Trong lĩnh vực CNTT hiện nay, ngôn ngữ lập trình là rất đa dạng: C#, c/c++, Java, Python, Golang,.... Và chủ đề của buổi thuyết trình ngày hôm nay đó là ngôn ngữ lập trình C#.  
Có thể nói C# là một ngôn ngữ lập trình cực kì mạnh mẽ , ta có thể làm hàng ngàn thứ với C# và đã có hàng triệu ứng dụng đã ra đời được viết bới ngôn ngữ mạnh mẽ này (Skype, Photoshop, Moffice),.và chúng ta sẽ đến với phần đầu tiên là tìm hiểu về các cú pháp, các cấu trúc cơ bản của C#.

**Vậy C# là gì?**

C# là một ngôn ngữ lập trình thuần hướng đối tượng được phát triển bởi Microsoft

C# ra đời năm 2000, được thiết kế chủ yếu bởi Anders Hejlsberg – kiến trúc sư phần mềm nổi tiếng với các sản phẩm Turbo Pascal, Delphi, . . .

Được xây dựng dựa trên nền tảng của 2 ngôn ngữ lập trình mạnh nhất đó là [C++](https://www.canva.com/link?target=https%3A%2F%2Fwww.howkteam.com%2Flearn%2Flap-trinh%2Flap-trinh-c-7-8&design=DAFPxpe3yu8&accessRole=viewer&linkSource=document) và [Java](https://www.canva.com/link?target=https%3A%2F%2Fwww.howkteam.com%2Flearn%2Flap-trinh%2Flap-trinh-java-7-18&design=DAFPxpe3yu8&accessRole=viewer&linkSource=document). Do đó C# được miêu tả là ngôn ngữ có sự cân bằng giữa C++, Visual Basic, Delphi và Java

**Đặc trưng:**

*Đơn giản(Simple)*

C# loại bỏ một vài sự phức tạp và rối rắm của những ngôn ngữ như Java và c++, bao gồm việc loại bỏ những macro, những template, đa kế thừa, và lớp cơ sở ảo (virtual base class)

Ngôn ngữ C# đơn giản vì nó dựa trên nền tảng C và C++

Ngôn ngữ lập trình hiện đại(Modern programming language)

C# có Những đặc tính của một ngôn ngữ hiện đại như là xử lý ngoại lệ, thu gom bộ nhớ tự động, những kiểu dữ liệu mở rộng, và bảo mật mã nguồn

Lập trình hướng đối tượng(Object oriented)

C# hỗ trợ cho chúng ta tất cả những đặc tính củaLập trình hướng đối tượng. Đó là tính trừu tượng (abstraction), tính đóng gói (encapsulation), tính đa hình (polymorphism) và tính kế thừa (inheritance)

Kiểu an toàn(Type safe)

Tất cả các lớp và biến trong C# là một kiểu và tất cả đều có dạng kiểu đối tượng.  
C# không cho phép kiểu biểu diễn untypesafe.  
Ví dụ : Khi bạn gán 1 kiểu float trực tiếp tới kiểu Boolean là không được phép. Khi đó compiler sẽ báo lỗi.

Khả năng tương tác(Interoperability)

Quá trình tương tác cho phép các chương trình C# thực hiện hầu hết mọi thứ mà ứng dụng C ++ gốc có thể làm.

Có thể mở rộng và cập nhật(Scalable and Updateable)

Phát triển phần mềm yêu cầu ngôn ngữ lập trình dễ dàng bảo trì và có thể mở rộng và C# là một ngôn ngữ như vậy. Tính nghiêm ngặt của mã hóa tĩnh làm cho các chương trình nhất quán với nhau. Điều này cho phép các developer dễ dàng thực hiện các điều chỉnh cũng như bảo trì.

Hướng thành phần(Component oriented)

C# là ngôn ngữ lập trình hướng thành phần. Đây là phương pháp phát triển phần mềm chiếm ưu thế được sử dụng để phát triển các ứng dụng mạnh mẽ hơn và có khả năng mở rộng cao hơn.

Ngôn ngữ lập trình có cấu trúc(Structured programming language)

C# là một ngôn ngữ lập trình có cấu trúc theo nghĩa chúng ta có thể chia chương trình thành các phần bằng các hàm. Vì vậy, nó rất dễ hiểu và sửa đổi.

