XUẤT NHẬP CONSOLE

Giao diện console (console user interface, CUI), còn gọi là giao diện dòng lệnh (command line interface, CLI), là loại giao diện đơn giản nhất, trong đó dữ liệu xuất nhập đều là văn bản. Mặc dù không có gì hấp dẫn, ứng dụng với giao diện console (thường gọi tắt là ứng dụng console) luôn được sử dụng để dạy ngôn ngữ lập trình. Tương tự, trong khi học lập trình C# căn bản bạn sẽ gắn bó với ứng dụng console. Sự đơn giản của loại ứng dụng này giúp bạn tập trung vào các đặc trưng của ngôn ngữ, thay vì phân tâm cho sự phức tạp của giao diện đồ họa (Graphical User Interface) hay giao diện web.

Bài học này sẽ cung cấp cho bạn đầy đủ các kỹ thuật để làm việc với giao diện console trong C#: nhập, xuất, định dạng. Nắm chắc các kỹ thuật này sẽ giúp đơn giản hóa một phần việc học lập trình C# về sau.

Giao diện console

Vai trò của ứng dụng console

Ứng dụng console là loại ứng dụng có giao diện đơn giản nhất, giúp người mới học lập trình tránh xa những sự phức tạp (chưa cần thiết) của các công nghệ GUI (Graphical User Interface – giao diện đồ họa).

Ứng dụng console sử dụng nhiều cho mục đích tính toán và xử lý, thay vì thể hiện dữ liệu. Nhiều chương trình tính toán nặng (trong khoa học) hay được viết bằng console thay vì dùng GUI. Các chương trình server thì hoàn toàn không cần đến GUI.

Ứng dụng console có hiệu suất cao và an toàn. Trên thực tế, rất nhiều ứng dụng dành cho quản trị hệ thống đều sử dụng giao diện console. Chắc bạn đã từng dùng Ping, IpConfig trên windows. Windows server còn có hẳn một giao diện dòng lệnh cao cấp (powershell) với hầu hết các công cụ đều thực hiện ở giao diện dòng lệnh. Nếu làm việc với Linux thì ... thôi rồi, toàn chương trình console chạy trên shell.

Ứng dụng console không phụ thuộc quá chặt chẽ vào shell đồ họa của hệ thống và dễ dàng phục vụ đa nền tảng. Ví dụ, một ứng dụng console viết trên .NET framework (cho windows) có thể dễ dàng chạy tiếp trên Linux (Mono).

Các vấn đề của giao diện console trong C

Vấn đề hiển nhiên nhất với ứng dụng console là việc xuất nhập dữ liệu. Mọi thứ xuất ra console đều là văn bản. Mọi thứ đọc vào từ console cũng đều là văn bản. Từ đây dẫn đến yêu cầu phải chuyển đổi kiểu dữ liệu khi xuất/nhập dữ liệu.

Vấn đề thứ hai là nhận lệnh và quyết định phương thức tương ứng nào sẽ được thực thi. Đối với các chương trình nhỏ thì đây không phải là vấn đề. Người mới học dễ dàng nghĩ tới sử dụng các cấu trúc rẽ nhánh để chọn phương thức phù hợp. Vậy nếu như có vài chục lệnh hoặc nhiều hơn nữa?

Vấn đề thứ ba là tham số đi kèm mỗi lệnh. Một lệnh người dùng nhập không thể thiếu tham số. Ví dụ lệnh “delete xyz.exe” có “xyz.exe” là tham số đi cùng lệnh. Làm thế nào để dễ dàng tiếp nhận, chuyển đổi và sử

dụng tham số đó với lệnh?

Lớp `System.Console`

Trong C# lớp System.Console để thực hiện các công việc với giao diện console. Tất cả các thao tác quan trọng như xuất/nhập, định dạng dữ liệu xuất ra đều được thực hiện qua lớp này.

