### **What is C#?**

#### Xin chào!

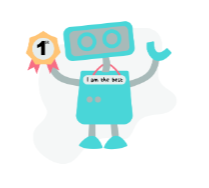
#### Chào mừng bạn đến với chương đầu tiên về C #.

#### Trước khi bất cứ điều gì được thảo luận, hãy làm rõ rằng ngôn ngữ này được phát âm là C Sharp.

#### C # là một ngôn ngữ lập trình hướng đối tượng - dựa trên lớp và hướng thành phần.

#### Nó được phát triển bởi Microsoft vào năm 2000.

#### Nó đã trở nên cực kỳ phổ biến trong số các nhà phát triển đang tạo phần mềm cho nhiều loại nền tảng khác nhau.



#### C # cung cấp cho chúng tôi các công cụ cần thiết để tạo phần mềm cho Windows, iOS, Android và hơn thế nữa!

#### Vì nó đến từ gia đình ngôn ngữ C và C ++, nó có nhiều lợi thế so với những ngôn ngữ đó.

### **Use of C#**

#### C # có thể được sử dụng để tạo các loại thư viện, chương trình và tiện ích mở rộng khác nhau.

#### Nó có nhiều SDK (Bộ công cụ phát triển phần mềm) để các nhà phát triển đáp ứng nhu cầu của họ và đạt được mục tiêu nhanh hơn!

#### Ngoài ra, C # có tích hợp đơn giản cho các API khác nhau (Giao diện lập trình ứng dụng) như cổng thanh toán!

#### Thật tuyệt vời phải không?!



#### Làm rất tốt!

#### Bạn đã có một ý tưởng rất cơ bản về C #. Bây giờ, chúng ta hãy đi đến những thứ thực tế để giúp bạn bắt đầu!

#### Đi thôi!

#### Đây là một sự thật thú vị để giúp bạn tiếp tục,

#### C # ban đầu được đặt tên là "C Cool" nhưng nó đã bị loại bỏ do vấn đề bản quyền.

### **Variables and Storing Data**



#### Lời giải thích cơ bản nhất của một biến là giữ mọi thứ trong bộ lưu trữ tương ứng của chúng.

#### Ví dụ: hãy xem xét áo khoác, sách và tài liệu giấy từ hình trên.



#### Bạn đang ở trong phòng của bạn, bạn đặt áo khoác của bạn ở đâu?

#### Đúng vậy, trong tủ quần áo của bạn!

#### Bạn đặt sách của mình ở đâu?

#### Vâng, trên giá sách!

#### Và tất nhiên, tài liệu giấy của bạn đi vào bên trong ngăn kéo bàn.



#### Không có nhiều sự khác biệt giữa lưu trữ dữ liệu trong các ứng dụng so với lưu trữ mọi thứ trong thế giới thực.

#### Điều quan trọng nhất là giữ mọi thứ ở vị trí được chỉ định của nó.

#### Trong lập trình, container lưu trữ dữ liệu được gọi là biến.

#### Bây giờ bạn sẽ tự hỏi làm thế nào chúng ta sẽ xác định loại dữ liệu nào nên được lưu trữ trong vùng chứa nào? Bên phải?

#### Chúng tôi sử dụng các kiểu dữ liệu để xác định loại dữ liệu sẽ được lưu trữ trong một biến cụ thể.

#### Để lưu trữ số bằng C #, chúng tôi sử dụng các kiểu dữ liệu như Số nguyên và Float. Để lưu trữ các từ hoặc câu bằng C #, chúng tôi sử dụng kiểu dữ liệu "Chuỗi".

#### Hãy xem xét một số ví dụ về dữ liệu:

* 10 là một số nguyên
* 10,15 là số thập phân
* 'C' là một ký tự duy nhất
* "Bạn bè" là một từ được tạo thành từ sáu ký tự, tức là một chuỗi



#### Tuyệt! Bạn đã làm rất tốt! Bây giờ bạn đã biết các loại biến và kiểu dữ liệu cơ bản giúp chúng ta lưu trữ dữ liệu trong các chương trình.

### **Storing Data**

#### Như chúng ta đã biết, dữ liệu số có thể được lưu trữ dưới dạng,

* Số nguyên
* Số thập phân

#### Thật đơn giản để hiểu sự khác biệt giữa chúng - Số nguyên không có bất kỳ phần thập phân nào.

#### Một số ví dụ về Số nguyên là:

* 186468
* 37886
* 100

#### Một số ví dụ về Thập phân là:

* 10.00005
* 120.25
* 1000.1



#### Trong C#, chúng ta có 2 loại biến cơ bản để biểu diễn số – Số nguyên và Số thập phân

#### Câu hỏi lớn là - Chúng ta lưu trữ chúng ở đâu?



#### Giả sử chúng ta có một số 5

#### Và chúng tôi muốn lưu trữ nó ở đâu đó trong mã của chúng tôi.

#### Với mục đích đó, chúng tôi có các biến.



#### Biến là gì?

#### Về cơ bản, nó là một thùng chứa nơi chúng ta lưu trữ mọi thứ.

#### Giống như chúng ta đã thấy rằng một cuốn sách đi trên giá sách và một chiếc áo khoác đi trong tủ quần áo.



#### Điều tương tự cũng áp dụng cho máy tính - chúng tôi sử dụng các biến để lưu trữ dữ liệu của chúng tôi.

#### Trong cuộc sống thực, chúng ta có thể đặt quần áo của mình ở những nơi khác nhau, tuy nhiên, điều này không thể xảy ra với máy tính.

#### Theo quy tắc lập trình, chúng tôi đã đặt quần áo của chúng tôi trong tủ quần áo của chúng tôi và chúng tôi phải giữ sách của chúng tôi trên giá sách của chúng tôi.

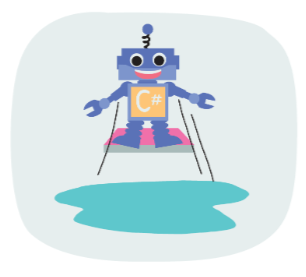
#### Mọi thứ phải được đặt một cách thích hợp!

#### Vì vậy, đối với dữ liệu mà chúng ta muốn lưu trữ, chúng ta cần chọn đúng loại biến.



#### Loại:

* Chúng ta sử dụng Integer type of variables để lưu trữ số nguyên – từ khóa "int"
* Để lưu trữ loại số thập phân, chúng tôi sử dụng biến thập phân – từ khóa "gấp đôi"
* Nếu chúng ta muốn lưu trữ một ký tự duy nhất, chúng ta sử dụng các biến ký tự – từ khóa "char"
* Trong trường hợp lưu trữ từ hoặc câu. chúng ta sử dụng biến String - từ khóa "chuỗi"



#### Bạn đã làm quen với các loại biến cơ bản.

#### Bây giờ chúng ta hãy đi sâu vào nhóm các biến! HOAN HÔ!

### **Working with Integers**

#### Chúng ta cần chọn đúng loại biến để lưu trữ Số nguyên trong C#

#### Khai báo một kiểu số nguyên của biến trong C # khá đơn giản.

#### Chúng tôi sử dụng từ khóa được gọi là "int" cho mục đích đó.



#### Trong C#, một biến cơ bản được khai báo như sau:

using System;

namespace CodeSample {

class Program {

static void Main(string[] args){

int A;

}

}

}

#### Đó là nó! Nó chỉ đơn giản như vậy!

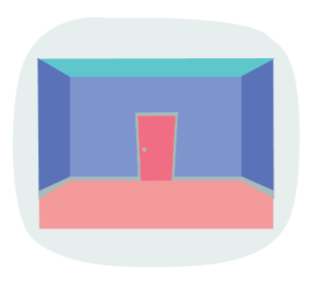
#### Bây giờ hãy chia nhỏ code của chúng ta:

#### **int** - đại diện cho loại biến.

#### **A** - đại diện cho tên của biến mà chúng ta đã khai báo.

#### Vì vậy, hãy xem những gì chúng ta đã học được cho đến nay!

#### **Lưu ý**: Đừng bị sa lầy bởi tất cả các biệt ngữ khác như "sử dụng", "không gian tên", "lớp", v.v. Khi chúng ta tiến về phía trước, chúng ta sẽ đón từng người một.



#### Chúng ta đã học cách khai báo một kiểu biến số nguyên.

#### Tuyệt vời!

#### Nhưng chúng ta chưa gán giá trị cho biến của mình. Một biến giống như một căn phòng trống. Hãy đưa ai đó vào đó!



#### Trong C#, có hai cách mà chúng ta có thể sử dụng để gán giá trị cho các biến:

* Trên cùng một dòng khi khai báo nó.
* Chỉ định sau trong khai báo bài mã

#### Các biến trong C # có thể thay đổi giá trị của chúng khi trải qua các hoạt động khác nhau.

#### Hãy xem nó trông như thế nào trong code của chúng ta!



#### Ví dụ đầu tiên minh họa quá trình khai báo và gán giá trị trên cùng một dòng mã.

using System;

namespace CodeSample {

class Program {

static void Main(string[] args){

int A = 10;

int B = -50;

int C = 0;

}

}

}

#### Vì vậy, những gì chúng ta nhận được ở đây là 3 biến số nguyên: A, B và C, chứa 3 giá trị khác nhau.

#### A chứa 10.

#### B chứa -50

#### Và C chứa 0 (không)



#### Ví dụ thứ hai là gán giá trị cho biến sau khi nó đã được khai báo.

using System;

namespace CodeSample {

class Program {

static void Main(string[] args){

int A;

int B;

A = 15;

B = -35;

}

}

}

#### Chúng ta có 2 biến số nguyên ở đây - A và B được khai báo mà không có bất kỳ giá trị nào. Lúc đầu, A và B trống rỗng.

#### Tuy nhiên, trong các dòng sau, chúng tôi đã gán các giá trị cho cả hai biến được tạo ở trên. Đầu tiên, chúng ta gán giá trị 15 cho biến A. Và sau đó, chúng ta gán -35 cho biến B.



