

# CHUYÊN ĐỀ 1: LÀM QUEN VỚI NGÔN NGỮ LẬP TRÌNH C++

## 1. Hướng dẫn cài đặt

Cài đặt IDE để biên dịch và thực thi C

Có một số IDE có sẵn và miễn phí để biên dịch và thực thi các chương trình C. Bạn có thể chọn **Dev-C++**, **Code:: Blocks**, hoặc **Turbo C**. Tuy nhiên, lựa chọn phổ biến nhất và hay được sử dụng nhất là Dev-C++ và các chương trình C trong loạt bài này cũng được biên dịch và thực thi trong Dev-C++.

Sau khi đã cài đặt xong, để biên dịch và thực thi một chương trình C:

vào **File -> New -> Project -> Console Application -> C project**, sau đó nhập tên vào hoặc (b) **File -> New -> Source File**. Cuối cùng, sao chép và dán chương trình C vào file bạn vừa tạo. Để biên dịch và thực thi, chọn **Execute -> Compile & Run**.

### **Cài đặt để chạy trên Command Prompt**

Nếu bạn muốn cài đặt để biên dịch và chạy trên Command Prompt, thì bạn nên đọc phần sau đây.

Nếu bạn muốn cài đặt môi trường C++, bạn cần hai phần mềm có sẵn trong máy sau đây:

#### **Text Editor**

Nó sẽ được sử dụng để soạn chương trình của bạn. Ví dụ như Windows Notepad, OS Edit command, Brief, Epsilon, EMACS và Vim hoặc Vi.

Tên và phiên bản của Text Editor có thể đa dạng trên các hệ điều hành khác nhau. Ví dụ, Notepad sẽ được sử dụng trên Windows và Vim hoặc Vi có thể được sử dụng trên Windows cũng như Linux hoặc UNIX.

Các file, mà bạn tạo với editor này, được gọi là các source file và với C++, chúng được đặt tên với đuôi mở rộng là .cpp, .cp, hoặc .c.

Trước khi bắt đầu lập trình, đảm bảo rằng bạn có một Text editor và bạn có đủ kinh nghiệm để soạn một chương trình C++.

#### **C++ Compiler:**

C++ Compiler được sử dụng để biên dịch source code của bạn thành chương trình có thể thực thi.

Hầu hết C++ compiler không quan tâm phần đuôi mở rộng bạn cung cấp cho source code, nhưng nếu bạn không xác định, thì theo mặc định, nó sẽ sử dụng đuôi là .cpp.

Compiler được sử dụng thường xuyên nhất là GNU C/C++ compiler, hoặc bạn có thể sử dụng các Compiler khác từ HP hoặc Solaris nếu bạn có Hệ điều hành tương ứng.

#### **Cài đặt GNU C/C++ Compiler**

## Cài đặt trên UNIX/Linux

Nếu bạn đang sử dụng **Linux** hoặc **UNIX**, thì kiểm tra xem GCC đã được cài đặt trên hệ thống chưa bằng việc nhập lệnh sau tại dòng lệnh (command line):

```
$ g++ -v
```

Nếu bạn đã cài đặt GCC, thì nó sẽ in thông báo sau:

```
Using built-in specs.
Target: i386-redhat-linux
Configured with: ../configure --prefix=/usr .....
Thread model: posix
gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)
```

## Cài đặt trên Mac OS X

Nếu bạn sử dụng Mac OS X, cách đơn giản nhất để cài đặt GCC là tải môi trường phát triển Xcode từ Website của Applet và theo các chỉ dẫn.

## Cài đặt trên Windows

Để cài đặt GCC trên Windows, bạn cần cài đặt MinGW. Tải phiên bản mới nhất của MinGW, sẽ có tên là MinGW-<version>.exe.

Trong khi cài đặt MinGW, tối thiểu bạn phải cài đặt gcc-core, gcc-g++, binutils, và MinGW runtime.

Thêm thư mục phụ bin của MinGW tới biến môi trường **PATH**, để mà bạn có thể xác định các tool trên command line bởi các tên đơn giản của chúng.

Khi việc cài đặt hoàn tất, bạn có thể chạy gcc, g++, ar, ranlib, dlltool, và một số GNU tool khác từ Windows command line.

## 2. Một số khái niệm cơ bản

Khi chúng ta xem xét một chương trình C++, nó có thể được định nghĩa như là một tập hợp của các đối tượng, mà giao tiếp thông qua việc triệu hồi các phương thức của mỗi đối tượng đó. Dưới đây, chúng tôi miêu tả ngắn gọn ý nghĩa của class (lớp), object (đối tượng), method (phương thức) và các biến đối tượng:

- **Đối tượng** - Đối tượng có các trạng thái và hành vi. Ví dụ: một đối tượng dog có các trạng thái là color, name, breed, và các hành vi là wagging, barking, eating. Một đối tượng là một minh họa của một lớp.
- **Lớp** - Một lớp có thể được định nghĩa như là một template/blueprint, mà miêu tả hành vi/trạng thái mà đối tượng hỗ trợ.
- **Phương thức** - Về cơ bản, một phương thức là một hành vi. Một lớp có thể chứa nhiều phương thức. Phương thức là nơi tính logic được viết, dữ liệu được thao tác và tất cả action được thực thi.

- **Biến instance** - Mỗi đối tượng có tập hợp biến đối tượng duy nhất của nó. Trạng thái của một đối tượng được tạo ra bởi các giá trị được gán cho các biến đối tượng của nó.

### 3. Cấu trúc chương trình C++

Bạn theo dõi một đoạn code đơn giản sẽ in *Hello World*.

```
#include <iostream>
using namespace std;

// Ham main() la noi su thuc thi chuong trinh bat dau
int main()
{
    cout << "Hello World"; // In dòng chữ Hello World
    return 0;
}
```

Chương trình trên có các phần sau:

- Ngôn ngữ C++ định nghĩa một số header, mà chứa thông tin cần thiết và hữu ích cho chương trình của bạn. Với chương trình này, header là **<iostream>** là cần thiết.
- Dòng **using namespace std;** nói cho compiler sử dụng std namespace. Namespace là phần bổ sung gần đây cho C++.
- Dòng tiếp theo **Ham main() la noi su thuc thi chuong trinh bat dau** là một comment đơn dòng trong C++. Các comment đơn dòng bắt đầu với // và kết thúc ở cuối dòng.
- Dòng **int main()** là hàm main, tại đây việc thực thi chương trình bắt đầu.
- Dòng tiếp theo **cout << "Hello World";** để in dòng chữ "Hello World" trên màn hình.
- Dòng tiếp theo **return 0;** kết thúc hàm main() và làm nó trả về giá trị 0 tới tiến trình đang gọi.

### 4. Biên dịch và thực thi chương trình C++

1. Dưới đây là cách lưu file, biên dịch và chạy chương trình với command prompt. Bạn theo các bước sau:

- Mở một text editor và thêm đoạn code trên.
- Lưu file với tên: hello.cpp.
- Mở một dòng nhắc lệnh (command prompt) và tới thư mục nơi bạn lưu file đó.

- Soạn 'g++ hello.cpp ' và nhấn Enter để biên dịch code trên. Nếu không có lỗi xảy ra trong code của bạn thì dòng nhắc lệnh sẽ đưa bạn tới dòng tiếp theo và tạo a.out file có thể thực thi.
- Bây giờ, bạn soạn 'a.out' để chạy chương trình.
- Bạn sẽ thấy 'Hello World' được in trên cửa sổ.

```
$ g++ hello.cpp
$ ./a.out
Hello World
```

Bạn chắc chắn rằng g++ là trong path của bạn và bạn đang chạy nó trong thư mục mà chứa hello.cpp file.

2. Đó là cách biên dịch và thực thi chương trình tại command prompt. Nếu bạn đang sử dụng **Dev-C++**, hoặc **Microsoft Visual Studio**, hoặc **Turbo C++**, bạn có thể sao chép ví dụ trên, sau đó:

Mở Dev-C++ chẳng hạn, chọn **File** -> **New Source**, sau đó dán ví dụ vào source file này và chọn **Execute** tab, chọn **Compile & Run** để thực thi chương trình. Bạn cũng có thể tạo một Project mới trong tùy chọn New để đặt tên và lưu cho ví dụ vừa thực hiện.

### 5. Dấu chấm phẩy và khối (block) trong C++

Trong C++, dấu chấm phẩy là ký tự kết thúc lệnh (statement terminator). Nghĩa là, mỗi lệnh đơn phải kết thúc bởi một dấu chấm phẩy. Nó chỉ dẫn sự kết thúc của một thực thể logic.

Dưới đây là ví dụ về 3 lệnh khác nhau:

```
x = y;
y = y+1;
add(x, y);
```

Một khối (block) là một tập hợp các lệnh được kết nối một cách logic, mà được bao quanh bởi các dấu ngoặc móc mở và đóng. Ví dụ:

```
{
    cout << "VietJack xin chào các bạn!"; // in dòng chữ VietJack xin chào các bạn!
    return 0;
}
```

Hai dạng sau là tương đương, C++ không quan tâm bạn đặt bao nhiêu lệnh trên một dòng. Ví dụ:

```
x = y;
y = y+1;
add(x, y);
```

Là giống với:

```
x = y; y = y+1; add(x, y);
```

## 6. Định danh (Identifier) trong C++

Một Định danh (Identifier) trong C++ là một tên được sử dụng để nhận diện một biến, hàm, lớp, module, hoặc bất kỳ user-defined item nào (người dùng tự định nghĩa). Một Định danh (Identifier) bắt đầu với một chữ cái từ A tới Z hoặc từ a tới z hoặc một dấu gạch dưới (\_) được theo sau bởi 0 hoặc nhiều chữ cái, dấu gạch dưới và chữ số (từ 0 tới 9).

C++ không cho phép các ký tự như @, \$ và % bên trong các Identifier. C++ là ngôn ngữ lập trình phân biệt kiểu chữ. Vì thế, **Manpower** và **manpower** là hai Identifier khác nhau trong C++.

Dưới đây là một số ví dụ về Identifier (Định danh) thích hợp:

```
hoang    nam    abc    sinh_vien    a_123  
caogia50 _nhanvien j    a23b9    vietJack
```

## 7. Từ khóa trong C++

Bảng dưới liệt kê các từ được dự trữ (dành riêng) trong C++. Những từ này không thể được sử dụng như là constant hoặc biến hoặc bất kỳ tên Identifier (Định danh) nào.

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile

do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

### **8. Trigraph trong C++**

Một Trigraph là một dãy 3 ký tự mà biểu diễn một ký tự đơn và dãy này luôn luôn bắt đầu với 2 dấu hỏi.

Các Trigraph được mở rộng bất cứ nơi đâu chúng xuất hiện, bao gồm bên trong String literal và Character literal, trong comment, và trong các preprocessor directive (các directive tiền xử lý).

Dưới đây là các dãy trigraph được sử dụng thường xuyên nhất:

Trigraph	Thay thế cho
??=	#
??/	\
??'	^
??(	[
??)	]
??!	
??<	{
??>	}
??-	~

Tất cả compiler không hỗ trợ Trigraph và chúng được khuyến là không nên sử dụng bởi vì tính khó hiểu của nó.

### **9. Khoảng trắng (Whitespace) trong C++**

Một dòng mà chỉ chứa khoảng trắng (Whitespace), có thể là một comment, được biết đến như là một dòng trống, và C++ hoàn toàn bỏ qua nó.

Khoảng trắng (Whitespace) là khái niệm được sử dụng trong C++ để miêu tả blank, tab, ký tự newline (dòng mới), và comment. Khoảng trắng (Whitespace) phân biệt các phần của lệnh và giúp compiler nhận diện vị trí một phần tử trong một lệnh, ví dụ như int, vị trí phần kết thúc và vị trí phần tử tiếp theo bắt đầu. Do đó, trong lệnh sau:

```
int diemthi;
```

Phải có ít nhất một Whitespace (thường là khoảng trống) giữa int và age để compiler có thể phân biệt chúng. Trong lệnh:

```
tongLuong = luongCoBan + phuCap; // tính tong luong
```

Các ký tự khoảng trắng (Whitespace) giữa luongCoBan và =, hoặc giữa = và phuCap là không cần thiết; tuy nhiên, để giúp cho code của bạn dễ đọc hơn, bạn có thể thêm chúng vào.

Comment của chương trình là các lời diễn giải, mà bạn có thể bao trong C/C++ code, và giúp cho bất kỳ ai đọc source code dễ dàng hơn. Tất cả ngôn ngữ lập trình đều cho phép một số mẫu comment nào đó.

C++ hỗ trợ các comment đơn dòng và đa dòng. Tất cả ký tự có trong comment được bỏ qua bởi C/C++ compiler.

Comment trong C/C++ bắt đầu với /\* và kết thúc với \*/. Ví dụ:

```
/* Day la mot comment don dong */

/* C/C++ cung ho tro cac comment
 * ma co nhieu dong
 */
```

Một comment cũng có thể bắt đầu với //, kéo dài tới phần cuối của dòng. Ví dụ:

```
#include <iostream>
using namespace std;

main()
{
    cout << "Hoc C/C++ co ban va nang cao"; // In dong chu Hoc C/C++ co ban
    va nang cao

    return 0;
}
```

Khi code trên được biên dịch, nó sẽ bỏ qua // **In dong chu Hello World** và cuối cùng cho kết quả sau:

```
Hello World
```

Bên trong một comment dạng /\* và \*/, các ký tự // không có ý nghĩa đặc biệt gì. Bên trong một comment dạng //, các ký tự /\* và \*/ không có ý nghĩa đặc biệt gì. Vì thế, bạn có thể "lồng" bất kỳ dạng comment nào bên trong dạng khác. Ví dụ:

```
/* Comment ve dong lenh in dong chu Hello World:

cout << "xin chao cac ban!"; // in dong chu xin chao cac ban!
```

## **10. Kiểu dữ liệu trong C/C++**

Trong khi làm việc với bất kỳ ngôn ngữ lập trình nào, bạn cần sử dụng các kiểu biến đa dạng để lưu giữ thông tin. Các biến, không gì khác ngoài các vị trí bộ nhớ được dành riêng để lưu giá trị. Nghĩa là, khi bạn tạo một biến, bạn dành riêng một số không gian trong bộ nhớ cho biến đó.

