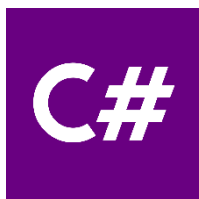


## Buổi 1.1

## Tổng quan về C#

## I. Tổng quan về C# và .NET Framework



C# là 1 ngôn ngữ lập trình hướng đối tượng mạnh mẽ, được phát triển bởi Microsoft. C# có thể được dùng để xây dựng rất nhiều loại ứng dụng, từ ứng dụng dòng lệnh, ứng dụng Windows, ứng dụng web cho đến ứng dụng di động, game...

**.NET Framework** là một nền tảng lập trình cho ngôn ngữ C# và cũng là một nền tảng thực thi ứng dụng chủ yếu trên hệ điều hành Microsoft Windows, được phát triển



bởi Microsoft. Các chương trình được viết trên nền .NET Framework sẽ được triển khai trong môi trường phần mềm được biết đến với tên Common Language Runtime (CLR).

Để phát triển 1 ứng dụng C#, lập trình viên cần sử dụng 1 IDE có khả năng biên dịch được ngôn ngữ C#, trong số đó phổ biến nhất là Visual Studio (VS).

Nhằm đáp ứng nhu cầu phát triển ứng dụng ngày càng cao, Microsoft luôn tiến hành cập nhật, nâng cấp, bổ sung tính năng cho ngôn ngữ C# cũng như cho trình biên dịch VS. Mỗi phiên bản C# mới ra đời thông thường sẽ đi kèm với 1 phiên bản .NET Framework mới và 1 phiên bản VS mới:

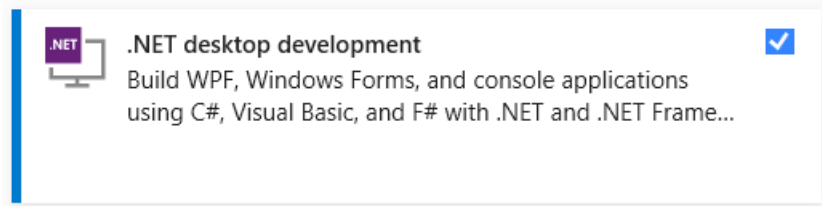
- C# 1.0 ra đời năm 2002 đi kèm với .NET Framework 1.0 và VS2002.
- C# 7.0 được giới thiệu cùng với VS2017 đánh dấu sự ra đời của .NET Core 2.0, là phiên bản .NET đa nền tảng kế thừa từ .NET Framework.
- C# 9.0 được giới thiệu cùng với VS2019 và .NET 5.0. Kể từ phiên bản này, MS đã loại bỏ "Core" khỏi tên của framework, mà chỉ đơn giản là .NET 5.0.
- Phiên bản C# mới nhất hiện nay là C# 10.0 được phát hành cùng với VS2022 và .NET 6.0.

## II. Các công cụ, phần mềm

Trong khuôn khổ môn học này, **khuyến cáo** sử dụng Visual Studio (VS) phiên bản mới nhất (hiện tại là phiên bản 2022).

Địa chỉ tải VS chính thức từ Microsoft: <https://visualstudio.microsoft.com/vs> (miễn phí đối với phiên bản Community).

Để có thể lập trình với C#, khi cài đặt VS, cần chọn vào workload sau:




Các chương trình được viết trong môn học này sẽ được viết trên nền .NET Framework, ngôn ngữ là C# dưới dạng 1 ứng dụng dòng lệnh (**Console application**).

Một số khái niệm mới được nhắc đến có thể sẽ không được giải thích ngay mà sẽ được giải thích ở các buổi sau.

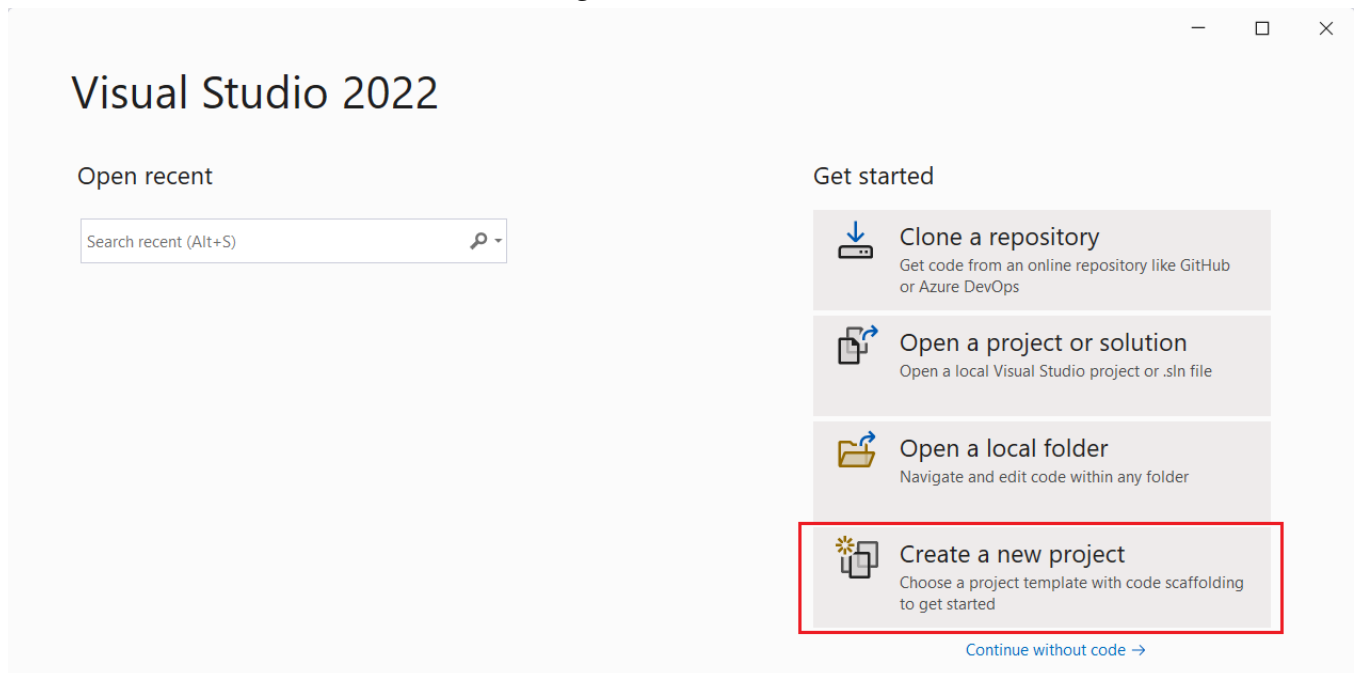
### III. Tạo project C#

Các chương trình C# được tổ chức dưới dạng các **project** (dự án). Trong 1 project có thể chứa rất nhiều file source code (mã nguồn), các file resource (tài nguyên) như hình ảnh, âm thanh... cũng như các file setting (cài đặt, thiết lập).

