Các cấu trúc điều khiển trong C#

Bài học này sẽ hướng dẫn bạn cách sử dụng các cấu trúc điều khiển cơ bản trong C#. Bạn có thể sẽ gặp những người quen như if-else, switch-case, while, do-while nếu bạn biết C hay Java. Tuy nhiên, việc sử dụng các cấu trúc điều khiển này trong C# có những điểm khác biệt nhất định mà bạn cần biết. C# 7 thậm chí còn đưa thêm một số đặc điểm của lập trình hàm (functional programming) vào cấu trúc điều khiển cơ bản.

Hãy theo dõi bài học đến cuối để nắm được các chi tiết.

NỘI DUNG CỦA BÀI [Ẩn]

Mặc định trong C# và hầu hết các ngôn ngữ lập trình imperative, các lệnh được thực hiện lần lượt theo thứ tự được chỉ định trong code. Trình tự thực hiện các lệnh thường được gọi là luồng thực thi (flow of execution). Tuy nhiên, nếu chỉ thực thi lệnh theo thứ tự thì khả năng của chương trình sẽ bị giới hạn. Từ đó dẫn tới trong các ngôn ngữ lập trình đều phải đưa ra các cấu trúc điều khiển (flow control). Các cấu trúc này có tác dụng thay đổi trật tự thực thi lệnh thông thường.

Nhìn chung các cấu trúc điều khiển được xếp vào các nhóm: cấu trúc điều kiện (còn gọi là cấu trúc rẽ nhánh), cấu trúc lặp, cấu trúc nhảy.

Cấu trúc điều kiện (rẽ nhánh) trong C#: if-else, switch-case

Cấu trúc điều kiện, còn gọi là cấu trúc rẽ nhánh cho phép phân tách việc thực thi code thành nhiều hướng khác nhau tùy thuộc vào một điều kiện nào đó. Điều kiện này thông thường được xác định theo giá trị của biến/biểu thức.

C# sử dụng hai cấu trúc điều kiện: cấu trúc if-else và cấu trúc switch-case.

Cấu trúc rẽ nhánh if else

Cấu trúc if-else của C# hoàn toàn thừa kế từ C/C++. Cú pháp của cấu trúc `if-else` như sau:

```
if (condition)
{
    statements 1
}
else
{
     statements 2
}
```

Trong đó, condition là *biểu thức logic* – biểu thức mà giá trị trả về là true hoặc false. Đây là kết quả thực hiện các phép so sánh, hoặc kết quả trả về của một số phương thức.

> Các *phép toán so sánh* trong C# bao gồm == (so sánh bằng), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng), != (khác). Các biểu thức hoặc giá trị logic có thể được kết hợp với nhau bởi các *phép toán logic*: && (và), || (hoặc), ! (phủ định).

Statements 1 là danh sách các lệnh sẽ thực thi nếu condition có giá trị true; **Statement 2** là danh sách lệnh sẽ thực thi nếu condition có giá trị false.

Có một số lưu ý sau khi dùng if-else:

- Nếu statements 1 hoặc statements 2 chỉ có một lệnh duy nhất thì có thể không cần dùng cặp dấu {}.
- Nhánh else{} là không bắt buộc; if thì bắt buộc phải có.
- Bình thường bạn chỉ có thể tạo ra 2 nhánh rẽ: 1 nhánh if, 1 nhánh else.
- Để tạo thêm nhiều nhánh rẽ nữa bạn có thể kết hợp thêm các nhánh else if vào cấu trúc trên. Số lượng nhánh else if không giới hạn.
- Bạn có thể lồng nhiều if-else với nhau.

Hãy cùng thực hiện ví dụ sau để hiểu rõ hơn cách sử dụng if-else

- Tạo solution rỗng đặt tên là S03_FlowControls.
- Trong solution này tạo thêm một project kiểu Console App đặt tên là P01_IfElse.