Bạn có thể thích lưu thông tin của các kiểu dữ liệu (Data Type) đa dạng như Character, Wide Character, integer, floating-point, double floating point, Boolean, .... Dựa trên kiểu dữ liệu của một biến, hệ thống sẽ cấp phát bộ nhớ và quyết định những gì có thể được lưu giữ trong bộ nhớ dành riêng đó.

Kiểu dữ liệu nguyên thủy trong C/C++

Tên tiếng Anh là Primitive Type, còn có thể gọi là kiểu dữ liệu gốc, kiểu dữ liệu có sẵn trong C/C++. Bên cạnh các kiểu dữ liệu gốc này, C/C++ cũng cung cấp các kiểu dữ liệu user-defined. Bảng dưới đây liệt kê 7 kiểu dữ liệu cơ bản trong C/C++:

Kiểu dữ liệu	Từ khóa
Boolean	bool
Ký tự	char
Số nguyên	int
Số thực	float
Số thực dạng Double	double
Kiểu không có giá trị	void
Kiểu Wide character	wchar_t

Một số kiểu cơ bản có thể được sửa đổi bởi sử dụng một hoặc nhiều modifier này:

- signed (kiểu có dấu)
- unsigned (kiểu không có dấu)
- short
- long

Bảng sau hiển thị kiểu biến, lượng bộ nhớ nó dùng để lưu giá trị trong bộ nhớ, và giá trị lớn nhất và nhỏ nhất có thể được lưu giữ với các kiểu biến đó:

Kiểu	Độ rộng bit	Dãy giá trị
------	-------------	-------------



char	1 byte	-127 tới 127 hoặc 0 tới 255
unsigned char	1 byte	0 tới 255
signed char	1 byte	-127 tới 127
int	4 byte	-2147483648 tới 2147483647
unsigned int	4 byte	0 tới 4294967295
signed int	4 byte	-2147483648 tới 2147483647
short int	2 byte	-32768 tới 32767
unsigned short int	Range	0 tới 65,535
signed short int	Range	-32768 tới 32767
long int	4 byte	-2,147,483,647 tới 2,147,483,647
signed long int	4 byte	Tương tự như long int
unsigned long int	4 byte	0 tới 4,294,967,295
float	4 byte	+/- 3.4e +/- 38 (~7 chữ số)
double	8 byte	+/- 1.7e +/- 308 (~15 chữ số)
long double	8 byte	+/- 1.7e +/- 308 (~15 chữ số)
wchar_t	2 hoặc 4 byte	1 wide character

Kích cỡ của các biến có thể khác với những gì hiển thị trên bảng, phụ thuộc vào compiler và máy tính bạn đang sử dụng.

Dưới đây là ví dụ sẽ đưa ra kích cỡ chính xác của các kiểu dữ liệu đa dạng trên máy tính của bạn.

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Kích co cua char la: " << sizeof(char) << endl;
    cout << "Kích co cua int la: " << sizeof(int) << endl;
    cout << "Kích co cua short int la: " << sizeof(short int) << endl;
    cout << "Kích co cua long int la: " << sizeof(long int) << endl;
    cout << "Kích co cua float la: " << sizeof(float) << endl;
```

```
cout << "Kích co cua double la: " << sizeof(double) << endl;
cout << "Kích co cua wchar_t la: " << sizeof(wchar_t) << endl;
return 0;
}
```

Ví dụ này sử dụng **endl**, mà chèn một ký tự newline (dòng mới) sau mỗi dòng, và toán tử << được sử dụng để truyền nhiều giá trị tới màn hình. Chúng tôi cũng sử dụng toán tử **sizeof()** để lấy kích cỡ của các kiểu dữ liệu đa dạng. Khi code trên được biên dịch và thực thi, nó cho kết quả sau (kết quả có thể đa dạng tùy thuộc vào compiler và máy tính bạn đang sử dụng).

```
Kích co cua char la: 1
Kích co cua int la: 4
Kích co cua short int la: 2
Kích co cua long int la: 4
Kích co cua float la: 4
Kích co cua double la: 8
Kích co cua wchar_t la: 4
```

### 11. Khai báo typedef trong C/C++.

Bạn có thể tạo một tên mới cho một kiểu dữ liệu đang tồn tại bởi sử dụng **typedef** trong C/C++. Cú pháp đơn giản sau để định nghĩa một kiểu dữ liệu mới bởi sử dụng typedef:

```
typedef kieu_du_lieu ten_moi;
```

Ví dụ sau nói cho compiler rằng sothuc là tên khác của float:

```
typedef float sothuc;
```

Bây giờ, khai báo sau là hoàn toàn hợp lệ và sẽ tạo một biến số thực gọi là *vantoc*:

```
sothuc vantoc;
```

### 12. Kiểu liệt kê enum trong C/C++

Kiểu liệt kê enum khai báo một tên kiểu tùy ý và một tập hợp của 0 hoặc nhiều Identifier (Định danh) mà có thể được sử dụng như là các giá trị của kiểu đó. Mỗi Enumerator là một constant có kiểu là kiểu liệt kê (enumeration).

Để tạo một Enumeration, bạn sử dụng từ khóa **enum** trong C/C++. Form chung của kiểu liệt kê enum là:

```
enum ten_cua_enum { danh_sach_cac_ten } danh_sach_bien;
```

Tại đây, ten\_cua\_enum là tên kiểu liệt kê. Danh sách tên được phân biệt bởi dấu phẩy.

Ví dụ, code sau định nghĩa một tên kiểu liệt kê hàng hóa gọi là *hanghoa* và biến *c* là kiểu của hanghoa. Cuối cùng, *c* được gán giá trị nuocngot.

```
enum hanghoa { sua, nuocngot, biachai } c;
```

```
c = nuocngot;
```

Theo mặc định, trong danh sách các tên thì giá trị của tên đầu tiên là 0, tên thứ hai là 1 và tên thứ 3 là 2, .... Nhưng bạn có thể cung cấp cho một tên một giá trị cụ thể bằng việc thêm một Initializer (giá trị khởi tạo). Ví dụ, trong enumeration sau, **nuocngot** sẽ có giá trị là 40:

```
enum hanghoa { sua, nuocngot=40, biachai };
```

Ở đây, **biachai** sẽ có giá trị là 41 bởi vì mỗi tên sẽ có giá trị lớn hơn của tên trước đó là 1.

### 13. Kiểu biến trong C/C++

Một biến cung cấp nơi lưu giữ được đặt tên để chúng ta có thể thao tác. Mỗi biến trong C/C++ có một kiểu cụ thể, mà quyết định: kích cỡ và cách bố trí bộ nhớ của biến; dãy giá trị có thể được lưu giữ bên trong bộ nhớ đó; và tập hợp hoạt động có thể được áp dụng cho biến đó.

Tên biến có thể gồm các ký tự, các chữ số, dấu gạch dưới. Nó phải bắt đầu bởi hoặc một ký tự hoặc một dấu gạch dưới. Các ký tự chữ hoa và chữ thường là khác nhau bởi C/C++ là ngôn ngữ phân biệt kiểu chữ. Biến có thể trong các kiểu giá trị đa dạng như kiểu char, int, float, ...

#### **Định nghĩa biến trong C/C++**

Định nghĩa biến trong C/C++ nghĩa là nói cho compiler nơi và lượng bộ nhớ cần tạo để lưu giữ biến đó. Một định nghĩa biến xác định một kiểu dữ liệu, và chứa danh sách của một hoặc nhiều biến có kiểu đó, như sau:

```
kieu_gia_tri danh_sach_bien;
```

Ở đây, **kieu\_gia\_tri** phải là một kiểu dữ liệu hợp lệ trong C/C++, gồm char, w\_char, int, float, double, bool hoặc bất kỳ đối tượng nào mà người dùng tự định nghĩa, ... và **danh\_sach\_bien** có thể chứa một hoặc nhiều tên Identifier (Định danh) phân biệt nhau bởi dấu phẩy. Sau đây là một số khai báo hợp lệ trong C/C++:

```
int j, q, k;  
char v, viet;  
float x, diemthi;  
double luong;
```

Dòng **int j, q, k;** vừa khai báo và định nghĩa các biến j, q, k, mà chỉ dẫn compiler để tạo các biến với tên là j, q, k với kiểu int.

Các biến có thể được khởi tạo (được gán giá trị ban đầu) trong các khai báo. Initializer (phần khởi tạo) gồm một ký hiệu bằng được theo sau bởi một Constant Expression (biểu thức hằng số), như sau:

```
kieu_gia_tri ten_bien = giaTri;
```

Ví dụ:

```
extern int d = 3, f = 5; // khai báo biến d và f.  
int d = 3, f = 5;      // định nghĩa và khởi tạo biến d và f.  
byte z = 22;          // định nghĩa và khởi tạo biến z.  
char x = 'k';         // biến x có giá trị là 'k'.
```

Với định nghĩa mà không có phần khởi tạo: các biến được khởi tạo với NULL (tất cả byte có giá trị 0); giá trị khởi tạo của tất cả biến khác không được định nghĩa.

### **Khai báo biến trong C/C++**

Khai báo biến trong C/C++ chắc chắn với compiler rằng có một biến đang tồn tại với kiểu và tên đã cho, để mà compiler tiếp tục trình biên dịch mà không cần biết đầy đủ chi tiết về biến đó. Một khai báo biến chỉ có ý nghĩa tại thời gian biên dịch, compiler cần khai báo biến thực sự tại thời điểm kết nối chương trình.

Khai báo biến là hữu ích khi bạn đang sử dụng nhiều file, và bạn định nghĩa biến của bạn ở một trong các file đó mà sẽ có sẵn tại thời điểm kết nối chương trình. Bạn sẽ sử dụng từ khóa `extern` để khai báo một biến tại bất kỳ đâu. Mặc dù, bạn có thể khai báo một biến nhiều lần trong chương trình C/C++, nhưng nó có thể chỉ được định nghĩa một lần trong một file, một hàm, hoặc một khối code.

Ví dụ

Trong ví dụ sau, một biến được khai báo tại ở phần đầu chương trình, nhưng nó đã được định nghĩa ở bên trong hàm `main`.

```
#include <iostream>  
using namespace std;  
  
// Phan khai báo biến:  
extern int a, b;  
extern int c;  
extern float f;  
  
int main ()  
{  
    // Phan định nghĩa biến:  
    int a, b;  
    int c;  
    float f;  
  
    // Phan khởi tạo biến  
    a = 10;  
    b = 20;  
    c = a + b;
```

```
cout << c << endl ;

f = 70.0/3.0;
cout << f << endl ;

return 0;
}
```

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

```
30
23.3333
```

Giống khái niệm áp dụng trên khai báo hàm, bạn cung cấp tên hàm tại thời điểm khai báo và định nghĩa thực sự của hàm đó có thể được cung cấp ở bất cứ đâu. Ví dụ:

```
// phan khai bao ham
int func();

int main()
{
    // phan goi ham
    int i = func();
}

// phan dinh nghia ham
int func()
{
    return 0;
}
```

### Lvalue và Rvalue trong C/C++

Có hai loại Expression trong C/C++:

- **lvalue** : Đây là expression liên quan tới một vị trí vùng nhớ. Bất kỳ lvalue có thể xuất hiện ở bên trái hoặc bên phải của một phép gán.
- **rvalue** : Khái niệm này liên quan tới một giá trị dữ liệu mà được lưu giữ tại một số địa chỉ trong vùng nhớ. Một rvalue là một expression mà không thể có một giá trị được gán tới nó, nghĩa là: rvalue có thể xuất hiện bên phải nhưng không thể xuất hiện bên trái của một phép gán.

Các biến là lvalue có thể xuất hiện bên trái của phép gán. Các Numeric literal (hằng số) là rvalue không thể được gán và không thể xuất hiện ở bên trái của phép gán. Dưới đây là một lệnh hợp lệ trong C/C++:

```
int t = 10;
```

Những lệnh sau là không hợp lệ và sẽ cho một compile-time error (lỗi tại thời điểm biên dịch):

```
16 = 30;
```

#### 14. Phạm vi biến trong C++

Một scope (phạm vi) là một khu vực của chương trình nơi biến hoạt động, và nói chung có thể có 3 khu vực mà biến có thể được khai báo:

- Bên trong một hàm hoặc một khối, được gọi là biến cục bộ (local).
- Trong định nghĩa của các tham số hàm, được gọi là các tham số chính thức (formal).
- Bên ngoài của tất cả hàm, được gọi là biến toàn cục (global).

Chúng ta sẽ học hàm và các tham số của hàm là gì trong chương tới. Dưới đây chúng tôi sẽ giải thích khái niệm về biến cục bộ và biến toàn cục.

Biến cục bộ trong C++

Các biến được khai báo bên trong một hàm hoặc khối là các biến cục bộ (local). Chúng chỉ có thể được sử dụng bởi các lệnh bên trong hàm hoặc khối code đó. Các biến cục bộ không được biết ở bên ngoài hàm đó (tức là chỉ được sử dụng bên trong hàm hoặc khối code đó). Dưới đây là ví dụ sử dụng các biến cục bộ:

```
#include <iostream>
using namespace std;

int main ()
{
    // khai báo biến cục bộ:
    int a, b;
    int c;

    // khởi tạo biến
    a = 10;
    b = 20;
    c = a + b;

    cout << c;

    return 0;
}
```

#### 15. Biến toàn cục trong C++

Biến toàn cục (global) trong C++ được định nghĩa bên ngoài các hàm, thường ở phần đầu chương trình. Các biến toàn cục giữ giá trị của nó trong suốt vòng đời chương trình của bạn.

Một biến toàn cục có thể được truy cập bởi bất kỳ hàm nào. Tức là, một biến toàn cục là có sẵn cho bạn sử dụng trong toàn bộ chương trình sau khi đã khai báo nó. Dưới đây là ví dụ sử dụng biến toàn cục và biến nội bộ trong C++:

```
#include <iostream>
using namespace std;