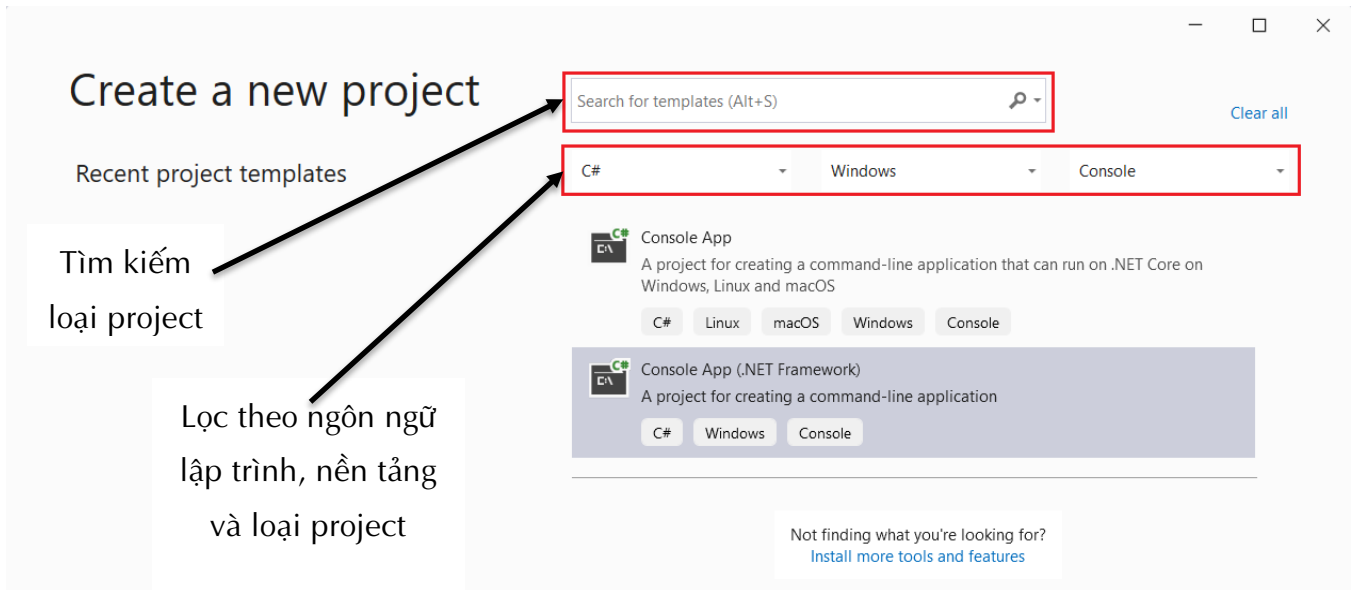
Với các chương trình phức tạp, có thể cần phải có **solution**. 1 solution có thể chứa nhiều project có liên quan.

Click vào biểu tượng VS2022 để khởi động chương trình: 

Màn hình đầu tiên sau khi khởi động VS như sau:



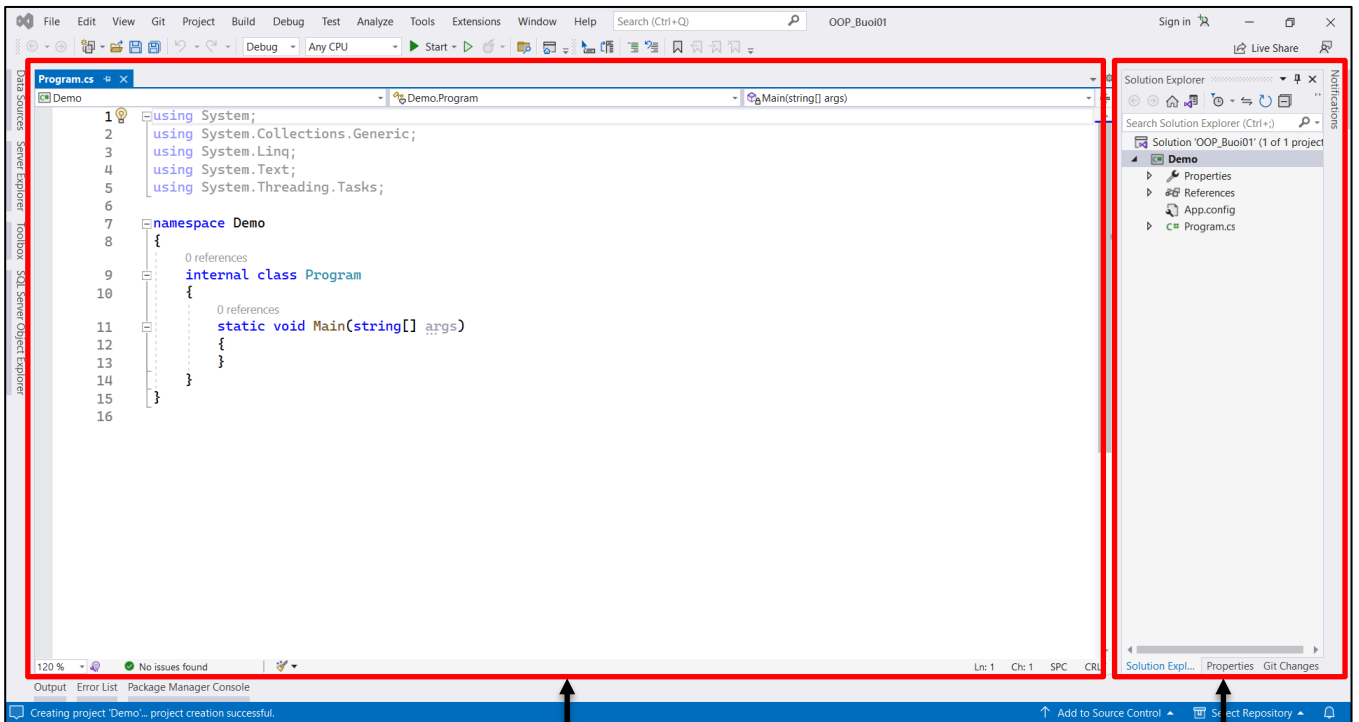
Để viết chương trình C# đầu tiên, click vào mục **Create a new project**. Ở màn hình tiếp theo, chọn loại project là **Console App (.NET Framework)** từ danh sách và click **Next**. Có thể tìm kiếm nhanh 1 loại project bằng cách nhập vào khung tìm kiếm hoặc lọc từ các menu ở trên cùng:



