LESSON 02

I. KIỂU DỮ LIỆU TRONG C#

❖ 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.

❖ 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**:

- Vùng nhớ được cấp phát khi chương trình biên dịch.
- 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.

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**:

- 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)
- 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.

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).

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:

Kiểu	Biểu diễn	Dãy giá trị	Giá trị mặc định
bool	Giá trị Boolean	True hoặc False	False
byte	Kiểu unsigned integer (8 bit)	0 tới 255	0
char	Kiểu Unicode character (16 bit)	U +0000 tới U +ffff	'\0'
decimal	Kiểu thập phân (128 bit)	(-7.9×10^{28} tới 7.9×10^{28}) / $10^{0 \text{ to } 28}$	0.0M
double	Kiểu double (64 bit)	(+/-)5.0×10^{-324} tới (+/-)1.7×10^{308}	0.0D
float	Kiểu float (32 bit)	-3.4×10^{38} tới $+ 3.4 \times 10^{38}$	0.0F
int	Kiểu integer (32 bit)	-2,147,483,648 tới 2,147,483,647	0
long	Kiểu signed integer (64 bit)	-9,223,372,036,854,775,808 tới 9,223,372,036,854,775,807	0L
sbyte	Kiểu signed integer (8 bit)	-128 tới 127	0
short	Kiểu signed integer (16 bit)	-32,768 tới 32,767	0
uint	Kiểu unsigned integer (32 bit)	0 tới 4,294,967,295	0
ulong	Kiểu unsigned integer (64 bit)	0 tới 18,446,744,073,709,551,615	0
ushort	Kiểu unsigned integer (16 bit)	0 tới 65,535	0

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

- 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

Mở 03 Ví dụ lên và chữa bài

II. TOÁN TỬ TRONG C#

❖ Toán tử là gì?

➢ Toán tử được định nghĩa như sau:

- Là một công cụ để thao tác với dữ liệu.
- Một toán tử là một ký hiệu dùng để đại diện cho một thao tác cụ thể được thực hiện trên dữ liệu.

➢ Có 6 loại toán tử cơ bản:

- Toán tử toán học.
- Toán tử quan hệ.
- Toán tử logic.
- Toán tử khởi tạo và gán.
- Toán tử so sánh trên bit.
- Toán tử khác.

❖ Các loại toán tử, cú pháp và ý nghĩa

Toán tử toán học:

Giả sử biến A có giá trị là 10 biến B có giá trị là 20

Toán tử	Miêu tả	Ví dụ
+	Thêm hai toán hạng	A + B sẽ cho kết quả là 30
-	Trừ giá trị toán hạng hai từ toán hạng đầu	A - B sẽ cho kết quả là -10
*	Nhân hai toán hạng	A * B sẽ cho kết quả là 200
/	Chia lấy phần nguyên hai toán hạng	B / A sẽ cho kết quả là 2
%	Chia lấy phần dư	B % A sẽ cho kết quả là 0
++	Lượng gia giá trị toán hạng thêm 1 đơn vị	A++ sẽ cho kết quả là 11
--	Lượng giảm giá trị toán hạng một đơn vị	A-- sẽ cho kết quả là 9

Lưu ý: đối với toán tử ++ và -- cần phân biệt i++ và ++i

- **i++**: là sử dụng giá trị của biến i để thực hiện biểu thức trước rồi mới thực hiện tăng lên 1 đơn vị. Tương tự cho i--.
- **++i**: là tăng giá trị biến i lên 1 đơn vị rồi mới sử dụng biến a để thực hiện biểu thức. Tương tự cho --i.

```
static void Main(string[] args)
    {
        int i = 5, j = 5;

        Console.WriteLine(i++); // Sử dụng giá trị i để in ra rồi mới tăng i
        Console.WriteLine(++j); // Tăng j lên rồi mới in giá trị j ra màn hình

        Console.ReadKey();
    }
```

Toán tử quan hệ

Toán tử	Miêu tả	Ví dụ
==	Kiểm tra nếu 2 toán hạng bằng nhau hay không. Nếu bằng thì điều kiện là true.	(A == B) là không đúng.
!=	Kiểm tra 2 toán hạng có giá trị khác nhau hay không. Nếu không bằng thì điều kiện là true.	(A != B) là true.
>	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không. Nếu lớn hơn thì điều kiện là true.	(A > B) là không đúng.
<	Kiểm tra nếu toán hạng bên trái nhỏ hơn toán hạng bên phải hay không. Nếu nhỏ hơn thì là true.	(A < B) là true.
>=	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn hoặc bằng giá trị của toán hạng bên phải hay không. Nếu đúng là true.	(A >= B) là không đúng.
<=	Kiểm tra nếu toán hạng bên trái có giá trị nhỏ hơn hoặc bằng toán hạng bên phải hay không. Nếu đúng là true.	(A <= B) là true.