// phan khai bao bien toan cuc:
int g;
int main ()
{
    // phan khai bao bien cuc bo:
    int a, b;

    // phan khoi tao bien
    a = 10;
    b = 20;
    g = a + b;

    cout << g;

    return 0;
}
```

Một chương trình có thể có các biến toàn cục và biến cục bộ cùng tên với nhau, nhưng trong một hàm thì giá trị của biến cục bộ sẽ được ưu tiên. Ví dụ:

```
#include <iostream>
using namespace std;

// phan khai bao bien toan cuc:
int g = 20;

int main ()
{
    // phan khai bao bien cuc bo:
    int g = 10;

    cout << g;

    return 0;
}
```

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

10

## **16. Khởi tạo biến cục bộ và biến toàn cục bởi hệ thống trong C++**

Khi một biến cục bộ được định nghĩa, nó không được khởi tạo bởi hệ thống, chính bạn phải khởi tạo nó. Các biến toàn cục được khởi tạo tự động bởi hệ thống khi bạn định nghĩa chúng, như sau:

Kiểu dữ liệu	Giá trị khởi tạo
int	0
char	'\0'
float	0
double	0
pointer	NULL

Khởi tạo biến một cách chính xác là một sự thực hành tốt, nếu không, đôi khi chương trình sẽ cho kết quả không mong đợi.

### 17. Hằng (Constant/Literal) trong C/C++

Constant liên quan tới các giá trị cố định mà chương trình không thể thay đổi và chúng được gọi là **literals**.

Constant là một kiểu dữ liệu thay thế cho Literal, còn Literal thể hiện chính nó. Trong ví dụ: `const PI = 3.14` thì Constant ở đây là PI, còn Literal là 3.14.

Constant có thể là bất kỳ kiểu dữ liệu cơ bản nào trong C/C++, và có thể được phân chia thành giá trị hằng số nguyên, hằng số thực, hằng ký tự, hằng chuỗi và Boolean literal (tạm dịch: hằng logic).

Ngoài ra, constant được đối xử giống như các biến thông thường, ngoại trừ việc giá trị của chúng là không thể thay đổi sau khi định nghĩa.

Hằng số nguyên trong C/C++

**17.1 Hằng số nguyên** có thể là decimal (cơ số 10), octal (cơ số 8) hay hexadecimal (cơ số 16). Giá trị có tiền tố (prefix) là 0 cho octal, là 0x hay 0X cho hexadecimal và không có gì cho decimal.

Một hằng số nguyên cũng có các hậu tố (suffix) U hay L thể hiện kiểu unsigned hay long. Hậu tố có thể là chữ hoa hoặc chữ thường và có thể trong bất kỳ thứ tự nào.

Ví dụ về các hằng số nguyên:

```
212      // Hop le
215u     // Hop le
0xFFeL   // Hop le
048      // Không hop le: 8 không phải là một ký số trong hệ octal
032UU    // Không hop le: hậu tố (suffix) không thể bị lặp
```

Dưới đây là các kiểu hằng số nguyên đa dạng:



```

85      // Hang so dang decimal
0213    // Hang so dang octal
0x4b    // Hang so dang hexadecimal
30      // Hang so dang so nguyen (int)
30u     // Hang so dang so nguyen khong dau (unsigned int)
30l     // Hang so dang long
30ul    // Hang so dang unsigned long

```

### Hằng số thực trong C/C++

Một hằng số thực bao gồm phần nguyên (integer part), dấu chấm thập phân (decimal point), phần lẻ (fraction part) và phần mũ (exponent part). Chúng ta có thể biểu diễn hằng số thực theo dạng decimal hay dạng mũ.

Khi thể hiện dạng decimal phải bao gồm dấu chấm thập phân, phần mũ hoặc cả hai. Khi thể hiện dạng mũ phải gồm phần nguyên, phần lẻ hoặc cả hai. Dạng mũ đi kèm với kí tự E hoặc e.

Xét các ví dụ sau:

```

3.14159    // Hop le
314159E-5L  // Hop le
510E       // Khong hop le: pham mu (exponent) con thieu
210f       // Khong hop le: khong co phan thập pha decimal hoac phan mu
exponent
.e55       // Khong hop le: thieu phan nguyen integer hoac phan fraction

```

Boolean literal trong C/C++

Có hai kiểu Boolean literal và chúng là một phần của các từ khóa C/C++:

- Giá trị **true**
- Và giá trị **false**

Bạn không nên đồng nhất giá trị true với 1 và giá trị false với 0.

### 17.2 Hằng ký tự trong C/C++

Các hằng ký tự trong C/C++ mở đầu và kết thúc bởi dấu nháy đơn. Nếu hằng ký tự bắt đầu với L (ví dụ L'x') thì nó là kiểu wchar\_t. Nếu không thì, nó là hằng ký tự kiểu **char**, ví dụ như 'x'.

Hằng ký tự có thể là một ký tự (như 'X'), một escape sequence (như '\t') hay một ký tự mở rộng (như '\u02c0').

Một số ký tự trong C/C++ khi được đứng trước bởi dấu \ thì chúng sẽ mang một ý nghĩa đặc biệt như bắt đầu dòng mới '\n' hay tạo một tab '\t'. Chúng được biết như là escape sequence (dãy thoát). Bảng dưới đây thể hiện một số mã escape sequence phổ biến:

Escape sequence	Ý nghĩa
-----------------	---------

\\	Ký tự \
\'	Ký tự '
\"	Ký tự "
\?	Ký tự ?
\a	Alert hoặc bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	tab ngang
\v	tab dọc
\ooo	Số hệ cơ số 8 của một tới 3 chữ số
\xhh . . .	Số hệ cơ số 16 của một hoặc nhiều chữ số

Dưới đây là ví dụ minh họa một số ký tự escape sequence:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello\tWorld\n\n";
    return 0;
}
```

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

```
Hello  World
```

### **17.3 Hằng chuỗi trong C/C++**

Hằng chuỗi chứa trong dấu nháy kép, ví dụ "abc". Một chuỗi sẽ chứa các kí tự tương tự hằng kí tự, gồm các ký tự thuần, escape sequence, và các ký tự mở rộng.

Có thể ngắt một dòng dài thành nhiều dòng bởi sử dụng hằng chuỗi và phân biệt chúng bởi sử dụng khoảng trắng (whitespace).

Xét ví dụ một hằng chuỗi trong C/C++ thể hiện theo 3 cách khác nhau:

```
"hoc, lap trinh"
```

```
"hoc, \  
lap trinh"  
  
"hoc, " "lap" "trinh"
```

#### 17.4 Định nghĩa hằng trong C/C++

Có hai cách định nghĩa hằng trong C/C++ là:

- Sử dụng bộ tiền xử lý **#define**.
- Sử dụng từ khóa **const**.

##### Sử dụng bộ tiền xử lý **#define** trong C/C++

Dưới đây là cú pháp để sử dụng tiền xử lý **#define** để định nghĩa một constant trong C/C++:

```
#define identifier value
```

Ví dụ sau giải thích cú pháp trên:

```
#include <iostream>  
using namespace std;  
#define CHIEUDAI 10  
#define CHIEURONG 5  
#define NEWLINE '\n'  
  
int main()  
{  
    int dientich;  
    dientich = CHIEUDAI * CHIEURONG;  
    cout << dientich;  
    cout << NEWLINE;  
    return 0;  
}
```

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

```
50
```

#### 17.5 Sử dụng từ khóa **const** trong C/C++

Bạn có thể sử dụng tiền tố **const** để định nghĩa hằng trong C/C++ với một kiểu cụ thể, như sau:

```
const kieu_gia_tri bien = giaTri;
```

Ví dụ sau giải thích chi tiết cú pháp trên:

```
#include <iostream>  
using namespace std;  
int main()
```

```

{
    const int CHIEUDAI = 10;
    const int CHIEURONG = 5;
    const char NEWLINE = '\n';
    int area;
    dientich = CHIEUDAI * CHIEURONG;
    cout << dientich;
    cout << NEWLINE;
    return 0;
}

```

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

50

**Ghi chú:** Nó là bước thực hành tốt cho bạn khi định nghĩa các hằng ở kiểu chữ hoa trong C/C++.

C++ cho phép các kiểu dữ liệu **char**, **int** và **double** có các Modifier đặt trước chúng. Một Modifier được sử dụng để thông báo ý nghĩa của kiểu cơ sở, giúp cho nó tăng sự chính xác hơn với sự cần thiết của các tình huống đa dạng.

Dưới đây là các Modifier trong C/C++:

- signed (có dấu)
- unsigned (không có dấu)
- long
- short

Các Modifier là: **signed**, **unsigned**, **long**, và **short** có thể được áp dụng cho kiểu integer. Ngoài ra, **signed** và **unsigned** có thể được áp dụng cho kiểu char, và **long** có thể áp dụng cho kiểu double.

Các Modifier là **signed** và **unsigned** cũng có thể được sử dụng như là tiền tố cho các Modifier là **long** hoặc **short** modifiers. Ví dụ: **unsigned long int**.

C++ cho phép kiểu khai báo tắt để khai báo các **unsigned**, **short**, or **long** integer. Bạn có thể chỉ đơn giản sử dụng từ **unsigned**, **short**, hoặc **long**, mà không cần int. Ví dụ sau minh họa hai khai báo là hợp lệ trong C/C++ để khai báo các biến unsigned integer:

```

unsigned x;
unsigned int y;

```

Để phân biệt sự khác nhau giữa hai Modifier là signed integer và unsigned integer được thông dịch bởi C/C++, bạn nên chạy chương trình sau:

```

#include <iostream>
using namespace std;
/* Chương trình này chỉ ra điểm khác nhau giữa
 * các số nguyên signed và unsigned.
 */

```

```
int main()
{
    short int i;          // mot so nguyen signed short int
    short unsigned int j; // mot so nguyen unsigned short int

    j = 32769;
    i = j;
    cout << i << " " << j;
    return 0;
}
```

Nó sẽ cho kết quả:

-32767 32769

Nếu bạn quay trở lại chương **Kiểu dữ liệu trong C/C++**, và đọc phần dãy giá trị của short int và unsigned short int, bạn sẽ nhận ra sự khác nhau khi chạy chương trình trên với  $j \leq 32767$  và với  $j \geq 32767$ .

Qualifier trong C/C++

Qualifier cung cấp thông tin bổ sung về các biến theo sau nó.

Qualifier	Ý nghĩa
const	Đối tượng của kiểu <b>const</b> không thể bị thay đổi bởi chương trình trong khi thực thi
volatile	Modifier này nói cho compiler rằng giá trị của biến có thể được thay đổi một cách không rõ ràng (không báo trước) bởi chương trình.
restrict	Một con trỏ được đặt là <b>restrict</b> thì có ý nghĩa là đối tượng nó trỏ đến có thể được truy cập. Restrict được thêm vào trong chuẩn C99.

### 18. Lớp lưu trữ (Storage Class) trong C/C++

Lớp lưu trữ (Storage Class) định nghĩa phạm vi và vòng đời của biến và/hoặc các hàm bên trong một chương trình C/C++. Chúng thường đứng trước kiểu dữ liệu mà chúng tác động. Dưới đây là các lớp lưu trữ có thể được sử dụng trong C/C++:

- auto
- register
- static
- extern
- mutable

Lớp lưu trữ auto trong C/C++

Lớp lưu trữ **auto** trong C/C++ là lớp lưu trữ mặc định cho tất cả biến cục bộ trong C/C++:

```
{  
    int diemthi;  
    auto int diemthi;  
}
```

Ví dụ trên định nghĩa hai biến với cùng lớp lưu trữ, auto chỉ có thể được sử dụng bên trong các hàm, ví dụ: cho các biến nội bộ.

### 18.1: Lớp lưu trữ register trong C/C++

Lớp lưu trữ **register** trong C/C++ được sử dụng để định nghĩa các biến cục bộ mà nên được lưu giữ trong một thanh ghi thay vì RAM. Nghĩa là, biến có kích cỡ tối đa bằng với kích cỡ thanh ghi (thường là 1 từ) và không thể có toán tử một ngôi '&' được áp dụng tới nó (vì không có địa chỉ bộ nhớ).

```
{  
    register int hocphi;  
}
```

Lớp lưu trữ register nên chỉ được dùng cho các biến yêu cầu truy cập nhanh như các biến đếm (counters). Cũng cần chú ý rằng, một biến định nghĩa với 'register' không có nghĩa là biến đó được lưu trữ trong thanh ghi. Tức là nó có thể được lưu trữ trong thanh ghi phụ thuộc vào phần cứng và giới hạn thực thi.

Lớp lưu trữ static trong C/C++

Lớp lưu trữ **static** trong C/C++ nói với compiler để giữ một biến cục bộ tồn tại trong toàn bộ thời gian sống của chương trình thay vì tạo và hủy biến mỗi lần nó vào và ra khỏi phạm vi biến. Vì vậy, các biến có static cho phép nó duy trì giá trị giữa các lần gọi hàm.

Lớp lưu trữ static cũng có thể được áp dụng cho các biến toàn cục (global). Khi áp dụng cho biến toàn cục, nó nói với trình biên dịch rằng, phạm vi của biến toàn cục bị giới hạn trong tập tin mà nó được khai báo.

Trong C/C++, khi static được sử dụng trên thành viên dữ liệu của lớp, nó gây ra: chỉ có một bản sao của thành viên đó được chia sẻ bởi tất cả đối tượng trong lớp của nó.

```
#include <iostream>  
  
// phan khai bao ham  
void func(void);  
static int biendem = 10; /* Day la bien toan cuc */  
main()  
{  
    while(biendem--)  
    {  
        func();  
    }  
}
```

```

    }
    return 0;
}
// Phan dinh nghia ham
void func( void )
{
    static int i = 5; // Day la bien cuc bo dang static
    i++;
    std::cout << "i co gia tri la " << i ;
    std::cout << " va biendem co gia tri la " << biendem << std::endl;
}

```

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

```

i co gia tri la 6 va biendem co gia tri la 9
i co gia tri la 7 va biendem co gia tri la 8
i co gia tri la 8 va biendem co gia tri la 7
i co gia tri la 9 va biendem co gia tri la 6
i co gia tri la 10 va biendem co gia tri la 5
i co gia tri la 11 va biendem co gia tri la 4
i co gia tri la 12 va biendem co gia tri la 3
i co gia tri la 13 va biendem co gia tri la 2
i co gia tri la 14 va biendem co gia tri la 1
i co gia tri la 15 va biendem co gia tri la 0

```

## 18.2: Lớp lưu trữ extern trong C/C++

Lớp lưu trữ **extern** trong C/C++ được dùng để cung cấp một tham chiếu của một biến toàn cục được nhìn thấy bởi TẤT CẢ các file chương trình. Khi bạn sử dụng 'extern', biến không thể được khởi tạo, khi nó trở tới tên biến tại một vị trí lớp lưu trữ mà đã được định nghĩa trước đó.