Ở màn hình **Configure your project**, thiết lập như sau, sau đó click **Create**:

- **Project name:** Đặt tên project là **Demo**.
- **Location:** Chọn nơi lưu project.
- **Solution name:** Đặt tên là solution là **OOP\_Buoi01**. Nếu không có nhu cầu tạo solution mà chỉ cần tạo 1 project đơn lẻ, check vào mục **Place solution and project in the same directory**.
- **Framework:** Chọn phiên bản .NET Framework. Trong ví dụ là phiên bản 4.8.

Giao diện làm việc của VS như sau:

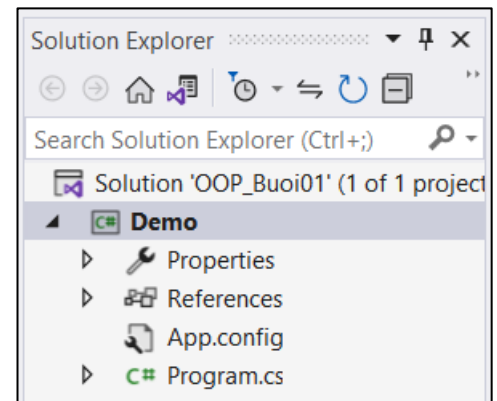


Vùng làm việc chính để  
viết mã nguồn

Cấu trúc project

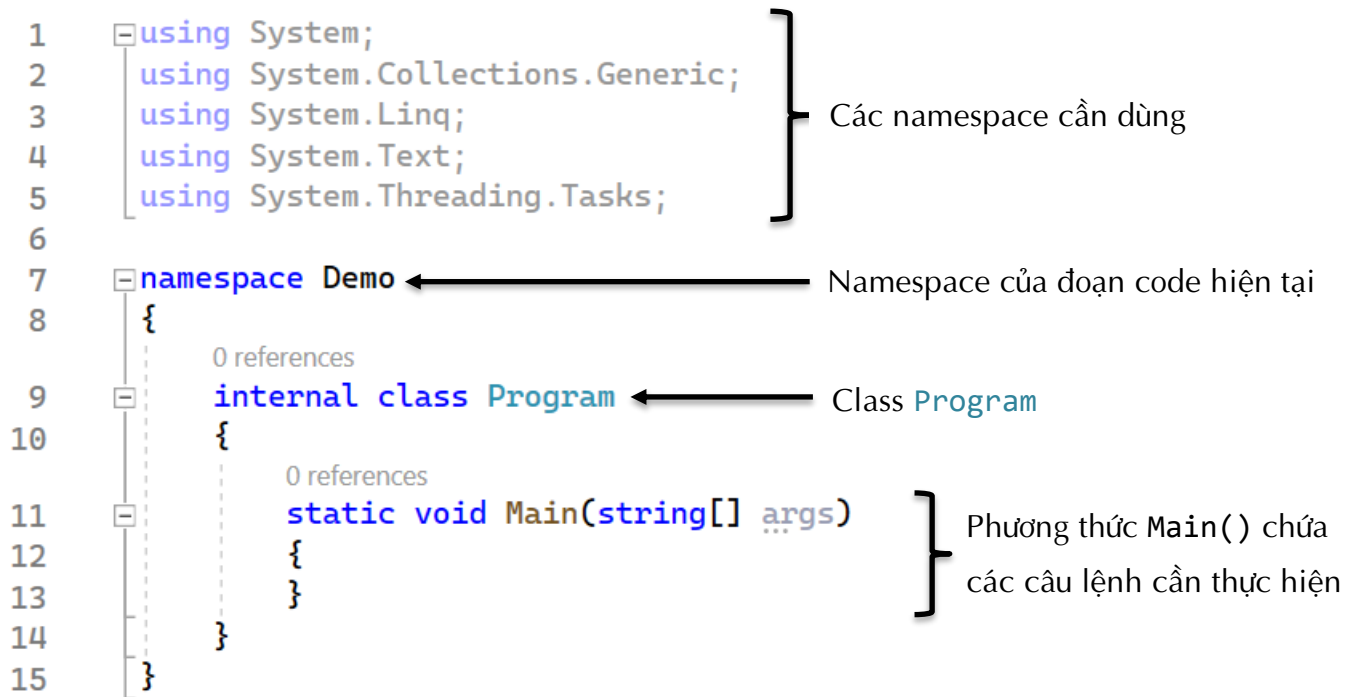
**Solution Explorer**<sup>1</sup> hiển thị cấu trúc solution và project hiện tại. Mặc định khi tạo mới 1 project sẽ gồm có:

- **Properties:** chứa các thiết lập của project cũng như các file resource đã được thêm vào project. Để thay đổi các thiết lập cho project, double-click vào thư mục **Properties**.
- **References:** chứa các namespace mà project hiện tại có sử dụng.
- **App.config:** file này chứa 1 số thiết lập cho ứng dụng hiện tại khi biên dịch.
- **Program.cs:** file mã nguồn C#. Project có thể chứa rất nhiều file mã nguồn như vậy, với phần mở rộng là **.cs** (C-Sharp).