#### Ngoài ra, trong C#, chúng ta có thể gán một giá trị cho một biến bằng cách sử dụng một biến khác.

using System;

namespace CodeSample {

class Program {

static void Main(string[] args){

int A;

int B;

A = 15;

B = A;

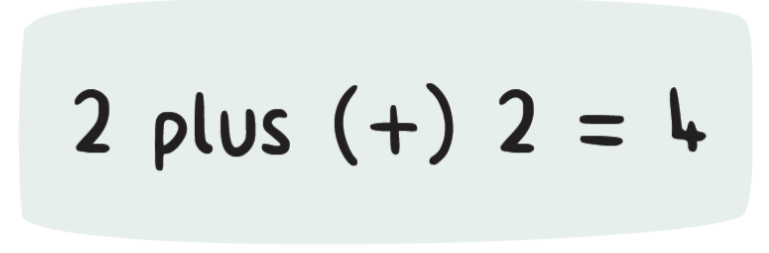
}

}

}

#### Ở phần đầu của mã, chúng ta thấy rằng cả hai biến A và B đều không được gán. Tiếp theo, chúng ta gán 15 cho biến A.

#### Trong dòng tiếp theo, chúng ta chỉ cần sao chép giá trị này vào biến B. Vì vậy, B cũng chứa giá trị, 15!



#### Ngoài ra, chúng ta có thể đảo ngược một số từ dương sang âm và ngược lại.

#### Điều này có thể được thực hiện bằng cách chỉ cần thêm một (-) trừ vào trước biến.

using System;

namespace CodeSample{

class Program{

static void Main(string[] args) {

int A;

int B;

A = 15;

B = -A;

}

}

}

#### Ở đây, biến B sẽ được gán giá trị đảo ngược của biến A là -15.



#### Tuyệt vời! Bây giờ bạn đã biết cách khai báo các biến và gán giá trị cho chúng!

#### Điều đó có nghĩa là bây giờ bạn đã biết cách dữ liệu được lưu trữ với sự trợ giúp của các biến trong C #.

#### Hãy tiếp tục công việc tốt!

### **Time for Decimals**

#### Để lưu trữ Decimals trong C#, chúng ta cần chọn đúng loại biến

#### Như chúng tôi đã đề cập trước đây, khá dễ dàng để khai báo một biến để lưu trữ các giá trị thập phân.

#### Chúng ta cần sử dụng từ khóa "kép" cho mục đích đó

### 

#### Trong C#, chúng ta có thể khai báo một kiểu biến kép như thế này::

### using System;

### namespace CodeSample{

### class Program{

### static void Main(string[] args){

### double A;

### }

### }

### }

#### Công việc dễ dàng! Bây giờ hãy chia nhỏ code của chúng ta:

#### **double** – là loại biến.

#### **A** - là tên của biến.

#### Vì vậy, hãy xem những gì bạn đã học được!

### 

#### Hãy thêm một vài ví dụ về khai báo biến và gán giá trị

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

double A = 1.2;

double B = -10.62;

double C = 0;

}

}

}

#### Ở đây, chúng ta có ba biến kép: A, B và C, chứa 3 giá trị khác nhau.

#### A chứa 1,2. B chứa -10,62. Và C chứa 0 (không)

#### Bạn có thể tự hỏi liệu bạn có thể sử dụng biến kép để lưu trữ số nguyên hay không.

#### Có! Chúng ta có thể!



#### Hãy thêm một vài ví dụ nữa.

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

double A = 12;

double B = -10;

double C = 0;

}

}

}

#### Tuyên bố này cũng đúng. Nhưng chờ đã? Nếu chúng ta có thể đơn giản sử dụng một kiểu biến thập phân, thì tại sao chúng ta lại sử dụng một kiểu số nguyên thay thế?

#### Đó là tất cả về tối ưu hóa mã, loại số nguyên đòi hỏi ít không gian hơn trong bộ nhớ so với loại thập phân.

#### Hiện tại, nó có vẻ khó hiểu, nhưng đừng lo lắng, chúng tôi sẽ đề cập đến chúng sau.

#### Khi bạn viết mã trong tương lai, hãy đảm bảo rằng mã của bạn được tối ưu hóa. Đó là một thực hành tuyệt vời trong lập trình!



using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

double A;

double B;

A = 20.5;

B = 10;

}

}

}

#### Hãy chia nhỏ code của chúng ta để giải thích những gì đang xảy ra: Lúc đầu, các biến A và B được khai báo là gấp đôi, nhưng chúng trống.

#### Trong dòng tiếp theo, giá trị 20,5 được gán cho biến A Và chúng tôi đã gán 10 cho biến B ở dòng cuối cùng



#### Tuyệt vời! Bây giờ bạn đã biết cách khai báo các biến kiểu kép và gán giá trị cho chúng!

#### Hãy tiếp tục công việc tốt!

### **Storing Characters**

#### Tuyệt vời! Bạn đang tiến bộ rất nhiều! Trước khi tiếp tục, có một điều nữa chúng ta cần đề cập

#### Chúng ta sẽ tìm hiểu cách các ký tự được lưu trữ trong C #. Đừng lo lắng, điều này rất giống với việc lưu trữ số!

#### Bắt đầu nào!



#### Để lưu trữ các ký tự trong C #, chúng tôi làm theo các bước tương tự như chúng tôi làm với các kiểu dữ liệu khác.

#### Chúng ta sử dụng kiểu biến "char" khi chúng ta cần lưu trữ các ký tự.

#### Dưới đây là các bước để khai báo một kiểu biến char:

* Loại biến - char
* Tên biến
* Và tất nhiên – kẻ thù của hầu hết các lập trình viên – DẤU CHẤM PHẨY.



#### Như chúng ta thấy, mọi thứ đều giống nhau ngoại trừ từ khóa "char"

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

char A;

}

}

}

#### Đã nói với bạn! Không có gì bạn chưa biết!

#### Chúng ta sử dụng khá nhiều cấu trúc cú pháp giống nhau khi chúng ta cần khai báo các biến.



#### Có một điểm quan trọng cần ghi nhớ khi gán giá trị cho loại biến char.

#### Khi chúng ta gán một giá trị cho chúng, chúng ta cần đặt nó bằng dấu ngoặc đơn.

#### Ví dụ: 'b'

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

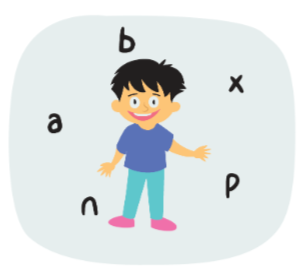
char A = ‘b’;

}

}

}

#### Đầu tiên, chúng ta khai báo một biến char và sau đó chúng ta gán một giá trị cho nó. Giá trị của chúng tôi là chữ 'b' cho ví dụ cụ thể này.



#### Điều quan trọng nhất về một biến char là nó chỉ có thể chứa một ký tự duy nhất - bất kể nó là gì - hãy để nó là một chữ cái, một chữ số, ký hiệu hoặc thậm chí là một khoảng trống.

#### Chỉ có một và đó là tất cả. Chúng tôi không thể gán một từ cho nó.

#### Nhưng nếu chúng ta cần gán một giá trị cho biến của chúng ta có nhiều hơn một ký tự thì sao? Chà, trong trường hợp đó, chúng ta sẽ sử dụng một kiểu dữ liệu khác gọi là String.

#### Nhưng đừng lo lắng, được chứ? Chúng tôi sẽ sớm giải quyết vấn đề đó.



#### Hãy củng cố kiến thức của chúng ta với một vài ví dụ nữa.

#### Khi nói đến việc khai báo các biến, chỉ cần nhớ rằng chúng ta sử dụng cùng một cú pháp.

#### Điều duy nhất thay đổi là loại biến.



#### Ê! Chúng tôi gần như đã hoàn thành với các biến!

#### Bây giờ bạn đã biết cách khai báo và gán một giá trị cho các biến kiểu "char"! Chỉ cần đừng quên dấu nháy đơn và "char" đó chỉ chứa một ký tự.

#### Bạn đã sẵn sàng để tiến về phía trước? Vậy thì chúng ta đi thôi!

### **Sequence of Characters**

#### Tuyệt! Đây là bước cuối cùng của bạn để học những điều cơ bản về biến!

#### Chúng ta sẽ tìm hiểu về một loại biến đặc biệt trong chương này.

#### Bạn đoán đúng, đó là loại String.

#### Như chúng ta đã biết - các biến kiểu "char" chỉ có thể chứa một ký tự.

#### Tuy nhiên, "Chuỗi" có thể chứa toàn bộ:

* Từ
* Câu
* Đoạn
* Hoặc thậm chí cả một cuốn sách!



#### Bạn có thể tự hỏi tại sao chúng ta thậm chí cần kiểu ký tự nếu chúng ta có thể sử dụng kiểu chuỗi thay thế?

#### Câu trả lời là tối ưu hóa. Chúng tôi đã nói về nó là một thực hành rất quan trọng trong lập trình, nhớ chứ?

#### Các biến "char" chiếm ít không gian hơn trong bộ nhớ so với các biến "chuỗi".



#### Bên cạnh khía cạnh bộ nhớ của nó, cũng có một sự khác biệt quan trọng trong việc khai báo các biến String

#### Cú pháp khai báo một biến chuỗi không hoàn toàn giống với cú pháp của biến char.

#### Như đã đề cập trước đây, chúng tôi sử dụng dấu nháy đơn khi chúng tôi gán một giá trị cho loại biến char của chúng tôi.

#### Tuy nhiên, chúng tôi sử dụng dấu ngoặc kép khi chúng tôi muốn làm tương tự với các biến chuỗi

#### Ngay cả khi nó chỉ là một ký tự duy nhất - chúng ta vẫn phải đặt nó bằng dấu ngoặc kép.

#### Tất cả đã sẵn sàng? Sẵn sàng, ổn định, đi!



#### Như chúng ta đã nói trước đây, chúng ta sử dụng các biến kiểu "chuỗi" khi giá trị của chúng ta chứa nhiều hơn một ký tự.

#### Không có gì mới về cú pháp khi khai báo một biến kiểu "chuỗi":

* Loại biến - "Chuỗi"
* Tên biến
* Và tất nhiên - DẤU CHẤM PHẨY.