Lưu ý:

- Các toán tử quan hệ này chỉ áp dụng cho số hoặc ký tự.
- Hai toán hạng hai bên phải cùng loại (cùng là số hoặc cùng là ký tự).
- Bản chất của việc so sánh 2 ký tự với nhau là so sánh mã ASCII của các ký tự đó.
- Không nên sử dụng các toán tử trên để so sánh các chuỗi với nhau vì bản chất việc so sánh chuỗi là so sánh từng ký tự tương ứng với nhau mà so sánh ký tự là so sánh mã ASCII của ký tự đó như vậy ký tự ‘T’ sẽ khác ký tự ‘t’. Để so sánh hai chuỗi người ta thường dùng hàm so sánh chuỗi đã được hỗ trợ sẵn (sẽ tìm hiểu ở những bài tiếp theo).

Toán tử logic

Giả sử mệnh đề A là đúng và mệnh đề B là sai:

Toán tử	Miêu tả	Ví dụ
&&	Được gọi là toán tử logic AND (và). Nếu cả hai toán tử đều có giá trị khác 0 thì điều kiện trở lên true.	(A && B) là false.
||	Được gọi là toán tử logic OR (hoặc). Nếu một trong hai toán tử khác 0, thì điều kiện là true.	(A || B) là true.
!	Được gọi là toán tử NOT (phủ định). Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false.	!(A && B) là true.

Lưu ý:

- Các toán tử && và || có thể áp dụng đồng thời nhiều toán hạng, ví dụ như: A && B && C || D || K..
- Các toán hạng trong biểu thức chứa toán tử logic phải trả về **true** hoặc **false**

Toán tử khởi tạo và gán

Toán tử khởi tạo và gán thường được sử dụng nhằm mục đích lưu lại giá trị cho một biến nào đó. Một số toán tử khởi tạo và gán hay được sử dụng:

Toán tử	Miêu tả	Ví dụ
=	Toán tử gán đơn giản. Gán giá trị toán hạng bên phải cho toán hạng trái.	C = A + B sẽ gán giá trị của A + B vào trong C
+=	Thêm giá trị toán hạng phải tới toán hạng trái và gán giá trị đó cho toán hạng trái.	C += A tương đương với C = C + A
-=	Trừ đi giá trị toán hạng phải từ toán hạng trái và gán giá trị này cho toán hạng trái.	C -= A tương đương với C = C - A
*=	Nhân giá trị toán hạng phải với toán hạng trái và gán giá trị này cho toán hạng trái.	C *= A tương đương với C = C * A
/=	Chia toán hạng trái cho toán hạng phải và gán giá trị này cho toán hạng trái.	C /= A tương đương với C = C / A
%=	Lấy phần dư của phép chia toán hạng trái cho toán hạng phải và gán cho toán hạng trái.	C %= A tương đương với C = C % A

Toán tử so sánh trên bit

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Toán tử	Miêu tả	Ví dụ
&	Toán tử AND (và) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong cả hai toán hạng.	(A & B) sẽ cho kết quả là 12, tức là 0000 1100
|	Toán tử OR (hoặc) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong một hoặc hai toán hạng.	(A | B) sẽ cho kết quả là 61, tức là 0011 1101
^	Toán tử XOR nhị phân sao chép bit mà nó chỉ tồn tại trong một toán hạng mà không phải cả hai.	(A ^ B) sẽ cho kết quả là 49, tức là 0011 0001
~	Toán tử đảo bit (đảo bit 1 thành bit 0 và ngược lại).	(~A) sẽ cho kết quả là -61, tức là 1100 0011.
<<	Toán tử dịch trái. Giá trị toán hạng trái được dịch chuyển sang trái bởi số các bit được xác định bởi toán hạng bên phải.	A << 2 sẽ cho kết quả 240, tức là 1111 0000 (dịch sang trái hai bit)
>>	Toán tử dịch phải. Giá trị toán hạng trái được dịch chuyển sang phải bởi số các bit được xác định bởi toán hạng bên phải.	A >> 2 sẽ cho kết quả là 15, tức là 0000 1111 (dịch sang phải hai bit)

Toán tử khác

Dưới đây là một số toán tử hỗn hợp quan trọng gồm **sizeof, typeof và ? :** được hỗ trợ bởi ngôn ngữ C#.

