

Bài: Kiểu dữ liệu trong C#

Xem bài học trên website để ủng hộ Kteam: [Kiểu dữ liệu trong C#](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

Dẫn nhập

Ở bài [BIỂU TRONG C#](#) chúng ta đã tìm hiểu về biến và có một thành phần không thể thiếu khi khai báo biến – Đó là kiểu dữ liệu. Bài học hôm nay chúng ta sẽ cùng tìm hiểu chi tiết về **Kiểu dữ liệu trong C#**.

Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- [Cấu trúc lệnh của C# viết trên nền Console Application](#).
- [Cấu trúc nhập xuất của C# trên nền Console Application](#).
- [Biến trong C#](#).

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- Kiểu dữ liệu là gì? Tại sao phải có kiểu dữ liệu?
- Phân loại kiểu dữ liệu và ý nghĩa của từng kiểu dữ liệu.
- Ví dụ chương trình sử dụng một số kiểu dữ liệu.

Kiểu dữ liệu là gì? Tại sao phải có kiểu dữ liệu?

Định nghĩa

Kiểu dữ liệu được định nghĩa như sau:

- Là tập hợp các nhóm dữ liệu có cùng đặc tính, cách lưu trữ và thao tác xử lý trên trường dữ liệu đó.
- Là một tín hiệu để trình biên dịch nhận biết kích thước của một biến (ví dụ như int là 4 bytes, sẽ được trình bày ở phần sau) và khả năng của nó (ví dụ biến kiểu int chỉ có thể chứa được các số nguyên).
- Là thành phần cốt lõi của một ngôn ngữ lập trình.

Tại sao phải có kiểu dữ liệu?

- Như trong định nghĩa đã trình bày, phải có kiểu dữ liệu để nhận biết kích thước và khả năng của một biến.
- Nhằm mục đích phân loại dữ liệu. Nếu như không có kiểu dữ liệu ta rất khó xử lý vì không biết biến này kiểu chuỗi hay kiểu số nguyên hay kiểu số thực, ...

Phân loại kiểu dữ liệu và ý nghĩa của từng kiểu dữ liệu

Trong C#, kiểu dữ liệu được chia thành 2 tập hợp kiểu dữ liệu chính:

- Kiểu dữ liệu dựng sẵn (built - in) mà ngôn ngữ cung cấp.
- Kiểu dữ liệu do người dùng định nghĩa (user - defined).

Mỗi tập hợp kiểu dữ liệu trên lại phân thành 2 loại:

Kiểu dữ liệu giá trị (value):

Một biến khi khai báo kiểu dữ liệu giá trị thì vùng nhớ của biến đó sẽ chứa giá trị của dữ liệu và được lưu trữ trong bộ nhớ **Stack**.

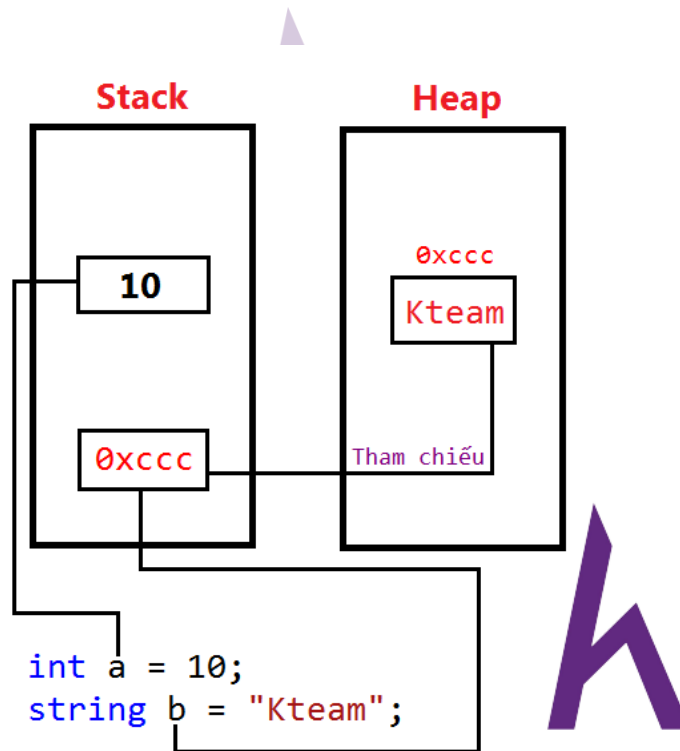
Một số kiểu dữ liệu thuộc kiểu giá trị: bool, byte, char, decimal, double, enum, float, int, long, sbyte, short, struct, uint, ulong, ushort. (các kiểu dữ liệu này sẽ được trình bày ngay sau đây)