Khi bạn có nhiều file và bạn định nghĩa một biến hay hàm toàn cục trong một file và cũng muốn dùng nó trong các file khác, thì **extern** được dùng trong file khác để cung cấp tham chiếu của biến hay hàm được định nghĩa. Cần nhớ rằng, **extern** dùng để khai báo một biến hay hàm toàn cục trong file khác.

Lớp lưu trữ extern được dùng phổ biến khi có hai hoặc nhiều file chia sẻ cùng biến hay hàm toàn cục. Xem ví dụ với hai file sau:

File đầu tiên: extern1.cpp

```

#include <iostream>

int biendem ;
extern void vidu_extern();

main()
{
    biendem = 5;
    vidu_extern();
}

```

File thứ hai: extern2.cpp

```
#include <iostream>

extern int biendem;

void vidu_extern(void)
{
    std::cout << "Gia tri biendem la " << count << std::endl;
}
```

Ở đây, từ khóa *extern* đang được sử dụng để khai báo *biendem* trong file khác. Bây giờ biên dịch hai file này như sau:

```
$g++ extern1.cpp extern2.cpp -o write
```

Nó sẽ tạo chương trình **write** có thể thực thi, bạn thử thực thi *write* và kiểm tra kết quả như sau:

```
$/write
5
```

### 18.3: Lớp lưu trữ mutable trong C/C++

Lớp lưu trữ **mutable** trong C/C++ chỉ áp dụng cho các đối tượng class, sẽ được bàn luận trong chương sau. Nó cho phép một thành viên của một đối tượng để override (ghi đè). Đó là, một thành viên là mutable có thể được sửa đổi bởi một hàm thành viên const.

### 19. Toán tử trong C++

Một toán tử là một biểu tượng, mà nói cho compiler thực hiện các thao tác toán học và logic cụ thể. C++ cung cấp nhiều toán tử có sẵn, đó là:

- Toán tử số học
- Toán tử quan hệ
- Toán tử logic
- Toán tử so sánh bit
- Toán tử gán
- Toán tử hỗn hợp

#### 19.1. Toán tử số học trong C++

Bảng dưới liệt kê các toán tử số học được hỗ trợ bởi ngôn ngữ C++:

Giả sử biến A giữ giá trị 10, biến B giữ 20 thì:

Toán tử	Miêu tả	Ví dụ
+	Cộng hai toán hạng	A + B kết quả là 30
-	Trừ toán hạng thứ hai từ toán hạng đầu	A - B kết quả là -10
*	Nhân hai toán hạng	A * B kết quả là 200



/	Phép chia	B / A kết quả là 2
%	Phép lấy số dư	B % A kết quả là 0
++	Toán tử tăng, tăng giá trị toán hạng thêm một đơn vị	A++ kết quả là 11
--	Toán tử giảm, giảm giá trị toán hạng bớt một đơn vị	A-- kết quả là 9

### 19.2. Toán tử quan hệ trong C++

Bảng dưới đây liệt kê các toán tử quan hệ được hỗ trợ bởi ngôn ngữ C++:

Giả sử biến A giữ giá trị 10, biến B giữ 20 thì:

Toán tử	Miêu tả	Ví dụ
==	Kiểm tra nếu 2 toán hạng bằng nhau hay không. Nếu bằng thì điều kiện là true.	(A == B) là không đúng
!=	Kiểm tra 2 toán hạng có giá trị khác nhau hay không. Nếu không bằng thì điều kiện là true.	(A != B) là true
>	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không. Nếu lớn hơn thì điều kiện là true.	(A > B) là không đúng
<	Kiểm tra nếu toán hạng bên trái nhỏ hơn toán hạng bên phải hay không. Nếu nhỏ hơn thì là true.	(A < B) là true
>=	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn hoặc bằng giá trị của toán hạng bên phải hay không. Nếu đúng là true.	(A >= B) là không đúng
<=	Kiểm tra nếu toán hạng bên trái có giá trị nhỏ hơn hoặc bằng toán hạng bên phải hay không. Nếu đúng là true.	(A <= B) là true

### 19.3. Toán tử logic trong C++

Bảng dưới đây chỉ rõ tất cả các toán tử logic được hỗ trợ bởi ngôn ngữ C.

Giả sử biến A có giá trị 1 và biến B có giá trị 0:

Toán tử	Miêu tả	Ví dụ
&&	Được gọi là toán tử logic AND (và). Nếu cả hai toán tử đều có giá trị khác 0 thì điều kiện trở lên true.	(A && B) là false.
	Được gọi là toán tử logic OR (hoặc). Nếu một trong hai toán tử khác 0, thì điều kiện là true.	(A    B) là true.
!	Được gọi là toán tử NOT (phủ định). Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false.	!(A && B) là true.

#### 19.4. Toán tử so sánh bit trong C++

Toán tử so sánh bit làm việc trên đơn vị bit, tính toán biểu thức so sánh từng bit.

Bảng dưới đây về &, |, và ^ như sau:

p	q	p & q	p   q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Giả sử nếu A = 60; và B = 13; thì bây giờ trong định dạng nhị phân chúng sẽ là như sau:

A = 0011 1100

B = 0000 1101

-----

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

Các toán tử so sánh bit được hỗ trợ bởi ngôn ngữ C++ được liệt kê trong bảng dưới đây. Giả sử ta có biến A có giá trị 60 và biến B có giá trị 13, ta có:

Toán tử	Miêu tả	Ví dụ
&	Toán tử AND (và) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong cả hai toán hạng.	(A & B) sẽ cho kết quả là 12, tức là 0000 1100
	Toán tử OR (hoặc) nhị phân sao chép một bit tới kết	(A   B) sẽ cho kết quả là

	quả nếu nó tồn tại trong một hoặc hai toán hạng.	61, tức là 0011 1101
$\wedge$	Toán tử XOR nhị phân sao chép bit mà nó chỉ tồn tại trong một toán hạng mà không phải cả hai.	$(A \wedge B)$ sẽ cho kết quả là 49, tức là 0011 0001
$\sim$	Toán tử đảo bit (đảo bit 1 thành bit 0 và ngược lại).	$(\sim A)$ sẽ cho kết quả là -61, tức là 1100 0011.
$\ll$	Toán tử dịch trái. Giá trị toán hạng trái được dịch chuyển sang trái bởi số các bit được xác định bởi toán hạng bên phải.	$A \ll 2$ sẽ cho kết quả 240, tức là 1111 0000 (dịch sang trái hai bit)
$\gg$	Toán tử dịch phải. Giá trị toán hạng trái được dịch chuyển sang phải bởi số các bit được xác định bởi toán hạng bên phải.	$A \gg 2$ sẽ cho kết quả là 15, tức là 0000 1111 (dịch sang phải hai bit)

### 19.5. Toán tử gán trong C++

Dưới đây là những toán tử gán được hỗ trợ bởi ngôn ngữ C++:

Toán tử	Miêu tả	Ví dụ
=	Toán tử gán đơn giản. Gán giá trị toán hạng bên phải cho toán hạng trái.	$C = A + B$ sẽ gán giá trị của $A + B$ vào trong $C$
+=	Thêm giá trị toán hạng phải tới toán hạng trái và gán giá trị đó cho toán hạng trái.	$C += A$ tương đương với $C = C + A$
-=	Trừ đi giá trị toán hạng phải từ toán hạng trái và gán giá trị này cho toán hạng trái.	$C -= A$ tương đương với $C = C - A$
*=	Nhân giá trị toán hạng phải với toán hạng trái và gán giá trị này cho toán hạng trái.	$C *= A$ tương đương với $C = C * A$
/=	Chia toán hạng trái cho toán hạng phải và gán giá trị này cho toán hạng trái.	$C /= A$ tương đương với $C = C / A$
%=	Lấy phần dư của phép chia toán hạng trái cho toán hạng phải và gán cho toán hạng trái.	$C \% = A$ tương đương với $C = C \% A$
<<=	Dịch trái toán hạng trái sang số vị trí là giá trị toán hạng phải.	$C \ll = 2$ tương đương với $C = C \ll 2$
>>=	Dịch phải toán hạng trái sang số vị trí là giá trị toán hạng phải.	$C \gg = 2$ tương đương với $C = C \gg 2$

&=	Phép AND bit	$C \&= 2$ tương đương với $C = C \& 2$
^=	Phép OR loại trừ bit	$C \wedge= 2$ tương đương với $C = C \wedge 2$
=	Phép OR bit.	$C  = 2$ tương đương với $C = C   2$

### 19.6. Các toán tử hỗn hợp trong C++

Dưới đây là một số toán tử hỗn hợp quan trọng được hỗ trợ bởi ngôn ngữ C++.

Toán tử	Miêu tả
sizeof	trả về kích cỡ của một biến. Ví dụ: sizeof(a), với a là integer, sẽ trả về 4
Điều kiện ? X : Y	Nếu Condition là true ? thì nó trả về giá trị X : nếu không thì trả về Y
,	làm cho một dãy hoạt động được thực hiện. Giá trị của toàn biểu thức comma là giá trị của biểu thức cuối cùng trong danh sách được phân biệt bởi dấu phẩy
. (dot) và -> (arrow)	được sử dụng để tham chiếu các phần tử đơn của các lớp, các cấu trúc, và union
Cast	biến đổi một kiểu dữ liệu thành kiểu khác. Ví dụ: int(2.2000) sẽ trả về 2
&	trả về địa chỉ của một biến. Ví dụ: &a; sẽ trả về địa chỉ thực sự của biến này
*	là trỏ tới một biến. Ví dụ: *var sẽ trỏ tới một biến var

### 19.7. Thứ tự ưu tiên toán tử trong C++

Thứ tự ưu tiên toán tử trong C++ xác định cách biểu thức được tính toán. Ví dụ, toán tử nhân có quyền ưu tiên hơn toán tử cộng, và nó được thực hiện trước.

Ví dụ,  $x = 7 + 3 * 2$ ; ở đây, x được gán giá trị 13, chứ không phải 20 bởi vì toán tử \* có quyền ưu tiên cao hơn toán tử +, vì thế đầu tiên nó thực hiện phép nhân  $3 * 2$  và sau đó thêm với 7.

Bảng dưới đây liệt kê thứ tự ưu tiên của các toán tử. Các toán tử với quyền ưu tiên cao nhất xuất hiện trên cùng của bảng, và các toán tử có quyền ưu tiên thấp nhất thì ở bên dưới cùng của bảng. Trong một biểu thức, các toán tử có quyền ưu tiên cao nhất được tính toán đầu tiên.

Loại	Toán tử	Thứ tự ưu tiên
------	---------	----------------

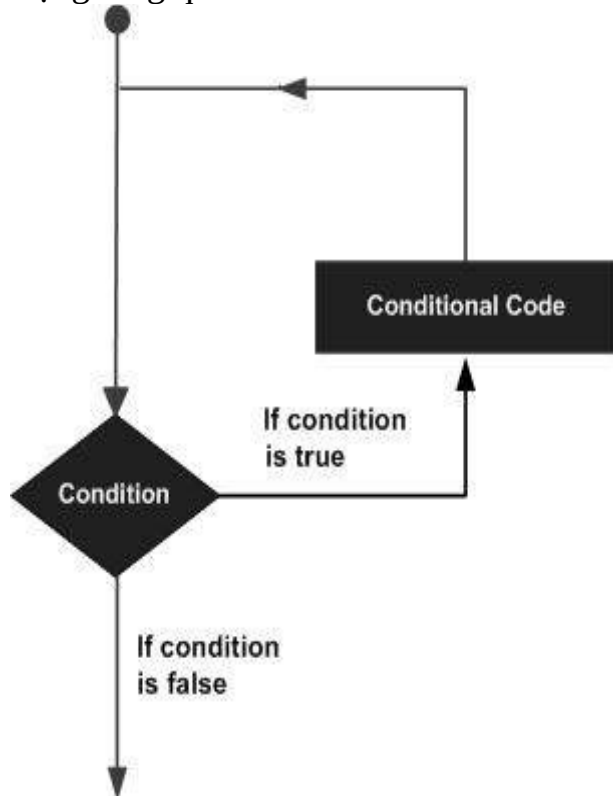
Postfix	() [] -> . ++ --	Trái sang phải
Unary	+ - ! ~ ++ -- (type)* & sizeof	Phải sang trái
Tính nhân	* / %	Trái sang phải
Tính cộng	+ -	Trái sang phải
Dịch chuyển	<< >>	Trái sang phải
Quan hệ	< <= > >=	Trái sang phải
Cân bằng	== !=	Trái sang phải
Phép AND bit	&	Trái sang phải
Phép XOR bit	^	Trái sang phải
Phép OR bit		Trái sang phải
Phép AND logic	&&	Trái sang phải
Phép OR logic		Trái sang phải
Điều kiện	?:	Phải sang trái
Gán	= += -= *= /= %= >>= <<= &= ^=  =	Phải sang trái
Dấu phẩy	,	Trái sang phải

## CHUYÊN ĐỀ 2: VÒNG LẶP TRONG C++

Có một tình huống mà bạn cần phải thực hiện một đoạn code một vài lần. Nhìn chung, các câu lệnh được thực hiện một cách tuần tự. Câu lệnh đầu tiên của hàm được thực hiện trước, sau đó đến câu thứ 2 và tiếp tục.