File **Program.cs** đang mở sẵn ở vùng làm việc chính và có sẵn 1 số dòng code như sau:

<sup>1</sup> Nếu không thấy cửa sổ Solution Explorer, vào menu **View** → **Solution Explorer** (hoặc nhấn phím tắt **Ctrl + Alt + L**)

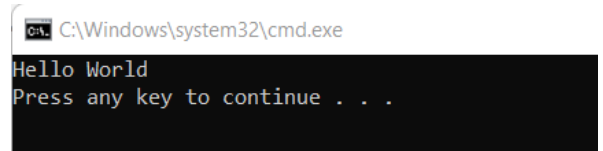


Để viết chương trình C# đầu tiên, chúng ta viết câu lệnh sau đây vào giữa cặp dấu { } của phương thức Main() như sau:

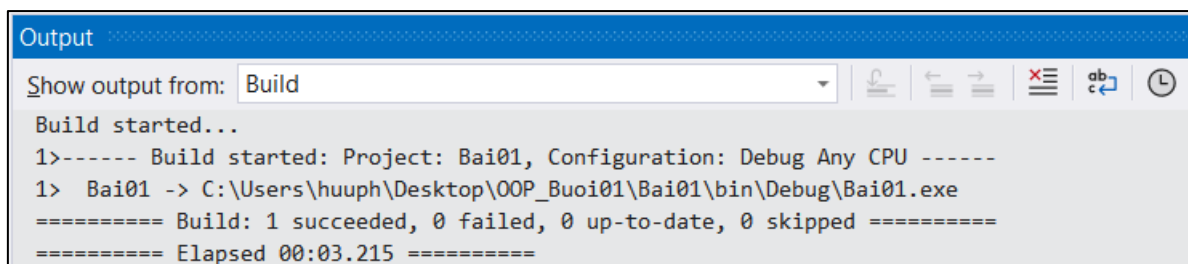
```
static void Main(string[] args)
{
    Console.WriteLine("Hello World");
}
```

Phương thức `Console.WriteLine()` dùng để hiển thị dữ liệu ra màn hình. Trong ví dụ trên, chương trình sẽ hiển thị dòng chữ *Hello World* ra màn hình console.

Để biên dịch và chạy 1 ứng dụng với VS, nhấn phím tắt **Ctrl + F5**. Kết quả như hình bên:



Ngoài ra, cửa sổ **Output** hiển thị kết quả biên dịch và cửa sổ **Error List** sẽ hiển thị các lỗi biên dịch (nếu có):



## IV. Các thành phần cơ bản của ngôn ngữ C#

### 1. Bộ ký tự

C# là một ngôn ngữ có phân biệt chữ hoa/chữ thường. C# sử dụng bộ ký tự sau:

Chữ hoa	A, B, C, ..., Z
Chữ thường	a, b, c, ..., z

Chữ số	0, 1, 2, ..., 9
Ký hiệu toán học	+ - * / % = > < >= <= ...
Ký hiệu đặc biệt	! ; : - ! # [ ] { } ( ) ? & \$ ^ ' " ...

## 2. Tên và cách đặt tên

Khi lập trình, thường phải đặt tên cho các biến số, hằng số, kiểu dữ liệu, hàm... trong chương trình để định danh và phân biệt chúng.

Tên trong C# chỉ được phép sử dụng chữ cái, chữ số, dấu gạch dưới nhưng không được phép có khoảng trắng, không có dấu tiếng Việt và không bắt đầu bằng chữ số.

Khi viết chương trình, nên đặt tên có tính gợi nhớ, diễn tả được ý nghĩa của đối tượng có tên đó nhằm giúp quá trình đọc hiểu code dễ dàng hơn. Trong cùng 1 phạm vi (scope) thì không được có 2 tên trùng nhau.

## 3. Từ khóa

Từ khóa (**keyword** hay **reserved word**) là các từ dành riêng cho một số mục đích nào đó của ngôn ngữ lập trình. Lập trình viên không được phép đặt tên trùng với các từ khóa. Các từ khóa trong C# được tô màu xanh lam.

Danh sách một số từ khóa của C#:

abstract	as	base	bool	break	byte
case	catch	char	checked	class	const
continue	decimal	default	delegate	do	double
else	enum	event	explicit	extern	false
finally	fixed	float	for	foreach	goto
if	implicit	in	int	interface	internal
is	lock	long	namespace	new	null
object	operator	out	override	params	private
protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string
struct	switch	this	throw	true	try
typeof	uint	ulong	unchecked	unsafe	ushort
using	virtual	void	volatile	while	

## 4. Chú thích

Chú thích (**comment**) là những đoạn văn bản mang ý nghĩa giải thích, ghi chú. Khi biên dịch, trình biên dịch sẽ bỏ qua những đoạn chú thích này. Chú thích trong C# được tô màu xanh lục.

Có 2 cách viết chú thích trong C#:

- Viết chú thích vào sau dấu `//`, có tác dụng đến hết dòng hiện tại:  
`// Đây là 1 dòng chú thích`
- Viết chú thích vào giữa 2 dấu `/*` và `*/`:  
`/* Đây là 1 đoạn chú thích  
gồm có nhiều dòng */`

Với ngôn ngữ C#, **khuyến cáo** sử dụng cách chú thích đầu tiên (dùng dấu `//`).

Phím tắt để chuyển đoạn code đang chọn thành chú thích là **Ctrl + K,C** (nếu không có đoạn code nào đang chọn thì áp dụng cho dòng code hiện tại). Phím tắt để bỏ chú thích là **Ctrl + K,U**.

## 5. Kết thúc câu lệnh

Mỗi câu lệnh trong C# phải được kết thúc bằng 1 dấu chấm phẩy (;). Với những đoạn lệnh dài đã nằm trong cặp dấu ngoặc { }, khi kết thúc đoạn lệnh đó không cần dấu ; nhưng mỗi câu lệnh trong đoạn lệnh đó vẫn phải kết thúc bằng dấu ;.

# V. Các kiểu dữ liệu cơ bản, biến và hằng số

## 1. Kiểu dữ liệu

Kiểu dữ liệu (**data type**) là tập hợp các giá trị mà kiểu dữ liệu đó có thể lưu trữ được.

Một số phép toán chỉ có thể được sử dụng trên một số kiểu dữ liệu nhất định. Mỗi kiểu dữ liệu khi lưu vào bộ nhớ sẽ chiếm 1 lượng bộ nhớ nhất định (tính bằng *byte*).

Một số kiểu dữ liệu cơ bản thường dùng trong C#:

Kiểu dữ liệu	Ý nghĩa	Kích thước (byte)	Phạm vi giá trị
<code>char</code>	Ký tự	1	-128 → 127
<code>int</code>	Số nguyên có dấu	4	-2.147.483.648 → 2.147.483.647
<code>unsigned int</code>	Số nguyên không dấu	4	0 → 4.294.967.295
<code>float</code>	Số thực ngắn	4	$3.4E^{-38}$ → $3.4E^{38}$
<code>double</code>	Số thực dài	8	$1.7E^{-308}$ → $1.7E^{308}$
<code>bool</code>	Logic (đúng/sai)	1	<code>true</code> hoặc <code>false</code>
<code>string</code>	Chuỗi ký tự		

## 2. Biến

Biến (**variable**) đại diện cho 1 vùng nhớ dùng để lưu trữ dữ liệu. Biến thường được dùng để chứa giá trị nhập vào từ bàn phím hoặc kết quả các câu lệnh, phép toán. Mỗi biến có 1 tên riêng và không được trùng nhau trong cùng 1 phạm vi.