Kiểu dữ liệu tham chiếu (reference):

Một biến khi khai báo kiểu dữ liệu tham chiếu thì vùng nhớ của biến đó chỉ chứa địa chỉ của đối tượng dữ liệu và lưu trong bộ nhớ **Stack**.

Đối tượng dữ liệu thực sự được lưu trong bộ nhớ **Heap**.

Một số kiểu dữ liệu thuộc kiểu tham chiếu: **object**, **dynamic**, **string** và tất cả các kiểu dữ liệu do người dùng định nghĩa. (sẽ được trình bày ở những bài sau).



Hình ảnh trên minh họa cho **kiểu dữ liệu giá trị** và **kiểu dữ liệu tham chiếu**: (đây chỉ là hình ảnh mang tính chất minh họa cho các bạn hiểu về cách lưu trữ của)

- Trong hình biến **a** được khai báo là kiểu int nên vùng nhớ được lưu trên **Stack** và lưu giá trị 10.
- Biến **b** được khai báo là kiểu string nên vùng nhớ trên **Stack** chỉ lưu địa chỉ đối tượng dữ liệu (**0xccc**) còn đối tượng dữ liệu thực sự được lưu trên **Heap** (ô nhớ có địa chỉ **0xccc** có giá trị là "Kteam").

Bộ nhớ **Stack** và bộ nhớ **Heap**: Cả hai đều là bộ nhớ trên RAM nhưng cách tổ chức, quản lý dữ liệu cũng như sử dụng thì rất khác nhau:

Stack:

- Vùng nhớ được cấp phát khi chương trình biên dịch.
- Được sử dụng cho việc thực thi thread (khái niệm về thread sẽ được trình bày trong bài [THREAD TRONG C#](#)), khi gọi hàm (khái niệm về hàm sẽ được trình bày trong bài [CẤU TRÚC HÀM CƠ BẢN TRONG C#](#)), các biến cục bộ kiểu giá trị và **tự động giải phóng** khi không còn sử dụng nữa.
- Kích thước vùng nhớ Stack là cố định và chúng ta không thể thay đổi.
- Khi vùng nhớ này không còn đủ dùng thì sẽ gây ra hiện tượng **tràn bộ nhớ** (stack overflow). Hiện tượng này xảy ra khi nhiều hàm lồng vào nhau hoặc gọi đệ quy nhiều lần dẫn đến không đủ vùng nhớ (khái niệm đệ quy sẽ được trình bày trong bài [CẤU TRÚC HÀM CƠ BẢN TRONG C#](#)).

Heap:

- Vùng nhớ được cấp phát khi chạy chương trình.
- Vùng nhớ Heap được dùng cho cấp phát bộ nhớ động (cấp phát thông qua toán tử new, sẽ được trình bày trong bài [TOÁN TỬ TRONG C#](#))
- Bình thường vùng nhớ trong Heap do người dùng tự giải phóng nhưng trong C# điều này được hỗ trợ mạnh mẽ bởi bộ tự động thu gom rác (Garbage Collection). Vì thế việc giải phóng vùng nhớ sẽ được thực hiện tự động.
- Kích thước vùng nhớ Heap có thể thay đổi được. Khi không đủ vùng nhớ để cấp phát thì hệ điều hành sẽ tự động tăng kích thước vùng nhớ Heap lên.