Toán tử	Miêu tả	Ví dụ
sizeof()	Trả về kích cỡ của một kiểu dữ liệu	sizeof(int), trả về 4
typeof()	Trả về kiểu của một lớp	typeof(StreamReader);
&	Trả về địa chỉ của một biến	&a; trả về địa chỉ thực sự của biến
*	Trỏ tới một biến	*a; tạo con trỏ với tên là a tới một biến
? :	Biểu thức điều kiện (Conditional Expression)	Nếu Condition là true ? Thì giá trị X : Nếu không thì Y
is	Xác định đối tượng là một kiểu cụ thể hay không	If(Ford is Car) // Kiểm tra nếu Ford là một đối tượng của lớp Car
as	Ép kiểu mà không tạo một exception nếu việc ép kiểu thất bại	Object obj = new StringReader("Hello"); StringReader r = obj as StringReader;

III. HẰNG TRONG C#

❖ Hằng là gì?

➢ Khái niệm về hằng:

- Là một biến những giá trị không thay đổi trong suốt chương trình.
- Bắt buộc phải khởi tạo giá trị khi khai báo.

➢ Mục đích:

- Để ngăn chặn việc gán giá trị khác vào biến.
- Hằng làm cho chương trình dễ đọc hơn bằng cách biến những con số vô cảm thành những tên có nghĩa.
- Hằng giúp cho chương trình dễ nâng cấp, dễ sửa chữa hơn.
- Hằng giúp cho việc tránh lỗi dễ dàng hơn. Nếu bạn vô ý gán giá trị cho một biến hằng ở đâu thì trình biên dịch sẽ báo lỗi ngay lập tức.

➢ Cú pháp

const **<kiểu dữ liệu>** **<tên biến>** = <giá trị hằng>**;**

➢ Lưu ý:

- Hằng bắt buộc phải được khởi tạo giá trị ngay khi khai báo và không được thay đổi giá trị trong suốt chương trình.
- Giá trị của hằng được tính toán vào lúc biên dịch nên không thể gán trực tiếp giá trị của biến vào hằng. Nếu muốn làm điều đó thì phải sử dụng từ khóa **readonly** trước khai báo biến (readonly là từ khóa chỉ cho trình biên dịch biết rằng biến này chỉ được đọc, lấy giá trị chứ không được gán giá trị)
- Hằng bao giờ cũng **static** nhưng ta không được đưa từ khóa này vào khai báo hằng (chi tiết về từ khóa static sẽ được trình bày sau)

IV. ÉP KIỂU TRONG C#.

❖ Ép kiểu là gì?

- Ép kiểu là **biến đổi dữ liệu** thuộc kiểu dữ liệu này thành kiểu dữ liệu khác.
- Tại sao phải ép kiểu?

  • Để chuyển dữ liệu sang một kiểu dữ liệu mong muốn phục vụ cho việc thao tác xử lý.

  • Đưa dữ liệu về định dạng mà mình mong muốn (ví dụ chuyển kiểu ngày tháng bên Mỹ sang dạng ngày tháng bên Việt Nam)

❖ Phân loại ép kiểu

- Trong C#, ép kiểu có 4 loại:
  - Chuyển đổi kiểu ngầm định (**implicit**).
  - Chuyển đổi kiểu tường minh (**explicit**).
  - Sử dụng phương thức, lớp hỗ trợ sẵn:
    - Dùng phương thức **Parse()**, **TryParse()**.
    - Dùng lớp hỗ trợ sẵn (**Convert**).
  - Người dùng tự định nghĩa kiểu chuyển đổi.

Chuyển đổi kiểu ngầm định (implicit)

- Việc chuyển đổi này được thực hiện bởi trình biên dịch và chúng ta không cần tác động gì.
- Được thực hiện khi chuyển kiểu từ
  - Kiểu dữ liệu nhỏ sang kiểu dữ liệu lớn hơn (xem lại bài KIỂU DỮ LIỆU TRONG C#)
  - Chuyển từ lớp con đến lớp cha (trong bài TÍNH KẾ THỪA TRONG C#).