C# là 1 ngôn ngữ **strongly-typed** (tạm dịch là nghiêm ngặt về kiểu dữ liệu), nghĩa là mỗi biến phải thuộc về 1 kiểu dữ liệu cụ thể ngay khi khai báo và cũng chỉ có thể chứa giá trị tương ứng vào kiểu dữ liệu đó. Ví dụ: biến kiểu **int** không thể chứa dữ liệu ở dạng chuỗi... Trái ngược với 1 vài ngôn ngữ như JavaScript, PHP... là ngôn ngữ **loosely-typed** (tạm dịch là linh hoạt về kiểu dữ liệu) khi mà 1 biến có thể thay đổi kiểu dữ liệu một cách linh hoạt tùy thuộc vào giá trị mà nó đang lưu trữ.

Muốn sử dụng biến phải khai báo trước bằng cú pháp sau (có thể khởi tạo giá trị ngay trong câu lệnh khai báo biến):

```
[kiểu dữ liệu] [tên biến];
```

Ví dụ:

```
int a;  
float b = 8.4;
```

Cách khai báo biến kèm theo kiểu dữ liệu ở trên được gọi là khai báo tường minh (**explicit**).

Ngoài ra, để khai báo 1 biến trong C#, có thể dùng từ khóa **var**, nhưng **bắt buộc** phải gán giá trị cho biến ngay khi khai báo theo cách này, do đó biến được khai báo bằng từ khóa **var** vẫn sẽ có kiểu dữ liệu cụ thể ngay khi khai báo và không vi phạm quy tắc strongly-typed. Cách khai báo biến dạng này được gọi là khai báo ngầm định (**implicit**) và kiểu dữ liệu của biến sẽ được trình biên dịch ngầm hiểu dựa vào giá trị đang gán cho nó:

```
var x = 5;           // x có kiểu int  
var y = "Hello";     // y có kiểu string  
var z;               // Báo lỗi "Implicitly-typed variables  
                    // must be initialized"
```

## 3. Hằng số

Hằng số (**constant**) dùng để lưu trữ dữ liệu tương tự như biến nhưng giá trị của hằng số không được thay đổi sau khi đã khai báo và khởi tạo. Muốn sử dụng hằng số phải khai báo trước bằng cú pháp sau:

```
const [kiểu dữ liệu] [tên hằng] = [giá trị];
```



Ví dụ:

```
const float PI = 3.14;
const int x = (4 + 9) / 2 - 1;
x = 17;    // Báo lỗi "The left-hand side of an assignment must be
           // a variable, property or indexer"
```

## 4. String

Chuỗi (**string**) là 1 cấu trúc dữ liệu cho phép lưu trữ 1 danh sách các ký tự để biểu diễn văn bản, bất kể là 1 ký tự, 1 từ, 1 cụm từ hay là 1 câu. Chuỗi trong C# phải được đặt trong cặp dấu nháy kép " ".

C# cung cấp kiểu dữ liệu **string** hoặc **String** để lưu trữ các chuỗi ký tự:

```
string msg1 = "Hello World";
String msg2 = "Xin chào";
```

Về bản chất, **String** là 1 class của C# và **string** là 1 alias<sup>2</sup> cho class đó. Do đó khai báo chuỗi bằng **String** hay **string** đều là như nhau.

Về lý thuyết, kích thước tối đa cho 1 chuỗi là 2 GB tương ứng với khoảng 1 tỷ ký tự. Tuy nhiên trên thực tế kích thước này sẽ thấp hơn tùy thuộc vào CPU và bộ nhớ của máy tính.

### a) Ký tự đặc biệt (Special character):

Một chuỗi có thể chứa bất kỳ ký tự nào, kể cả các ký tự đặc biệt như ". Do đó, nếu muốn biểu diễn chuỗi *My mom is a "hero" at home* bằng C# sẽ gây ra lỗi:

```
string s = "My mom is a \"hero\" at home";
```

Để sử dụng một số ký tự đặc biệt trong chuỗi, C# cung cấp ký tự \ gọi là **escape character**. Chuỗi nói trên sẽ được biểu diễn như sau:

```
string s = "My mom is a \"hero\" at home";
string t = "C:\\Users\\Admin";    // Biểu diễn chuỗi C:\\Users\\Admin
```

### b) Verbatim string

Nếu trong chuỗi có nhiều ký tự đặc biệt thì việc luôn phải ghi kèm ký tự \ là khá phiền phức. Do đó C# hỗ trợ cách viết chuỗi dưới dạng **Verbatim string** (tạm dịch là chuỗi nguyên gốc). Cách viết này cho phép trong chuỗi có chứa ký tự đặc biệt lẫn xuống dòng, bằng cách bổ sung ký tự @ ở đầu chuỗi như sau:

```
string t = @"C:\Users\Admin";
string v = @"Đây là 1 chuỗi
            nằm trên nhiều dòng";
```

---

<sup>2</sup> Xem phần VI

Tuy nhiên, để biểu diễn ký tự " trong Verbatim string, chúng ta phải dùng ký tự "" để biểu diễn 1 ký tự ":

```
string s = @"My mom is a ""hero"" at home";
```

#### c) Nối chuỗi

Phép nối chuỗi (**string concatenation**) cho phép ghép nối nhiều chuỗi lại với nhau. Phép nối chuỗi được thực hiện bằng toán tử +.

