Contents

A.C++ TUTORIAL

myfirstprogram.cpp

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hello World!";
  return 0;
}
```

Omitting Namespace(Bỏ qua không gian tên
)

You might see some C++ programs that runs without the standard namespace library. The using namespace std line can be omitted and replaced with the std keyword, followed by the :: operator for some objects:

Bạn có thể thấy một số chương trình C ++ chạy mà không có thư viện không gian tên tiêu chuẩn. Các using namespace stddòng có thể được bỏ qua và thay thế bằng các stdtừ khóa, tiếp theo là các ::nhà điều hành cho một số đối tượng:

```
#include <iostream>
int main() {
  std::cout << "Hello World!";
  return 0;
}
```

The cout object is used together with the << operator to display variables.

To combine both text and a variable, separate them with the << operator:

Đối couttượng được sử dụng cùng với << toán tử để hiển thị các biến.

Để kết hợp cả văn bản và một biến, hãy tách chúng với << toán tử:

C++ Variable

```
char a = 65, b = 66, c = 67;//output :A,B,C
string greeting = "Hello";
int myAge = 35;
cout << "I am " << myAge << " years old.";
```

When you do not want others (or yourself) to override existing variable values, use the const keyword (this will declare the variable as "constant", which means **unchangeable and read-only**):

Khi bạn không muốn người khác (hoặc chính mình) ghi đè các giá trị biến hiện có, hãy sử dụng consttừ khóa (điều này sẽ khai báo biến là "hằng", có nghĩa là **không thể thay đổi và chỉ đọc**):

```
const int myNum = 15;  // myNum will always be 15
myNum = 10;  // error: assignment of read-only variable 'myNum'
```

C++ User Input

```
#include <iostream>
using namespace std;
int main() {
  int x, y;
  int sum;
  cout << "Type a number: ";cin >> x;
  cout << "Type another number: "; cin >> y;
  sum = x + y; cout << "Sum is: " << sum; return 0;
}
```

```
Type a number:  1,2
Type another number: 3
Sum is: 3
```

C++ Boolean Data Types

A boolean data type is declared with the bool keyword and can only take the values true or false. When the value is returned, true = 1 and false = 0.

Một kiểu dữ liệu boolean được khai báo bằng booltừ khóa và chỉ có thể lấy các giá trị truehoặc false. Khi giá trị được trả về, true = 1và false = 0.

```
#include <iostream>
using namespace std;
int main() {
  bool isCodingFun = true;
  bool isFishTasty = false;
  cout << isCodingFun << "\n";
  cout << isFishTasty;
```

```
  return 0;
}//output : 1 0
```

C++ Strings

To use strings, you must include an additional header file in the source code, the <string> library:
Để sử dụng chuỗi, bạn phải bao gồm một tệp tiêu đề bổ sung trong mã nguồn, <string>thư viện:

```
// Include the string library
#include <string>
// Create a string variable
string greeting = "Hello";
```

String Concatenation **Nối chuỗi** C ++

The + operator can be used between strings to add them together to make a new string. This is called **concatenation**:
In the example above, we added a space after firstName to create a space between John and Doe on output. However, you could also add a space with quotes (" " or ' '):
Các +nhà điều hành có thể được sử dụng giữa các chuỗi để thêm chúng lại với nhau để tạo ra một chuỗi mới. Điều này được gọi là **nối** : Trong ví dụ trên, chúng tôi đã thêm một khoảng trắng sau FirstName để tạo khoảng trắng giữa John và Doe trên đầu ra. Tuy nhiên, bạn cũng có thể thêm khoảng trắng bằng dấu ngoặc kép (" "hoặc ' '):

```
string firstName = "John";
string lastName = "Doe";
string fullName = firstName + " " + lastName;
cout << fullName;
```

Append

A string in C++ is actually an object, which contain functions that can perform certain operations on strings. For example, you can also concatenate strings with the append() function:

```
string fullName = firstName.append(lastName);
```

Adding Numbers and Strings

WARNING!

C++ uses the + operator for both **addition** and **concatenation**.
Numbers are added. Strings are concatenated.
CẢNH BÁO!
C ++ sử dụng +toán tử cho cả phép **cộng** và **phép nối** .
Số được thêm vào. Các chuỗi được nối.
If you add two numbers, the result will be a number:
If you add two strings, the result will be a string concatenation:
If you try to add a number to a string, an error occurs:

```
int x = 10;
int y = 20;
int z = x + y;      // z will be 30 (an integer)
```

```
string x = "10";
string y = "20";
string z = x + y;   // z will be 1020 (a string)
```

```
string x = "10";
int y = 20;
string z = x + y;=>error
```

C++ String Length

To get the length of a string, use the length() function:

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is: " << txt.length();
```

Tip: You might see some C++ programs that use the size() function to get the length of a string. This is just an alias of length(). It is completely up to you if you want to use length() or size():

Mẹo: Bạn có thể thấy một số chương trình C ++ sử dụng size()hàm để lấy độ dài của chuỗi. Đây chỉ là một bí danh của length(). Nó hoàn toàn phụ thuộc vào bạn nếu bạn muốn sử dụng length()hoặc size():

Access Strings

You can access the characters in a string by referring to its index number inside square brackets []. Bạn có thể truy cập các ký tự trong một chuỗi bằng cách tham khảo số chỉ mục của nó trong dấu ngoặc vuông [].

```
string myString = "Hello";
cout << myString[0];
// Outputs H
```

User Input Strings Chuỗi nhập của người dùng

It is possible to use the extraction operator >> on cin to display a string entered by a user: However, cin considers a space (whitespace, tabs, etc) as a terminating character, which means that it can only display a single word (even if you type many words):

Có thể sử dụng các nhà điều hành khai thác >>trên cinđể hiển thị một chuỗi nhập vào bởi người dùng: Tuy nhiên, cincoi một khoảng trắng (khoảng trắng, tab, v.v.) là ký tự kết thúc, có nghĩa là nó chỉ có thể hiển thị một từ duy nhất (ngay cả khi bạn nhập nhiều từ):

```
string fullName;
cout << "Type your full name: ";
cin >> fullName;
cout << "Your name is: " << fullName;
// Type your full name: John Doe// Your name is: John
```

From the example above, you would expect the program to print "John Doe", but it only prints "John".

That's why, when working with strings, we often use the getline() function to read a line of text. It takes cin as the first parameter, and the string variable as second:

Từ ví dụ trên, bạn sẽ mong đợi chương trình in "John Doe", nhưng nó chỉ in "John". Đó là lý do tại sao, khi làm việc với các chuỗi, chúng ta thường sử dụng getline() hàm để đọc một dòng văn bản. Nó nhận cinlàm tham số đầu tiên và biến chuỗi là thứ hai:

```
string fullName;
cout << "Type your full name: ";
getline (cin, fullName);
cout << "Your name is: " << fullName;
```

C++ Math

C++ has many functions that allows you to perform mathematical tasks on numbers.
The max(*x*,*y*) function can be used to find the highest value of *x* and *y*:

```
cout << max(5, 10);
```

Other functions, such as sqrt (square root), round (rounds a number) and log (natural logarithm), can be found in the <cmath> header file:

```
// Include the cmath library
#include <cmath>
cout << sqrt(64);
cout << round(2.6);
cout << log(2);
```

C++ Booleans

A boolean variable is declared with the bool keyword and can only take the values true or false:

```
bool isCodingFun = true;
bool isFishTasty = false;
cout << isCodingFun;  // Outputs 1 (true)
cout << isFishTasty;  // Outputs 0 (false)
```

From the example above, you can read that a true value returns 1, and false returns 0.
However, it is more common to return boolean values from boolean expressions (see next page).
Từ ví dụ trên, bạn có thể đọc rằng truegiá trị trả về 1và falsetrả về 0.
Tuy nhiên, thông thường hơn là trả về các giá trị boolean từ các biểu thức boolean (xem trang tiếp theo).