Viết code cho file Program.cs như sau:

```
1.  using System;
2.  using System.Text;
3.
4.  namespace P01_IfElse
5.  {
6.      class Program
7.      {
8.          static void Main(string[] args)
9.          {
10.             Console.OutputEncoding = Encoding.UTF8;
11.
12.             Console.Write("Nhập nhiệt độ (oC): ");
13.             var input = Console.ReadLine();
14.
15.             var temperature = int.Parse(input);
16.
```

```
            if(temperature <= 5)
            {
                Console.WriteLine("Lạnh quá!");
            }
            else
            {
                if(temperature <= 15)
                {
                    Console.WriteLine("Mát mẻ, dễ chịu!");
                }
                else
                {
                    if(temperature <= 25)
                    {
                        Console.WriteLine("Ấm áp!");
                    }
                    else
                    {
                        Console.WriteLine("Nóng quá!");
                    }
                }
            }

            Console.ReadKey();
        }
    }
}
```

Đoạn code trên phân chia việc thực hiện code vào nhiều nhánh với các cấu trúc if-else lồng nhau. Các nhánh bao gồm: dưới 5 độ, từ 5 đến 15 độ, từ 15 đến 25 độ, trên 25 độ.

Nếu bạn không thích sử dụng các khối if-else lồng nhau, bạn có thể mở nhánh bằng cụm else if. Khối if-else bên trên hoàn toàn tương đương với cách viết dưới đây:

```
if(temperature < 5)
{
    Console.WriteLine("Lạnh quá!");
}
else if(temperature <= 15)
{
    Console.WriteLine("Mát mẻ, dễ chịu!");
}
else if(temperature <= 15)
{
    Console.WriteLine("Ấm áp!");
}
else
{
    Console.WriteLine("Nóng quá!");
}
```

Có thể thấy, các nhánh else if thực chất chỉ là dạng viết khác của các cấu trúc if else lồng nhau để tránh rối rắm (vì nguyên tắc if-else chỉ có hai nhánh). Việc lựa chọn cách viết nào hoàn toàn mang tính cá nhân.

Tuy nhiên, lưu ý rằng, sau if và else NÊN sử dụng code block ngay cả khi có 1 lệnh duy nhất.

Cấu trúc rẽ nhiều nhánh switch-case

Ở bên trên bạn đã gặp cấu trúc rẽ nhánh if-else. Cấu trúc này chỉ cho phép rẽ tới 2 nhánh. Nếu muốn rẽ nhiều nhánh, bạn phải lồng ghép các nhánh else if khiến code trở nên khó đọc.

C# cung cấp một cấu trúc khác để thực hiện rẽ nhiều nhánh thay cho việc lồng ghép nhiều if-else: cấu trúc switch-case. Cú pháp như sau:

```
switch(expression)
{
    case <value1>
        // code block
    break;
    case <value2>
        // code block
    break;
    case <valueN>
        // code block
    break;
    default
        // code block
    break;
}
```

Cấu trúc này yêu cầu phải cung cấp một biểu thức "expression" (lệnh tính toán). Giá trị của expression sẽ được tính ra và lần lượt so sánh với value1, value2, .., valueN. Các value này bắt buộc phải là các hằng số hoặc biểu thức tính ra hằng số, KHÔNG được sử dụng biến.

Nếu trùng với value nào, khối lệnh tương ứng sẽ được thực hiện, sau đó sẽ bỏ qua tất cả các kiểm tra còn lại. Vì lý do này, C# bắt buộc mỗi "case" phải được kết thúc bằng lệnh "break" hoặc "return". Quy định này khiến cấu trúc switch-case của C# an toàn hơn một chút so với trong C/C++ (vốn không bắt buộc dùng break).

Khi một case được thực hiện, bạn có thể tiếp tục nhảy sang một case khác bằng lệnh nhảy `goto case`. Cách sử dụng switch-case mà thực hiện được nhiều case cùng lúc như vậy có tên gọi là *fall-through*.

Nếu giá trị của biểu thức tính ra không trùng với bất kỳ "case" nào, khối lệnh default sẽ được thực hiện. Khối "default" không bắt buộc. Trong trường hợp không có khối default và giá trị của expression không trùng với bất kỳ case nào, cấu trúc switch đơn giản là không thực hiện bất kỳ khối lệnh nào.

Các lệnh đi sau mỗi case không cần viết trong cặp {}, kể cả khi có nhiều lệnh.

Hãy cùng thực hiện một ví dụ để hiểu rõ cú pháp của cấu trúc này.