Ví dụ: Hiển thị giá trị của biến x ra màn hình:

```
int x = 5;
Console.WriteLine("Giá trị x = " + x);
```

#### d) String interpolation

Với việc nối chuỗi trong ví dụ trên, đoạn code sẽ khá rối rắm và dễ sai sót, đặc biệt là nếu muốn chèn nhiều giá trị từ biến/biểu thức vào chuỗi:

```
Console.WriteLine("Giá trị x = " + x + ", giá trị y = " + y);
```

C# cung cấp cơ chế **String interpolation** (tạm dịch là chuỗi nội suy) giúp chèn giá trị trực tiếp vào chuỗi một cách đơn giản hơn bằng cách bổ sung ký tự \$ ở đầu chuỗi như sau:

```
Console.WriteLine($"Giá trị x = {x}, giá trị y = {y}");
```

## VI. Alias cho các struct/class kiểu dữ liệu của C#

Ngoài **string** và **String** như đã trình bày ở phần II ở trên, các kiểu dữ liệu trong C# (thuộc namespace **System**) dưới dạng **struct** và **class** hầu hết đều có alias tương ứng. Khi khai báo đối tượng, có thể sử dụng alias hoặc tên của kiểu dữ liệu đều được:

Alias	Kiểu dữ liệu .NET
<b>byte</b>	Byte
<b>short</b>	Int16
<b>int</b>	Int32
<b>long</b>	Int64
<b>float</b>	Single

Alias	Kiểu dữ liệu .NET
<b>double</b>	Double
<b>char</b>	Char
<b>bool</b>	Boolean
<b>string</b>	String
<b>decimal</b>	Decimal

Ví dụ: 2 cách khai báo biến sau đây là như nhau:

```
int x = 5;
Int32 x = 5;
```

## VII. Chuyển đổi kiểu dữ liệu trong C#

Với 1 lượng lớn kiểu dữ liệu cơ bản, C# hỗ trợ nhiều cách chuyển đổi giữa chúng:

- Ép kiểu (**casting**) gồm có 2 dạng là ép kiểu ngầm định (**implicit casting**) và ép kiểu tường minh (**explicit casting**).
- Dùng phương thức **Parse()**.

- Dùng class **Convert**.

## 1. Ép kiểu ngầm định (implicit casting)

Việc ép kiểu ngầm định được trình biên dịch thực hiện tự động khi gán 1 kiểu dữ liệu có kích thước nhỏ hơn cho 1 kiểu dữ liệu có kích thước lớn hơn.

Ví dụ: Khi gán 1 giá trị kiểu **int** cho 1 giá trị kiểu **double**, trình biên dịch sẽ tự động chuyển đổi giá trị **int** thành kiểu **double** để thực hiện phép gán:

```
int x = 5;  
double d = x;           // Implicit casting: int → double với d = 5
```

## 2. Ép kiểu tường minh (explicit casting)

Việc ép kiểu ngầm định phải được lập trình viên thực hiện thủ công khi gán 1 kiểu dữ liệu có kích thước lớn hơn cho 1 kiểu dữ liệu có kích thước nhỏ hơn bằng cách ghi rõ kiểu dữ liệu cần chuyển đổi trong cặp dấu ngoặc (). **Lưu ý:** việc này có thể khiến mất mát dữ liệu khi chuyển đổi.

Ví dụ: Khi gán 1 giá trị kiểu **double** cho 1 giá trị kiểu **int**, cần thực hiện ép kiểu tường minh theo cú pháp sau:

```
double d = 7.92;  
int x = (int)d;           // Explicit casting: double → int với x = 7  
  
int y = d;                // Báo lỗi: "Cannot implicitly convert type  
                           // 'double' to 'int'. An explicit conversion  
                           // exists (are you missing a cast?)"
```

## 3. Dùng phương thức Parse()

Một số kiểu dữ liệu cung cấp phương thức **Parse()** để chuyển 1 chuỗi về kiểu dữ liệu đó:

- Nếu chuyển đổi thành công, phương thức **Parse()** trả về giá trị ở kiểu dữ liệu mới:

```
int x = int.Parse("297");           // x = 297
```

- Nếu chuỗi là **null**, trả về **ArgumentNullException**:

```
int x = int.Parse(null);           // ArgumentNullException
```

- Nếu chuỗi không đúng định dạng của kiểu dữ liệu cần chuyển, trả về **FormatException**:

```
int x = int.Parse("");           // FormatException  
int y = int.Parse("12a");         // FormatException
```

- Nếu chuỗi vượt quá khoảng giá trị của kiểu dữ liệu, trả về **OverflowException**:

```
int x = int.Parse("50000000000"); // OverflowException
```

Lưu ý: Khái niệm về **Exception** (ngoại lệ) sẽ được trình bày ở các buổi sau.

#### 4. Dùng class `Convert`

C# cung cấp class `Convert` gồm rất nhiều phương thức giúp chuyển đổi giữa các kiểu dữ liệu cơ bản một cách đơn giản. Các phương thức này có dạng `.To[Kiểu dữ liệu]()`.

Ví dụ:

Mục đích	Cú pháp ví dụ
Chuyển đổi <code>int</code> → <code>double</code> :	<code>Convert.ToDouble(352)</code>
Chuyển đổi <code>double</code> → <code>int</code> :	<code>Convert.ToInt32(984574.62304)</code>
Chuyển đổi <code>int</code> → <code>bool</code> :	<code>Convert.ToBoolean(1)</code>
Chuyển đổi <code>string</code> → <code>DateTime</code> :	<code>Convert.ToDateTime("2022/02/18")</code>
Chuyển đổi <code>DateTime</code> → <code>string</code> :	<code>Convert.ToString(DateTime.Now);</code>

Ngoài ra, do kiểu `string` vô cùng phổ biến khi viết ứng dụng với C#, do đó, bất kỳ đối tượng thuộc kiểu dữ liệu nào cũng có thể chuyển đổi về kiểu `string` bằng phương thức `.ToString()` (thay vì phải dùng `Convert.ToString()`):

```
int x = 846;  
Console.WriteLine(x.ToString());  
Console.WriteLine(DateTime.Now.ToString());
```

#### VIII. Nhập, xuất cơ bản

Để xuất dữ liệu ra màn hình dòng lệnh, chúng ta dùng phương thức `Console.Write()` hoặc `Console.WriteLine()`. Điểm khác biệt là phương thức `Console.WriteLine()` sẽ tự động xuống dòng sau khi xuất dữ liệu còn phương thức `Console.Write()` thì không.

Snippet<sup>3</sup> cho phương thức `Console.WriteLine()` là cw.

Để nhận dữ liệu nhập từ bàn phím, chúng ta dùng phương thức `Console.ReadLine()`. Phương thức này sẽ đọc toàn bộ 1 dòng dữ liệu nhập và trả kết quả về dưới dạng `string`.