Dưới đây là một số thành viên của lớp Console:

- `Write()`, `WriteLine()`: in thông tin ra console
- `Read()`, `ReadLine()`, `ReadKey()`: đọc thông tin từ console
- `Beep()`: phát tiếng bíp ra loa
- `BackgroundColor`: đặt màu nền cho văn bản
- `ForegroundColor`: đặt màu văn bản
- `Title`: Đặt tiêu đề cho cửa sổ console
- `Clear()`: xóa nội dung của console
- `BufferWidth`/`BufferHeight`: đặt kích thước buffer cho console
- `WindowWidth`/`WindowHeight`: đặt kích thước của console
- `WindowTop`/`WindowLeft`: đặt vị trí của console
- `OutputEncoding`: Loại mã của console

Xuất dữ liệu ra console

Để xuất dữ liệu ra giao diện console trong C# có thể sử dụng phương thức `Console.Write` hoặc `Console.WriteLine`. Hai phương thức này có thể nhận tham số thuộc bất kỳ kiểu nào, bất kỳ số lượng nào. `Write` hoặc `WriteLine` sau đó sẽ gọi tới phương thức `ToString()` của kiểu dữ liệu đó để chuyển tham số sang chuỗi ký tự và in ra console.

Lưu ý rằng, `ToString()` là phương thức được tất cả các kiểu kế thừa từ kiểu object (`System.Object`). Mỗi kiểu sẽ ghi đè `ToString()` theo cách riêng cho phù hợp. Bạn cũng có thể tự ghi đè phương thức này trong class do mình xây dựng.

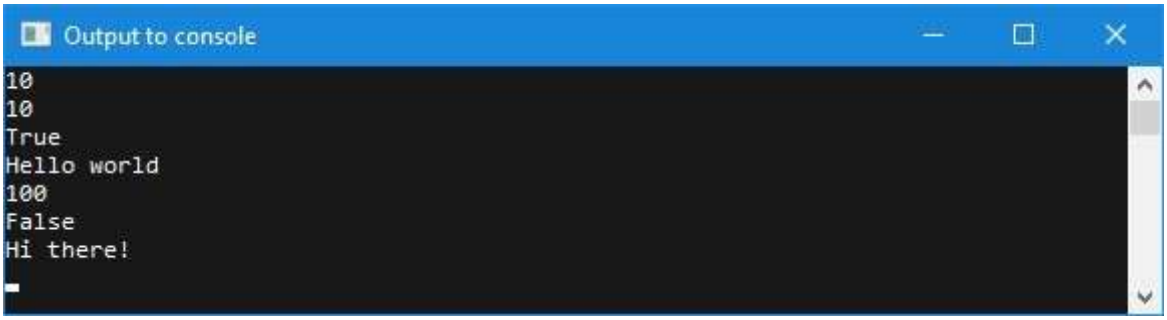
`WriteLine` khác biệt với Write ở duy nhất một điểm: Sau khi in thông tin ra console, `WriteLine` sẽ xuống một dòng mới và di chuyển con trỏ văn bản về đầu dòng mới, trong khi `Write` giữ nguyên vị trí con trỏ ở sau ký tự cuối cùng được in ra.

Cùng thực hiện ví dụ đơn giản sau đây.

Tạo solution trống đặt tên là S04_Console. Thêm project Console App đặt tên là P01_ConsoleOutput. Viết code cho Program.cs như sau:

```
using System;
namespace P01_ConsoleOutput
{
    class Program
    {
        static void Main()
```

```
        {
            Console.WriteLine(10); // in ra một số nguyên
            Console.WriteLine(10.0); // in ra một số thực
            Console.WriteLine(true); // in ra một giá trị logic
            Console.WriteLine("Hello world"); // in ra một chuỗi
            int a = 100;
            Console.WriteLine(a); // in ra một biến kiểu số nguyên
            bool b = false;
            Console.WriteLine(b); // in ra một biến kiểu logic
            string c = "Hi there!";
            Console.WriteLine(c); // in ra một biến kiểu xâu ký tự
            Console.ReadKey();
        }
    }
}
```



in ra console bằng writeline

Bạn hãy tự thử thay WriteLine bằng Write trong ví dụ trên để thấy sự khác biệt.