#### Mã trông như thế này khi chúng ta khai báo biến chuỗi,

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

string A;

}

}

}

#### Dễ!



#### Nếu chúng ta muốn gán một từ, nó sẽ trông như thế này:

System;

namespace CodeSample{

class Program{

static void Main(string[] args){

string A = “buddy”;

}

}

}

#### Điều quan trọng nhất bạn cần ghi nhớ là giá trị phải nằm giữa dấu ngoặc kép.



#### Congrats!

#### Bây giờ bạn đã biết cách khai báo một biến thuộc loại "chuỗi" và gán một giá trị cho nó!

#### Hãy nhớ rằng - các biến kiểu chuỗi có thể chứa nhiều ký tự và khi chúng ta muốn gán giá trị cho chúng, chúng ta cần viết các giá trị đó giữa dấu ngoặc kép!

#### Chúng tôi đã hoàn thành khá nhiều với những điều cơ bản về lưu trữ dữ liệu và các biến trong C #.

#### Bạn đã làm một công việc tuyệt vời! Bây giờ chúng ta hãy tiếp tục sang chương tiếp theo!

### **What is a Program?**

#### Các chương trình máy tính đã trở thành một phần thiết yếu trong cuộc sống hàng ngày của chúng ta.

#### Giống như BÂY GIỜ! - Bạn đang sử dụng ứng dụng tuyệt vời này để tìm hiểu những điều cơ bản về C #!

#### Và đó là một điều tuyệt vời để làm!

#### Trong chương này, chúng tôi sẽ đề cập đến những điều cơ bản về cách một chương trình cơ bản được viết bằng C #.

#### Bắt đầu nào!

### **First C# Program**

#### Đối với những điều cơ bản về C #, chúng ta sẽ sử dụng Ứng dụng Bảng điều khiển để bắt đầu!

#### Về cơ bản, nó có nghĩa là một cửa sổ trên màn hình của bạn hiển thị một số dữ liệu.

#### Chúng tôi đã chọn cụ thể các Ứng dụng Bảng điều khiển để bắt đầu.

#### Thật dễ dàng để xây dựng các ứng dụng Bảng điều khiển thực hiện hầu hết các thao tác nền.

#### Ngoài ra, việc chọn các ứng dụng bảng điều khiển sẽ cho phép chúng tôi bắt đầu chạy!

#### Chương trình đầu tiên của chúng tôi sẽ hiển thị cụm từ nổi tiếng trong thế giới lập trình - Hello World! - trên cửa sổ bảng điều khiển.



#### Vì vậy, chúng tôi có một tệp có phần mở rộng '.cs' là tệp mã C #!

#### Để xây dựng một ứng dụng console, trước tiên, chúng ta cần thêm các thư viện cần thiết vào ứng dụng của mình.

#### Điều thứ hai là - không gian tên!

#### Điều tiếp theo cần làm là tạo một lớp.

#### Hãy nhớ rằng, C # là một ngôn ngữ lập trình hướng đối tượng.

#### Và sau khi tạo một lớp chúng ta cần tạo một hàm entry point, mặc định là hàm main(). Đó là nơi thực thi mã bắt đầu.

#### Hãy xem tất cả những gì được giải thích ở trên trông như thế nào trong mã!

using System;

namespace ConsoleApp

{

class Program

{

static void Main(string[] args)

{

}

}

}

#### Không phức tạp lắm phải không?!

#### Chúng ta có thể thấy rằng chương trình này hiện không có bất kỳ đầu ra nào.

#### Hãy tiếp tục và làm một cái!



#### Trong C#, việc thực thi mã bắt đầu từ hàm Main.

#### Vì vậy, tất cả các chú thích và hành động mà chúng tôi muốn chương trình của chúng tôi thực hiện phải được viết giữa các dấu ngoặc nhọn - trong chức năng Chính.

#### Để tạo một output, chúng ta sẽ sử dụng phương thức Console.WriteLine().

#### Chúng tôi phải đính kèm các từ chúng tôi muốn chương trình của chúng tôi hiển thị cho chúng tôi bằng dấu ngoặc kép.



using System;

namespace ConsoleApp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

Hello World!

#### Khá đơn giản phải không?

#### Bạn lo lắng về phần Console.WriteLine()?

#### Bây giờ, chúng ta hãy hiểu rằng đó là một cách để xuất bất kỳ tin nhắn nào trên màn hình.

#### Trong các chương sắp tới, chúng ta sẽ đưa sự hiểu biết này đi trước một chút.



#### Bắt đầu với C # trong thế giới lập trình là một lựa chọn tuyệt vời!

#### C # có tất cả các loại khả năng và đó là một ngôn ngữ lập trình hướng đối tượng rất phổ biến.

#### Sự lựa chọn tuyệt vời!

#### Hãy tiếp tục!

#### Để tạo một ứng dụng C#, chúng ta cần:

* Bao gồm các thư viện mà chúng ta sẽ sử dụng. Ví dụ: sử dụng Hệ thống;
* Tạo không gian tên. không gian tên MyFirstProgram
* Tạo một lớp học bên trong nó. lớp MyFirstClass
* Tạo hàm Main nơi bắt đầu thực thi mã. static void main(string[] args)



#### Ngoạn mục!

#### Bạn vừa chứng minh rằng bạn biết cấu trúc cơ bản của chương trình C #.

#### Đã đến lúc nói Xin chào C#!

#### Hãy đi sâu hơn và tìm hiểu một số điều tuyệt vời sẽ giúp bạn đưa kỹ năng C # của mình lên một tầm cao mới!

### **Introduction**



#### Các phép toán trong C # khá giống với các ngôn ngữ lập trình khác.

#### Các phép toán chính là cộng, trừ, nhân và chia.

#### Tất cả các thao tác này đều cần thiết cho việc lập trình, vì vậy hãy đảm bảo bạn chú ý đến một vài màn hình tiếp theo.



#### Để thực hiện các phép toán, chúng tôi sử dụng các phép toán sau:

* + Để bổ sung
* - để trừ
* \* Đối với phép nhân
* / cho Bộ phận

#### Như bạn có thể thấy, bạn đã rất quen thuộc với các toán tử này từ toán học cơ bản.

### **Working with Addition**

#### Hãy thử một ví dụ để xem chúng trông như thế nào trong mã.

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

int A = 10;

int B = 0;

B = A + 10;

}

}

}

#### Hãy chia nhỏ những gì code của chúng ta nói. Đầu tiên chúng ta khai báo hai biến số nguyên - A và B.

#### Sau đó, chúng tôi gán cho họ các giá trị lần lượt là 10 và 0.

#### Trong dòng cuối cùng, chúng tôi đã thay đổi giá trị của B thành A + 10.

#### Vì A = 10, giá trị hiện tại của B sẽ là A + 10 = 20.



#### Hãy thử một ví dụ khác.

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

int A = 10;

double B = 0;

B = A + 10;

}

}

}

#### Hãy giải thích những gì chúng tôi vừa làm:

#### Đầu tiên, chúng ta đã khai báo hai biến - A và B.

#### A là một biến số nguyên mà chúng ta đã gán 10. B là một loại biến kép có giá trị bằng 0.

#### Trong dòng tiếp theo, chúng ta đã thay đổi giá trị của B thành A + 10, tuy nhiên, nó sẽ không có kết quả giống như ví dụ trước.

#### Bởi vì biến B thuộc loại kép, không phải là số nguyên!

#### Kết quả của thao tác cuối cùng, biến B sẽ có giá trị 20.0000000! Chúng ta có được phép thực hiện các phép toán này với các loại biến khác nhau không?

#### Vâng, chúng tôi là!



#### Chúng tôi cũng có một toán tử gia tăng - "++" trong C #.

#### Chúng ta có thể sử dụng toán tử này để tăng giá trị lên 1.

#### Hãy xem nó hoạt động:

int c = 5;

c++;

#### Chúng tôi đã tăng giá trị của c lên 1. Như vậy, c bằng 6.

int c = 5;

c++;

c++;

#### Vì toán tử gia tăng đã được sử dụng hai lần, C sẽ là 7 lần này! Cũng giống như gia tăng, chúng ta cũng có một toán tử giảm: "--".



#### Bây giờ chúng ta hãy thử một loại ví dụ khác:

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

double A = 10.55;

int B = 0;

B = A + 10;

}

}

}

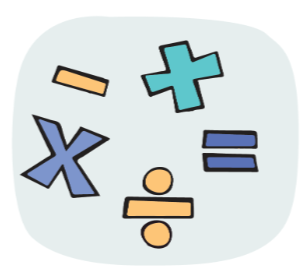
#### Mã này không chính xác!