Nhiều thư viện(Rich Library)

Sở hữu thư viện khổng lồ, cung cấp các chức năng cấp cao so với các ngôn ngữ khác như Java hay C++.

Tốc độ nhanh(Fast speed)

Vì đã được loại bỏ vài sự phức tạp của C++ và Java, nên thời gian biên dịch và thực thi ngôn ngữ C # rất nhanh.

Đây là một đoạn code cơ bản “Hello world” của C#, như ta đã thấy; nếu như ở C++ t sẽ viết code trong hàm int main thì ở đây tất cả phần code chính sẽ được viết trong hàm nhưu trên. Và trong đây t có các lệnh nhập xuất cơ bản: Console.WriteLine để in ra màn hình, Console.Readkey dùng để dừng màn hình hoặc là nhập từ bàn phím khi ta gán cho một biến.

Tiếp theo ta sẽ cùng tìm hiểu về các cú pháp cũng như là các cấu trúc điều khiển cơ bản, vì là ngôn ngữ được xây dựng dựa trên C++, nên các cú pháp của C# cũng không khác biệt so với C++

Đầu tiên ta sẽ đến với các kiểu dữ liệu

Kiểu số nguyên: ta vẫn có các kiểu số nguyên quen thuộc như là int, long, byte ngoài ra còn có các kiểu ulong, ushort cho các kiểu số nguyên không dấu lớn hơn 0, ngoài ra ta còn có sbyte, short

Kiểu ký tự vẫn là kiểu char

Kiểu logic vẫn là kiểu bool

Kiểu số thực, ta có 3 kiểu là float, double, decimal biểu diễn kiểu số thực, trong đó decimal là kiểu biểu diễn số thực chính xác nhất, float sẽ biểu diễn số thực ít chính xác hơn decimal, và nằm giữa float và decimal là double.

**Kiểu tham chiếu**

String mặc định là null dùng để lưu chuỗi ký tự

Object Là một kiểu dữ liệu cơ bản của tất cả các kiểu dữ liệu trong .NET.  Mọi kiểu dữ liệu đều được kế thừa từ System.Object

Type name của kiểu dữ liệu Object là System.Object

**Kiểu con trỏ**

chứa địa chỉ bộ nhớ của giá trị biến.

**&** còn được gọi là toán tử Address, được sử dụng để xác định địa chỉ của một biến.

**\*** còn được gọi là toán tử Indirection, được sử dụng để truy cập giá trị của một địa chỉ.

trong C # bất kỳ mã nào liên quan đến con trỏ đều yêu cầu một ngữ cảnh không an toàn. Các câu lệnh này được đánh dấu là không an toàn bằng cách sử dụng từ khóa unsafe sẽ chạy ngoài sự điều khiển của Garbage Collector.

Sau đây là một số lưu ý về các kiểu dữ liệu trên

Kiểu dữ liệu có miền giá trị lớn hơn sẽ chứa được kiểu dữ liệu có miền giá trị nhỏ hơn.

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.

float BienFloat = 10.9f;

Giá trị của biến kiểu decimal phải có chữ m hoặc M làm hậu tố.

decimal BienDecimal = 10.9m;

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? . . .

Tiếp theo là đến các toán tử cơ bản trong C#

Toán tử toán học: Giả sử biến a có giá trị là 10 biến b có giá trị là 9

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| + | Thực hiện **cộng**hai toán hạng | **a + b** kết quả bằng 19 |
| - | Thực hiện **trừ**hai toán hạng | **a - b** kết quả bằng 1 |
| \* | Thực hiện **nhân**hai toán hạng | **a \* b** kết quả bằng 90 |
| / | Thực hiện **chia lấy phần nguyên** hai toán hạng nếu 2 toán hạng là số nguyên. Ngược lại thì thực hiện chia bình thường | **a / b** kết quả bằng 1 |
| % | Thực hiện **chia lấy dư** | **a%b** kết quả bằng 1 |
| ++ | **Tăng** giá trị lên 1 đơn vị | **a++** kết quả bằng 11 |
| -- | **Giảm** giá trị xuống 1 đơn vị | **a--**kết quả bằng 9 |

Toán tử quan hệ: Giả sử biến a có giá trị bằng 10 và biến b có giá trị bằng 9:

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| == | So sánh 2 toán hạng có bằng nhau hay không. Nếu **bằng**thì trả về true nếu **không bằng** thì trả về false | a == b sẽ trả về false |
| != | So sánh 2 toán hạng có bằng nhau hay không. Nếu **không bằng** thì trả về true nếu **bằng** thì trả về false | a != b sẽ trả về true |
| > | So sánh 2 toán hạng bên trái có **lớn hơn** toán hạng bên phải hay không. Nếu **lớn hơn** thì trả về true nếu **không lớn hơn** thì trả về false | a > b sẽ trả về true |
| < | So sánh 2 toán hạng bên trái có **nhỏ hơn** toán hạng bên phải hay không. Nếu **nhỏ hơn** thì trả về true nếu **không nhỏ hơn** thì trả về false | a < b sẽ trả về false |
| >= | So sánh 2 toán hạng bên trái **có lớn hơn** **hoặc bằng** toán hạng bên phải hay không. Nếu **lớn hơn hoặc bằng** thì trả về true nếu **nhỏ hơn** thì trả về false | a >= b sẽ trả về true |
| <= | So sánh 2 toán hạng có **nhỏ hơn hoặc bằng** hay không. Nếu **nhỏ hơn hoặc bằng** thì trả về true nếu **lớn hơn** thì trả về false | a <= b sẽ trả về false |

Toán tử logic: Giả sử mệnh đề A là đúng và mệnh đề B là sai:

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| && | Hay còn gọi là toán tử logic AND (và). Trả về true nếu **tất cả** toán hạng đều mang giá trị true. Và trả về false nếu có **ít nhất 1**toán hạng mang giá trị false. | A && B kết quả là false |
| || | Hay còn gọi là toán tử logic OR (hoặc). Trả về true nếu có **ít nhất 1** toán hạng mang giá trị true. Và trả về false nếu **tất cả** toán hạng đều mang giá trị false. | A || B kết quả là true. |
| ! | Hay còn gọi là toán tử logic NOT (phủ định). Có chức năng **đảo ngược** trạng thái logic của toán hạng. Nếu toán hạng đang mang giá trị true thì kết quả sẽ là false và ngược lại. | !A kết quả là false |

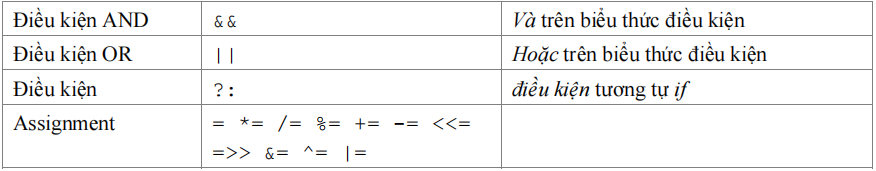
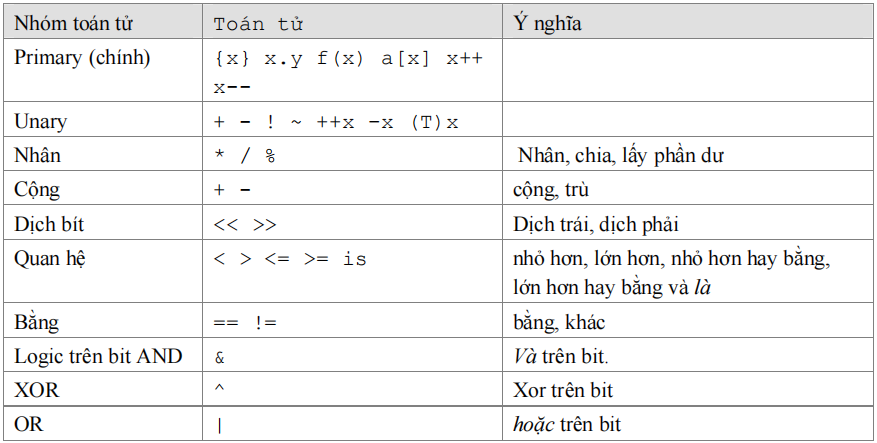
Toán tử khởi tạo và gán: thường được sử dụng nhằm mục đích lưu lại giá trị cho một biến nào đó.