➢ **Ví dụ:**

```
0 references
static void Main6(string[] args)
{
    int intValue = 10;
    long longValue = intValue;
    /* Chuyển kiểu ngầm định vì kiểu long có miền giá trị
       lớn hơn kiểu int nên có thể chuyển từ int sang long.*/

    float floatValue = 10.9f;
    double doubleValue = floatValue;
    /* Tương tự vì kiểu double có miền giá trị lớn
        hơn kiểu float nên có thể chuyển từ float sang double.*/
}
```

Chuyển đổi kiểu tường minh (explicit)

➢ Chuyển đổi kiểu tường minh là việc chuyển kiểu một cách rõ ràng và dùng từ khóa chỉ định chứ không dùng phương thức.

➢ Một số đặc điểm của chuyển đổi kiểu tường minh:

- Thường được dùng để chuyển đổi giữa **các kiểu dữ liệu có tính chất tương tự nhau** (thường thì với số).
- Hỗ trợ trong việc boxing và unboxing đối tượng (trong bài KIỂU DỮ LIỆU OBJECT TRONG C#).
- Cú pháp ngắn gọn do không sử dụng phương thức.
- Chỉ khi chúng ta biết rõ biến đó thuộc kiểu dữ liệu nào mới thực hiện chuyển đổi. Ngược lại có thể lỗi khi chạy chương trình.

➢ Cú pháp:

(<**kiểu dữ liệu**>) <**biến cần ép kiểu**>

➢ Ý nghĩa:

Ép kiểu của <biến cần ép kiểu> về <kiểu dữ liệu> nếu thành công sẽ trả ra giá trị kết quả ngược lại sẽ báo lỗi chương trình.

Đặc biệt đối với số:

- Ta có thực hiện ép **kiểu dữ liệu lớn hơn** về **kiểu dữ liệu nhỏ hơn** mà không báo lỗi.

- Nếu dữ liệu cần ép kiểu **vượt quá miền giá trị** của kiểu dữ liệu muốn ép kiểu về thì chương trình sẽ **tự cắt bit** cho phù hợp với khả năng chứa kiểu dữ liệu đó (cắt từ bên trái qua)

```
int i = 300; // 300 có mã nhị phân là 100101100
byte b = (byte)i;
/* do kiểu byte có giới hạn đến giá trị 255 thôi nên không thể chứa số 300 được
mà kiểu byte có kích thước là 1 bytes tương đương 8 bit. Như vậy ta cần cắt mã
nhị phân của số 300 về còn 8 bit là được. Mã nhị phân 300 là 100101100 cắt từ trái
qua 1 bit ta được 00101100 (đủ 8 bit) tương đương với số 44. Cuối cùng biến b sẽ
mang giá trị là 44.*/

Console.WriteLine(" i = " + i);
Console.WriteLine(" b = " + b);
```

➢ **Ví dụ:**

```
static void Main6(string[] args)
{
    double d = 2 / 3;
    // kết quả ra 0 vì 2 và 3 đều là số nguyên nên thực hiện 2 chia lấy phần nguyên với 3 được 0

    double k = (double)2 / 3;
    // Ép kiểu số 2 từ kiểu nguyên sang kiểu số thực. Như vậy kết quả phép chia sẽ ra số thực

    double t = 1.0 * 2 / 3;
    // Thực hiện nhân 1.0 với 2 mục đích để biến số 2 (sốnguyên) thành 2.0(số thực)

    Console.WriteLine(" d = {0} \n k = {1} \n t = {2}", d, k, t);
}
```

Sử dụng phương thức, lớp hỗ trợ sẵn Phương thức Parse(), TryParse()

Parse()

➢ Cú pháp:

<kiểu dữ liệu>.**Parse(**<dữ liệu cần chuyển đổi>**);**

➢ Ý nghĩa:

- Chuyển đổi một chuỗi sang một kiểu dữ liệu cơ bản tương ứng.
- Phương thức trả về giá trị kết quả chuyển kiểu nếu chuyển kiểu thành công. Ngược lại sẽ báo lỗi chương trình.