#### Tại sao? Bởi vì chúng ta không được phép gán giá trị thập phân cho một loại biến số nguyên

#### Và đó là bởi vì nếu chúng ta có thể - chúng ta sẽ mất độ chính xác.

#### Khi kết thúc phép toán, giá trị của B sẽ là 20,55, đây không phải là giá trị phù hợp cho biến số nguyên.

#### Hãy nhớ rằng, các biến số nguyên chỉ có thể chứa các số nguyên!



#### Chúng tôi chưa đề cập đến phép trừ trong C #.

#### Chà, đó sẽ là chủ đề tiếp theo của chúng tôi.

#### Đi thôi!

### **Time for Subtraction**

#### Phép trừ trông như thế này - toán tử "-" (trừ).

#### Bạn đã đoán nó được sử dụng để làm gì!

#### Hãy xem một ví dụ:

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

int B;

B = 25 - 10;

}

}

}

#### Kết quả của hoạt động, giá trị của biến B sẽ là 15.



#### Trong phần cộng, chúng ta đã nói về toán tử gia tăng, nhớ chứ?

#### Vâng, chúng ta cũng có một toán tử tương tự trong phép trừ.

#### Nhưng lần này, nó được gọi là toán tử giảm "--", thay vì gia tăng!

int c = 5;

c--;

#### Giá trị của cis bây giờ là 4.

int c = 5;

c--;

c--;

#### Giá trị của c sẽ là 3!



#### Bây giờ chúng ta đã đề cập đến phép cộng và trừ, hãy chuyển sang hai cái tiếp theo - phép nhân và phép chia!

### **Multiplication**

#### Toán tử nhân trong C# trông như thế này - "\*"

#### Hãy thử một ví dụ để xem nó hoạt động:

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

int A = 4;

int B = 4;

int C = A \* B;

}

}

}

#### Khi việc thực thi mã của chúng ta được thực hiện, giá trị của biến C sẽ là 16.



#### Trong phép nhân, chúng ta không có bất cứ thứ gì tương tự như toán tử ++ và -.

#### Bây giờ chúng ta hãy điều tra sự phân chia trong C #!

### **Some Division**

#### Chúng ta sử dụng toán tử chia - "\" trong C# khi chúng ta cần thực hiện thao tác chia.

#### Chúng ta hãy xem một ví dụ đơn giản:

using System;

namespace CodeSample{

class Program{

static void Main(string[] args){

int A = 6;

int B = 2;

int C = A \ B;

}

}

}

#### Kết quả của việc thực thi mã, giá trị của biến C sẽ là 3!



#### Chúng tôi cũng không có bất cứ điều gì tương tự như các toán tử ++ và -- trong bộ phận.

#### bất chấp! Hãy kết thúc với phép toán này!

#### Hãy đi đến "toán học từ"!

#### Chỉ đùa thôi, không có chuyện đó cả, đừng lo lắng!

#### Hãy tiếp tục với chương tiếp theo!

### **Operations with Strings**

#### Chúng ta có thể thực hiện các thao tác khác nhau với các chuỗi và ký tự trong C #.

#### Dưới đây là phác thảo về những gì chúng ta sẽ tìm hiểu trong chương này:

* Chuyển đổi ký tự thành số nguyên
* Thêm chuỗi lại với nhau
* Chuỗi dưới dạng mảng char
* Chuyển đổi bất kỳ loại dữ liệu nào thành một chuỗi

### **Converting Chars to Integers**

#### Tất cả các ký tự đều có giá trị số (Mã ASCII).

#### Ví dụ - chữ 'a' có thể được biểu thị bằng số 97 theo mã ASCII của nó.

#### Chữ 'b' có giá trị là 98, v.v.

#### Tất nhiên, không cần phải ghi nhớ giá trị int của từng ký tự, nhưng điều quan trọng là phải hiểu cách chuyển đổi hoạt động.

#### Hãy xem chuyển đổi đang hoạt động!

using System;

namespace ConsoleApp

{

class Program

{

static void Main(string[] args)

{

char x = 'a';

int i = (int)x;

Console.WriteLine(i);

}

}

}

97

#### Hãy chia nhỏ code của chúng ta.

#### Đầu tiên, chúng ta khai báo một biến kiểu ký tự với tên x, chứa một giá trị là 'a'.

#### Tiếp theo, chúng tôi khai báo một biến khác, lần này là số nguyên kiểu và gán cho nó giá trị nguyên của biến x, là 97 theo ASCII.

#### Có lẽ bạn đang tự hỏi, quá trình chuyển đổi hoạt động như thế nào?

#### Hãy xem nào!

using System;

namespace ConsoleApp

{

class Program

{

static void Main(string[] args)

{

char x = (char)97;

Console.WriteLine(x);

}

}

}

a

#### Chúng ta khai báo một biến thuộc loại char – x.

#### Sau đó, chúng tôi đã chuyển đổi giá trị 97 thành một ký tự!

### **Addition of Strings**

#### Cũng có thể thêm các chuỗi lại với nhau.

#### Chúng tôi gọi đó là quá trình ghép nối!

#### Nghe có vẻ lạ, nhưng nó thực sự có ý nghĩa!

#### Ngoài ra, nó có ích khi bạn cần tạo một câu từ các từ khác nhau.

#### Hãy xem nó trông như thế nào.

using System;

namespace ConsoleApp

{

class Program

{

static void Main(string[] args)

{

string name = "John";

string surname = "Doe";

string Fullname = name + surname;

Console.WriteLine(Fullname);

}

}

}

JohnDoe

#### Chúng tôi đã kết hợp hai biến chuỗi để tạo thành một tên đầy đủ!

#### Nhưng chờ đã, chúng tôi đã quên mất không gian! Hãy sửa nó!

using System;

namespace ConsoleApp

{

class Program

{

static void Main(string[] args)

{

string name = "John";

string surname = "Doe";

string Fullname = name + " " + surname;

Console.WriteLine(Fullname);

}

}

}

John Doe

#### Bây giờ kết quả chính xác là những gì chúng tôi đang tìm kiếm!

#### Chúng tôi đã thêm khoảng trắng bằng " " ở giữa để làm cho nó đúng!

#### Hãy nhớ rằng toán tử "+" được sử dụng để nối chuỗi.

#### Tuy nhiên, nó không tự thêm không gian.

#### Do đó, chúng ta cần phải làm điều đó bằng tay.

#### Đó là nó. Hãy đi xa hơn!

#### Ngoài ra, cùng một toán tử khi được sử dụng với các biến số sẽ thực hiện một thao tác cộng.

### **Strings as a Char Array**

#### Hãy tưởng tượng chúng ta cần truy cập một chữ cái duy nhất từ một từ được lưu trữ trong một loại biến chuỗi.

#### Chúng ta có thể làm gì?

#### Chúng ta cần sử dụng chuỗi như một mảng.

#### Bạn có thể hoang mang như, một mảng là gì?

#### Đừng lo lắng. Chúng ta sẽ thảo luận về chúng sau.

#### Hiện tại, chỉ cần nhớ rằng đó là một nhóm các biến được nhập giống nhau được gọi bằng một tên chung.

#### Hãy xem một ví dụ:

using System;

namespace ConsoleApp

{

class Program

{

static void Main(string[] args)

{

string word = "Friend";

char character = word[0];

Console.WriteLine(character);

}

}

}

F

#### Nhưng nó không nên. Bởi vì chỉ mục trong C # bắt đầu từ con số không. Đầu ra của chúng tôi là ký tự đầu tiên 'F'.

#### Nếu chỉ mục của chúng ta là 1 thay vì 0, đầu ra sẽ là ký tự thứ hai trong từ, đó là 'r'.

#### Bạn sẽ quen với các quy tắc lập chỉ mục khi bạn tiến lên phía trước.



#### Điều đó cực kỳ hữu ích và tiện dụng vì biết những điều cơ bản về chỉnh sửa văn bản rất hữu ích khi bạn viết bất kỳ loại chương trình nào có giao diện người dùng.

#### Ngoài ra, một điều bạn cần nhớ là chuỗi chỉ là một mảng các ký tự có thể được truy cập bất cứ khi nào bạn muốn.

### **Converting any type of data to String**

#### Bạn sắp học một phương pháp quan trọng liên quan đến chuỗi.

#### Phương pháp này sẽ được sử dụng gần như trong mọi chương trình.

#### Trong C #, bạn có thể chuyển đổi hầu hết mọi loại dữ liệu thành kiểu chuỗi ngay cả khi nó được lưu trữ trong một biến, số nguyên, số thập phân, ký tự hoặc bất kỳ biến nào khác.

#### Bạn chỉ cần thêm phương thức ma thuật ToString();

#### Hãy nhảy vào một ví dụ:

int a = 58;

string sentence = “a equals ”+ a.ToString();

Console.WriteLine(sentence);

#### a bằng 58

#### Hãy xem một ví dụ khác mà chúng ta sẽ cố tình mắc lỗi để cho thấy tầm quan trọng của phương thức ToString():

int a = 58;

string sentence = “a equals ”+ a;

Console.WriteLine(sentence);

#### Mã này sẽ không biên dịch và do đó, sẽ không hoạt động!

#### Nếu không sử dụng phương thức ToString(), code của chúng ta sẽ ném một ngoại lệ vì chúng ta không được phép thêm một số nguyên và một giá trị chuỗi mà không chuyển đổi chúng!



#### Tuyệt vời!

#### Bây giờ bạn đã biết các hoạt động với các từ và ký tự mà bạn có thể sử dụng trong các chương trình trong tương lai của mình!

#### Một lập trình viên giỏi phải biết những điều cơ bản về hoạt động chuỗi trong C #!

#### Tốt!

### **Input**

#### Đầu vào và đầu ra được coi là một phần không thể thiếu của lập trình.

#### Có hai cách chúng ta trao đổi thông tin với máy tính:

* Chèn dữ liệu
* Truy xuất dữ liệu



#### Trong Ứng dụng Bảng điều khiển C #, chúng tôi sử dụng phương pháp kết hợp sau để lấy thông tin:

Console.Read();

#### Hoặc

Console.Readline();

#### Các phương pháp này được sử dụng để đọc đầu vào của người dùng từ bảng điều khiển.

#### Hãy xem một ví dụ để xem điều này trong thực tế:

using System;

namespace ConsoleApp{

class Program {

static void Main(string[] args){

int a;

a = int.Parse(Console.ReadLine());

}

}

}

#### Hãy chia nhỏ code của chúng ta.

#### Ở đây, chúng ta đã khai báo một biến số nguyên có tên là 'a' và nó đang chờ giá trị của nó được nhập từ bàn phím.

using System;

namespace ConsoleApp{

class Program{

static void Main(string[] args){

int a;

a = int.Parse(Console.ReadLine());

}

}

}

#### Đợi! Có một cái gì đó mới ngoài kia phải không? Các "int. Phương pháp" phân tích! Bạn có thể đang suy nghĩ về lý do tại sao chúng tôi sử dụng nó trong mã của chúng tôi.

#### Vâng, chức năng của phương thức Parse() là chuyển đổi dữ liệu từ loại này sang loại khác.

#### Biến mà chúng ta khai báo là số nguyên kiểu, tuy nhiên, dữ liệu được thu thập bằng phương thức Console.ReadLine() luôn nằm trong kiểu chuỗi.