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| = | Gán giá trị của toán hạng **bên phải** cho toán hạng **bên trái**. | **K = 10**sẽ gán 10 cho biến**K** |
| += | Lấy toán hạng **bên trái cộng** toán hạng **bên phải** sau đó gán kết quả lại cho toán hạng **bên trái**. | **K += 1** tương đương với**K = K + 1** |
| -= | Lấy toán hạng **bên trái trừ** toán hạng**bên phải** sau đó gán kết quả lại cho toán hạng **bên trái**. | **K -= 1**tương đương với **K = K – 1** |
| \*= | Lấy toán hạng **bên trái nhân** toán hạng **bên phải** sau đó gán kết quả lại cho toán hạng **bên trái**. | **K \*= 1** tương đương với **K = K \* 1** |
| /= | Lấy toán hạng **bên trái chia lấy phần nguyên** với toán hạng **bên phải** sau đó gán kết quả lại cho toán hạng **bên trái**. | **K /= 1** tương đương với **K = K / 1** |
| %= | Lấy toán hạng **bên trái chia lấy dư** với toán hạng **bên phải** sau đó gán kết quả lại cho toán hạng **bên trái**. | **K %= 1**tương đương với **K = K % 1** |

Toán tử so sánh trên bit: Giả sử a có giá trị bằng 10 và b có giá trị bằng 9. Giá trị biến a đổi ra nhị phân là 1010 và giá trị biến b đổi ra nhị phân là 1001

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| & | Sao chép bit 1 tới kết quả nếu nó **tồn tại trong cả hai** toán hạng tại vị trí tương ứng, ngược lại thì bit kết quả bằng 0 | **a&b** sẽ cho kết quả là 1000 tương đương với số 8 trong hệ thập phân |
| | | Sao chép bit 1 tới kết quả nếu nó **tồn tại ở một trong hai** toán hạng tại vị trí tương ứng, ngược lại thì bit kết quả bằng 0 | **a|b**sẽ cho kết quả 1011 tương đương với số 11 trong hệ thập phân |
| ^ | Sao chép bit 1 tới kết quả nếu**nó chỉ tồn tại ở một** toán hạng tại vị trí tương ứng, ngược lại thì bit kết quả bằng 0 | **a^b** sẽ cho kết quả 0011 tương đương với số 3 trong hệ thập phân |
| ~ | Dùng để **đảo bit** 0 thành 1 và ngược lại 1 thành 0 | **~a** sẽ cho kết quả 0101 |
| << | Dịch trái n bit. Giá trị toán hạng bên trái sẽ được **dịch trái** n bit với n được xác định bởi toán hạng bên phải | **a<<2** sẽ cho kết quả 101000 |
| >> | Dịch phải n bit. Giá trị toán hạng bên trái sẽ được **dịch phải** n bit với n được xác định bởi toán hạng bên phải | **a>>2** sẽ cho kết quả 0010 |

**Thứ tự ưu tiên của các nhóm toán tử (chiều ưu tiên từ trên xuống)**



**Cấu trúc điều khiển cơ bản**

1. **Các lệnh rẽ nhánh có điều kiện** 
   1. **If ... else....**

* **Dạng thiếu:**

**Cú pháp:**

**If**(**[Biểu thức điều kiện]**)  **<Câu lệnh thực hiện>**

* If là từ khóa bắt buộc.
* <Biểu thức điều kiện> là biểu trức dạng boolean (trả về true hoặc false).
* <Câu lệnh thực hiện> là câu lệnh muốn thực hiện nếu <Biểu thức điều kiện> là đúng.

**Ý nghĩa:** Nếu <Biểu thức điều kiện> trả về true thì thực hiện <Câu lệnh thực hiện> ngược lại thì không làm gì cả.

* **Dạng đủ:**

**Cú pháp:**

**If <Biểu thức điều kiện>**

**<Câu lệnh thực hiện 1>**

**else**

**<Câu lệnh thực hiện 2>**

* **If, else** là từ khóa bắt buộc.
* <Biểu thức điều kiện> là biểu trức dạng boolean (trả về true hoặc false).
* <Câu lệnh thực hiện 1> là câu lệnh muốn thực hiện nếu <Biểu thức điều kiện> là đúng.
* <Câu lệnh thực hiện 2> là câu lệnh muốn thực hiện nếu <Biểu thức điều kiện> là sai.

