỗi con không giới hạn.

```
> string greeting = string.Concat("Hello ", "world ", "from ", "C#");
> greeting
"Hello world from C#"
>
```

Concat cũng cho phép ghép các object thành chuỗi. Khi ghép object, phương thức ToString() của object đó sẽ được gọi tự động để chuyển đổi nó thành chuỗi.

Ngoài ra, nếu bạn có một mảng (kiểu phần tử bất kỳ), bạn cũng có thể dùng Concat để ghép chúng thành chuỗi.

---

static string Join(string separator, params string[] value): hoạt động giống Concat nhưng tự động chèn thêm chuỗi separator vào giữa các chuỗi con.

```
> var greeting = string.Join("|", "Hello", "world", "from", "C#");
> greeting
"Hello|world|from|C#"
>
```

---

static string Copy(string str): tạo bản sao của một chuỗi.

```
> var hello = "Hello";
> var world = string.Copy(hello);
> world
"Hello"
>
```

Bạn cần lưu ý rằng, Copy tạo ra một object bản sao, nghĩa là hai biến xâu trỏ vào hai object khác biệt (chỉ là có giá trị bằng nhau). Nó khác biệt với phép gán xâu, khi biến mới cùng trỏ vào object ban đầu.

---

static bool IsNullOrEmpty(string value): kiểm tra xem chuỗi value là null hoặc là một chuỗi rỗng

```
> string str = string.Empty;
> string str2;
> string.IsNullOrEmpty(str)
true
> string.IsNullOrEmpty(str2)
true
> string str3 = "Hello world";
> string.IsNullOrEmpty(str3)
false
>
```

---

static string Format(string format, Object arg0): giúp tạo ra xâu "động" từ giá trị của các biến. Cách sử dụng của Format rất giống phương thức Write/WriteLine với placeholder mà bạn đã biết trong bài học về Console trong C#.

```
> var x1 = 1.234;
> var x2 = 5.678;
> var str = string.Format("Nghiệm của phương trình là: {0} và {1}", x1, x2);
> str
"Nghiệm của phương trình là: 1.234 và 5.678"
>
```

Trong ví dụ trên, vị trí đánh dấu (placeholder) được biểu diễn bằng cụm {0} và {1}. Trong đó {0} sẽ được thay thế bởi biến thứ nhất trong danh sách (tức là x1); {1} sẽ được thay thế bởi biến thứ hai trong danh sách (tức là x2).

Như vậy có thể để ý, các biến trong danh sách được đánh số thứ tự từ 0, và sẽ được thay thế vào vị trí đánh dấu có chỉ số tương ứng.

> Write/WriteLine với placeholder chỉ là một cách làm tắt. Trên thực tế Write/WriteLine đều tự động gọi tới phương thức Format để định dạng chuỗi trước khi in ra console.

Chúng ta sẽ xem xét chi tiết cách dùng Format trong phần định dạng chuỗi.

Định dạng chuỗi trong C#, phương thức string.Format()

Format() là phương thức rất mạnh giúp bạn tạo chuỗi sử dụng giá trị từ các biến và định dạng cụ thể cho giá trị. Để định dạng cho giá trị trong xâu, bạn cần biết một số "cú pháp" – chuỗi định dạng – đặc biệt. Chuỗi định dạng là một nhóm ký tự viết theo những quy tắc nhất định để phương thức Format() hiểu được ý định của bạn.

Chúng ta cùng xem xét một số trường hợp phổ biến.

Định dạng cho số

Trong nhiều tình huống bạn cần định dạng số khi tạo chuỗi. Ví dụ trong chuỗi chứa giá tiền, bạn muốn kèm theo đơn vị tiền tệ và chỉ với 2 chữ số thập phân. Bạn cũng có thể muốn dành một độ rộng cố định để sau in số ra console (như trong bảng biểu). Bạn muốn số in ra căn lề trái hoặc căn lề phải.

Định dạng cho mỗi số bao gồm 3 phần: alignment, format và precision.

Format specifier

Ví dụ, để định dạng giá tiền, bạn dùng **:C** hoặc **:c** phía sau chỉ số trong placeholder, hoặc sau tên biến trong interpolated string. Đơn vị tiền tệ sẽ phụ thuộc vào cấu hình ngôn ngữ của windows.

```
1.  > string.Format("The value: {0}", 500)
2.  The value: 500
3.  > string.Format("The value: {0:C}", 500)
4.  The value: $500.00
5.  >  string.Format ("The value: {0:c}", 500)
6.  The value: $500.00
7.  > decimal value = 500;
```

C hoặc c là định dạng số chuẩn dành cho kiểu tiền tệ (currency). Ngoài C (c), bạn có thể gặp thêm các trường hợp khác: D, d – Decimal; F, f – Fixed point; G, g – General; X, x – Hexadecimal; N, n – Number; P, p – Percent; R, r – Round-trip; E, e – Scientific.

