Biến và hằng trong C#

Biến trong c# hay bất kỳ ngôn ngữ lập trình nào được dùng để lưu trữ thông tin tạm thời để sau tái sử dụng. Tương tự, hằng cũng được dùng để lưu trữ thông tin cho tái sử dụng. Thông tin chứa trong biến có thể thay đổi, còn hằng thì không. Những vấn đề cơ bản này chắc chắn bạn đã nắm rất rõ khi học nhập môn lập trình, dù là bằng ngôn ngữ nào cũng vậy.

Cách khai báo và sử dụng biến/hằng trong C# rất gần gũi với C/C++ hay Java (và các ngôn ngữ dùng cú pháp của C). Tuy nhiên, có một số điểm khác biệt nhất định phải biết.

C# cung cấp một số loại biến khác nhau:

- Biến cục bộ: lưu trữ thông tin trong phạm vi phương thức.
- Biến thành viên: lưu trữ thông tin trong phạm vi class hoặc struct.
- Tham số: tạm thời lưu trữ thông tin để truyền vào phương thức.

Bài học này sẽ cung cấp những thông tin đầy đủ và chi tiết về biến cục bộ trong C#. Biến thành viên sẽ xem xét khi học về class. Tham số sẽ xem xét khi học chi tiết về phương thức.

NỘI DUNG CỦA BÀI [Ẩn]

> Để thử nghiệm code trong bài, bạn có thể sử dụng C# interactive cho tiện lợi.

Khai báo biến trong C#

Biến trong C# được **khai báo** với cú pháp sau:

```
datatype identifier;
```

Trong đó *datatype* là **tên kiểu** dữ liệu mà biến đó có thể lưu giữ, *identifier* là **định danh** (tên) của biến.

Ví dụ sau khai báo biến có tên là i, lưu giữ được các giá trị số nguyên (int).

```
int i;
```

Tuy nhiên, trong C#, lệnh khai báo biến trên mặc dù đúng cú pháp nhưng compiler lại không cho phép bạn dùng ngay biến i trong các biểu thức. **C# bắt buộc biến phải được gán giá trị trước khi sử dụng**.

Để **gán giá trị** cho biến bạn dùng **phép gán** (assignment operator):

```
i = 10;
```

Phép gán trong C# đơn giản chỉ là một dấu bằng (=).

Có thể **kết hợp cả khai báo biến và gán giá trị** vào cùng một lệnh:

```
int i = 10;
```

Có thể **khai báo và gán giá trị cho nhiều biến cùng kiểu** trong cùng một lệnh:

```
int x = 10, y = 20; // x và y có cùng kiểu int
```

Nếu các biến khác kiểu, bạn bắt buộc phải khai báo trong các lệnh khác nhau:

```
int x = 10;
bool y = true; // biến chứa giá trị logic true/false

// lệnh khai báo dưới đây là SAI
int x = 10, bool y = true; // compiler sẽ báo lỗi ở dòng này
```

Biến cục bộ trong C# được đặt tên theo quy tắc đặt định danh, đồng thời nên tuân thủ quy ước **camelCase**.

> Quy ước camelCase: tên bắt đầu bằng chữ cái thường; nếu tên bao gồm nhiều từ, chữ cái đầu mỗi từ tiếp theo sẽ viết hoa. Ví dụ: camelCase, bigInteger, complexNumber.

Khởi tạo biến trong C#

Trình biên dịch C# bắt buộc mọi biến phải được **khởi tạo** với một giá trị nào đó trước khi sử dụng trong biểu thức. Đây là một ví dụ về sự chú trọng tới sự an toàn trong C#. Trong khi các ngôn ngữ khác thường chỉ coi việc sử dụng biến mà không gán giá trị trước là một dạng cảnh báo (warning), C# coi đây là một lỗi.

> Các bạn có thể gặp các từ *khởi tạo* (initialize) biến hay *gán* giá trị (assign) đôi khi được sử dụng lẫn lộn. Trong C#, đối với biến thuộc các kiểu như int, bool thì khởi tạo hay gán là như nhau. Đối với các kiểu dữ liệu tham chiếu (như class sẽ học trong phần lập trình hướng đối tượng), khởi tạo và gán giá trị là các lệnh khác nhau.

C# có hai phương pháp để đảm bảo biến được khởi tạo trước khi sử dụng.

Nếu biến là một trường dữ liệu trong class hoặc struct (bạn sẽ học sau): nếu không được lập trình viên trực tiếp gán giá trị, C# compiler sẽ tự động gán cho biến một giá trị mặc định tùy từng kiểu dữ liệu (ví dụ 0 cho số nguyên).

Nếu biến nằm trong thân phương thức (gọi là biến cục bộ – local variable): bắt buộc lập trình viên phải trực tiếp gán giá trị trước khi sử dụng nó trong biểu thức. Trong trường hợp này, biến không nhất thiết phải được khởi tạo ngay khi khai báo. Miễn sao nó phải có giá trị trước khi sử dụng là được. Vi phạm này bị C# coi là lỗi và compiler sẽ dừng lại.

Dưới đây là một ví dụ LỖI về sử dụng biến không khởi tạo:

```
static void Main()
{
    int d;
    Console.WriteLine(d); // Sẽ báo lỗi ở đây. Biến d chưa có giá trị.
}
```

Nếu biên dịch đoạn code trên bạn sẽ gặp lỗi `Use of unassigned local variable 'd'`.