**Ý nghĩa:** Nếu <Biểu thức điều kiện> trả về true thì thực hiện <Câu lệnh thực hiện 1> ngược lại thì thực hiện <Câu lệnh thực hiện 2>.

**Chú ý:**

* <Biểu thức điều kiện> có thể chứa nhiều biểu thức con bên trong và các biểu thức con liên kết với nhau bằng các toán tử quan hệ (toán tử quan hệ đã được trình bày trong bài [TOÁN TỬ TRONG C#](http://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/toan-tu-trong-c-54)) nhưng tất cả phải trả về kiểu boolean (true hoặc false).
* Nếu muốn thực hiện nhiều câu lệnh thì ta có thể nhóm chúng vào trong cặp ngoặc nhọn **{ }**.
  1. **switch case**

**Cú pháp:**

switch (<biểu thức>)

{

case <giá trị thứ 1>: <câu lệnh thứ 1>;

                            break;

case <giá trị thứ 2>: <câu lệnh thứ 2>;

                            break;

. . .

case <giá trị thứ n>: <câu lệnh thứ n>;

                            break;

default: <câu lệnh mặc định>;

                break;

}

* switch, case, default là từ khóa bắt buộc.
* <biểu thức> phải là biểu thức trả về kết quả kiểu:
  + Số nguyên (int, long, byte, . . .)
  + Ký tự hoặc chuỗi (char, string)
  + Kiểu liệt kê (enum, sẽ được trình bày trong bài [ENUM TRONG LẬP TRÌNH C#](http://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/enum-trong-lap-trinh-c-1367))
* <giá trị thứ i> với i = 1..n là giá trị muốn so sánh với giá trị của <biểu thức>.
* <câu lệnh thứ i> với i = 1..n là câu lệnh muốn thực hiện khi <giá trị thứ i> tương ứng bằng với giá trị của <biểu thức>.
* <câu lệnh mặc định> là câu lệnh sẽ được thực hiện nếu giá trị <biểu thức> không bằng với <giá trị thứ i> nào.

**Ý nghĩa:** Duyệt lần lượt từ trên xuống dưới và kiểm tra xem giá trị của <biểu thức> có bằng với <giá trị thứ i> đang xét hay không. Nếu bằng thì thực hiện <câu lệnh thứ i> tương ứng. Nếu không bằng tất cả các <giá trị thứ i> thì sẽ thực hiện <câu lệnh mặc định>.

**Chú ý**

* Trường hợp không có lệnh break như trong hình đồng nghĩa với việc case đó rỗng (không có câu lệnh thực hiện).
* Đối với case cuối cùng dù có câu lệnh để thực hiện hay không vẫn phải có lệnh break để thoát khỏi cấu trúc.

1. **Các lệnh lặp:**
   1. **while**

**Cú pháp:**

**while**(**<Điều kiện lặp>**)

            {

**// khối lệnh lặp lại**

            }

* **Điều kiện lặp**là một biểu thức logic bắt buộc phải có với kết quả trả về bắt buộc là true hoặc false.
* Từ khóa while biểu thị đây là một vòng lặp while. Các câu lệnh trong khối lệnh sẽ được lặp lại đến khi không còn thỏa mãn **điều kiện lặp** sẽ kết thúc vòng lặp while.

**Tiến trình:**

* Đầu tiên trình biên dịch sẽ đi vào dòng while (**<Điều kiện lặp>**). Kiểm tra điều kiện lặp có thỏa mãn hay không. Nếu kết quả là true thì sẽ đi vào bên trong thực hiện khối code. Khi gặp ký tự } sẽ quay lên kiểm tra điều kiện lặp và tiếp tục thực hiện khối code. Quá trình chỉ kết thúc khi điều kiện lặp là false.
* Điều kiện lặp luôn bằng true thì vòng lặp while sẽ trở thành vòng lặp vô tận.
* Điều kiện lặp luôn bằng false thì vòng lặp sẽ không được thực thi.
  1. **do - while**

**Cú pháp:**

**do**

            {

                 // khối lệnh lặp lai

            } **while**(**<Điều kiện lặp>**);

* **Điều kiện lặp**là một biểu thức logic bắt buộc phải có với kết quả trả về bắt buộc là true hoặc false.
* Từ khóa do while biểu thị đây là một vòng lặp do while. Các câu lệnh trong khối lệnh sẽ được lặp lại đến khi không còn thỏa mãn điều kiện lặp sẽ kết thúc vòng lặp do while.