#### Đây là lý do tại sao chúng tôi cần int. Phương thức Parse() để chuyển đổi dữ liệu từ chuỗi sang số nguyên!

#### Hãy xem một ví dụ khác:

using System;

namespace ConsoleApp{

class Program {

static void Main(string[] args){

string a;

a = Console.ReadLine());

}

}

}

#### Bạn có thể tự mình thấy sự khác biệt.

#### Vì chúng ta khai báo biến của chúng ta là một chuỗi, chúng ta không cần sử dụng phương thức Parse() nữa.

#### Hãy xem lại những gì chúng ta vừa học được.



#### Tuyệt vời, bây giờ bạn đã biết cách cung cấp đầu vào cho các chương trình của mình và giao tiếp một chiều với chúng!

#### Chỉ cần nhớ rằng dữ liệu được thu thập với Console.ReadLine () là loại chuỗi, vì vậy nếu bạn muốn sử dụng nó như int, double hoặc char, bạn cần phải chuyển đổi nó trước!

#### Tuyệt vời! Hãy cùng tìm hiểu sâu hơn!

### **Output**

#### Cho đến nay chúng tôi đã nói về cách cung cấp đầu vào cho các chương trình của chúng tôi. Mát!

#### Nhưng còn việc nhận đầu ra từ họ thì sao?

#### Hộp thoại với các chương trình bao gồm hai phần - chúng tôi cung cấp đầu vào và lần lượt, nhận đầu ra.

#### Bây giờ chúng ta hãy chuyển sang phần còn thiếu - cách chúng ta có thể truy xuất dữ liệu từ chương trình của mình.



#### Hãy bắt đầu với cụm từ nổi tiếng trong lập trình - "Xin chào thế giới"

#### Chúng tôi sử dụng phương thức kết hợp của Console.WriteLine() để hiển thị nó trên bảng điều khiển.

#### Tất cả những gì chúng ta cần làm là nhập giá trị hoặc tên của biến giữa dấu ngoặc và boom, chúng ta đã hiểu rồi!

#### Đây là giao diện của nó: Console.WriteLine ("Hello World");

#### Trong C#, có hai phương pháp để hiển thị đầu ra trong Bảng điều khiển:

* WriteLine () - luôn gắn thêm một ký tự dòng mới vào cuối chuỗi, có nghĩa là bất kỳ đầu ra tiếp theo nào sẽ bắt đầu trên một dòng mới
* Write() - xuất một hoặc nhiều giá trị ra màn hình mà không có ký tự dòng mới

#### Hãy xem một ví dụ mã thực tế để xem điều này trong thực tế.

#### Mã đầy đủ của việc hiển thị "Hello World" trên màn hình sẽ trông như thế này:

using System;

namespace ConsoleApp{

class Program{

static void Main(string[] args){

string a = “Hello World!”;

Console.WriteLine(a);

}

}

}

Hello World

#### Hãy chia nhỏ code của chúng ta để giải thích những gì đang xảy ra:

#### Đầu tiên, chúng tôi khai báo một biến chuỗi chứa giá trị - "Hello World!"

#### Sau đó, chúng tôi sử dụng phương thức Console.WriteLine() yêu quý của chúng tôi để hiển thị nó trong bảng điều khiển.

#### Chà, điều đó thật dễ dàng!

#### Hãy xem một ví dụ khác:

using System;

namespace ConsoleApp{

class Program{

static void Main(string[] args) {

Console.WriteLine(“Hello World”);

}

}

}

Hello World

#### Như bạn có thể thấy, lần này chúng tôi đã thực hiện một cách tiếp cận khác trong khi có cùng đầu ra.

#### Điều này cho thấy rằng chúng ta không nhất thiết phải khai báo một biến để hiển thị giá trị của nó.

#### Chúng ta có thể nhập giá trị trực tiếp trong dấu ngoặc và hiển thị nó.

using System;

namespace ConsoleApp{

class Program{

static void Main(string[] args){

double a;

a = double.Parse(Console.ReadLine());

Console.WriteLine(a);

}

}

}

10.55

#### Alright! Hãy giải thích những gì đang xảy ra với code của chúng ta.

#### Chúng tôi có một loại biến kép mà chúng tôi đã gán đầu vào đến từ bàn phím.

#### Khi chúng tôi bắt được đầu vào, chúng tôi hiển thị nó trong bảng điều khiển bằng cách sử dụng phương thức Console.WriteLine ().

#### Nói cách khác, chúng tôi đã biến đầu vào thành đầu ra của mình.

#### Như bạn có thể thấy từ dòng cuối cùng, không cần chuyển đổi kiểu khi hiển thị đầu ra.



#### Tuyệt vời!

#### Bây giờ bạn đã quen thuộc với cả hai phần của cuộc đối thoại giữa người dùng và chương trình!

#### Đó là một bước tiến lớn!

#### Chỉ cần lưu ý rằng giá trị chúng tôi muốn hiển thị phải được viết giữa các dấu ngoặc và kèm theo dấu ngoặc kép.

#### Mặt khác, nếu bạn muốn hiển thị giá trị được gán cho một biến, chỉ cần viết tên của biến mà không sử dụng dấu ngoặc kép.

#### Hãy xem làm thế nào chúng ta có thể sử dụng kiến thức mới có được của chúng tôi trong các chủ đề sắp tới!

### **Decision Making**

#### Trong cuộc sống hàng ngày, chúng ta phải so sánh mọi thứ để đưa ra quyết định theo các khía cạnh khác nhau như:

* Tôi nên đến phòng tập thể dục hay thư giãn trước TV?
* Tôi có nên mua kem thay vì kẹo mút?
* Tôi nên đi chơi với bạn bè hay ở nhà ngủ cả ngày?



#### Nhưng, chờ đã! Làm thế nào để chúng ta đưa ra những quyết định này?

#### Như bạn đã biết, chúng tôi so sánh các điều kiện khác nhau và đưa ra quyết định đúng đắn cho một tình huống nhất định.

#### Tương tự như con người, máy tính cũng có khả năng đưa ra quyết định.

#### Quá trình ra quyết định hoạt động khá giống nhau và nó dựa trên việc đánh giá tình huống.

#### Được rồi, chúng ta hãy xem một ví dụ thực tế cơ bản về việc ra quyết định.

#### Hãy tưởng tượng chúng ta có hai điều kiện: **Tôi nên đến GYM hay thư giãn trước TV?**

#### Điều kiện:

* Hôm nay tôi cảm thấy mệt mỏi!
* Tôi đang cảm thấy tràn đầy năng lượng!

#### Vì vậy, nếu tôi cảm thấy mệt mỏi hôm nay, tôi sẽ bỏ qua phòng tập thể dục và có thể ở nhà xem TV.

#### **Nhưng**, nếu tôi cảm thấy tràn đầy năng lượng và động lực - tôi chắc chắn sẽ đến phòng tập thể dục!

### **Simple Condition Statement**

#### Máy tính đưa ra quyết định khá giống nhau - chúng so sánh một số điều kiện và đi đến quyết định!

#### Chúng tôi sử dụng câu lệnh có điều kiện - **IF** - để cho phép máy tính đưa ra quyết định.

#### Hãy bắt đầu với một tình huống thực tế mà chúng ta đã sử dụng vài phút trước - **Bên ngoài trời rất nóng và bạn muốn hạ nhiệt - bạn sẽ mua gì?**

#### Trời nóng và tôi muốn hạ nhiệt nên tôi sẽ mua những thứ có thể làm mát tôi, phải không?

#### Vì vậy, chúng ta có thể viết: IF (**Kem** sẽ làm mát tôi **xuống hoặc Lollipop** sẽ làm mát tôi xuống

#### Như chúng ta đã biết - **Kem** sẽ giúp bạn hạ nhiệt tốt hơn - do đó, bạn chọn Kem!



#### Hãy nhảy vào một ví dụ để xem câu lệnh điều kiện đang hoạt động:

int A = 15;

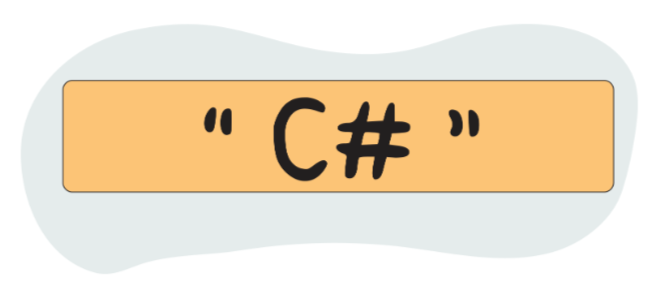
int B = 10;

if(A>B){

Console.WriteLine(“A is higher than B”);

}

A is higher than B



#### Nhưng chờ đã? Điều gì sẽ xảy ra nếu B lớn hơn? Đầu ra sẽ trông như thế nào?

#### Yup, bạn nói đúng - vì chúng tôi đã không tính đến điều kiện đó, sẽ không có bất kỳ đầu ra nào cả.

#### Hãy sửa nó!

#### Để tính đến điều kiện khác, chúng ta cần thêm câu lệnh điều kiện thứ hai tương ứng với ĐIỀU **GÌ NẾU ĐIỀU KIỆN KHÔNG CHÍNH XÁC!**

#### Chà, đây là lúc tuyên bố **ELSE** phát huy tác dụng

#### Hãy sử dụng một ví dụ để xem các câu lệnh IF/ELSE đang hoạt động:

int A = 15;

int B = 20;

if(A>B){

Console.WriteLine(“A is higher than B”);

}

else{

Console.WriteLine(“B is higher than A”);

}

B is higher than A

#### Hãy chia nhỏ code của chúng ta:

#### Đầu tiên, chúng ta khai báo hai biến số nguyên có tên A và B và gán cho chúng các giá trị 15 và 20 tương ứng.

#### Điều tiếp theo là tuyên bố có điều kiện của chúng tôi.

#### Câu lệnh IF đánh giá điều kiện và vì điều kiện không đúng (A không cao hơn B), phép toán được chuyển sang câu lệnh ELSE.