Các ký tự định dạng số chuẩn này phải đi ngay dấu hai chấm, ví dụ :C, :c, :P, :p, như bạn đã thấy ở trên.

Precision specifier

Sau ký tự định dạng bạn có thể sử dụng thêm một con số để mô tả độ chính xác (Precision specifier) của giá trị được in ra.

Ví dụ, mặc định :C sẽ in ra hai chữ số thập phân. Bạn có thể yêu cầu in ra con số với độ chính xác cao hơn (3-4 chữ số thập phân) hoặc thấp hơn. Chẳng hạn **:c3** chỉ định in ra 3 chữ số thập phân, **:c4** chỉ định in 4 chữ số thập phân.

```
> string.Format("The value: {0:c3}", 500);
The value: $500.000
>  string.Format ("The value: {0:c4}", 500);
The value: $500.0000
```

Cách thể hiện của "độ chính xác" phụ thuộc vào loại số và định dạng của nó. Bạn hãy xem ví dụ sau đây:

```
> string.Format("{0 :C}", 12.5);
$12.50
> string.Format("{0 :D4}", 12);
0012
> string.Format("{0 :F4}", 12.3456789);
12.3457
> string.Format("{0 :G4}", 12.3456789);
12.35
> string.Format("{0 :x}", 180026);
2bf3a
> string.Format("{0 :N2}", 12345678.54321);
12,345,678.54
> string.Format("{0 :P2}", 0.1221897);
12.22%
> string.Format("{0 :e4}", 12.3456789);
1.2346e+001
>
```

Alignment specifier

Để dễ hiểu, hãy xem ví dụ sau:

```
> int myInt = 500;
> string.Format("|{0, 10}|", myInt);
. string.Format("|{0,-10}|", myInt);
|       500|
|500       |
>
```

Ở đây chúng ta mô phỏng lại việc in giá trị ra thành cột. Con số 10 và -10 viết tách với chỉ số placeholder bằng dấu phẩy được gọi là **alignment specifier**. Alignement specifier chỉ định độ rộng (số lượng ký tự) tối thiểu để in giá trị số đó. Giá trị dương báo hiệu căn lề phải; Giá trị âm báo hiệu căn lề trái.

Format specifier mà bạn đã biết viết về phía phải của alignment specifier như dưới đây:

```
> double myDouble = 12.345678;
> string.Format("{0,-10:G} -- General", myDouble)
"12.345678  -- General"
> string.Format("{0,-10} -- Default, same as General", myDouble)
"12.345678  -- Default, same as General"
> string.Format("{0,-10:F4} -- Fixed Point, 4 dec places", myDouble)
"12.3457    -- Fixed Point, 4 dec places"
> string.Format("{0,-10:E3} -- Sci. Notation, 3 dec places", myDouble)
"1.235E+001 -- Sci. Notation, 3 dec places"
> string.Format("{0,-10:x} -- Hexadecimal integer", 1194719)
"123adf     -- Hexadecimal integer"
>
```

Định dạng thời gian

C# cung cấp kiểu dữ liệu DateTime để lưu trữ thông tin về thời gian.

Tương tự như đối với số, bạn cũng có thể định dạng thời gian khi tạo xâu với Format(). Quy tắc viết định dạng cho thời gian giống hệt như đối với số. Khác biệt là bạn cần sử dụng format specifier riêng cho thời gian:

- d – chỉ in thông tin về ngày ở dạng ngắn gọn;
- D – chỉ in thông tin về ngày ở dạng đầy đủ;
- t – chỉ in thông tin về thời gian ở dạng ngắn gọn;
- T – chỉ in thông tin về thời gian ở dạng đầy đủ.

Ví dụ:

```
> var day = new DateTime(2025, 2, 14);// tạo object chứa ngày 14 tháng 2 năm 2025
> string.Format("A future day: {0}", day)
"A future day: 2/14/2025 12:00:00 AM"
> string.Format("A future day: {0:d}", day)
"A future day: 2/14/2025"
> string.Format("A future day: {0:D}", day)
"A future day: Friday, February 14, 2025"
> string.Format("A future day: {0,-20:d}", day)
"A future day: 2/14/2025           "
> string.Format("A future day: {0,20:d}", day)
"A future day:            2/14/2025"
> string.Format("A future time: {0,20:t}", day)
"A future time:              12:00 AM"
> string.Format("A future time: {0,20:T}", day)
"A future time:          12:00:00 AM"
>
```

Cách hiển thị cụ thể phụ thuộc vào thiết lập vùng và ngôn ngữ của hệ điều hành windows bạn đang dùng. Trong ví dụ trên, windows đang thiết lập là tiếng Anh-Mỹ.