Boolean Expression Biểu thức Boolean

A **Boolean expression** is a C++ expression that returns a boolean value: 1 (true) or 0 (false).
You can use a comparison operator, such as the **greater than** (>)(<,==) operator to find out if an expression (or a variable) is true:
Một **biểu thức Boolean** là một ++ biểu C mà trả về một giá trị boolean: 1(true) hoặc 0(false).
Bạn có thể sử dụng toán tử so sánh, chẳng hạn như toán tử **lớn hơn** (>) để tìm hiểu xem một biểu thức (hoặc một biến) có đúng không:

```
int x = 10;
int y = 9;
cout << (x > y); // returns 1 (true), because 10 is higher than 9
```

Or even easier: cout << (10 > 9); // returns 1 (true), because 10 is higher than 9

C++ Conditions

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

--

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

--

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

--

Short Hand If...Else (Ternary Operator)

```
variable = (condition) ? expressionTrue : expressionFalse;
```

C++ Switch

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

C++ Loop

```
while (condition) {
  // code block to be executed
}
```

```
do {
  // code block to be executed
```

```
}
while (condition);

for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

C++ Break and Continue

C++ Break

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a **loop**.

Bạn đã thấy breaktuyên bố được sử dụng trong một chương trước của hướng dẫn này. Nó được sử dụng để "nhảy ra" một switchtuyên bố.

Câu breaklệnh cũng có thể được sử dụng để nhảy ra khỏi **vòng lặp** .

C++ Continue

Câu continuelệnh phá vỡ một lần lặp (trong vòng lặp), nếu một điều kiện được chỉ định xảy ra và tiếp tục với lần lặp tiếp theo trong vòng lặp.

C++ Arrays

To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store:

Để khai báo một mảng, xác định loại biến, chỉ định tên của mảng theo **dấu ngoặc vuông** và chỉ định số phần tử cần lưu trữ:

```
string cars[4];
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
int myNum[3] = {10, 20, 30};
```

Omit Array Size- Kích thước mảng bỏ qua

You don't have to specify the size of the array. But if you don't, it will only be as big as the elements that are inserted into it:

Bạn không phải xác định kích thước của mảng. Nhưng nếu bạn không, nó sẽ chỉ lớn bằng các yếu tố được chèn vào nó:

```
string cars[] = {"Volvo", "BMW", "Ford"}; // size of array is always 3
```

This is completely fine. However, the problem arise if you want extra space for future elements. Then you have to overwrite the existing values:

Điều này là hoàn toàn tốt. Tuy nhiên, vấn đề phát sinh nếu bạn muốn có thêm không gian cho các yếu tố trong tương lai. Sau đó, bạn phải ghi đè lên các giá trị hiện có:

```
~~string cars[] = {"Volvo", "BMW", "Ford"};~~
string cars[] = {"Volvo", "BMW", "Ford", "Mazda", "Tesla"};
```

If you specify the size however, the array will reserve the extra space:

Tuy nhiên, nếu bạn chỉ định kích thước, mảng sẽ dành thêm không gian:

C++ References- Tạo tài liệu tham khảo

A reference variable is a "reference" to an existing variable, and it is created with the & operator:
Biến tham chiếu là "tham chiếu" cho biến hiện có và được tạo bằng &toán tử:

```
string food = "Pizza";  // food variable
string &meal = food;    // reference to food
```

Now, we can use either the variable name food or the reference name meal to refer to the food variable:

```
string food = "Pizza";
string &meal = food;

cout << food << "\n";  // Outputs Pizza
cout << meal << "\n";  // Outputs Pizza
```

Memory Address- Địa chỉ bộ nhớ

In the example from the previous page, the & operator was used to create a reference variable. But it can also be used to get the memory address of a variable; which is the location of where the variable is stored on the computer.
When a variable is created in C++, a memory address is assigned to the variable. And when we assign a value to the variable, it is stored in this memory address.
To access it, use the & operator, and the result will represent where the variable is stored:
Trong ví dụ từ trang trước, &toán tử đã được sử dụng để tạo biến tham chiếu. Nhưng nó cũng có thể được sử dụng để lấy địa chỉ bộ nhớ của một biến; đó là vị trí nơi lưu trữ biến trên máy tính.
Khi một biến được tạo trong C ++, một địa chỉ bộ nhớ được gán cho biến đó. Và khi chúng ta gán một giá trị cho biến, nó được lưu trong địa chỉ bộ nhớ này.
Để truy cập nó, sử dụng & toán tử và kết quả sẽ biểu thị nơi lưu trữ biến:

```
string food = "Pizza";

cout << &food; // Outputs 0x6dfed4
```

Note: The memory address is in hexadecimal form (0x..). Note that you may not get the same result in your program.
And why is it useful to know the memory address?
References and **Pointers** (which you will learn about in the next chapter) are important in C++, because they give you the ability to manipulate the data in the computer's memory - **which can reduce the code and improve the perfomance**.
These two features are one of the things that make C++ stand out from other programming languages, like Python and Java.
Và tại sao nó hữu ích để biết địa chỉ bộ nhớ?
Tài liệu tham khảo và **con trỏ** (mà bạn sẽ tìm hiểu trong chương tiếp theo) rất quan trọng trong C ++, vì chúng cho bạn khả năng thao tác dữ liệu trong bộ nhớ của máy tính - **có thể giảm mã và cải thiện độ hoàn hảo** .

Hai tính năng này là một trong những điều làm cho C ++ nổi bật so với các ngôn ngữ lập trình khác, như Python và Java

C++ Pointers

Creating Pointers

You learned from the previous chapter, that we can get the **memory address** of a variable by using the & operator:

```
string food = "Pizza"; // A food variable of type string
cout << food;  // Outputs the value of food (Pizza)
cout << &food; // Outputs the memory address of food (0x6dfed4)
```

A **pointer** however, is a variable that **stores the memory address as its value**.

A pointer variable points to a data type (like int or string) of the same type, and is created with the * operator. The address of the variable you're working with is assigned to the pointer:

Một **con trỏ** tuy nhiên, là một biến **lưu trữ địa chỉ bộ nhớ như giá trị của nó** .

Một biến con trỏ trỏ đến một loại dữ liệu (như inthoặc string) cùng loại và được tạo với *toán tử. Địa chỉ của biến bạn đang làm việc được gán cho con trỏ:

```
string food = "Pizza";  // A food variable of type string
string* ptr = &food;    // A pointer variable, with the name ptr, that stores the address of food
// Output the value of food (Pizza)
cout << food << "\n";

// Output the memory address of food (0x6dfed4)
cout << &food << "\n";
// Output the memory address of food with the pointer (0x6dfed4)
cout << ptr << "\n";
```

Tip: There are three ways to declare pointer variables, but the first way is preferred:

```
string* mystring; // Preferred
string *mystring;    string * mystring;
```

C++ **Dereference -** toán tử gián tiếp

Get Memory Address and Value - Nhận địa chỉ và giá trị bộ nhớ

In the example from the previous page, we used the pointer variable to get the memory address of a variable (used together with the & **reference** operator). However, you can also use the pointer to get the value of the variable, by using the * operator (the **dereference** operator):

Trong ví dụ từ trang trước, chúng tôi đã sử dụng biến con trỏ để lấy địa chỉ bộ nhớ của một biến (được sử dụng cùng với toán tử & **tham chiếu**). Tuy nhiên, bạn cũng có thể sử dụng con trỏ để lấy giá trị của biến, bằng cách sử dụng *toán tử (toán tử **dereference**):

```
string food = "Pizza";  // Variable declaration
string* ptr = &food;    // Pointer declaration
// Reference: Output the memory address of food with the pointer (0x6dfed4)
cout << ptr << "\n";
```

```
// Dereference: Output the value of food with the pointer (Pizza)
cout << *ptr << "\n";
```

Note that the * sign can be confusing here, as it does two different things in our code:
When used in declaration (string* ptr), it creates a pointer variable.
When not used in declaration, it act as a dereference operator.
Lưu ý rằng *dấu hiệu có thể gây nhầm lẫn ở đây, vì nó có hai điều khác nhau trong mã của chúng tôi:
Khi được sử dụng trong khai báo (chuỗi * ptr), nó tạo ra một biến con trỏ .
Khi không được sử dụng trong khai báo, nó hoạt động như một toán tử dereference .

Modify Pointers- Con trỏ sửa đổi

Modify the Pointer Value
You can also change the pointer's value. But note that this will also change the value of the original variable:
Bạn cũng có thể thay đổi giá trị của con trỏ. Nhưng lưu ý rằng điều này cũng sẽ thay đổi giá trị của biến ban đầu:

```
string food = "Pizza";
string* ptr = &food;
// Output the value of food (Pizza)
cout << food << "\n";
// Output the memory address of food (0x6dfed4)
cout << &food << "\n";
// Access the memory address of food and output its value (Pizza)
cout << *ptr << "\n";
// Change the value of the pointer
*ptr = "Hamburger";
// Output the new value of the pointer (Hamburger)
cout << *ptr << "\n";
// Output the new value of the food variable (Hamburger)
cout << food << "\n";
```

B. C++ FUNTION

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.
Hàm là một khối mã chỉ chạy khi được gọi.
Bạn có thể truyền dữ liệu, được gọi là tham số, vào một hàm.

Các hàm được sử dụng để thực hiện một số hành động nhất định và chúng rất quan trọng để sử dụng lại mã: Xác định mã một lần và sử dụng nhiều lần.
Create a Function

```
void myFunction() {
  // code to be executed
}
```

Call a Function

```
// Create a function
void myFunction() {
  cout << "I just got executed!";
}

int main() {
  myFunction(); // call the function
  return 0;
}

// Outputs "I just got executed!"
```

A function can be called multiple times:
Function Declaration and Definition
Khai báo và định nghĩa hàm
A C++ function consist of two parts:

- **Declaration:** the function's name, return type, and parameters (if any)
- **Definition:** the body of the function (code to be executed)

Hàm C ++ bao gồm hai phần:

- **Khai báo:** tên, kiểu trả về và tham số (nếu có) của hàm
- **Định nghĩa**: phần thân của hàm (mã được thực thi)

```
void myFunction() { // declaration
  // the body of the function (definition)
}
```

Note: If a user-defined function, such as myFunction() is declared after the main() function, **an error will occur**. It is because C++ works from top to bottom; which means that if the function is not declared above main(), the program is unaware of it:
Lưu ý: Nếu một hàm do người dùng định nghĩa, chẳng hạn như myFunction()được khai báo sau main()hàm, **sẽ xảy ra lỗi** . Đó là bởi vì C ++ hoạt động từ trên xuống dưới; có nghĩa là nếu hàm không được khai báo ở trên main(), chương trình không biết về nó:
However, it is possible to separate the declaration and the definition of the function - for code optimization.
You will often see C++ programs that have function declaration above main(), and function definition below main(). This will make the code better organized and easier to read:

Tuy nhiên, có thể tách biệt khai báo và định nghĩa của hàm - để tối ưu hóa mã.
Bạn sẽ thường thấy các chương trình C ++ có khai báo hàm ở trên main()và định nghĩa hàm bên dưới main(). Điều này sẽ làm cho mã được tổ chức tốt hơn và dễ đọc hơn:

```
// Function declaration
void myFunction();

// The main method
int main() {
  myFunction();  // call the function
  return 0;
}

// Function definition
void myFunction() {
  cout << "I just got executed!";
}
```

C++ Function Parameters

Thông số chức năng C ++
Parameters and Arguments
Information can be passed to functions as a parameter. Parameters act as variables inside the function.
Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:
Thông tin có thể được truyền cho các chức năng như một tham số. Các tham số đóng vai trò là các biến bên trong hàm.
Các tham số được chỉ định sau tên hàm, bên trong dấu ngoặc đơn. Bạn có thể thêm bao nhiêu tham số tùy ý, chỉ cần tách chúng bằng dấu phẩy:

```
void functionName(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

The following example has a function that takes a string called **fname** as parameter. When the function is called, we pass along a first name, which is used inside the function to print the full name:
Ví dụ sau có một chức năng mà phải mất một stringgọi là **fname** là tham số. Khi hàm được gọi, chúng ta chuyển một tên đầu tiên, được sử dụng bên trong hàm để in tên đầy đủ:

```
void myFunction(string fname) {
  cout << fname << " Refsnes\n";
```

```
}

int main() {
  myFunction("Liam");
  myFunction("Jenny");
  myFunction("Anja");
  return 0;
}

// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

When a **parameter** is passed to the function, it is called an **argument**. So, from the example above: fname is a **parameter**, while Liam, Jenny and Anja are **arguments**.

Khi một **tham số** được truyền cho hàm, nó được gọi là **đối số** . Vì vậy, từ ví dụ trên: fnamelà một **tham số** , trong khi Liam, Jennyvà Anjalà **đối số** .

C++ Default Parameters

Thông số mặc định của C ++

Default Parameter Value

Giá trị tham số mặc định

You can also use a default parameter value, by using the equals sign (=).

If we call the function without an argument, it uses the default value ("Norway"):

Bạn cũng có thể sử dụng một giá trị tham số mặc định, bằng cách sử dụng dấu bằng (=).

Nếu chúng ta gọi hàm mà không có đối số, nó sẽ sử dụng giá trị mặc định ("Na Uy"):

```
void myFunction(string country = "Norway") {
  cout << country << "\n";
}

int main() {
  myFunction("Sweden");
  myFunction("India");
  myFunction();
  myFunction("USA");
  return 0;
}

// Sweden
// India
// Norway
// USA
```

A parameter with a default value, is often known as an "**optional parameter**". From the example above, country is an optional parameter and "Norway" is the default value.
Một tham số có giá trị mặc định, thường được gọi là " **tham số tùy chọn** ". Từ ví dụ trên, countrylà một tham số tùy chọn và "Norway"là giá trị mặc định.

C++ Multiple Parameters
C ++ Nhiều tham số
Multiple Parameters
Inside the function, you can add as many parameters as you want:

```
void myFunction(string fname, int age) {
  cout << fname << " Refsnes. " << age << " years old. \n";
}

int main() {
  myFunction("Liam", 3);
  myFunction("Jenny", 14);
  myFunction("Anja", 30);
  return 0;
}

// Liam Refsnes. 3 years old.
// Jenny Refsnes. 14 years old.
// Anja Refsnes. 30 years old.
```

Note that when you are working with multiple parameters, the function call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.
Lưu ý rằng khi bạn đang làm việc với nhiều tham số, lệnh gọi hàm phải có cùng số lượng đối số như có các tham số và các đối số phải được truyền theo cùng một thứ tự.

C++ The Return Keyword
Return Values
The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int, string, etc.) instead of void, and use the return keyword inside the function:
Các voidtừ khóa, sử dụng trong các ví dụ trước đó, cho thấy rằng các chức năng không nên trả về một giá trị. Nếu bạn muốn các chức năng để trả về một giá trị, bạn có thể sử dụng một kiểu dữ liệu (chẳng hạn như int, string, vv) thay vì void, và sử dụng các return từ khóa bên trong hàm:

```
int myFunction(int x) {
  return 5 + x;
}

int main() {
  cout << myFunction(3);
```

```
  return 0;
}

// Outputs 8 (5 + 3)
```

vd2:

```
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  int z = myFunction(5, 3);
  cout << z;
  return 0;
}
// Outputs 8 (5 + 3)
```

C++ Functions - Pass By Reference

Hàm C ++ - Truyền qua tham chiếu

Pass By Reference

In the examples from the previous page, we used normal variables when we passed parameters to a function. You can also pass a reference to the function. This can be useful when you need to change the value of the arguments:

Trong các ví dụ từ trang trước, chúng tôi đã sử dụng các biến thông thường khi chúng tôi truyền tham số cho hàm. Bạn cũng có thể chuyển một tham chiếu đến hàm. Điều này có thể hữu ích khi bạn cần thay đổi giá trị của các đối số:

```
void swapNums(int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}

int main() {
  int firstNum = 10;
  int secondNum = 20;

  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  // Call the function, which will change the values of firstNum and secondNum
  swapNums(firstNum, secondNum);

  cout << "After swap: " << "\n";
```

```
  cout << firstNum << secondNum << "\n";

  return 0;
}
```