Trong phạm vi bài học hôm nay chúng ta chỉ tìm hiểu qua các kiểu dữ liệu dựng sẵn cơ bản những kiểu dữ liệu còn lại chúng ta sẽ tìm hiểu trong các bài học sau. Ý nghĩa của một số kiểu dữ liệu cơ bản:

Nhóm	Kiểu dữ liệu	Kích thước (bytes)	Ý nghĩa
Kiểu số nguyên	byte	1	Số nguyên dương không dấu có giá trị từ 0 đến 255
	sbyte	1	Số nguyên có dấu có giá trị từ -128 đến 127
	short	2	Số nguyên có dấu có giá trị từ -32,768 đến 32,767
	ushort	2	Số nguyên không dấu có giá trị từ 0 đến 65,535
	int	4	Số nguyên có dấu có giá trị từ -2,147,483,647 đến 2,147,483,647
	uint	4	Số nguyên không dấu có giá trị từ 0 đến 4,294,967,295
	long	8	Số nguyên có dấu có giá trị từ -9,223,370,036,854,775,808 đến 9,223,370,036,854,775,807
	ulong	8	Số nguyên không dấu có giá trị từ 0 đến 18,446,744,073,709,551,615
Kiểu ký tự	char	2	Chứa một ký tự Unicode
Kiểu logic	bool	1	Chứa 1 trong 2 giá trị logic là true hoặc false
Kiểu số thực	float	4	Kiểu số thực dấu chấm động có giá trị dao động từ 3.4E - 38 đến 3.4E + 38 , với 7 chữ số có nghĩa
	double	8	Kiểu số thực dấu chấm động có giá trị dao động từ 1.7E - 308 đến 1.7E + 308 , với 15, 16 chữ số có nghĩa
	decimal	8	Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính

- Khác với những kiểu dữ liệu trên, **string** là kiểu dữ liệu tham chiếu và dùng để lưu chuỗi ký tự.

Một số lưu ý khi sử dụng các kiểu dữ liệu trên:

Khác với những kiểu dữ liệu trên, **string** là kiểu dữ liệu tham chiếu và dùng để lưu chuỗi ký tự. Trong phạm vi bài học hôm nay chúng ta chỉ tìm hiểu qua các kiểu dữ liệu dựng sẵn cơ bản những kiểu dữ liệu còn lại chúng ta sẽ tìm hiểu trong các bài học sau.

Ý nghĩa của một số kiểu dữ liệu cơ bản:

- Kiểu dữ liệu có miền giá trị lớn hơn sẽ chứa được kiểu dữ liệu có miền giá trị nhỏ hơn. Như vậy biến kiểu dữ liệu nhỏ hơn có thể gán giá trị qua biến kiểu dữ liệu lớn hơn (sẽ được trình bày trong phần tiếp theo).
- Giá trị của kiểu **char** sẽ nằm trong dấu **' '** (nháy đơn).
- Giá trị của kiểu **string** sẽ nằm trong dấu **" "** (nháy kép).
- Giá trị của biến kiểu **float** phải có chữ **F** hoặc **f** làm hậu tố.
- Giá trị của biến kiểu **decimal** phải có chữ **m** hoặc **M** làm hậu tố.
- Trừ kiểu **string**, tất cả kiểu dữ liệu trên đều **không được có giá trị null**:
 - Null là giá trị rỗng, không tham chiếu đến vùng nhớ nào.
 - Để có thể gán giá trị null cho biến thì ta thêm ký tự **?** vào sau tên kiểu dữ liệu là được. Ví dụ: **int?** hay **bool?** ...

Ví dụ chương trình sử dụng kiểu dữ liệu

Ví dụ 1: Khai báo biến với các kiểu dữ liệu đã học.

C#:

```
static void Main(string[] args)
{
    // Kiểu số nguyên
    byte BienByte = 10;
    short BienShort = 10;
    int BienInt = 10;
    long BienLong = 10;

    // Kiểu số thực
    float BienFloat = 10.9f; // Giá trị của biến kiểu float phải có hậu tố f hoặc F.
    double BienDouble = 10.9; // Giá trị của biến kiểu double không cần hậu tố.
    decimal BienDecimal = 10.9m; // Giá trị của biến kiểu decimal phải có hậu tố m.

    // Kiểu ký tự và kiểu chuỗi
    char BienChar = 'K'; // Giá trị của biến kiểu ký tự nằm trong cặp dấu ' ' (nháy đơn).
    string BienString = "Kteam"; // Giá trị của biến kiểu chuỗi nằm trong cặp dấu " " (nháy kép).

    Console.ReadKey();
}
```

Các bạn chú ý vào cách khai báo, khởi tạo giá trị cho các biến.

Ví dụ 2: Một số thao tác liên quan đến biến và kiểu dữ liệu.