Ví dụ: Viết chương trình nhập vào 2 số nguyên. Xuất ra màn hình tổng của 2 số đó:

```
Console.OutputEncoding = Encoding.UTF8;  
int a, b;  
Console.Write("Nhập số nguyên a: ");  
a = int.Parse(Console.ReadLine());  
Console.Write("Nhập số nguyên b: ");  
b = int.Parse(Console.ReadLine());  
int c = a + b;  
Console.WriteLine("Tổng 2 số là " + c);
```

---

<sup>3</sup> Từ viết tắt để gõ nhanh 1 đoạn lệnh. Thông thường sẽ gõ snippet và nhấn Tab để VS tự hoàn tất phần còn lại.

Trong đó:

- Câu lệnh `Console.OutputEncoding = Encoding.UTF8;` chuyển đổi bảng mã của màn hình console thành UTF8 (Unicode) cho phép hiển thị tiếng Việt có dấu.
- Phương thức `int.Parse()` chuyển đổi chuỗi ký tự nhận được từ phương thức `Console.ReadLine()` thành số nguyên.

## IX. Toán tử

Trong lập trình, toán tử (**operator**) là các kí hiệu dùng để thực hiện các phép toán.

Ví dụ: Trong câu lệnh `c = a + b;` thì `a`, `b`, `c` là toán hạng (**operand**) còn `+` và `=` là các toán tử.

### 1. Toán tử gán

Toán tử gán (**assignment operator**) dùng để gán giá trị cho biến. Giá trị này có thể là 1 hằng số hoặc 1 biểu thức. Có thể kết hợp phép gán và câu lệnh khai báo biến. Cú pháp của phép gán:

```
[tên biến] = [giá trị/biểu thức];
```

Ví dụ:

```
int a = 5;  
int b, c;  
b = a + 2;  
a = a + 1;  
c = 7;
```

### 2. Toán tử số học

Toán tử số học (**arithmetic operator**) dùng để thực hiện các phép toán số học:

Toán tử	Ý nghĩa	Ví dụ
+	Cộng	$5 + 7 \rightarrow 12$
-	Trừ	$8.5 - 6 \rightarrow 2.5$
*	Nhân	$4 * 9 \rightarrow 36$
/	Chia lấy phần nguyên Chia lấy thương	$7 / 2 \rightarrow 3$ $7.0 / 2 \rightarrow 3.5$
%	Chia lấy phần dư (modulo)	$7 \% 2 \rightarrow 1$

Toán tử số học chỉ sử dụng trên các kiểu số nguyên hoặc số thực. Nếu tất cả các toán hạng trong biểu thức là số nguyên, kết quả biểu thức sẽ là số nguyên. Ngược lại, nếu có ít nhất 1 toán hạng là số thực, kết quả biểu thức sẽ là số thực.

Ngoại lệ duy nhất là toán tử + có thể sử dụng đối với kiểu ký tự (**char**) hoặc chuỗi (**string**). Tuy nhiên, khi dùng theo cách này thì đây là phép nối chuỗi (**string concatenate**) chứ không phải là 1 phép toán số học:

```
| string s = "Tổng 2 số là " + (5 + 9);
```

### 3. Toán tử phức hợp

Toán tử phức hợp (**compound-assignment operator**) kết hợp toán tử gán và toán tử số học.

Ví dụ:

```
| a += 5;    // Tương đương phép toán a = a + 5;  
| x -= y;    // Tương đương phép toán x = x - y;
```

### 4. Toán tử so sánh

Toán tử so sánh (**comparison operator**) dùng để so sánh 2 giá trị với nhau và trả về đúng (**true**)/sai (**false**):

Toán tử	Ý nghĩa	Ví dụ
==	Bằng	5 == 2 + 3 → <b>true</b>
!=	Khác	7 != 9 → <b>true</b>
>	Lớn hơn	4 > 8 → <b>false</b>
>=	Lớn hơn hoặc bằng	7 >= 3 + 4 → <b>true</b>
<	Nhỏ hơn	-5.2 < 9 → <b>true</b>
<=	Nhỏ hơn hoặc bằng	-6.9 <= -9.6 → <b>false</b>

### 5. Toán tử logic

Toán tử logic (**logical operator**), còn gọi là toán tử luận lý, dùng để nối các mệnh đề logic và cho kết quả là **true/false**:

Toán tử	Ý nghĩa	Ví dụ
!	Phủ định	!(5 < 7) → <b>false</b>
&&	Và	(8.5 > 16) && (7 != 2) → <b>false</b>
	Hoặc	(4 > 8)    (2 == 2) → <b>true</b>

### 6. Toán tử tăng, giảm

Toán tử tăng, giảm (**increment & decrement operator**) dùng để tăng, giảm giá trị của biến 1 đơn vị.

Câu lệnh **x++**; cho kết quả tương đương với câu lệnh **x = x + 1**; hoặc **x += 1**;

Câu lệnh **x--**; cho kết quả tương đương với câu lệnh **x = x - 1**; hoặc **x -= 1**;

## X. Các cấu trúc điều khiển cơ bản

### 1. Cấu trúc rẽ nhánh

Cấu trúc rẽ nhánh (còn gọi là cấu trúc điều kiện) dùng để cài đặt 1 công việc sẽ được thực hiện khi 1 điều kiện cụ thể được thỏa mãn. Cấu trúc rẽ nhánh gồm có các dạng sau:

- Câu lệnh **if** và **if-else**
- Câu lệnh **switch**
- Toán tử điều kiện

#### a) Câu lệnh **if** và **if-else**

Cú pháp của câu lệnh **if**:

```
if ([biểu thức điều kiện])  
{  
    // Các câu lệnh cần thực hiện nếu điều kiện đúng  
    // (Nếu điều kiện sai, không có gì xảy ra)  
}
```

Cú pháp của câu lệnh **if-else**:

```
if ([biểu thức điều kiện])  
{  
    // Các câu lệnh cần thực hiện nếu điều kiện đúng  
}  
else  
{  
    // Các câu lệnh cần thực hiện nếu điều kiện sai  
}
```

Ví dụ: Kiểm tra biến x là số chẵn hay số lẻ:

```
if (x % 2 == 0)  
{  
    Console.WriteLine("x là số chẵn");  
}  
else  
{  
    Console.WriteLine("x là số lẻ");  
}
```

#### b) Câu lệnh **switch**

Câu lệnh **switch** dùng để cài đặt 1 công việc sẽ được thực hiện khi 1 biến/biểu thức mang 1 giá trị cụ thể.