#### Do đó, mã trong câu lệnh ELSE được thực thi và đầu ra được hiển thị trên bảng điều khiển.

### **Statements with Multiple Conditions**

#### Cho đến nay, các ví dụ dựa trên hai điều kiện: Cho dù A > B hay A < B.

#### Nhưng nếu điều kiện là A = B thì sao? Chúng tôi đã không xem xét điều đó?!

#### Chà, nếu chúng ta cũng muốn tính đến điều kiện cụ thể đó, chúng ta cần sử dụng câu lệnh **ELSE IF**.

#### Chúng ta hãy xem nó trông như thế nào trong một mẫu mã C #.

#### Hãy sửa đổi ví dụ trước:

int A = 10;

int B = 10;

if(A>B) {

Console.WriteLine(“A is higher than B”);

}

else if(A<B){

Console.WriteLine(“B is higher than A”);

}

else{

Console.WriteLine(“A is equal to B”);

}

A is equal to B

#### Bạn đã có cách nó hoạt động, phải không?

#### Đầu tiên, điều kiện IF được kiểm tra và vì nó không đúng, thao tác được chuyển đến câu lệnh ELSE IF.

#### Điều kiện ELSE IF cũng không đúng!

#### Một lần nữa, hoạt động được chuyển đến câu lệnh tiếp theo (ELSE) cuối cùng hiển thị đầu ra được hiển thị ở trên.



#### Ngoài ra, bạn có thể so sánh nhiều điều kiện trong một câu lệnh **IF**.

#### Chúng tôi sử dụng các toán tử đặc biệt cho mục đích đó: && - có nghĩa là **VÀ**, || - có nghĩa là **OR**.

#### Như bạn đã biết, chúng tôi sử dụng toán tử "=" để gán giá trị.

#### Nhưng toán tử "==" được sử dụng để so sánh. Nó kiểm tra xem hai giá trị có bằng nhau hay không và trả về true nếu không, nó trả về false.

#### Xem mẫu mã:

int A = 15;

int B = 15;

int C = 20;

if(A == B && C > B)

{

Console.WriteLine(“A and B are equal, and C is higher than B”)

}

#### Vì cả hai điều kiện đều được đáp ứng (A == B &&; C > B), toàn bộ điều kiện IF sẽ trả về TRUE và mã trong câu lệnh IF sẽ được thực thi.

#### Tuyệt vời, chúng ta hãy tiếp tục!

#### Chúng ta hãy xem mã với || - **HOẶC nhà** điều hành.

int A = 15;

int B = 15;

int C = 20;

if(A == B || C > B)

{

Console.WriteLine(“A and B are equal, or C is higher than B”);

}

else if(A == B || C < B)

{

Console.WriteLine(“A and B are equal, B is higher than C”) n}

else

{

(Console.WriteLine(“This is Else”);

}

A and B are equal, or C is higher than B

#### Tuyệt!

#### Bạn có thể đã hiểu các câu lệnh có điều kiện quan trọng như thế nào trong C #.

#### Học chúng là một bước tiến lớn.

#### Xin chúc mừng và chúng ta hãy chuyển sang chủ đề tiếp theo!

### **The Concept of Loops**



#### Đôi khi trong lập trình, chúng ta cần lặp lại cùng một thao tác nhiều lần.

#### Để làm điều đó, có hai cách mà chúng ta có thể chọn:

* Viết cùng một mã nhiều lần tùy thích.
* HOẶC tạo một vòng lặp để lặp lại cùng một thao tác bao nhiêu lần tùy thích!

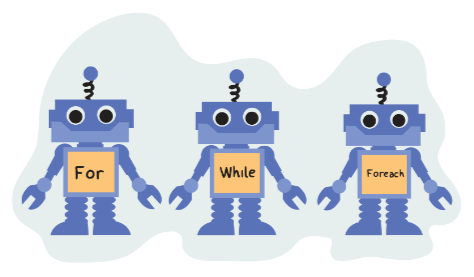
#### Tất nhiên, khi chúng ta viết mã, chúng ta muốn mã của chúng ta có hiệu quả.



#### Vòng lặp là gì?

#### Đây là một trong những công cụ hữu ích và thường được sử dụng nhất trong lập trình, cho phép chúng ta lặp lại cùng một thao tác nhiều lần.

#### Các vòng lặp rất hữu ích, đặc biệt là khi chúng ta không chắc chắn về việc cùng một thao tác sẽ phải lặp lại bao nhiêu lần. Có thể hai, có thể năm hoặc có thể một triệu lần. Đó là nơi các vòng lặp là cần thiết.



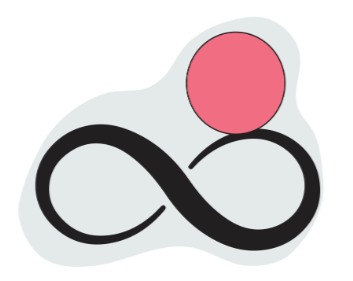
#### Có ba loại vòng lặp trong C #:

* FOR
* WHILE
* FOREACH

#### Như đã nêu ở trên, tất cả đều có một nhiệm vụ chung, đó là lặp lại các thao tác.

#### Chúng có thể thay thế nhau trong phần lớn các trường hợp.

#### Tuy nhiên, chúng tôi chọn và chọn loại vòng lặp chúng tôi cần sử dụng dựa trên tác vụ trong tay để làm cho mã của chúng tôi sạch sẽ và hiệu quả.



#### Tất cả các vòng lặp đều có ưu điểm riêng vì vậy chúng ta cần sử dụng đúng loại vòng lặp để có được hiệu suất tốt nhất có thể.

#### Khi làm việc với các vòng lặp, chúng ta phải luôn chỉ định điều kiện mà vòng lặp dừng hoạt động.

#### Trong trường hợp tình trạng xấu hoặc không chính xác, chúng ta kết thúc với vô tận, hay nói cách khác, các vòng lặp vô hạn.

#### Nếu một vòng lặp là vô hạn, điều đó có nghĩa là vòng lặp chỉ lặp lại hoạt động và không bao giờ kết thúc.

#### Đây là một tình huống có vấn đề thường dẫn đến chương trình bị lỗi do sự lặp lại của cùng một thao tác.



#### Tuyệt! Vì vậy, bây giờ bạn biết nhiệm vụ lặp lại là gì!

#### Hãy tiến thêm một bước nữa.

#### Hãy bắt đầu với vòng lặp FOR.

### **For Loop in C#**

#### Vòng lặp For khá đơn giản để hiểu.

#### Trong hầu hết các trường hợp, chúng ta sử dụng vòng lặp For khi chúng ta biết chính xác tác vụ bên trong vòng lặp phải được lặp lại bao nhiêu lần.

#### Với For loop, trong hầu hết các trường hợp, bạn có thể giữ bình tĩnh và không lo lắng về việc kết thúc với một vòng lặp vô hạn.

#### Tại sao? Hãy nhảy vào một ví dụ để hiểu.

#### Cú pháp vòng lặp FOR trong C# trông như thế này:

for(int i = 0; i < n; i++) {

task to be done n – times

}

#### Hãy chia nhỏ code của chúng ta và xem những gì chúng ta nhận được ở đây:

* for – toán tử mà chúng ta sử dụng để khai báo một vòng lặp FOR.
* int i = 0 là một biến sẽ chỉ tồn tại trong phạm vi của vòng lặp. Chúng ta cần điều này để đếm số lần vòng lặp sẽ lặp lại tác vụ.
* I < n là điều kiện vòng lặp – vòng lặp sẽ tiếp tục lặp lại tác vụ cho đến khi i bằng hoặc cao hơn n. Khi điều kiện i < n trở nên sai - vòng lặp dừng lại.
* i ++ là toán tử gia tăng. Nó làm tăng giá trị của i gần n hơn mỗi vòng lặp thời gian được lặp lại. Nó có thể là gia tăng \ giảm hoặc thậm chí i = i, nếu chúng ta đang thay đổi giá trị của biến i trong vòng lặp.



#### Ngoài ra, chúng ta có thể tạo các vòng lặp bên trong các vòng lặp khác.

#### Có một điều mà bạn cần chú ý và ghi nhớ.

#### Chúng ta không thể khai báo một biến có cùng tên trong vòng lặp thứ hai (bên trong) đã được khai báo trong vòng lặp đầu tiên.

#### Hãy sử dụng một ví dụ để làm rõ:

for(int i = 0; i < 10; i++){

for(int i = 0; i < 5; i++){

}

}

\\(incorrect example)

#### Như bạn đã đoán, mã này sẽ không biên dịch!

#### Đó là bởi vì biến có tên int 'i' đã được khai báo trong vòng lặp đầu tiên và nó không thể được sao chép trong vòng lặp thứ hai một lần nữa.

#### Hãy thay đổi tên và làm cho nó hoạt động!

for(int i = 0; i < 10; i++)

{

for(int j = 0; j < 5; j++){

}

}

\\correct example



#### Vì vậy, bây giờ bạn đã biết những điều cơ bản của vòng lặp FOR và cách thức hoạt động của nó!

#### Hãy nhớ rằng vòng lặp sẽ tiếp tục trong khi điều kiện là đúng. Sau khi điều kiện trở nên sai - vòng lặp ngừng lặp lại nhiệm vụ!

#### Bây giờ chúng ta hãy điều tra vòng lặp WHILE!

### **C# While Loop**

#### Chúng tôi chủ yếu sử dụng vòng lặp WHILE khi chúng tôi không biết chính xác lượng thời gian tác vụ phải được lặp lại.