Ngôn ngữ lập trình cung cấp cho chúng ta nhiều cấu trúc điều khiển và cho phép bạn thực hiện những phần phức tạp.

Vòng lặp cho phép thực hiện một lệnh và một nhóm lệnh nhiều lần , dưới đây là dạng tổng quát:



C++ hỗ trợ vòng lặp sau đây.

Vòng lặp	Miêu tả
<b>Vòng lặp While</b>	Lặp lại một hoặc một nhóm các lệnh trong khi điều kiện đã cho là đúng. Nó kiểm tra điều kiện trước khi thực hiện thân vòng lặp.
<b>Vòng lặp For</b>	Thực thi một dãy các lệnh nhiều lần và tóm tắt đoạn code mà quản lý biến vòng lặp.
<b>Vòng lặp Do ... While</b>	Giống lệnh While, ngoại trừ ở điểm là nó kiểm tra điều kiện ở cuối thân vòng lặp.
<b>Lồng vòng lặp trong C++</b>	Bạn có thể sử dụng một hoặc nhiều vòng lặp trong các vòng lặp while, for hoặc do..while khác.

## 1. Vòng lặp while trong C++

Vòng lặp **while** trong Ngôn ngữ chương trình C/C++ thực hiện lặp đi lặp lại một lệnh mục tiêu đến khi nào điều kiện đã cho còn là đúng.

### Cú pháp

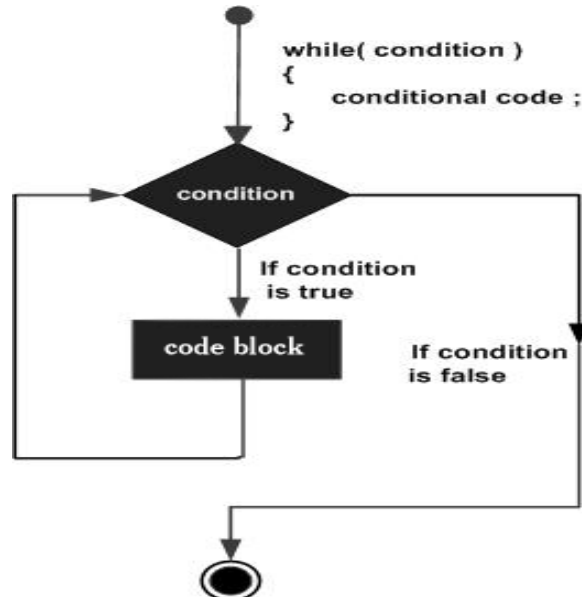
Cú pháp của vòng lặp while trong Ngôn ngữ chương trình C/C++ là:

```
while(dieu_kien)
{
    cac_lenh;
}
```

Ở đây, **cac\_lenh** có thể là lệnh đơn hoặc một khối các lệnh. **dieu\_kien** có thể là bất kỳ biểu thức nào, và giá trị true là bất kỳ giá trị nào khác 0. Vòng lặp lặp đi lặp lại trong khi dieu\_kien là true.

Khi điều kiện trở thành false, chương trình điều khiển ngay lập tức chuyển tới dòng lệnh ngay sau vòng lặp.

### Sơ đồ



Ở đây, điểm chính của vòng lặp **while** là nó có thể không chạy. Bởi vì khi kiểm tra điều kiện và kết quả là false, phần thân vòng lặp được bỏ qua và lệnh đầu tiên ngay sau vòng lặp sẽ được thực thi.

### Ví dụ

```
#include <iostream>
using namespace std;
```

```

int main ()
{
    // Khai bao bien cuc bo:
    int a = 4;
    // vong lap while
    while( a < 10 )
    {
        cout << "Gia tri cua a la: " << a << endl;
        a++;
    }
    return 0;
}

```

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

```

Gia tri cua a la: 4
Gia tri cua a la: 5
Gia tri cua a la: 6
Gia tri cua a la: 7
Gia tri cua a la: 8
Gia tri cua a la: 9
-----

```

## 2. Vòng lặp for trong C++

Vòng lặp for trong C++ là một cấu trúc điều khiển lặp đi lặp lại mà cho phép bạn viết một vòng lặp một cách hiệu quả, mà cần thực hiện trong một khoảng thời gian cụ thể nào đó.

### Cú pháp

Cú pháp của một vòng lặp for trong Ngôn ngữ chương trình C++ là:

```

for ( bien; dieu_kien; tang_giam )
{
    cac_lenh;
}

```

Dưới đây là miêu tả dòng điều khiển trong một vòng lặp for:

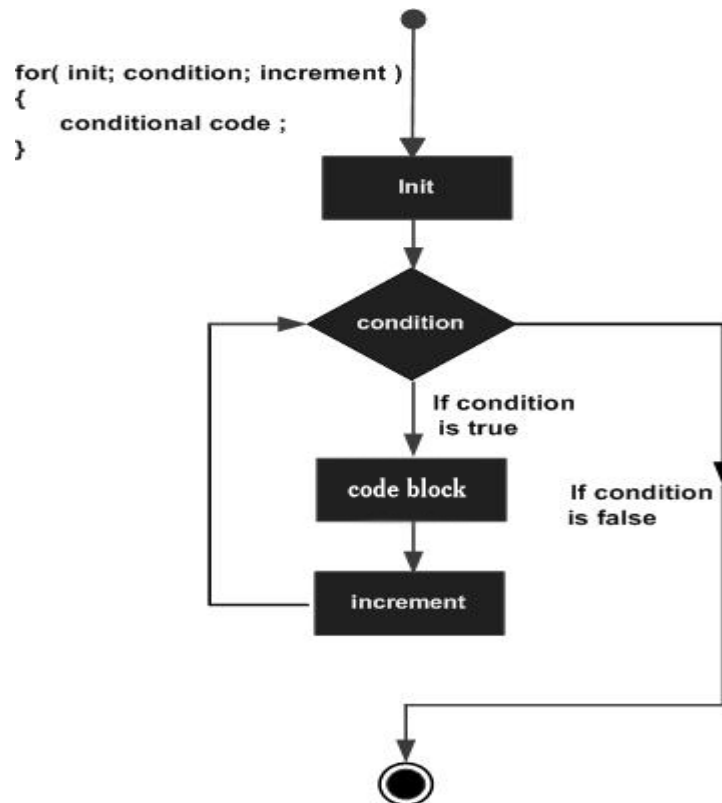
- Bước **bien** được thực hiện đầu tiên và chỉ một lần. Bước này cho phép bạn khai báo và khởi tạo bất kỳ biến điều khiển vòng lặp nào. Bạn không được yêu cầu để đặt một lệnh ở đây, miễn là một dấu chấm phẩy xuất hiện.
- Tiếp theo, **dieu\_kien** được ước lượng. Nếu điều kiện là true, phần thân vòng lặp được thực thi. Nếu nó là false, phần thân vòng lặp không được thực thi và dòng điều khiển nhảy tới lệnh tiếp theo ngay sau vòng lặp for.
- Sau khi phần thân vòng lặp for thực thi, dòng điều khiển nhảy tới lệnh tang\_giam. Lệnh này cho phép bạn cập nhật bất kỳ biến điều khiển vòng lặp



nào. Lệnh này có thể để trống, miễn là một dấu chấm phẩy xuất hiện sau điều kiện.

- `dieu_kien` bây giờ được ước lượng lần nữa. Nếu là `true`, vòng lặp thực thi và tiến trình lặp đi lặp lại chính nó (phần thân vòng lặp, sau đó là `tang_giam`, và sau đó kiểm tra điều kiện lần nữa). Sau khi điều kiện trở thành `false`, vòng lặp for kết thúc.

### Sơ đồ



### Ví dụ

```
#include <iostream>
using namespace std;
int main ()
{
    // vong lap for
    for( int a = 5; a < 15; a = a + 1 )
    {
        cout << "Gia tri cua a la: " << a << endl;
    }
}
```

```
return 0;  
}
```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
Gia tri cua a la: 5  
Gia tri cua a la: 6  
Gia tri cua a la: 7  
Gia tri cua a la: 8  
Gia tri cua a la: 9  
Gia tri cua a la: 10  
Gia tri cua a la: 11  
Gia tri cua a la: 12  
Gia tri cua a la: 13  
Gia tri cua a la: 14
```

Không giống như các vòng lặp **for** và **while**, mà kiểm tra điều kiện vòng lặp ở ngay bước đầu tiên của vòng lặp, vòng lặp **do...while** trong Ngôn ngữ C++ kiểm tra điều kiện của nó tại phần cuối của vòng lặp.

### 3. Vòng lặp Do...While trong C++

Vòng lặp **do...while** là tương tự như vòng lặp **while**, ngoại trừ ở điểm một vòng lặp **do...while** bảo đảm thực hiện vòng lặp ít nhất một lần.

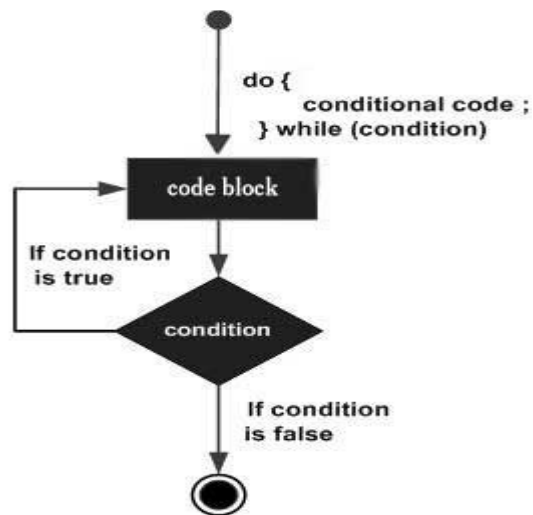
#### Cú pháp

Cú pháp của một vòng lặp **do...while** trong Ngôn ngữ chương trình C++ là:

```
do  
{  
    cac_lenh;  
}while( dieu_kien );
```

- Bạn chú ý rằng, biểu thức điều kiện xuất hiện ở cuối cùng của vòng lặp, vì thế các lệnh trong vòng lặp thực hiện một lần trước khi điều kiện được kiểm tra.
- Nếu điều kiện là true, dòng điều khiển vòng lặp quay trở lại, và các lệnh trong vòng lặp được thực hiện lần nữa. Tiến trình này lặp đi lặp lại tới khi nào điều kiện đã cho trở thành false.

#### Sơ đồ



Ví dụ

```
#include <iostream>
using namespace std;
int main ()
{
    // Khai bao bien cuc bo:
    int a = 5;
    // Vong lap do...while
    do
    {
        cout << "Gia tri cua a la: " << a << endl;
        a = a + 1;
    }while( a < 15 );
    return 0;
}
```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
Gia tri cua a la: 5
Gia tri cua a la: 6
Gia tri cua a la: 7
Gia tri cua a la: 8
Gia tri cua a la: 9
Gia tri cua a la: 10
Gia tri cua a la: 11
Gia tri cua a la: 12
Gia tri cua a la: 13
Gia tri cua a la: 14
-----
```

#### 4. Lồng vòng lặp trong C++

Ngôn ngữ chương trình C++ cho phép bạn sử dụng một vòng lặp bên trong một vòng lặp. Dưới đây là một số ví dụ minh họa khái niệm này.

##### Cú pháp

Cú pháp để **lồng vòng lặp for** trong C++ như sau:

```
for ( bien; dieu_kien; tang_giam )
{
    for ( bien; dieu_kien; tang_giam )
    {
        cac_lenh;
    }
    cac_lenh; // bạn có thể đặt nhiều lệnh tại đây.
}
```

Cú pháp để **lồng vòng lặp while** trong C++ như sau:

```
while(dieu_kien)
{
    while(dieu_kien)
    {
        cac_lenh;
    }
    cac_lenh; // bạn có thể đặt nhiều lệnh tại đây.
}
```

Cú pháp để **lồng vòng lặp do...while** trong C++ như sau:

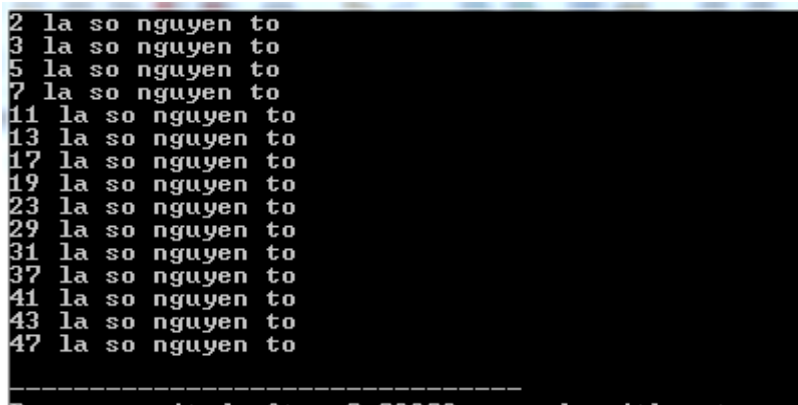
```
do
{
    cac_lenh; // bạn có thể đặt nhiều lệnh tại đây.
    do
    {
        cac_lenh;
    }while( dieu_kien );
}while( dieu_kien );
```

Ví dụ

Chương trình sau sử dụng lồng vòng lặp for để tìm các số nguyên tố từ 2 đến 50:

```
#include <iostream>
using namespace std;
int main ()
{
    int i, j;
    for(i=2; i<50; i++) {
        for(j=2; j <= (i/j); j++)
            if(!(i%j)) break; // neu tim thay he so, thi khong la so nguyen to
        if(j > (i/j)) cout << i << " la so nguyen to\n";
    }
    return 0;
}
```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:



### 5. Các lệnh điều khiển vòng lặp trong C++

Các lệnh điều khiển vòng lặp thay đổi sự thực thi lệnh từ dãy thông thường của nó. Khi sự thực thi lệnh rời khỏi một phạm vi, tất cả các đối tượng tự động mà được tạo ra trong phạm vi đó bị hủy.

C++ hỗ trợ các lệnh điều khiển vòng lặp sau đây.

Lệnh điều khiển	Miêu tả
Lệnh Break trong C++	Kết thúc <b>vòng lặp</b> hoặc lệnh <b>switch</b> và chuyển sang thực thi vòng lặp hoặc lệnh switch ngay sau nó.