Bạn cũng có thể tự đưa ra định dạng ngày tháng riêng độc lập khỏi hệ điều hành như sau:

```
> string.Format("Một ngày nào đó trong tương lai: {0:dd-MM-yyyy}", day)
"Một ngày nào đó trong tương lai: 14-02-2025"
> string.Format("Một ngày nào đó trong tương lai: {0:dd-MMM-yyyy}", day)
"Một ngày nào đó trong tương lai: 14-Feb-2025"
> string.Format("Một ngày nào đó trong tương lai: {0, 25:dd/MM/yyyy}", day)
"Một ngày nào đó trong tương lai:                14/02/2025"
```

Trong loại định dạng tự do này bạn có thể dùng các ký tự d thay cho ngày, M thay cho tháng, y thay cho năm. Số lượng ký tự có ý nghĩa riêng. Ví dụ:

```
> string.Format("Một ngày nào đó trong tương lai: {0, 25:dd/MMMM/yyyy}", day)
"Một ngày nào đó trong tương lai:          14/February/2025"
> string.Format("Một ngày nào đó trong tương lai: {0, 25:d/MMMM/yyyy}", day)
"Một ngày nào đó trong tương lai:          14/February/2025"
> string.Format("Một ngày nào đó trong tương lai: {0, 25:ddd/MMMM/yyyy}", day)
"Một ngày nào đó trong tương lai:         Fri/February/2025"
> string.Format("Một ngày nào đó trong tương lai: {0, 25:ddd/M/yyyy}", day)
"Một ngày nào đó trong tương lai:                Fri/2/2025"
```

String interpolation

String interpolation là một khả năng tạo chuỗi động từ giá trị của biến mới đưa vào trong C# 6. String interpolation có lối viết giản dị và dễ đọc hơn nhiều so với Format().

```
> string s1 = "World";
> string s2 = $"Hello, {s1}";
> s2
```

```
4.  "Hello, World"
```

String interpolation phải bắt đầu bằng ký tự $, theo sau là string literal thông thường. Trong chuỗi ký tự, ở đâu cần thay bằng giá trị của biến/biểu thức thì đặt tên biến/biểu thức trong cặp dấu ngoặc nhọn {}.

Như trong ví dụ trên bạn đã thấy, cách viết này rất dễ đọc và dễ hiểu.

> Trên thực tế, string interpolation cũng chỉ là một dạng cú pháp tắt của Format(). Nếu gặp ký tự $ trước xâu, compiler sẽ tự động gọi đến phương thức Format.
> Ví dụ trên sẽ chuyển thành string.Format("Hello, {0}", s1);
> Đây cũng là lý do chúng ta xem xét rất chi tiết phương thức Format() ở trên.

Đây cũng là lý do chúng ta xem xét rất chi tiết phương thức Format() ở trên.

Bên trong placeholder {} bạn có thể đặt bất kỳ biến, biểu thức, lời gọi phương thức nào, miễn là nó trả về giá trị.

```
1.  > string world = "world";
2.  > string greeting = $"Hello, {world.ToUpper()}";
3.  > greeting
4.  "Hello, WORLD"
```

Trong chuỗi interpolation, nếu bạn muốn viết ký tự { thì bạn phải viết nó hai lần:

```
1.  > string s = "Hello";
2.  > string s2 = $"{{s}} displays the value of s: {s}";
3.  > s2
4.  "{s} displays the value of s: Hello"
```

Bạn thậm chí có thể viết các biểu thức phức tạp bên trong chuỗi interpolation:

```
1.  > int a = 10, b = 20;
2.  > string str = $"Giá trị lớn hơn là: {(a > b ? a : b)}";
3.  > str
4.  "Giá trị lớn hơn là: 20"
```

Khi này lưu ý đặt biểu thức trong cặp dấu ngoặc tròn ().

String interpolation sử dụng cách định dạng số và ngày tháng giống hệt như của phương thức Format():

```
1.  > decimal d = 500;
2.  > string str = $"Giá tiền là: {d,10:c2}";
3.  > str
4.  "Giá tiền là:    $500.00"
```

Kết luận

Bài học này đã cung cấp cho bạn đầy đủ những thông tin cần thiết để làm việc với chuỗi ký tự trong C#. Đây là kiểu dữ liệu phổ biến hàng đầu. Hầu như trong mọi chương trình bạn đều sẽ phải xử lý chuỗi ký tự. Do đó, mặc dù đây là một bài học rất dài, hi vọng bạn sẽ nắm bắt tốt nó.

+ Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
+ Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
+ Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
Cảm ơn bạn!