Đối với các kiểu dữ liệu phức tạp (như class) thì phương thức `WriteLine/Write` chỉ đơn giản là in ra tên đầy đủ của kiểu dữ liệu mà không thể in ra các dữ liệu mà object đang chứa. Do đó, bạn cần tự mình ghi đè phương thức `ToString()` trên class mình xây dựng nếu muốn dùng trực tiếp với `Write`/`WriteLine`.

Thay đổi màu sắc cho console

Giao diện console nền đen chữ trắng không chỉ nhàm chán mà còn khó theo dõi khi hiển thị nhiều dữ liệu. Lớp Console cho phép thay đổi màu nền và màu văn bản in ra trên console thông qua thiết lập giá trị `BackgroundColor` và `ForegroundColor` trước khi gọi lệnh in Write hoặc WriteLine. Nếu muốn trả lại màu sắc mặc định chỉ cần gọi phương thức `ResetColor()`.

Hãy cùng thực hiện ví dụ sau:

Thêm project P03_ColorfulConsole vào solution và viết code cho Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace P03_ColorfulConsole
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Colorful Console";
            Console.ForegroundColor = ConsoleColor.Magenta;
            Console.Write("Your name: ");
            Console.ResetColor();
            var name = Console.ReadLine();
            Console.BackgroundColor = ConsoleColor.Blue;
            Console.Write(name);
            Console.ReadLine();
        }
    }
}
```

Các hằng số màu được định nghĩa trong kiểu liệt kê (enum) `ConsoleColor` với 16 giá trị màu khác nhau. Một khi giá trị màu sắc đã được thiết lập nó sẽ có tác dụng đến khi gặp thiết lập màu khác, hoặc đến khi gặp `ResetColor()`.

Trong ví dụ trên chúng ta cũng sử dụng property `Console.Title` để đặt tiêu đề cho cửa sổ console.

Nhập dữ liệu từ console

So với xuất dữ liệu, việc đọc dữ liệu từ console phức tạp hơn một chút. Để nhập dữ liệu từ giao diện dòng lệnh có thể sử dụng các phương thức của lớp `Console`, bao gồm `ReadLine`, `ReadKey`, `Read`.

`ReadLine`

`ReadLine` đọc một dòng và trả về một chuỗi ký tự.

Khi sử dụng phương thức `ReadLine`, màn hình console sẽ dừng lại chờ nhập dữ liệu. Người dùng có thể nhập nhiều ký tự liên tục cho đến khi bấm phím Enter để kết thúc nhập.

Do `ReadLine` luôn trả về một chuỗi ký tự, chúng ta cần chuyển đổi chuỗi sang các kiểu khác theo nhu cầu.

Phương thức `ReadLine` cũng có thể dùng để dừng màn hình console chờ người dùng bấm Enter để tiếp tục.

Cùng thực hiện ví dụ sau. Tạo thêm project mới P02_ReadLine trong solution và viết code cho Program.cs:

```
using System;
namespace P02_ReadLine
{
    class Program
    {
        static void Main(string[] args)
```

```
        {
            Console.Write("Enter a number: ");
            string number = Console.ReadLine();
            Console.WriteLine(number);
            Console.Write("Enter true or false: ");
            string logic = Console.ReadLine();
            Console.WriteLine(logic);
            Console.Write("Enter a string: ");
            string message = Console.ReadLine();
            Console.WriteLine(message);
            Console.Write("Press enter to quit");
            Console.ReadLine(); // dừng màn hình chờ ấn enter
        }
    }
}
```

ReadKey

`ReadKey` đọc một ký tự và trả về kiểu `ConsoleKeyInfo`. Khi sử dụng `ReadKey`, màn hình console sẽ dừng lại chờ nhập dữ liệu.

Khi người dùng bấm một phím bất kỳ, `ReadKey` sẽ đọc và kết thúc nhập ngay lập tức (không cần dùng phím Enter để kết thúc nhập như `ReadLine`). Thông tin về ký tự người dùng đã nhập được trả về một biến thuộc kiểu `ConsoleKeyInfo`.

Phương thức `ReadKey` cũng thường được sử dụng để dừng màn hình console chờ người dùng bấm một phím bất kỳ để tiếp tục.

Thêm project P04_ReadKey vào solution và viết code cho Program.cs như sau:

```
using System;
namespace P04_ReadKey
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "ReadKey";
            Console.Write("Press any key: ");
            var key = Console.ReadKey();
            Console.WriteLine();
            Console.WriteLine(key.KeyChar);
            Console.WriteLine(key.Modifiers);
            if (key.Key == ConsoleKey.H)
                Console.WriteLine("Hello!");
            Console.ReadKey();
        }
    }
}
```

Kiểu `ConsoleKeyInfo` lưu trữ một số thông tin về phím người dùng đã bấm:

- `KeyChar` (kiểu char): ký tự người dùng đã nhập;
- Modifiers (kiểu enum `ConsoleModifiers`): các phím điều khiển bấm cùng như Ctrl, Shift, Alt;
- Key (kiểu enum `ConsoleKey`): phím chuẩn trên console.

ConsoleKey và char hoàn toàn không giống nhau, và cũng không thể sử dụng lẫn lộn nhau. Kiểu char bạn đã biết qua bài học các kiểu dữ liệu cơ sở của C#. Bạn có thể đọc thêm bài viết về ConsoleKey enum.

Read

`Read` đọc một ký tự và trả về mã của ký tự đó. Khi sử dụng `Read`, màn hình console sẽ dừng chờ nhập dữ liệu.

Người dùng có thể nhập cả một chuỗi ký tự và bấm phím Enter để kết thúc nhập. Tuy nhiên, `Read` chỉ trả về mã của ký tự đầu tiên của chuỗi người dùng nhập.

Một số vấn đề khi làm việc với console

Đơn giản hóa lời gọi phương thức của Console

Từ C# 6 bắt đầu đưa vào cấu trúc **using static** để gọi tắt tới các thành viên static của một class. Lớp Console chứa toàn các thành viên static và rất phù hợp với cách dùng này. Hãy cùng làm lại ví dụ trên:

```
using System;
using static System.Console;
namespace P03_ColorfulConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            Title = "Colorful Console";
            ForegroundColor = ConsoleColor.Magenta;
            Write("Your name: ");
            ResetColor();
            var name = ReadLine();
            BackgroundColor = ConsoleColor.Blue;
            Write(name);
            ReadLine();
        }
    }
}
```

Bạn đã bớt kha khá code với cách viết này khi không cần lặp lại tên class Console khi gọi phương thức hay thành viên tĩnh của nó nữa.

Biến đổi dữ liệu đọc từ console

Lấy ví dụ, bạn yêu cầu người dùng nhập vào một số nguyên. Do dữ liệu nhập từ console sử dụng ReadLine đều là string, bạn phải biến đổi chuỗi đó thành số thì mới sử dụng được trong biểu thức. Ngoài ra, do người dùng có thể vô tình nhập lẫn cả chữ cái vào chuỗi chữ số dẫn đến lỗi khi biến đổi.

Như vậy, khi đọc dữ liệu từ console bạn phải thực hiện hai nhiệm vụ: kiểm tra tính chính xác của chuỗi, biến đổi chuỗi về giá trị thuộc kiểu mình cần.

Hầu hết các kiểu dữ liệu cơ bản của C# đã hỗ trợ thực hiện các nhiệm vụ này với phương thức `Parse()` và `TryParse()`. Loại biến đổi dữ liệu này được gọi là **chuyển kiểu** (**type conversion**), để phân biệt với **ép kiểu** (type casting).