<b>Lệnh Continue trong C++</b>	Khi gặp lệnh này thì chương trình sẽ bỏ qua các câu lệnh ở dưới nó (trong cùng một câu lệnh lặp) để thực hiện vòng lặp mới.
<b>Lệnh Goto trong C++</b>	Chuyển tới lệnh được gán. Mặc dù vậy, nó được khuyên rằng không nên sử dụng lệnh goto trong chương trình của bạn.

### 5.1 Lệnh break trong C++

Lệnh break trong C++ có hai cách sử dụng:

- Khi lệnh **break** được sử dụng trong vòng lặp, vòng lặp ngay lập tức kết thúc và điều khiển chương trình bắt đầu lệnh tiếp theo sau vòng lặp.
- Nó có thể được sử dụng trong lệnh **switch** (sẽ được nhắc đến trong chương tới).

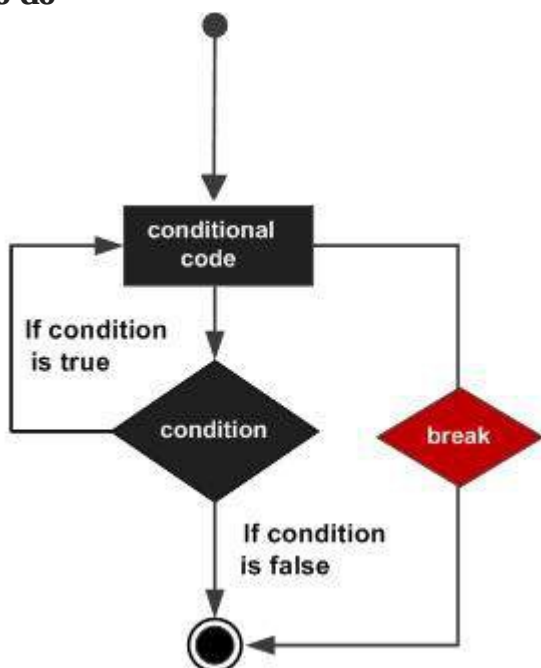
Nếu bạn đang sử dụng các vòng lặp lồng nhau (ví dụ, một vòng lặp bên trong vòng lặp khác), lệnh break sẽ dừng thực thi một lệnh nào đó trong một vòng lặp và bắt đầu thực thi lệnh tiếp theo của đoạn code sau khối code đó.

#### Cú pháp

Cú pháp của lệnh break trong C++ như sau:

```
break;
```

#### Sơ đồ



#### Ví dụ

```
#include <iostream>
using namespace std;
```

```

int main ()
{
    // Khai bao bien cuc bo:
    int a = 10;

    // Vong lap do...while
    do
    {
        cout << "Gia tri cua a la: " << a << endl;
        a = a + 1;
        if( a > 15)
        {
            // Ket thuc vong lap
            break;
        }
    }while( a < 20 );

    return 0;
}

```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```

Gia tri cua a la: 10
Gia tri cua a la: 11
Gia tri cua a la: 12
Gia tri cua a la: 13
Gia tri cua a la: 14
Gia tri cua a la: 15
=====

```

## 5.2: Lệnh continue trong C++

Lệnh **continue** trong C++ làm việc hơi giống với lệnh break. Thay vì bắt buộc kết thúc, nó bắt buộc vòng lặp tiếp theo diễn ra, bỏ qua bất kỳ đoạn code nào ở giữa.

Với vòng lặp **for**, lệnh continue làm cho bước kiểm tra điều kiện và phần tang\_giam của vòng lặp thực thi. Với **while** và **do...while**, lệnh continue làm điều khiển chương trình chuyển tới các kiểm tra điều kiện.

### Cú pháp

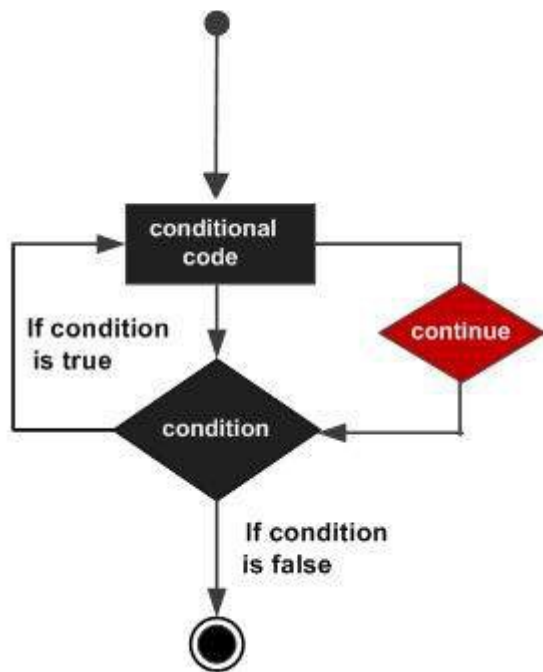
Cú pháp của lệnh continue trong C++ như sau:

```

continue;

```

### Sơ đồ



Ví dụ

```
#include <iostream>
using namespace std;
int main ()
{
    // Khai bao bien cuc bo:
    int a = 10;
    // vong lap do...while
    do
    {
        if( a == 15)
        {
            // nhay qua buoc lap.
            a = a + 1;
            continue;
        }
        cout << "Gia tri cua a la: " << a << endl;
        a = a + 1;
    }while( a < 20 );
```



```
return 0;  
}
```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
Gia tri cua a la: 10  
Gia tri cua a la: 11  
Gia tri cua a la: 12  
Gia tri cua a la: 13  
Gia tri cua a la: 14  
Gia tri cua a la: 16  
Gia tri cua a la: 17  
Gia tri cua a la: 18  
Gia tri cua a la: 19  
-----
```

### 5.3: Lệnh goto trong C++

Lệnh **goto** trong Ngôn ngữ chương trình C++ cung cấp một bước nhảy không điều kiện từ lệnh goto tới lệnh được gán nhãn trong cùng một hàm.

**Ghi chú:** Sử dụng lệnh **goto** gây khó khăn cho bất kỳ ngôn ngữ chương trình nào bởi vì nó gây khó khăn cho việc theo dấu dòng điều khiển của một chương trình, làm cho chương trình khó để hiểu và khó để chỉnh sửa. Bất kỳ chương trình nào sử dụng một lệnh goto có thể được viết lại để có thể không cần lệnh goto này.

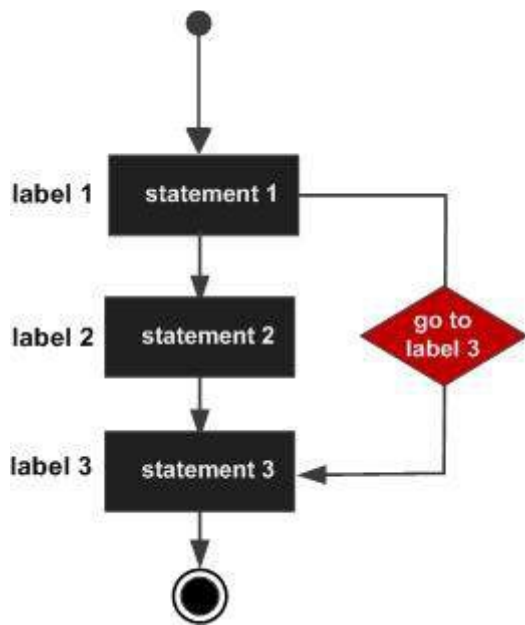
#### Cú pháp

Cú pháp của lệnh goto trong C++ như sau:

```
goto label;  
..  
.  
label: lenh;
```

Ở đây, **label** có thể là bất kỳ phần thuần văn bản nào ngoại trừ các từ khóa trong C++, và nó có thể được thiết lập bất cứ đâu trong chương trình C++, bên trên hoặc dưới lệnh goto này.

#### Sơ đồ



Ví dụ

```
#include <iostream>
using namespace std;

int main ()
{
    // Khai bao bien cuc bo:
    int a = 10;

    // Vong lap do...while
    VONGLAP:do // tai day lable la VONGLAP, va lenh la do
    {
        if( a == 15)
        {
            // nhay qua buoc lap.
            a = a + 1;
            goto VONGLAP;
        }
        cout << "Gia tri cua a la: " << a << endl;
        a = a + 1;
    }while( a < 20 );

    return 0;
}
```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
Gia tri cua a la: 10
Gia tri cua a la: 11
Gia tri cua a la: 12
Gia tri cua a la: 13
Gia tri cua a la: 14
Gia tri cua a la: 16
Gia tri cua a la: 17
Gia tri cua a la: 18
Gia tri cua a la: 19
```

Một sự sử dụng tốt của lệnh `goto` là để thoát khỏi một vòng lặp sâu. Ví dụ, bạn xét code sau:

```
for(...) {
    for(...) {
        while(...) {
            if(...) goto stop;
            .
            .
            .
        }
    }
}
stop:
cout << "Xuat hien loi trong chuong trinh.\n";
```

Với việc không có **goto**, chương trình sẽ thực hiện thêm nhiều kiểm nghiệm bổ sung. Trong khi đó, một lệnh **break** sẽ không nên được dùng ở đây, bởi vì nó sẽ chỉ làm chương trình thoát khỏi vòng lặp trong cùng.

## 6. Vòng lặp vô hạn trong C++

Một vòng lặp là vòng lặp vô hạn khi một điều kiện không bao giờ false. Vòng lặp **for** thường được sử dụng cho mục đích này. Khi bạn để ba biểu thức điều kiện trong vòng lặp *for* trống thì bạn sẽ tạo ra một vòng lặp vô hạn.

```
#include <iostream>
using namespace std;

int main ()
{
    for(;;)
    {
        printf("Vong lap nay se chay mai mai.\n");
    }
    return 0;
}
```

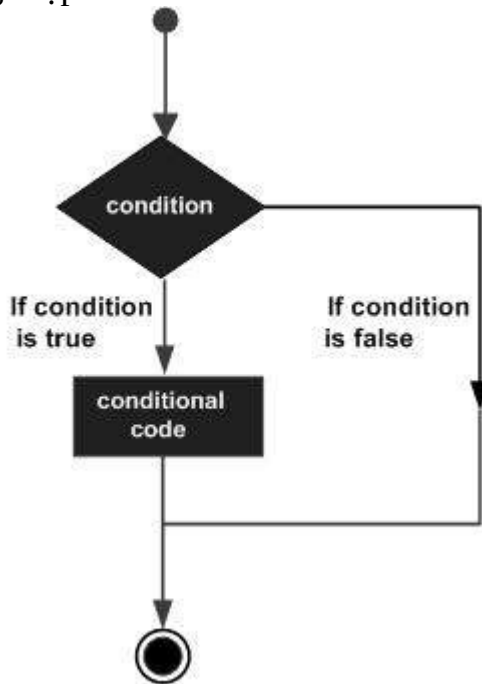
Khi biểu thức điều kiện vắng mặt, nó được giả sử là luôn đúng. Bạn có thể có một biểu thức khởi tạo và biểu thức tăng, giảm, nhưng các lập trình viên C++ thường sử dụng `for(;;)` để biểu thị một vòng lặp vô hạn.

**Ghi chú:** Bạn có thể dừng (kết thúc) một vòng lặp vô hạn bởi nhấn Ctrl + C.

### CHUYÊN ĐỀ 3: LỆNH ĐIỀU KHIỂN LƯỒNG (LỆNH ĐIỀU KIỆN) TRONG C++

Các cấu trúc điều khiển luồng yêu cầu lập trình viên xác định một hoặc nhiều điều kiện để được đánh giá và kiểm tra bởi chương trình, cùng với các lệnh được thực hiện nếu điều kiện được xác định là đúng, hoặc các lệnh khác được thực hiện nếu điều kiện xác định là sai.

Dưới đây là mẫu chung của một cấu trúc điều khiển luồng hay gặp trong ngôn ngữ lập trình.



Dưới đây liệt kê các lệnh điều khiển luồng được cung cấp bởi C++.

Lệnh	Miêu tả
<b>Lệnh if trong C++</b>	Một lệnh if bao gồm một biểu thức logic theo sau bởi một hoặc nhiều lệnh khác.
<b>Lệnh if...else trong C++</b>	Một lệnh if có thể theo sau bởi một lệnh else (tùy ý: có hoặc không), mà có thể được thực hiện khi biểu thức logic có giá trị false.
<b>Lệnh switch trong C++</b>	Một lệnh switch cho phép kiểm tra điều kiện của một biến trước khi thực thi các lệnh.
<b>Lồng các lệnh if trong C++</b>	Bạn có thể sử dụng lệnh if hoặc else if bên trong lệnh if hoặc else if khác.
<b>Lồng các lệnh switch trong C++</b>	Bạn có thể sử dụng một lệnh switch bên trong một lệnh switch khác.

**Toán tử điều kiện ? trong C++**

Chúng ta đã bàn về **Toán tử điều kiện ?** trong chương trước mà có thể được dùng để thay thế cho lệnh **if...else**. Nó có mẫu chung như sau:

```
Exp1 ? Exp2 : Exp3;
```

Trong đó Exp1, Exp2 và Exp3 là các biểu thức. Chú ý việc sử dụng và đặt của dấu hai chấm.

Giá trị của biểu thức Exp1 trước dấu ? có giá trị **true**, Exp2 được thực hiện, và giá trị của nó là giá trị của biểu thức. Nếu Exp1 là **false** thì Exp3 được thực hiện và giá trị của nó là giá trị của biểu thức.

### 1. Lệnh if trong C++

Một lệnh **if** trong Ngôn ngữ C++ chứa một biểu thức boolean được theo sau bởi một hoặc nhiều lệnh.

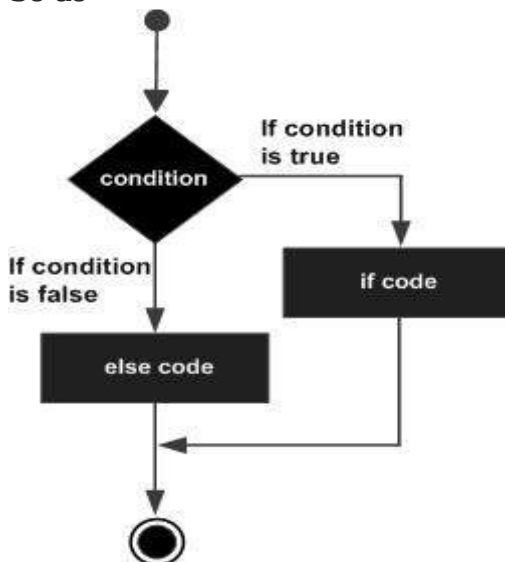
Cú pháp

Sau đây là cú pháp của một lệnh if trong C++:

```
if(bieu_thuc_boolean)
{
    // cac lenh se duoc thuc thi neu bieu thuc boolean la true
}
```

Nếu biểu thức boolean được ước lượng là **true**, thì sau đó khối code bên trong lệnh if sẽ được thực thi. Nếu biểu thức boolean được ước lượng là **false**, thì khi đó, lệnh ngay sau lệnh if sẽ được thực thi.

Sơ đồ



Ví dụ

```
#include <iostream>
using namespace std;
int main ()
{
    // Khai bao bien cuc bo:
```

```

int a = 10;
// kiểm tra điều kiện của biểu thức boolean
if( a < 20 )
{
    // Nếu điều kiện là true thì in dòng sau
    cout << "a là nhỏ hơn 20." << endl;
}
cout << "Giá trị của a là: " << a << endl;
return 0;
}

```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```

a là nhỏ hơn 20.
Giá trị của a là: 10

```

## 2. Lệnh if...else trong C++

Lệnh **if** trong C++ có thể được theo sau bởi một lệnh **else** tùy ý, mà thực hiện khi biểu thức boolean là false.