➢ Ví dụ:

```
0 references
static void Main8(string[] args)
{
    string stringValue = "10";
    int intValue = int.Parse(stringValue);
    // Chuyển chuỗi stringValue sang kiểu int và lưu giá trị vào biến intValue - Kết quả intValue = 10

    double TTDesign = double.Parse("10.9");
    // Chuyển chuỗi giá trị hằng "10.9" sang kiểu int và lưu giá trị vào biến TTDesign - Kết quả TTDesign = 10.9
}
```

- ***TryParse()***

➢ Cú pháp:

<kiểu dữ liệu>.**TryParse(**<dữ liệu cần chuyển đổi>**, out** <biến chứakết quả>**);**

➢ Ý nghĩa:

- Chuyển một chuỗi sang một kiểu dữ liệu cơ bản tương ứng.
- Phương thức sẽ trả về **true** nếu chuyển kiểu thành công và giá trị kết quả chuyển kiểu sẽ lưu vào <biến chứa kết quả>. Ngược lại sẽ trả về **false** và <biến chứa kết quả> sẽ mang giá trị 0.

➢ Ví dụ:

```
0 references
static void Main9(string[] args)
{
    int Result; // Biến chứa giá trị kết quả khi ép kiểu thành công
    bool isSuccess; // Biến kiểm tra việc ép kiểu có thành công hay không

    string Data1 = "10", Data2 = "TTDesign"; // Dữ liệu cần ép kiểu

    isSuccess = int.TryParse(Data1, out Result);
    // Thử ép kiểu Data1 về int nếu thành công thì Result
    //sẽ chứa giá trị kết quả ép kiểu và isSuccess sẽ mang giá trị true.
    // Ngược lại Result sẽ mang giá trị 0 và isSuccess mang giá trị false

    Console.Write(isSuccess == true ? " Success !" : " Failed !");
    // Sử dụng toán tử 3 ngôi để in ra màn hình việc ép kiểu đã thành công hay thất bại.

    Console.WriteLine(" Result = " + Result); // In giá trị Result ra màn hình

    isSuccess = int.TryParse(Data2, out Result); // Tương tự như trên nhưng thao tác với Data2
    Console.Write(isSuccess == true ? " Success !" : " Failed !"); // Tương tự như trên

    Console.WriteLine(" Result = " + Result); // Tương tự như trên
}
```

➢ Ngoài TryParse() ra thì vẫn có một cách ép kiểu không báo lỗi chương trình. Đó là sử dụng toán tử **as**:

Nếu không thành sẽ trả về **null**, và chỉ sử dụng được cho các kiểu **nullable**

Lớp hỗ trợ sẵn (Convert).

- **Convert** là lớp tiện ích được C# hỗ trợ sẵn cung cấp cho chúng ta nhiều phương thức chuyển đổi kiểu dữ liệu.
- Các đặc điểm của các phương thức trong lớp **Convert**:
  - Tham số truyền vào của các phương thức **không nhất thiết là chuỗi** (có thể là int, bool, . . .).
  - Nếu tham số truyền vào là null thì các phương thức sẽ **trả về giá trị mặc định** của kiểu dữ liệu.
  - Các trường hợp tham số truyền vào sai định dạng hoặc vượt quá giới hạn thì chương trình sẽ báo lỗi như phương thức **Parse()**

- Một số phương thức chuyển kiểu mà **Convert** có:

Tên phương thức	Ý nghĩa
ToBoolean	Chuyển đổi một kiểu thành một bool (true, false) nếu có thể
ToByte	Chuyển đổi một kiểu thành một byte
ToChar	Chuyển đổi một kiểu thành một char nếu có thể
ToDateTime	Chuyển đổi một kiểu thành các cấu trúc date-time
ToDecimal	Chuyển đổi một kiểu thành một kiểu thập phân
ToDouble	Chuyển đổi một kiểu thành kiểu double
ToInt32	Chuyển đổi một kiểu thành kiểu int
ToString	Chuyển đổi một kiểu thành kiểu string

❖ Ví dụ.

Ví dụ: viết chương trình nhập vào 2 số a và b sau đó in ra tổng, hiệu, tích, thương của 2 số đó:

- Đầu tiên ta yêu cầu người dùng nhập vào 2 số a và b (lúc này giá trị nhập vào đang ở kiểu chuỗi).
- Ép kiểu giá trị vừa nhập vào kiểu chuỗi. Nếu thất bại sẽ thông báo lỗi và kết thúc chương trình, ngược lại thì thực hiện tiếp.
- Tính tổng, hiệu, tích, thương thì 2 số đó và in kết quả ra màn hình.

- **Mở VS và chữa bài**