Suy luận kiểu của biến, từ khóa var

Trong C# có một cách khác để khai báo và khởi tạo biến khác biệt với các ngôn ngữ kiểu C (nhưng lại nhìn giống JavaScript!): tự suy luận kiểu với từ khóa `var`.

```
var i = 0;
```

Trong cách khai báo (và khởi tạo) này, tên kiểu được thay bằng từ khóa `var`. C# compiler khi gặp dòng lệnh này sẽ tự "suy đoán" ra kiểu của biến dựa vào giá trị gán cho nó. Nghĩa là dòng lệnh trên được C# tự động hiểu thành:

```
int i = 0;
```

Kết quả biên dịch của cả hai lệnh trên là như nhau.

> Khai báo và khởi tạo biến với từ khóa var hiện được ưa thích hơn so với kiểu khai báo "truyền thống". Trong một số tình huống (sẽ gặp sau) bạn thậm chí không thể sử dụng được kiểu khai báo biến thông thường mà bắt buộc phải dùng var.

Khi khai báo biến với từ khóa `var` bạn phải tuân thủ quy tắc: Biến phải được khởi tạo lúc khai báo. Nếu không, compiler sẽ không có căn cứ gì để suy đoán cả.

> Ngoài ra còn một số quy tắc quan trọng nữa bạn sẽ gặp khi làm việc với object và class.

Phạm vi của biến trong C#

Phạm vi (scope), còn gọi là phạm vi tác dụng, của biến là vùng code mà các lệnh trong đó có thể truy xuất biến. Phạm vi được xác định theo quy tắc sau:

Biến (trong thân phương thức) có thể truy xuất từ vị trí khai báo đến khi gặp dấu } báo hiệu kết thúc của nhóm lệnh. Một nhóm lệnh đặt trong cặp dấu { } như vậy gọi là một **khối code** (code block). Nói cách khác, biến có phạm vi tác dụng là khối code mà nó được khai báo. Ra khỏi khối code này, biến không sử dụng được nữa.

Các khối code có thể nằm lồng nhau. Biến khai báo ở khối code lớn (nằm ngoài) sẽ có phạm là cả khối code lớn, tức là bao trùm cả các khối code con bên trong. Biến khai báo ở khối code con (bên trong) thì có phạm vi là khối code đó thôi. Khối code bên ngoài không thuộc phạm vi của biến đó.

Hãy xem ví dụ sau đây để hiểu rõ hơn về phạm vi của biến

```
1.  static void Main()
2.  {// bắt đầu khối code thân của Main()
3.      var i = 10;
4.      Console.WriteLine(i);
5.
6.      {// bắt đầu một khối code tự do
7.          var j = 100; // j có phạm vi từ đây đến hết khối code này
8.          Console.WriteLine(i); // phạm vi của i bao trùm khối code này
9.          Console.WriteLine(j); // vẫn trong phạm vi của j
10.     }// kết thúc khối code tự do
11.
12.     Console.WriteLine(i); // i vẫn còn tác dụng
13.     Console.WriteLine(j); // đã ra ngoài phạm vi của j (không dùng được j nữa) => báo lỗ
14. }// kết thúc khối code thân của Main()
```

Trong phạm vi của một biến, bạn không thể khai báo một biến khác trùng tên. Với ví dụ trên, trong phạm vi của biến i (là thân của Main) bạn không thể khai báo một biến i khác. Trong khối code tự do bạn không thể khai báo một biến j khác.

> Biến được khai báo trong cấu trúc điều khiển như for, while, foreach (bạn sẽ học sau) chỉ có tác dụng trong khối code thân của cấu trúc đó. Trong phần lập trình hướng đối tượng bạn sẽ còn gặp biến thành viên (member variable) – loại biến có phạm vi tác dụng là toàn bộ class, bất kể vị trí khai báo.

Hằng trong C#

Hằng có thể xem tương tự như biến về khía cạnh lưu trữ dữ liệu. Khác biệt duy nhất là giá trị của hằng không thể thay đổi.

Hằng được khai báo và khởi tạo với cú pháp như sau:

```
const int a = 1000; // giá trị của a sẽ không thể thay đổi được sau khai báo này
```

Việc sử dụng hằng hoàn toàn tương tự như biến. Vai trò của hằng trong lập trình chắc chắn bạn đã biết rõ. Do đó chúng ta sẽ không trình bày cụ thể nữa.

Tuy nhiên có một số sự khác biệt sau giữa hằng và biến:

Hằng bắt buộc phải được khởi tạo ngay lúc khai báo. Sau khi gán giá trị (lúc khai báo), giá trị này sẽ không thể thay đổi.

Giá trị của hằng phải tính toán được ở giai đoạn compile time. Do vậy, bạn không thể khởi tạo một hằng nhưng sử dụng giá trị lấy từ một biến.

> Đến phần lập trình hướng đối tượng bạn sẽ còn gặp một đặc điểm nữa của hằng: hằng thành viên được mặc định xem là thành viên static của class. Nghĩa là có thể truy xuất qua tên class, thay vì truy xuất qua object.

Kết luận

Bài học đã trình bày những vấn đề tiêu biểu về biến và hằng trong C#. Nếu bạn đã học qua C/C++/Java, biến và hằng trong C# rất đơn giản để tiếp thu, ngoại trừ một số đặc điểm riêng như đã nêu trong bài.

Để có thể khai thác được biến và hằng, bạn cần biết thêm về các kiểu dữ liệu của C# – nội dung sẽ được trình bày trong bài học tiếp theo.

> + 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!