#### Vòng lặp WHILE rất hữu ích trong những trường hợp đó.

#### Tuy nhiên, chúng ta cần đảm bảo rằng điều kiện vòng lặp được đặt đúng.

#### Nếu không, chúng ta có thể kết thúc với một vòng lặp vô hạn!

#### Hãy xem một ví dụ để xem cú pháp của vòng lặp WHILE:

while(a <= 25)

{

task to be repeated while a reaches value of 25 or higher

}

#### Hãy chia nhỏ code của chúng ta:

* Trong khi - toán tử khai báo vòng lặp.
* (a < = 25) - điều kiện vòng lặp được viết giữa các dấu ngoặc.
* { } - nhiệm vụ cần hoàn thành, giống như chúng ta đã có trong vòng lặp

#### Chúng ta hãy xem một ví dụ để xem nó hoạt động:

int a = 5;

while(a <= 25)

{

a = a + 5;

Console.WriteLine(a);

}

10 15 20 25 30

int a = 5;

while(a <= 25)

{

a = a + 5;

Console.WriteLine(a);

}

#### Chúng ta hãy xem xét kỹ hơn những gì đang diễn ra sau mỗi hoạt động:

* Sau lần lặp đầu tiên, a + 5, a trở thành 10 - điều kiện 10 < = 25 là đúng.
* Sau giây a + 5, a trở thành 15 - điều kiện 15 < = 25 là đúng.
* Sau a + 5 thứ ba, a trở thành 20 - điều kiện 20 < = 25 là đúng
* Sau thứ tư a + 5, a trở thành 25 - điều kiện 25 < = 25 là đúng
* Sau lần thứ năm a + 5, a trở thành 30 - điều kiện 30 < = là sai - vì vậy vòng lặp sẽ không lặp lại tác vụ nữa.



#### Tốt! Tuyệt!

#### Bạn vừa học cách khai báo và sử dụng các vòng lặp để lặp lại các tác vụ!

#### Tạm biệt Copy-Paste!

#### Làm việc thông minh hơn - Không khó hơn!

### **What is Array?**

#### Như chúng ta đã biết - dữ liệu được lưu trữ trong các biến.

#### Nhưng không phải lúc nào chúng ta cũng biết mình sẽ sử dụng bao nhiêu biến.

#### Và đó là nơi chúng ta cần sử dụng mảng.

#### Hãy tưởng tượng một chồng 10 tấm.

#### Chúng ta có thể coi những tấm đó là một mảng có kích thước 10 trong đó mỗi tấm có kích thước cố định.

#### Điều này có nghĩa là tất cả các phần tử của một mảng sẽ thuộc cùng một kiểu dữ liệu.



#### Vì vậy, trong C #, một mảng là một kệ khổng lồ với nhiều biến trong đó.

#### Các biến với các chỉ mục giúp chúng ta điều hướng nó và tìm ngăn kéo chính xác để đặt đồ đạc trong đó.

#### Mỗi phần tử ("ngăn kéo") có chỉ mục riêng và chúng ta có thể đặt dữ liệu của mình bên trong nó bằng cách sử dụng chỉ mục đó.



#### Chúng tôi cũng có mảng 2D (hai chiều) trong C #.

#### Nó có thể khá khó tưởng tượng, nhưng đây là một ví dụ thực tế - bàn cờ.

#### Mỗi ô vuông, bất kể màu đen hay trắng đều có một chỉ mục.



#### Chúng ta đã quen thuộc với mảng hoặc ít nhất chúng ta có thể tưởng tượng mảng là gì.

#### Bây giờ chúng ta có thể thảo luận về nơi chúng được sử dụng!

#### Hãy tiếp tục!

### **Use of Arrays**

#### Mảng được sử dụng trong những tình huống mà chúng ta không biết từ bước nhảy, chúng ta sẽ cần bao nhiêu biến riêng biệt.

#### Đây là một ví dụ: Chúng tôi cần một chương trình để đăng ký tất cả các học sinh đã theo học tại trường mỗi ngày.

#### Lúc đầu, chúng tôi đếm chúng - sau đó chúng tôi biết chúng tôi sẽ cần bao nhiêu phần tử mảng.

#### Và sau đó - điền vào mảng với tên của họ.

#### Nếu không có mảng, nó sẽ vô cùng khó khăn.

#### Hãy tưởng tượng tạo một biến mới cho mỗi học sinh. Tedious!

#### Hãy xem mã liên quan đến tình huống được đề cập ở trên:

class Program

{

static void Main(string[] args)

{

Console.WriteLine("How many attended today: ");

int c = int.Parse(Console.ReadLine());

string[] students = new string[c];

for(int i = 0; i < c; i++)

{

Console.WriteLine("Name of student: ");

students[i] = Console.ReadLine();

}

Console.WriteLine("Attended to school today: ");

for (int i = 0; i < c; i++)

{

Console.WriteLine(students[i]);

}

}

}

#### **ĐẦU VÀO**:

#### Có bao nhiêu người tham dự ngày hôm nay: 3 Tên học sinh: John Doe Tên học sinh: Lisa Doe Tên học sinh: George Doe

Attended school today: John Doe Lisa Doe George Doe

#### Trong mã này, chúng tôi đã yêu cầu người dùng nhập số lượng sinh viên đã tham dự - và sau đó chúng tôi tạo một mảng với 4 phần tử nơi chúng tôi có thể lưu trữ tên của họ.

#### Tại sao lại là 4??? Bởi vì một mảng bắt đầu bằng index zero – 0. Điều đó có nghĩa là nếu chúng ta nhập như trong ví dụ này, chúng ta có 3 sinh viên tham dự - chúng ta tạo ra một mảng với 4 phần tử. Nó không tệ, nhưng chúng ta có thể làm cho nó hiệu quả hơn.

#### Chỉ cần lưu ý rằng lập chỉ mục bắt đầu từ con số không!

#### Vì vậy, mảng cho chúng ta một cơ hội tuyệt vời để làm cho mã của chúng ta hiệu quả hơn và thích ứng hơn nhiều với các tình huống khác nhau!

#### Điều đó thật tuyệt phải không?

#### Hãy cùng tìm hiểu cách khai báo một mảng trong C# nhé!

### **Array Initialization**

#### Việc khai báo mảng hơi khác so với khai báo của các biến.

#### Nhưng không cần phải lo lắng! Chúng ta sắp tìm hiểu nó!

#### Hãy bắt đầu nào!

#### Mảng có thể thuộc bất kỳ loại nào bạn muốn - int, double, char, string, v.v.

#### Khai báo mảng trong mã trông như thế này:

int elements = 100; // for example - it can be any positive value that is more than zero

int[] numbers = new int[elements];

#### Dưới đây là tóm tắt về mã ở trên:

* int elements – Để khởi tạo một mảng, chúng ta cần cho máy tính biết chúng ta sẽ sử dụng bao nhiêu phần tử. Ngoài ra, nó phải là một giá trị nguyên!
* int[] – toán tử khởi tạo được sử dụng để khai báo một mảng – về cơ bản với int, chúng ta đang nói rằng nó sẽ lưu trữ các số nguyên và với các dấu ngoặc vuông đó chỉ ra rằng đó là một mảng!
* Số – tên của mảng.
* new int[elements] – ở đây, chúng ta đang bảo lưu số lượng phần tử mà chúng ta cần! Với dấu ngoặc vuông, chúng ta đang nói số tiền chúng ta cần – nó có thể là một biến int hoặc một giá trị số nguyên dương (cao hơn không).

#### Đây rồi! Một mảng số nguyên!

#### Dưới đây là một ví dụ về cách khai báo một mảng để lưu trữ số thập phân:

int elements = 100; // for example - it can be any positive value that is more than zero

double[] numbers = new double[elements];

#### Ở đây, chúng ta chỉ cần thay đổi kiểu mảng và các phần tử dành riêng có thể chứa các giá trị thập phân.

#### Để nói ít nhất - một mảng là một biến với các chỉ mục!

#### Không có nhiều thách thức ở đây!

#### Tuyệt vời! Bây giờ bạn đã biết cách khai báo một mảng!

#### Hãy tiến lên một bước và học cách sử dụng nó!

#### Đi! Đi! Đi!

### **Accessing Array Elements**

#### Trong việc truy cập các yếu tố của một mảng, không có nhiều điều mới để ghi nhớ!

#### Tin tốt?

#### Được rồi, chúng ta hãy xem chúng ta có gì ở đây!

#### Khi chúng ta có một mảng được khai báo và chúng ta muốn gán một giá trị cho một trong các phần tử, chúng ta phải viết:

* Đó là tên
* Chỉ mục của phần tử chúng ta muốn gán giá trị cho
* Và tất nhiên, giá trị chính nó sau dấu bằng

static void Main(string[] args)

{

int elements = 5;

double[] numbers = new double[elements];

numbers[0] = 10;

numbers[1] = 25;

numbers[2] = 27;

for (int i = 0; i < elements; i++)

{

Console.WriteLine(numbers[i]);

}

}

10 25 27

#### Đơn giản vậy thôi!

#### Chúng ta hãy xem xét một ví dụ khác – nơi chúng ta sẽ gán các phần tử cho một mảng chuỗi.

static void Main(string[] args)

{

int elements = 3;

string[] words = new string[elements];

words[0] = "Have";

words[1] = "some";

words[2] = "Coffee";

for (int i = 0; i < elements; i++)

{

Console.WriteLine(words[i]);

}

}

Have Some Coffee

#### Đã nói với bạn! Không có nhiều sự khác biệt xung quanh đây nếu chúng ta so sánh nó với các biến!

#### Làm thế nào để gán một giá trị từ một mảng? Hãy cùng xem!

static void Main(string[] args)

{

int elements = 3;

string[] words = new string[elements];

words[0] = "Have";

words[1] = "some";

words[2] = "Coffee";

string WORD = words[2];

Console.WriteLine(WORD);

}

Coffee

#### Để phá vỡ nó, trước tiên, chúng ta đã tạo một biến và chúng ta đã gán giá trị phần tử thứ ba cho một biến có tên WORD từ mảng – words.



#### Tuyệt! Vì vậy, bây giờ bạn đã sẵn sàng để sử dụng mảng trong C #!

#### Tốt!

#### Mảng sẽ giúp bạn tối ưu hóa mã của mình và giải quyết các vấn đề khó khăn khác nhau.

#### Chúc may mắn với họ!

#### Hãy chuyển sang cái tiếp theo.

### **What is a Function?**

#### Ê! Thật tuyệt khi thấy rằng bạn đã đi xa đến mức này! Chúng tôi sắp đề cập đến một trong những khái niệm quan trọng và hữu ích nhất về C # và lập trình nói chung.