**Tiến trình:**

* Đầu tiên trình biên dịch sẽ đi vào dòng do và thực hiện khối lệnh bên trong. Sau đó khi gặp ký tự } sẽ kiểm tra điều kiện lặp có thỏa mãn hay không. Nếu kết quả là true thì sẽ quay lại ký tự { thực hiện khối code. Quá trình chỉ kết thúc khi điều kiện lặp là false.
* Điều kiện lặp luôn bằng true thì vòng lặp while sẽ trở thành vòng lặp vô tận.
* Điều kiện lặp luôn bằng false thì vòng lặp sẽ không được thực thi.

**Chú ý:** vòng lặp do - while sẽ thực hiện câu lệnh trong khối code xong rồi mới kiểm tra điều kiện lặp. Cuối vòng lặp do - while có dấu ; ở cuối.

* 1. **for**

**Cú pháp:**

**for**([Khởi tạo]; [Điều kiện lặp]; [Bước lặp lại])

{

// Khối lệnh được lặp lại. Có thể bỏ trống

}

* Các phần [Khởi tạo]; [Điều kiện lặp]; [Bước lặp lại] hoàn toàn có thể để trống như ví dụ sau.
* Mỗi đoạn [Khởi tạo]; hay [Điều kiện lặp]; hay [Bước lặp lại] là một câu lệnh riêng.

**Tiến trình:**

* Ban đầu trình biên dịch sẽ di vào phần khởi tạo chạy đoạn lệnh khởi tạo.
* Tiếp theo kiểm tra điều kiện lặp. Rồi thực hiện khối code bên trong vòng lặp for. Khi đến ký hiệu } thì sẽ quay lên bước lặp lại.
* Sau đó lại kiểm tra điều kiện lặp rồi tiếp tục thực hiện đoạn code trong khối lệnh. Đến khi điều kiện lặp không còn thõa mãn thì sẽ kết thúc vòng lặp for.

**Trường hợp khác:**

**for**(; ;)   // lưu ý dấu ;

{

// Khối lệnh được lặp lại. Có thể bỏ trống

}

* Vòng lặp for này trở thành vòng lặp vô tận.
* Lưu ý dấu ; vẫn phải có.
  1. **foreach**

Một số đặc trưng của foreach:

* Foreach không duyệt mảng hoặc tập hợp thông qua chỉ số phần tử như cấu trúc lặp for.
* Foreach duyệt tuần tự các phần tử trong mảng hoặc tập hợp.
* Foreach chỉ dùng để duyệt mảng hoặc tập hợp ngoài ra không thể làm gì khác.

**Cú pháp:**

**foreach**(**<kiểu dữ liệu>** **<tên biến tạm>** **in <tên mảng hoặc tập hợp>**)

{

              // Code xử lý

 }

* Các từ khoá **foreach**, **in**là từ khoá bắt buộc.
* <kiểu dữ liệu> là kiểu dữ liệu của các phần tử trong mảng hoặc tập hợp.
* <tên biến tạm> là tên 1 biến tạm đại diện cho phần tử đang xét khi duyệt mảng hoặc tập hợp.
* <tên mảng hoặc tập hợp> là tên của mảng hoặc tập hợp cần duyệt.

**Nguyên tắc hoạt động:**

* Ở vòng lặp đầu tiên sẽ gán giá trị của phần tử đầu tiên trong mảng vào biến tạm.
* Thực hiện khối lệnh bên trong vòng lặp foreach.
* Qua mỗi vòng lặp tiếp theo sẽ thực hiện kiểm tra xem đã duyệt hết mảng hoặc tập hợp chưa. Nếu chưa thì tiếp gán giá trị của phần tử hiện tại vào biến tạm và tiếp tục thực hiện khối lệnh bên trong.
* Nếu đã duyệt qua hết các phần tử thì vòng lặp sẽ kết thúc.
  1. **Điều khiển câu lệnh**
* Lệnh break: phá vỡ vòng lặp. Khi gặp lệnh break, tất cả các câu lệnh đứng sau break sẽ không thực hiện nữa, vòng lặp kết thúc.
* 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.