### Cú pháp

Sau đây là cú pháp của một lệnh if...else trong C++:

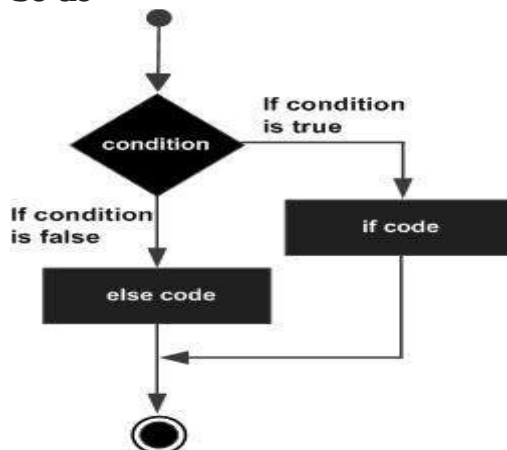
```

if(bieu_thuc_boolean)
{
    // các lệnh sẽ được thực thi nếu biểu thức boolean là true
}
else
{
    // các lệnh sẽ được thực thi nếu biểu thức boolean là false
}

```

Nếu biểu thức boolean được ước lượng là **true**, thì khi đó khối **if** sẽ được thực thi, nếu không thì khối **else** sẽ được thực thi.

### Sơ đồ



### Ví dụ

```

#include <iostream>
using namespace std;

int main ()
{
    // Khai bao bien cuc bo:
    int a = 100;

    // kiem tra dieu kien cua bieu thuc boolean
    if( a < 20 )
    {
        // Neu dieu kien la true thi in dong sau
        cout << "a la nho hon 20." << endl;
    }
    else
    {
        // Neu dieu kien la false thi in dong sau
        cout << "a khong nho hon 20." << endl;
    }
    cout << "Gia tri cua a la: " << a << endl;

    return 0;
}

```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```

a la nho hon 20.
Gia tri cua a la: 100

```

### 3. Lệnh if...else if...else trong C++

Một lệnh **if** có thể được theo sau bởi một lệnh **else if...else** tùy ý, mà rất có ích để kiểm tra các điều kiện đa dạng.

Khi sử dụng lệnh if, else if, else, có một số điểm chính cần ghi nhớ:

- Một lệnh if có 0 hoặc một lệnh else và chúng phải theo sau bởi bất kỳ lệnh else if nào.
- Một lệnh if có 0 tới nhiều lệnh else if và chúng phải ở trước lệnh else.
- Một khi lệnh else if thực hiện, nó sẽ không kiểm tra lại bất kỳ lệnh else if hoặc lệnh else còn lại khác.

Cú pháp

Cú pháp của một lệnh if...else if...else trong Ngôn ngữ C++ như sau:

```

if(bieu_thuc_boolean 1)
{
    // thuc thi khi bieu thuc boolean 1 la true
}

```



```

else if( bieu_thuc_boolean 2)
{
    // thuc thi khi bieu thuc boolean 2 la true
}
else if( bieu_thuc_boolean 3)
{
    // thuc thi khi bieu thuc boolean 3 la true
}
else
{
    // thuc thi khi tat ca cac dieu kien tren khong la true.
}

```

Ví dụ

```

#include <iostream>
using namespace std;
int main ()
{
    // Khai bao bien cuc bo:
    int a = 100;

    // kiem tra dieu kien cua bieu thuc boolean
    if( a == 10 )
    {
        // Neu dieu kien la true thi in dong sau
        cout << "Gia tri cua a la 10" << endl;
    }
    else if( a == 20 )
    {
        // neu dieu kien else if la true
        cout << "Gia tri cua a la 20" << endl;
    }
    else if( a == 30 )
    {
        // eu dieu kien else if la true
        cout << "Gia tri cua a la 30" << endl;
    }
    else
    {
        // neu cac dieu kien tren khong la true thi in dong sau
        cout << "Gia tri cua a khong ket noi voi cac dieu kien tren" << endl;
    }
    cout << "Gia tri chinh xac cua a la: " << a << endl;

    return 0;
}

```

```
}
```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
Gia tri cua a khong ket noi voi cac dieu kien tren
Gia tri chinh xac cua a la: 100
=====
```

#### 4. Lệnh switch case trong C/C++.

**[switch/case trong C/C++]** Một lệnh **switch** trong C/C++ cho một biến được kiểm tra một cách bình đẳng trong danh sách các giá trị. Mỗi giá trị được gọi là một **case - trường hợp** và biến được chuyển tới được kiểm tra cho mỗi trường hợp switch.

Cú pháp

Cú pháp của lệnh switch trong Ngôn ngữ C/C++ như sau:

```
switch(bieu_thuc){
    case bieu_thuc_hang :
        statement(s);
        break; //optional
    case bieu_thuc_hang :
        statement(s);
        break; //optional

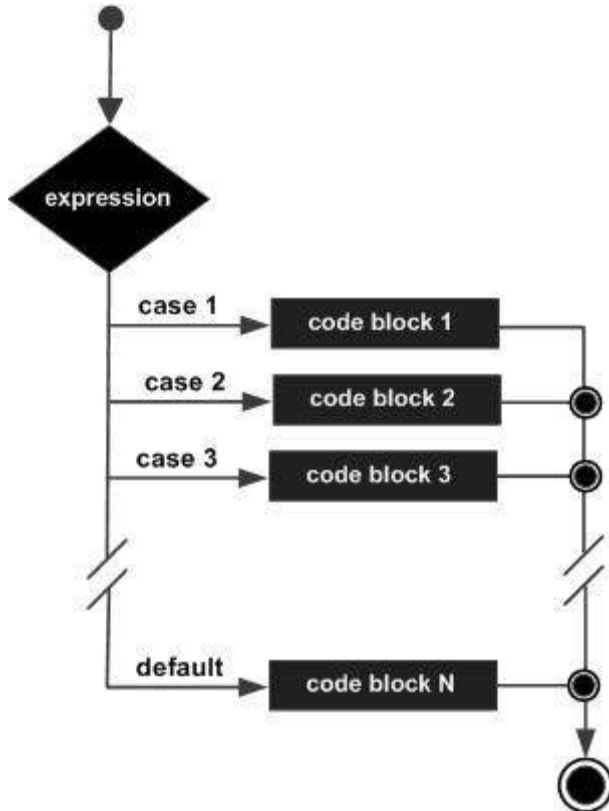
    // you can have any number of case statements.
    default : //Optional
        statement(s);
}
```

Các quy tắc sau được áp dụng tới một lệnh **switch**:

- Biểu thức **bieu\_thuc** được sử dụng trong một lệnh switch phải có kiểu là integer hoặc liệt kê, hoặc là một trong các kiểu lớp trong đó lớp có một hàm biến đổi đơn tới một kiểu integer hoặc kiểu liệt kê.
- Bạn có thể có bất kỳ số lệnh case nào trong một switch. Mỗi case được theo sau bởi giá trị để được so sánh và một dấu hai chấm.
- **bieu\_thuc\_hang**, là biểu thức hằng, cho một case phải cùng kiểu dữ liệu với biến trong switch, và nó phải là hằng số.
- Khi biến được chuyển tới là cân bằng với một case, lệnh theo sau case đó sẽ thực thi tới khi gặp lệnh **break**.
- Khi gặp lệnh **break**, switch kết thúc, và dòng điều khiển nhảy tới dòng lệnh tiếp theo của lệnh switch đó.
- Không phải mỗi case cần chứa một lệnh **break**. Nếu không có lệnh **break** nào xuất hiện, dòng điều khiển sẽ **không tới được** case tiếp theo cho tới khi bắt gặp một lệnh break.

- Một lệnh **switch** có thể có một case **mặc định** tùy chọn, mà phải xuất hiện ở cuối cùng của switch. Case mặc định này có thể được sử dụng để thực hiện một nhiệm vụ khi không có case nào true. Trong trường hợp case mặc định này thì không cần lệnh **break**.

Sơ đồ



Ví dụ

```
#include <iostream>
using namespace std;

int main ()
{
    // Khai bao bien cuc bo:
    char hocluc = 'D';
    switch(hocluc)
    {
        case 'A' :
            cout << "Gioi!" << endl;
            break;
        case 'B' :
        case 'C' :
            cout << "Kha" << endl;
            break;
        case 'D' :
            cout << "Trung binh" << endl;
```

```

    break;
case 'F' :
    cout << "Phai hoc lai!!" << endl;
    break;
default :
    cout << "Gia tri khong hop le" << endl;
}
cout << "Hoc luc cua ban la " << hocluc << endl;

return 0;
}

```

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

```

Trung binh
Hoc luc cua ban la D
-----

```

### 5. Lồng các lệnh if trong C++.

Nó là hợp lệ để lồng các lệnh **if-else** trong C++, nghĩa là bạn có thể sử dụng một lệnh if hoặc else bên trong lệnh if hoặc else khác.

Cú pháp

Cú pháp để lồng các lệnh **if** trong C++ như sau:

```

if( bieu_thuc_boolean 1)
{
    // Thuc thi khi bieu thuc boolean 1 la true
    if(bieu_thuc_boolean 2)
    {
        // Thuc thi khi bieu thuc boolean 2 la true
    }
}

```

Bạn có thể lồng **else if...else** theo cách tương tự như bạn đã lồng lệnh if.

Ví dụ

```

#include <iostream>
using namespace std;

int main ()
{
    // Khai bao bien cuc bo:
    int a = 100;
    int b = 200;

    // kiem tra dieu kien cua bieu thuc boolean
    if( a == 100 )
    {
        // neu dieu kien la true thi kiem tra tiep dieu kien sau
    }
}

```

```

if( b == 200 )
{
    // neu dieu kien la true thi in dong sau
    cout << "Gia tri cua a la 100 va b la 200" << endl;
}
}
cout << "Gia tri chinh xac cua a la: " << a << endl;
cout << "Gia tri chinh xac cua b la: " << b << endl;
return 0;
}

```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```

Gia tri cua a la 100 va b la 200
Gia tri chinh xac cua a la: 100
Gia tri chinh xac cua b la: 200
-----

```

## 6. Lồng các lệnh switch trong C++

Nó là có thể để có một lệnh switch như là một phần của dãy lệnh trong một lệnh switch ở vòng ngoài. Ngay cả khi hằng số case trong và ngoài lệnh switch chứa các giá trị bình thường, sẽ không có sự xung đột diễn ra ở đây.

C++ xác định có ít nhất 256 mức độ lồng cho lệnh switch được cho phép.

Cú pháp

Cú pháp để lồng các lệnh **switch** trong C++ như sau:

```

switch(ch1) {
    case 'A':
        cout << "A nay la mot phan cua lenh switch ben ngoai";
        switch(ch2) {
            case 'A':
                cout << "A nay la mot phan cua lenh switch ben trong";
                break;
            case 'B': // ...
        }
        break;
    case 'B': // ...
}

```

Ví dụ

```

#include <iostream>
using namespace std;
int main ()
{
    // Khai bao bien cuc bo:
    int a = 100;
    int b = 200;
}

```

```

switch(a) {
    case 100:
        cout << "Day la mot phan cua lenh switch ben ngoai" << endl;
        switch(b) {
            case 200:
                cout << "Day la mot phan cua lenh switch ben trong" << endl;
            }
        }
    cout << "Gia tri chinh xac cua a la: " << a << endl;
    cout << "Gia tri chinh xac cua b la: " << b << endl;

    return 0;
}

```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```

Day la mot phan cua lenh switch ben ngoai
Day la mot phan cua lenh switch ben trong
Gia tri chinh xac cua a la: 100
Gia tri chinh xac cua b la: 200
-----

```

## CHUYÊN ĐỀ 4: HÀM TRONG C/C++

Một hàm là một nhóm các lệnh đi cùng nhau để thực hiện một nhiệm vụ. Mỗi chương trình C/C++ có ít nhất một hàm là hàm **main()**, và tất cả hầu hết các chương trình bình thường đều định nghĩa thêm các hàm.

Bạn có thể chia đoạn code của bạn thành những hàm riêng biệt. Cách bạn chia đoạn code của bạn thành các hàm khác nhau phụ thuộc vào bạn, nhưng theo tính logic, một hàm thường có một nhiệm vụ nhất định.

Một sự **khai báo** hàm thông báo với bộ biên dịch về tên của hàm, kiểu trả về và tham số. Một **định nghĩa** hàm cung cấp phần thân của một hàm.