```
Console.Write("Enter an integer: ");
string input = Console.ReadLine(); // đọc vào một chuỗi chữ số
int i = int.Parse(input); // biến đổi input thành int
Console.Write("Enter true or false: ");
input = Console.ReadLine(); // đọc chuỗi "true" hoặc "false"
bool b = bool.Parse(input); // biến đổi chuỗi thành bool
Console.Write("Enter a double: ");
input = Console.ReadLine(); // đọc vào một chuỗi chữ số
double d = double.Parse(input); // biến đổi input thành int
```

In chuỗi có định dạng với placeholder

Hãy cùng xem ví dụ sau đây:

```
int x1 = 123, x2 = 456;
Console.WriteLine("Nghiệm thứ nhất: {0}, Nghiệm thứ hai: {1}", x1, x2);
```

Phương thức `Write`/`WriteLine` có thể in ra một xâu ký tự đồng thời ghép các biến vào những vị trí đánh dấu sẵn trong một xâu mẫu.

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.

In chuỗi có định dạng với string interpolation

Bắt đầu từ C# 6 xuất hiện một phương pháp định dạng xâu mới gọi là *interpolated*.

Một xâu interpolated được bắt đầu bằng ký tự `$`, ở những vị trí cần thay thế bằng giá trị từ biểu thức/biến, biểu thức/biến đó sẽ đặt trong cặp dấu `{}`.

Khi gặp loại xâu này, C# sẽ tính giá trị của biểu thức/lấy giá trị biến và đặt vào đúng vị trí quy định.

Ví dụ trên có thể viết lại theo cách mới như sau:

```
int x1 = 123, x2 = 456;
Console.WriteLine($"Nghiệm thứ nhất: {x1}, Nghiệm thứ hai: {x2}");
```

Cách viết này rõ ràng, sáng sủa và dễ đọc hơn nhiều. Trong cặp dấu `{}` bạn có thể để bất kỳ biểu thức nào của C#, thậm chí cả những biểu thức phức tạp như phép toán điều kiện hay các đoạn code.

Định dạng số khi in ra console

Trong nhiều tình huống bạn cần in số có định dạng. Ví dụ khi in 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 in số (như trong bảng biểu). Bạn muốn số in ra căn lề trái hoặc căn lề phải.

Các lệnh in của Console hỗ trợ thực hiện các thao tác định dạng này. Định dạng in cho mỗi số bao gồm 3 phần: alignment, format và precision.

Để thử nghiệm, bạn hãy dùng C# Interactive.

> Lưu ý: cách viết số này chỉ áp dụng được trong chuỗi có định dạng với placeholder hoặc string interpolation.

Format specifier

Ví dụ, để in ra 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.

```
> Console.WriteLine("The value: {0}", 500)
The value: 500
> Console.WriteLine("The value: {0:C}", 500)
The value: $500.00
> Console.WriteLine("The value: {0:c}", 500)
The value: $500.00
> decimal value = 500;
. Console.WriteLine($"The value: {value:C}");
The value: $500.00
> Console.WriteLine($"The value: {value:c}");
The value: $500.00
>
```

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.

Bạn có thể tự mình thử nghiệm các định dạng trên để xem kết quả.

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.

```
> Console.WriteLine($"The value: {value:c3}");
The value: $500.000
> Console.WriteLine($"The value: {value:c4}");
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:

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

```
1.2346e+001
>
```

Alignment specifier

Để dễ hiểu, hãy xem ví dụ sau:

```
> int myInt = 500;
> Console.WriteLine("|{0, 10}|", myInt);
. Console.WriteLine("|{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:

```
> double myDouble = 12.345678;
. Console.WriteLine("{0,-10:G} -- General", myDouble);
. Console.WriteLine("{0,-10} -- Default, same as General", myDouble);
. Console.WriteLine("{0,-10:F4} -- Fixed Point, 4 dec places", myDouble);
. Console.WriteLine("{0,-10:C} -- Currency", myDouble);
. Console.WriteLine("{0,-10:E3} -- Sci. Notation, 3 dec places", myDouble);
. Console.WriteLine("{0,-10:x} -- Hexadecimal integer", 1194719);
12.345678  -- General
12.345678  -- Default, same as General
12.3457    -- Fixed Point, 4 dec places
$12.35     -- Currency
1.235E+001 -- Sci. Notation, 3 dec places
123adf     -- Hexadecimal integer
>
```