C#:

```
static void Main(string[] args)
{
    // Kiểu số nguyên
    byte BienByte = 3;
    short BienShort = 9;
    int BienInt = 10;
    long BienLong = 0;

    BienLong = BienByte; // BienLong có kiểu dữ liệu lớn hơn BienByte nên giá trị BienByte có thể gán qua cho BienLong
    Console.WriteLine(" BienLong = " + BienLong);

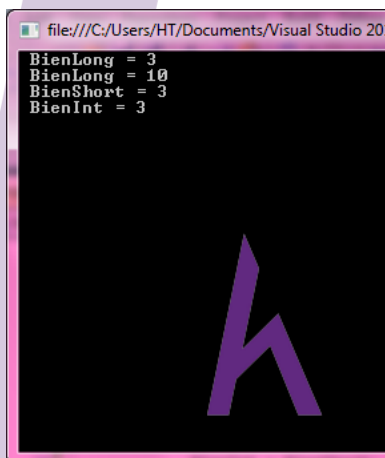
    BienLong = BienInt; // tương tự như trên
    Console.WriteLine(" BienLong = {0}", BienLong);

    BienShort = BienByte; // tương tự như trên
    Console.WriteLine(" BienShort = " + BienShort);

    BienInt = BienShort; // tương tự như trên
    Console.WriteLine(" BienInt = " + BienInt);

    Console.ReadKey();
}
```

Ở ví dụ này chúng ta kiểm chứng lại xem việc gán giá trị từ biến có kiểu dữ liệu nhỏ hơn sang biến có kiểu dữ liệu lớn hơn có được hay không. Và kết quả sau khi chạy chương trình:



Qua kết quả chạy, ta thấy giá trị của các biến thay đổi nên việc gán giá trị như vậy là có thể được.

Ví dụ 3: Những lỗi cần lưu ý.

C#:

```
static void Main(string[] args)
{
    int a;
    Console.WriteLine(" a = " + a); // Lỗi vì biến a không thể sử dụng khi chưa có giá trị.

    int b = 10.9; // Lỗi vì b là biến kiểu số nguyên nên không thể nhận giá trị ngoài số nguyên.

    byte c = 1093; // Lỗi vì c là biến kiểu byte mà kiểu byte có miền giá trị từ -128 đến 127 nên không thể nhận giá ngoài vùng này
    được.

    string d = 'K'; // Lỗi vì không thể gán giá trị ký tự vào biến kiểu chuỗi được mặc dù chuỗi có thể hiểu là tập hợp nhiều ký tự. Có
    thể sửa bằng cặp dấu "" thay vì ''.

    long e = null; // Lỗi vì không thể gán null cho biến kiểu long, int, byte, . . .
    long? f = null; // Cách khắc phục là thêm dấu ? vào sau kiểu dữ liệu. Lúc này kiểu dữ liệu của f là long?

    int g = 10;
    byte h = g; // Lỗi vì giá trị của biến có kiểu dữ liệu lớn hơn không thể gán cho biến có kiểu dữ liệu nhỏ hơn mặc dù trong trường
    hợp này ta thấy số 10 đều có thể gán cho 2 biến.

    string k = "Kteam";
    Console.WriteLine(" k = " + K); // Lỗi vì phía trên khai báo biến k còn khi sử dụng là biến K (C# có phân biệt chữ hoa, thường cần
    lưu ý để tránh gặp lỗi)

    Console.ReadKey();
}
```

Chương trình này là minh họa những lỗi thường gặp khi làm việc với biến và kiểu dữ liệu. Các bạn cần lưu ý để tránh những lỗi cơ bản này.

Kết luận

Nội dung bài này giúp các bạn nắm được:

- Khái niệm về kiểu dữ liệu và tại sao lại phải sử dụng kiểu dữ liệu.
- Phân loại kiểu dữ liệu và ý nghĩa của các kiểu dữ liệu dựng sẵn cơ bản.
- Phân biệt giữa bộ nhớ Stack và Heap.
- Ví dụ chương trình sử dụng kiểu dữ liệu và những lỗi lập trình thường gặp về phần này.

Bài học sau chúng ta sẽ cùng tìm hiểu một khái niệm tiếp theo đó là [TOÁN TỬ TRONG C#](#).

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**