#### CÁC CHỨC NĂNG!

#### Chà, để giải thích khái niệm này với một ví dụ thực tế, hãy tưởng tượng nó như một công cụ!



#### Trong cuộc sống thực, chúng ta tự mua hoặc chế tạo những công cụ đó. Điều này cũng áp dụng cho thế giới lập trình nhưng trong hầu hết các trường hợp, chính chúng tôi là người tạo ra những công cụ đó.

#### Hãy tưởng tượng nhiệm vụ xây dựng 5 bảng. Bạn có các công cụ cần thiết và bạn biết các bước để hoàn thành công việc!

#### Bạn đang sử dụng các công cụ tương tự để tạo tất cả 5 bảng này.

#### Vì vậy, tại sao bạn không làm điều tương tự trong lập trình với C #!

#### Trong các ngôn ngữ lập trình hướng đối tượng như C #, chúng ta cũng có thể tạo ra các công cụ có thể được sử dụng nhiều lần!

#### Các "công cụ" mà chúng tôi sử dụng trong ngữ cảnh được gọi là các hàm trong C #.

#### Chúng ta hãy xem một ví dụ cơ bản của các hàm.

class Program

{

static void Main(string[] args)

{

int c = -5;

int x = 2;

int y = 8;

while(c <= 250)

{

c = c + DoMath(x, y);

for(int i = 5; i < 8; i++)

{

c = c + DoMath(x, y);

}

}

}

static int DoMath(int a, int b) \\ function that does some kind of operation

{

int variable = 52;

int variable1 = a + b + variable;

variable1 = variable1 \* 2;

return variable;

}

}

#### Đó là một ví dụ hiệu quả - bởi vì chúng tôi đã tạo ra hàm cung cấp cho chúng tôi một số loại giá trị và nó được sử dụng hai lần.

#### Điều này cũng tiết kiệm rất nhiều thời gian.

#### Hãy xem một phiên bản không hiệu quả của điều đó:

class Program

{

static void Main(string[] args)

{

int c = -5;

int x = 2;

int y = 8;

while(c <= 250)

{

c = c + x + y;

c = c + 52;

c = c \* 2;

for(int i = 5; i < 8; i++)

{

c = c + x + y;

c = c + 52;

c = c \* 2;

}

}

}

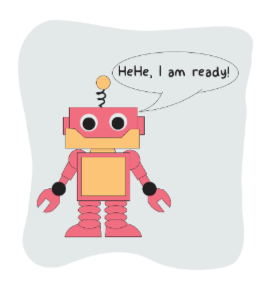
}

#### Nó thực hiện hoạt động tương tự.

#### Tuy nhiên, mọi hoạt động phải được viết riêng và nếu giá trị của, giả sử biến c, phải được thay đổi thì sẽ cần phải làm lại rất nhiều trong mã.

#### Tại sao nó xấu? Bởi vì nó không hiệu quả, và nếu bạn đang tham gia vào một dự án lớn - biết các chức năng là cần thiết và cực kỳ quan trọng.

#### Nó sẽ giúp bạn tiết kiệm rất nhiều thời gian trong khi vẫn giữ cho mã của bạn sạch sẽ!



#### Vì vậy, bây giờ bạn đã biết tất cả các chức năng là gì!

#### Bây giờ, đã đến lúc đi sâu hơn một chút!

### **Types of Functions**

#### Phần quan trọng nhất của chức năng là loại của chúng.

#### Trong C# chúng ta có hai loại hàm:

* Những thứ trả về giá trị thông qua các biến mà chúng ta đang cung cấp cho chúng. Các hàm void chỉ trả về giá trị của các biến bằng cách sử dụng từ khóa "out"
* Những người trả về một giá trị thông qua tên của họ. int \ double \ char \ chuỗi và v.v.

#### Những cái trả về một giá trị thông qua các biến khá tiện dụng khi chúng ta muốn trả về nhiều hơn một biến.

class Program

{

static void Main(string[] args)

{

int c = -5;

int x = 2;

int y = 8;

while(c <= 250)

{

DoMath(out x, out y);

c = c + x + y;

}

}

static void DoMath(out int a, out int b) \\ function that does some kind of operation

{

a = 15;

b = 3;

}

}

#### Dưới đây là một ví dụ về một hàm truyền các biến x và y đến hàm DoMath nơi chúng nhận được các giá trị mới 15 và 3.

#### Và chúng được trả về thông qua các tham số (biến) của chúng được truyền từ mã chính.

#### Cả hai đều có từ khóa ở phía trước các biến - ra.

#### Loại hàm này có thể trả về nhiều biến với các giá trị mới hơn.

#### Các hàm trả về một giá trị duy nhất thông qua tên của chúng trông như thế này:

class Program

{

static void Main(string[] args)

{

int c = -5;

int x = 2;

int y = 8;

while(c <= 250)

{

c = c + DoMath(x, y);

for(int i = 5; i < 8; i++)

{

c = c + DoMath(x, y);

}

}

}

static int DoMath(int a, int b) \\ function that does some kind of operation

{

int variable = 52;

int variable1 = a + b + variable;

variable1 = variable1 \* 2;

return variable;

}

}

#### Như tên gọi của nó, toán tử trả về được sử dụng để trả về một giá trị duy nhất.

#### Toán tử trả về là điểm kết thúc cho một hàm, do đó, ngay cả khi có một số mã sau toán tử trả về, nó sẽ không được thực thi.

#### Toán tử trả về trả về giá trị ước tính của biến mà trong trường hợp này là biến.

#### Tóm lại, chúng ta có hai loại hàm.

#### void – có thể trả về nhiều giá trị hoặc không trả về bất cứ thứ gì cả.

#### int\double\char\string – những giá trị trả về một giá trị duy nhất thông qua tên của chúng.

#### Đó là nó! Các hàm hoạt động như thế nào? Chúng tôi sắp tìm ra điều đó.

### **Use of Functions**

#### Hãy bắt đầu với các hàm trả về một giá trị thông qua tên của chúng.

#### Ví dụ: hãy lấy hàm này:

static int DoMath(int a, int b)

{

return a + b;

}

* static int – là một loại hàm. static có nghĩa là một hàm có thể được gọi từ một lớp cục bộ mà không cần toán tử của nó. int là kiểu, cho biết loại biến mà hàm này trả về. Bạn có thể mất một chút thời gian để tìm hiểu về lớp học như một khái niệm. Nó được đề cập trong khóa học Advanced C #.
* DoMath – tên của hàm. Cùng một lớp không thể chứa tên trùng lặp. Cũng giống như với các biến.
* (int a, int b) – là tham số hàm. Đây chỉ là các biến được khai báo trong nguyên mẫu hàm. Các giá trị đó phải được chuyển đến mã mà chúng tôi hoạt động của hàm được viết. Nó giống như một đầu vào cho hàm.
* Trả về a + b là toán tử hàm trả về một giá trị mà trong trường hợp này là a + b và kết thúc công việc của hàm.

class Program

{

static void Main(string[] args)

{

int y = 8;

int c = DoMath(x, y);

Console.WriteLine(c);

}

static int DoMath(int a, int b) \\ function that does some kind of operation

{

return a+b;

}

}

#### Vậy là ta có một class Program với hàm chính Main và các biến x và y khai báo.

#### Chúng ta đang khai báo biến c sẽ chứa giá trị mà hàm DoMath sẽ trả về.

#### Khi chúng ta gọi hàm DoMath(x,y), chúng ta đang cho hàm hai biến x và y mà trong hàm sẽ được sử dụng với các tên khác nhau – a và b.

#### Hàm DoMath trả về 2 + 8 sẽ là 10. Vì vậy, đầu ra của chương trình sẽ là 10!



#### Bây giờ là lúc cho các hàm có thể trả về giá trị thông qua các tham số của chúng.

#### Ưu điểm lớn nhất là có thể trả về nhiều giá trị có thể thuộc các loại khác nhau - chuỗi, char, double hoặc int.

#### Chúng ta hãy xem một ví dụ nơi nó được sử dụng!

class Program

{

static void Main(string[] args)

{

int x = 2;

string y = null; \\ null means it does not contain any value

DoMath(out x,out y);

Console.WriteLine("I have " + x + " " + y);

}

static void DoMath(out int a, out string b)

{

int v = 15;

a = v + 10;

b = "Friends";

}

}

I have 25 Friends

#### Những gì chúng ta thấy ở đây là:

* Khoảng trống tĩnh – một loại chức năng.
* DoMath – tên của hàm
* (out int a, out string b) – là một tham số hàm với keyword out – có nghĩa là các giá trị này sẽ được thay đổi khi trả về.

#### Hãy xem một ví dụ khác!

class Program

{

static void Main(string[] args)

{

int x = 2;

string y = null;

DoMath(out x,out y);

Console.WriteLine("I have " + x + " " + y);

}

static void DoMath(out int a, out string b)

{

int v = 15;

a = v + 10;

b = "Friends";

}

}

I have 25 Friends

#### Hãy xem chú thích hàm trong hàm Main.

#### Chúng ta đang cho nó biến x và một biến trống y của chuỗi kiểu.

#### Trong hàm, x trở thành 15 + 10 vì vậy nó là 25 và b nhận được giá trị là - "Bạn bè". Và tất cả những điều đó được kết hợp bên trong đầu ra để đưa toàn bộ câu lên màn hình!

#### Đó là nó!



#### Tốt! Chúng tôi vừa hoàn thành bài học về các chức năng và cách chúng hoạt động!

#### Tuyệt vời!

#### Chúng tôi đã có rất nhiều thông tin mới, vì vậy hãy kiểm tra những gì bạn hiểu, được chứ?

#### Hãy lăn!

### **Let's sum up**

#### Câu hỏi khó, phải không?

#### Dù sao thì, bạn đã xử lý nó vừa phải! Và bạn nên tự hào về bản thân!

#### Các hàm có thể có chú thích bên trong cho các chức năng khác - nghe có vẻ hơi khó hiểu, nhưng bạn sẽ thấy nó tiện dụng như thế nào trong tương lai!



#### Làm rất tốt! Đó là nơi chúng ta sẽ kết thúc chủ đề của chúng ta về các chức năng!

#### Bạn vừa bước vào thế giới lập trình hướng đối tượng!

#### Chúc may mắn!