Thêm project P02_SwitchCase vào solution. Viết code cho file Program.cs như sau:

```
1.  using System;
2.  using System.Text;
3.
4.  namespace P02_SwitchCase
5.  {
6.      class Program
7.      {
8.          static void Main(string[] args)
9.          {
10.             Console.OutputEncoding = Encoding.UTF8;
11.
12.             Console.Write("Nhập một số từ 1 đến 8: ");
13.             var day = Console.ReadLine();
14.
```

```
            switch (day)
            {
                case "2":
                    Console.WriteLine("Thứ hai");
                    break;

                case "3":
                    Console.WriteLine("Thứ ba");
                    break;

                case "4":
                    Console.WriteLine("Thứ tư");
                    break;

                case "5":
                    Console.WriteLine("Thứ năm");
                    break;

                case "6":
                    Console.WriteLine("Thứ sáu");
                    break;

                case "7":
                    Console.WriteLine("Thứ bảy");
                    break;

                // nhập 1 và 8 sẽ đều thực hiện chung lệnh viết ra "Chủ nhật", rồi quay
                case "1":
                case "8":
                    Console.WriteLine("Chủ nhật");
                    goto case "2";

                // nếu nhập bất kỳ giá trị nào khác sẽ thực hiện lệnh ở nhóm này
                default:
                    Console.WriteLine("Bạn nhập sai rồi");
                    break;
            }

            Console.ReadKey();
        }
    }
}
```

Các cấu trúc lặp trong C#

C# cung cấp 4 cấu trúc lặp khác nhau: do-while, while, for, foreach.

> **foreach** là cấu trúc lặp đặc biệt của C# chuyên dùng với các kiểu dữ liệu tập hợp như mảng (array), danh sách (list). Chúng ta sẽ học cấu trúc này trong bài học về mảng trong C#.

Hãy cùng thực hiện một ví dụ trước.

Thêm project mới đặt tên là P03_Loops và viết code cho Program.cs như sau:

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

namespace P03_Loops
{
    class Program
    {
        static void Main(string[] args)
        {
            var i = 0;

```

```
            Console.WriteLine("While loop");
            while(i < 10)
            {
                Console.Write($"{i}\t");
                i++;
            }

            Console.WriteLine("\r\nDo-While loop");
            i = 0;
            do
            {
                Console.Write($"{i}\t");
                i++;
            } while (i < 10);

            Console.WriteLine("\r\nFor loop");
            for (i = 0; i < 10; i++)
            {
                Console.Write($"{i}\t");
            }

            Console.ReadKey();
        }
    }
}
```

Ví dụ này minh họa cách sử dụng 3 loại vòng lặp while, do-while và for để in ra console các số từ 0 đến 9.

Cấu trúc while

while (<biểu_thức_logic>) { [<danh_sách_lệnh>] }

Chừng nào biểu thức logic còn nhận giá trị true thì danh sách lệnh sẽ được thực hiện. Cấu trúc này sẽ luôn kiểm tra biểu thức logic trước, sau đó mới thực hiện danh sách lệnh.

```
var i = 0;
 while(i < 10)
 {
     Console.Write($"{i}\t");
     i++;
 }
```

Trong cấu trúc *while*, danh sách lệnh có thể không được thực hiện lần nào. Tình huống này xảy ra khi biểu thức logic nhận giá trị false ngay từ đầu.

Lưu ý rằng trong thân của while phải có lệnh làm thay đổi giá trị của biểu thức logic. Nếu không sẽ tạo ra vòng lặp vô hạn.

Cấu trúc do-while

do { [<danh_sách_lệnh>] } **while** (<biểu_thức_logic>);

Thực hiện danh sách lệnh rồi mới kiểm tra giá trị của biểu thức logic. Nếu biểu thức logic vẫn nhận giá trị true, danh sách lệnh sẽ lại được thực hiện.

```
i = 0;
 do
```

```
 {
     Console.Write($"{i}\t");
     i++;
 } while (i < 10);
```

Cấu trúc *do-while* khác biệt với *while* ở chỗ, danh sách lệnh sẽ được thực hiện trước, sau đó mới kiểm tra giá trị của biểu thức logic. Do đó, khi sử dụng cấu trúc *do-while*, danh sách lệnh luôn luôn thực hiện ít nhất một lần.

Lưu ý rằng, sau while(<biểu_thức_logic>) phải có dấu chấm phẩy.