**Nguyên tắc phát triển lập trình C++:**

* **Đơn giản, hướng đối tượng và dễ hiểu.**
* **Code phải được tiến hành trong một môi trường mạnh mẽ và an toàn.**
* **Code cần tuân theo công cụ specific architecture và phải có tính di động.**

**Nguyên tắc phát triển lập trình C#:**

* **C# là ngôn ngữ lập trình đơn giản, hiệu quả, hướng đối tượng.**
* **C# là một lựa chọn lý tưởng để xây dựng các ứng dụng cho cả hệ thống hosted và**[**Embedded systems.**](https://fsoft-academy.edu.vn/tin-tuc-su-kien/bi-quyet-giup-hoc-lap-trinh-nhung-hieu-qua-nhanh-chong/)
* **Tính đơn giản: C # đơn giản về cách sử dụng các toán tử và nó hoàn toàn tách biệt kiểu dữ liệu boolean và số nguyên.**
* **Khả năng tương thích: C # cung cấp khả năng tương tác trong các ngôn ngữ nền tảng .NET bằng cách thực thi các đặc tả ngôn ngữ chung.**
* **An toàn kiểu: Nó kết hợp một số biện pháp an toàn kiểu như các đối tượng được cấp phát động và mảng được khởi tạo bằng 0, không cho phép sử dụng các biến chưa được khởi tạo, kiểm tra phạm vi tự động cho các mảng, không cho phép phôi không an toàn, kiểm tra tràn, vân vân.**
* **Hướng đối tượng: Nó hoàn toàn là hướng đối tượng trong đó mọi thứ là một đối tượng thay vì các hàm, biến và hằng số toàn cục.**

**C++**

**Phụ thuộc vào nền tảng: C ++ là ngôn ngữ phụ thuộc vào nền tảng mà các chương trình được viết trong một hệ điều hành không thể được thực thi trong hệ điều hành khác.**

**Dựa trên trình biên dịch: C ++ được cho là ngôn ngữ biên dịch vì nó không thực thi bất kỳ mã nào mà không có biên dịch.**

**TỔNG KẾT VỀ TƯƠNG ĐỒNG**

**Là một ngôn ngữ được phát triển để cạnh tranh với Java, C # có những điểm tương đồng gần với Java hơn là C ++. Tuy nhiên, vẫn có một số điểm tương đồng chính giữa C # và C ++, chẳng hạn như:**

**Cả hai ngôn ngữ đều hướng đối tượng, mặc dù có một chút khác biệt giữa cú pháp của C # và C ++. Tuy nhiên, các khái niệm OOP như đa hình, kế thừa và các lớp đều được tìm thấy trong cả hai.**

**Cả C ++ và C # đều được coi là ngôn ngữ biên dịch, có nghĩa là trước khi các chương trình có thể chạy trên PC hoặc trên máy chủ, mã của chúng trước tiên phải chuyển đổi thành mã nhị phân. Nếu bạn nhìn vào các ứng dụng của mình, bạn sẽ thấy các tệp .EXE - đây là những ví dụ điển hình về các tệp đã biên dịch.**

**C++ != C# NTN?**

**C # là ngôn ngữ lập trình cấp cao, trong khi C ++ được coi là ngôn ngữ lập trình cấp trung gian bổ sung một số tính năng hướng đối tượng cho C, ngôn ngữ cơ sở của nó.**

**C # biên dịch các chương trình của nó thành CLR, hoặc Common Language Runtime (trong trường hợp này là .NET). C ++ biên dịch thành mã gốc, có nghĩa là nó không cần bất kỳ hệ thống thời gian chạy nào.**



Đây là một ngôn ngữ lập trình hướng đối tượng. Đây là một đặc điểm ít ngôn ngữ nào khác có được. Lập trình hướng đối tượng (Object-Oriented Programming – OOP) mang lại hiệu quả và tính linh hoạt cao. Có thể kể đến như xây dựng phần mềm Quản lý hiệu suất ứng dụng (Applications Performance Management – APM). C#, cùng với một số ngôn ngữ khác, có thể xây dựng các phần mềm giống như vậy.

Việc triển khai giải pháp APM có thể giúp các nhà phát triển phần mềm tối ưu hóa hiệu suất trong quá trình dàn dựng, sản suất cũng như hậu kỳ. Nó cung cấp thông tin chi tiết về hiệu suất ứng dụng cũng như cách khắc phục các sự cố nếu phát sinh.