Cú pháp của câu lệnh **switch**:

```
switch ([biến/biểu thức])  
{  
    case [giá trị #1]:
```

```

        // Các câu lệnh cần thực hiện với giá trị #1
        break;
    case [giá trị #2]:
        // Các câu lệnh cần thực hiện với giá trị #2
        break;
    .....
    default:
        // Các câu lệnh mặc định nếu các giá trị đều không khớp
        break;
}

```

Ví dụ: Hiển thị tên tiếng Việt của tháng được lưu trong biến month:

```

switch (month)
{
    case 1: Console.WriteLine("Tháng Một"); break;
    case 2: Console.WriteLine("Tháng Hai"); break;
    case 3: Console.WriteLine("Tháng Ba"); break;
    // Tương tự cho các case còn lại từ 4 -> 12
    default: Console.WriteLine("Tháng không hợp lệ"); break;
}

```

### c) Toán tử điều kiện

Toán tử điều kiện (**conditional operator**) hay còn gọi là toán tử 3 ngôi (**ternary operator**) trả về 1 giá trị tùy thuộc vào biểu thức điều kiện là đúng hay sai.

Cú pháp của toán tử điều kiện:

[biểu thức điều kiện] ? [giá trị nếu đúng] : [giá trị nếu sai]

Ví dụ: Tính giá trị biểu thức sau:  $f(x) = \begin{cases} 3x + 2 & \text{nếu } x \geq 0 \\ 7 - 9x & \text{nếu } x < 0 \end{cases}$

```
int fx = (x >= 0) ? (3 * x + 2) : (7 - 9 * x);
```

## 2. Cấu trúc lặp

Cấu trúc lặp (còn gọi là vòng lặp – **loop**) dùng để cài đặt 1 công việc sẽ được thực hiện lặp đi lặp lại khi 1 điều kiện cụ thể được thỏa mãn. Cấu trúc lặp gồm có các dạng sau:

- Vòng lặp biết trước số lần lặp: **for**
- Vòng lặp không biết trước số lần lặp: **while** và **do-while**

### a) Vòng lặp **for**

Vòng lặp **for** thường được dùng cho các công việc được lặp đi lặp lại với 1 số lần biết trước.

Cú pháp của vòng lặp **for**:

```

for ([khởi tạo biến lặp]; [biểu thức điều kiện]; [thay đổi biến lặp])
{
    // Các câu lệnh cần thực hiện
}

```



Cơ chế hoạt động của vòng lặp **for**:

- Bước 1: Biểu thức đầu tiên của vòng lặp **for** sẽ được thực hiện 1 lần duy nhất. Biểu thức này thường dùng để khởi tạo biến lặp để điều khiển số lần lặp.
- Bước 2: Nếu biểu thức điều kiện đúng, các câu lệnh trong vòng lặp sẽ được thực hiện. Nếu không, vòng lặp kết thúc.
- Bước 3: Biểu thức cuối cùng của vòng lặp **for** sẽ được thực hiện. Biểu thức này thường dùng để thay đổi giá trị biến lặp. Sau đó quay lại bước 2.

Ví dụ: Hiển thị ra màn hình 10 lần dòng chữ "I love C#":

```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine("I love C#");  
}
```

#### b) Vòng lặp **while** và **do-while**

Vòng lặp **while** và **do-while** thường được dùng cho các công việc được lặp đi lặp lại với số lần không biết trước.

Cú pháp của vòng lặp **while**:

```
while ([biểu thức điều kiện])  
{  
    // Các câu lệnh cần thực hiện  
}
```

Cú pháp của vòng lặp **do-while**:

```
do  
{  
    // Các câu lệnh cần thực hiện  
} while ([biểu thức điều kiện]);
```

Cơ chế hoạt động của vòng lặp **while** và **do-while** giống nhau, đó là chỉ cần biểu thức điều kiện đúng là các câu lệnh sẽ được thực hiện. Khác biệt giữa 2 vòng lặp này đó là vòng lặp **while** kiểm tra điều kiện trước khi thực hiện công việc, còn vòng lặp **do-while** thì thực hiện công việc trước 1 lần rồi mới kiểm tra điều kiện.

Ví dụ: Tìm giá trị x nhỏ nhất sao cho tổng các số từ 1 đến x lớn hơn 50:

```
int x = 0, sum = 0;  
while (sum <= 50)  
{  
    x++;  
    sum += x;  
}  
Console.WriteLine("Giá trị x cần tìm là: " + x);
```

## XI. Bài tập

Thực hiện các bài tập sau, mỗi bài nằm trong 1 project của solution **OOP\_Buoi01** đã tạo:

**Bài 1:** Nhập số nguyên dương N. Tính giá trị các biểu thức sau:

$$S_1 = 1 + 2 + 3 + \dots + N$$

$$S_2 = 1 \times 2 \times 3 \times \dots \times N$$

$$S_3 = 1 - 2 + 3 - 4 + 5 - 6 + \dots + N$$

$$S_4 = 1^2 + 2^2 + 3^2 + \dots + N^2$$

$$S_5 = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{N}$$

$$S_6 = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \dots + \frac{N}{N+1}$$

$$S_7 = 1! + 2! + 3! + \dots + N!$$

**Bài 2:** Nhập 1 số nguyên dương có 3 chữ số. Tính tổng 3 chữ số đó.

**Bài 3:** Nhập 1 số nguyên dương bất kỳ. Tính tổng tất cả chữ số của số đó.

**Bài 4:** Nhập 1 số nguyên dương. Cho biết số đó có phải số nguyên tố không (Số nguyên tố là số chỉ có đúng 2 ước số là 1 và chính nó). Ví dụ: 2, 3, 17, 31, 53... là số nguyên tố.

**Bài 5:** Nhập 1 số nguyên dương. Cho biết số đó có phải số hoàn chỉnh không (Số hoàn chỉnh là số bằng tổng của tất cả các ước số của nó, ngoại trừ nó).

Ví dụ:

- Số 6 là số hoàn chỉnh vì 6 có các ước số là 1, 2, 3, 6, và  $6 = 1 + 2 + 3$ .
- Số 28 là số hoàn chỉnh vì 28 có các ước số 1, 2, 4, 7, 14, 28, và  $28 = 1 + 2 + 4 + 7 + 14$ .

**Bài 6:** Nhập 2 số nguyên. Tìm ước chung lớn nhất và bội chung nhỏ nhất của 2 số này.