```
do {...}
while(i < 10);
```

Giống như đối với while, phải có lệnh làm thay đổi giá trị của biểu thức logic trong khối code của do. Nếu không sẽ tạo ra vòng lặp vô hạn.

Cấu trúc for

for (<khởi tạo giá trị đầu của biến điều khiển>; <kiểm tra điều kiện dừng của biến điều khiển>; <bước nhảy> { [<danh_sách_lệnh>] }

Cấu trúc này sẽ thực hiện danh sách lệnh một số lần xác định (trong khi hai cấu trúc trên không xác định được số lần thực hiện).

```
for (i = 0; i < 10; i++)
 {
     Console.Write($"{i}\t");
 }
```

Trong cấu trúc for, biến điều khiển, cách thay đổi giá trị của biến điều khiển cũng như điều kiện kiểm tra biến điều khiển đều viết chung trong khai báo. C# sẽ tự thay đổi giá trị biến điều khiển theo công thức chúng ta cung cấp.

Bạn có thể thực hiện đồng thời khai báo và khởi tạo giá trị của biến điều khiển ngay trong cấu trúc for, thay vì phải khai báo biến riêng.

```
for (var i = 0; i < 10; i++) { ... }
```

Bạn có thể lồng nhiều vòng for với nhau, ví dụ, để duyệt một ma trận.

```
// duyệt qua các hàng
for (int i = 0; i < 100; i += 10)
 {
     // duyệt qua các cột trong một hàng
     for (int j = i; j < i + 10; j++)
     {
         Console.Write($" {j}");
     }
```

```
    Console.WriteLine();
}
```

Điều khiển vòng lặp

Trong vòng lặp có thể sử dụng lệnh `break` hoặc `continue` để điều khiển hoạt động của vòng lặp. Cụ thể như sau:

1. Lệnh break: phá vỡ vòng lặp. Khi gặp lệnh break, tất cả các lệnh đứng sau break sẽ không thực hiện nữa, vòng lặp kết thúc.
2. Lệnh continue: phá vỡ chu kỳ hiện tại của vòng lặp. Khi gặp lệnh continue, tất cả lệnh đứng sau continue không thực hiện nữa, vòng lặp sẽ chuyển sang chu kỳ tiếp theo.

Ví dụ sử dụng `break` và `continue` để điều khiển vòng lặp. Thêm project P04_BreakContinue vào solution và viết code cho Program.cs như sau:

```
using System;

namespace P04_BreakContinue
{
    class Program
    {
        static void Main(string[] args)
        {
            var i = 0;
            while (true)
            {
                if (i == 10)
                    break;

                Console.Write($"{i}\t");
                i++;
            }

            var j = 0;
            do
            {
                j++;

                if (j == 5) continue;

                Console.Write($"{j}\t");

            } while (j < 10);

            Console.ReadKey();
        }
    }
}
```

Khi chạy ví dụ trên bạn sẽ thấy, vòng lặp while chỉ in ra các số từ 0 đến 10, mặc dù biểu thức logic luôn luôn nhận giá trị true – tức là nhẽ ra đây phải là một vòng lặp vô hạn. Vòng lặp do-while tiếp theo chỉ in ra các số từ 1 đến 10 nhưng lại bỏ qua giá trị số 5.

Ở vòng lặp thứ nhất, nếu biến điều khiển i có giá trị bằng 10 thì sẽ thực hiện lệnh break. Lệnh này sẽ phá vỡ (kết thúc) vòng lặp và thoát ra ngoài.

Ở vòng lặp thứ hai, nếu biến điều khiển i có giá trị bằng 5 thì sẽ phá vỡ chu kỳ đó, tức là bỏ qua hết tất cả các lệnh phía sau continue. Dẫn đến giá trị 5 không được in ra console. Nhưng continue không phá vỡ vòng lặp như break. Nên một chu kỳ mới lại bắt đầu như bình thường.

> Bạn có thể sử dụng return thay cho break. Khi này return không những phá vỡ vòng lặp mà còn kết thúc luôn việc thực thi của phương thức.

Kết luận

Bài viết này cung cấp cho bạn các cấu trúc điều khiển của C#. Dễ dàng nhận thấy, các cấu trúc này hầu như không khác biệt so với C/C++ hay Java.

C# cung cấp thêm một số tính năng đặc biệt cho cấu trúc điều khiển. Ví dụ pattern matching dành cho switch-case, cấu trúc foreach dành cho dữ liệu tập hợp. Tuy nhiên, các vấn đề này chúng ta sẽ xem xét ở các bài học khác.