Các thư viện tiêu chuẩn của ngôn ngữ C/C++ cung cấp rất nhiều hàm có sẵn để chương trình của bạn có thể gọi. Ví dụ, hàm **strcat()** có thể nối hai đoạn chuỗi, hàm **memcpy()** dùng để copy một vùng nhớ đến một vùng nhớ khác và rất nhiều hàm khác nữa.

Một hàm được biết đến với các tên khác nhau như một phương thức, một tuyến phụ hoặc một thủ tục.

### 1. Định nghĩa một hàm trong C/C++

Mẫu chung của định nghĩa hàm trong Ngôn ngữ C/C++ như sau:

```
Kieu_tra_ve Ten_ham( Danh_sach_tham_so )
{
    Than_ham
}
```

Một định nghĩa hàm trong ngôn ngữ C/C++ bao gồm *đầu hàm* và một *thân hàm*. Dưới đây là các phần của một hàm:

- **Kiểu trả về:** Một hàm có thể trả về một giá trị. **Kieu\_tra\_ve** là dạng dữ liệu của giá trị mà hàm trả về. Vài hàm cung cấp các hoạt động và không trả về giá trị nào cả. Đó là hàm **void**.
- **Tên hàm:** Đây là tên thực sự của hàm. Tên hàm và danh sách tham số cấu tạo nên dấu hiệu hàm.
- **Danh sách tham số:** Khi hàm được gọi, bạn phải truyền vào danh sách các tham số. Một giá trị hướng đến một tham số thực tế. Danh sách tham số có các kiểu, thứ tự và số lượng các tham số của hàm. Các tham số trong hàm là tùy chọn, nghĩa là một hàm có thể không có tham số.
- **Thân hàm:** Phần thân của một hàm bao gồm tập hợp các lệnh xác định những gì mà hàm thực hiện.

Ví dụ

Sau đây phần code cho một hàm có tên gọi là **max()**. Hàm này có 2 tham số: **so1** và **so2** và trả về giá trị lớn nhất giữa hai số:

```
// ham tra ve so lon nhat cua hai so
int max(int so1, int so2)
{
    // Khai bao bien cuc bo
    int result;
    if (so1 > so2)
        result = so1;
    else
        result = so2;
    return result;
}
```

## 2. Khai báo hàm trong C/C++

Một **khai báo** hàm thông báo cho trình biên dịch về tên hàm và cách gọi của hàm. Phần thân hàm có thể định nghĩa một cách rời rạc.

Một khai báo hàm có các phần sau đây:

```
kieu_tra_ve ten_ham( danh sach tham so );
```

Ví dụ khi định nghĩa hàm max(), dưới đây là câu khai báo hàm:

```
int max(int so1, int so2);
```

Tên các tham số không quan trọng trong việc khai báo hàm, và kiểu dưới đây là cách khai báo hợp lệ:

```
int max(int, int);
```

Một khai báo hàm được yêu cầu khi bạn định nghĩa một hàm và mã nguồn và khi gọi một hàm từ một file nguồn khác. Trong trường hợp này, bạn nên khai báo hàm trước khi gọi hàm đó.

Gọi hàm trong C/C++

Trong khi tạo một hàm, bạn định nghĩa những gì hàm phải làm. Để sử dụng một hàm, bạn phải gọi hàm đó để thực hiện một nhiệm vụ cụ thể.

Khi một chương trình gọi một hàm, phần điều khiển được chuyển đến hàm được gọi. Một hàm được gọi thực hiện các nhiệm vụ được định nghĩa và trả về giá trị sau khi thực hiện chương trình.

Để gọi hàm, bạn đơn giản cần truyền các tham số được yêu cầu cùng với tên của hàm và nếu hàm trả về các giá trị, bạn có thể dự trữ các giá trị trả về này, ví dụ:

```
#include <iostream>
using namespace std;
// khai bao ham
int max(int so1, int so2);
int main ()
{
```



```

// Khai bao bien cuc bo:
int a = 100;
int b = 200;
int ketqua;
// goi ham de tim gia tri lon nhat.
ketqua = max(a, b);
cout << "Gia tri lon nhat la: " << ketqua << endl;
return 0;
}
// ham tra ve so lon nhat cua hai so
int max(int so1, int so2)
{
    // Khai bao bien cuc bo
    int result;
    if (so1 > so2)
        result = so1;
    else
        result = so2;
    return result;
}

```

Mình giữ giá trị hàm max() trong hàm main vào biến ketqua. Khi chạy chương trình C/C++ trên sẽ cho kết quả sau:

Gia tri lon nhat la: 200

### 3. Tham số của hàm trong C/C++:

Một hàm sử dụng các danh sách tham số, nó phải khai báo các biến và chấp nhận giá trị các biến này. Các biến này được gọi là các biến chính thức.

Các biến chính thức giống các biến cục bộ khác bên trong hàm.

Khi bạn gọi hàm, có 2 cách để bạn truyền các giá trị vào cho hàm:

Kiểu gọi	Miêu tả
<b>Gọi hàm bởi giá trị trong C/C++</b>	Phương thức này sao chép giá trị thực sự của tham số vào trong tham số chính thức của một hàm. Trong trường hợp này, các thay đổi của bản thân các tham số bên trong hàm không ảnh hưởng tới các tham số.
<b>Gọi hàm bởi con trỏ trong C/C++</b>	Phương thức này sao chép địa chỉ của tham số vào trong biến chính thức. Bên trong hàm này, địa chỉ này được sử dụng để truy cập tham số thực sự được sử dụng trong lời gọi hàm.
<b>Gọi hàm bởi tham chiếu trong C/C++</b>	Phương thức này sao chép địa chỉ của tham số vào trong tham số chính thức. Bên trong hàm, địa chỉ được

	dùng để truy cập tham số thực sự được sử dụng khi gọi hàm. Có nghĩa là các thay đổi tới tham số làm tham số thay đổi.
--	---

Theo mặc định, C/C++ sử dụng **gọi bởi giá trị** để truyền các tham số. Nhìn chung, code đó trong một hàm không thể thay đổi các tham số được dùng để gọi hàm đó và trong ví dụ trên, khi gọi hàm `max()` là dùng phương thức tương tự.

Phương thức **gọi hàm bởi giá trị** của các tham số đã truyền tới một hàm sao chép giá trị thực sự của một tham số vào trong tham biến chính thức của hàm. Trong trường hợp này, các thay đổi được tạo ra tới tham biến bên trong hàm không có ảnh hưởng tới tham số.

### 3.1 Gọi hàm bởi giá trị trong C/C++

Theo mặc định, Ngôn ngữ C++ sử dụng hàm gọi bởi giá trị để truyền các tham số. Hiểu theo nghĩa chung là code bên trong một hàm không thể thay đổi tham số đã sử dụng để gọi hàm. Giả sử định nghĩa hàm **traodoi()** như sau:

```
// phan dinh nghĩa ham de trao doi cac gia tri.
void traodoi(int x, int y)
{
    int temp;
    temp = x; /* luu giu gia tri cua x */
    x = y;    /* dat y vao trong x */
    y = temp; /* dat x vao trong y */
    return;
}
```

Bây giờ chúng ta gọi hàm **traodoi()** bởi truyền các giá trị thực sự như ví dụ sau:

```
#include <iostream>
using namespace std;
// Phan khai bao ham
void traodoi(int x, int y);
int main ()
{
```

```
// Khai bao bien cuc bo:
int a = 100;
int b = 200;

cout << "Truoc khi trao doi, gia tri cua a la: " << a << endl;
cout << "Truoc khi trao doi, gia tri cua b la: " << b << endl;

// goi ham de trao doi cac gia tri.
traodoi(a, b);

cout << "Sau khi trao doi, gia tri cua a la: " << a << endl;
cout << "Sau khi trao doi, gia tri cua b la: " << b << endl;

return 0;
}
```

Bạn đặt phần định nghĩa hàm trên vào cuối đoạn code này, sau đó biên dịch và chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
Truoc khi trao doi, gia tri cua a la: 100
Truoc khi trao doi, gia tri cua b la: 200
Sau khi trao doi, gia tri cua a la: 100
Sau khi trao doi, gia tri cua b la: 200
-----
```

Nó chỉ rằng không có thay đổi về giá trị mặc dù chúng đã được thay đổi bên trong hàm.

### 3.2 Gọi hàm bởi con trỏ trong C/C++

Phương thức **gọi hàm bởi con trỏ** trong C++ truyền các tham số tới một hàm, sao các địa chỉ của một tham số vào trong tham số chính thức. Bên trong hàm, địa chỉ này được sử dụng để truy cập tham số thực sự được sử dụng trong lời gọi hàm. Nghĩa là các thay đổi được tạo ra cho tham số chính thức ảnh hưởng tới tham số đã truyền.

Để truyền giá trị bởi con trỏ, các con trỏ tham số được truyền tới các hàm giống như bất kỳ giá trị khác. Vì thế, bạn cần khai báo các tham số hàm ở dạng kiểu con trỏ như trong hàm **traodoi()** sau, mà trao đổi giá trị của hai biến integer được trỏ bởi tham số của nó.

```
// phan dinh nghĩa ham de trao doi cac gia tri.
void traodoi(int *x, int *y)
{
    int temp;
```

```

temp = *x; /* luu giu gia tri tai dia chi x */
*x = *y; /* dat y vao trong x */
*y = temp; /* dat x vao trong y */
return;
}

```

Bây giờ, gọi hàm **traodoi()** bằng việc truyền các giá trị bởi con trỏ như trong ví dụ sau:

```

#include <iostream>
using namespace std;
// Phan khai bao ham
void traodoi(int *x, int *y);
int main ()
{
    // Khai bao bien cuc bo:
    int a = 100;
    int b = 200;
    cout << "Truoc khi trao doi, gia tri cua a la: " << a << endl;
    cout << "Truoc khi trao doi, gia tri cua b la: " << b << endl;
    /* goi ham traodoi de trao doi cac gia tri cua cac bien.
     * &a chi rang con tro dang tro toi a (dia chi cua bien a) va
     * &b chi rang con tro dang tro toi b (dia chi cua bien b).
     */
    traodoi(&a, &b);
    cout << "Sau khi trao doi, gia tri cua a la: " << a << endl;
    cout << "Sau khi trao doi, gia tri cua b la: " << b << endl;
    return 0;
}

```

Bạn đặt phần định nghĩa hàm trên vào cuối đoạn code này, sau đó biên dịch và chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```

Truoc khi trao doi, gia tri cua a la: 100
Truoc khi trao doi, gia tri cua b la: 200
Sau khi trao doi, gia tri cua a la: 200
Sau khi trao doi, gia tri cua b la: 100

```

### 3.3 Gọi hàm bởi tham chiếu trong C/C++

Phương thức **gọi hàm bởi tham chiếu** của các tham số đã truyền tới một hàm sao chép địa chỉ của một tham số vào trong tham biến chính thức. Bên trong hàm đó, địa chỉ được sử dụng để truy cập tham số thực sự được sử dụng trong gọi hàm. Điều này có nghĩa rằng các thay đổi được tạo ra với tham biến ảnh hưởng tới tham số được truyền.

Để truyền giá trị bởi tham chiếu, các con trỏ tham số được truyền tới các hàm giống như bất kỳ giá trị khác. Vì thế theo đó, bạn cần khai báo các tham số hàm

như là các kiểu con trỏ như trong hàm **traodoi()** như sau, mà thay đổi giá trị của hai biến integer được trỏ tới bởi các tham số của nó.

```
// phân định nghĩa hàm để trao đổi các giá trị.
```

```
void traodoi(int &x, int &y)
{
    int temp;
    temp = x; /* lưu giữ giá trị tại địa chỉ x */
    x = y;    /* đặt y vào trong x */
    y = temp; /* đặt x vào trong y */
    return;
}
```

Bây giờ, chúng ta gọi hàm traodoi() bởi truyền các giá trị bởi tham chiếu như sau:

```
#include <iostream>
using namespace std;
// phân khai báo hàm
void traodoi(int &x, int &y);
int main ()
{
    // Khai báo biến cục bộ:
    int a = 100;
    int b = 200;
    cout << "Trước khi trao đổi, giá trị của a là: " << a << endl;
    cout << "Trước khi trao đổi, giá trị của b là: " << b << endl;
    /* gọi hàm traodoi để trao đổi các giá trị bởi sử dụng tham chiếu biến.*/
    traodoi(a, b);
    cout << "Sau khi trao đổi, giá trị của a là: " << a << endl;
    cout << "Sau khi trao đổi, giá trị của b là: " << b << endl;
    return 0;
}
```

Bạn đặt phần định nghĩa hàm trên vào cuối đoạn code này, sau đó biên dịch và chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
Trước khi trao đổi, giá trị của a là: 100
Trước khi trao đổi, giá trị của b là: 200
Sau khi trao đổi, giá trị của a là: 200
Sau khi trao đổi, giá trị của b là: 100
-----
```

Điều này chỉ ra rằng các thay đổi đã phản ánh bên ngoài hàm, không giống như gọi hàm bởi giá trị mà các thay đổi không phản ánh bên ngoài hàm.

#### 4. Giá trị mặc định cho các tham số trong C/C++

Khi bạn định nghĩa một hàm, bạn có thể xác định một giá trị mặc định cho mỗi tham số cuối cùng. Giá trị này sẽ được sử dụng nếu tham số tương ứng là để trống bên trái khi gọi hàm đó.

Việc này được thực hiện bởi sử dụng toán tử gán và gán các giá trị cho các tham số trong định nghĩa hàm. Nếu một giá trị cho tham số đó không được truyền khi hàm được gọi, thì giá trị mặc định đã cung cấp sẽ được sử dụng, nhưng nếu một giá trị đã được xác định, thì giá trị mặc định này bị bỏ qua và, thay vào đó, giá trị đã truyền được sử dụng. Bạn theo dõi ví dụ sau:

```
#include <iostream>
using namespace std;

int sum(int a, int b=20)
{
    int ketqua;
    ketqua = a + b;
    return (ketqua);
}

int main ()
{
    // Khai bao bien cuc bo:
    int a = 100;
    int b = 200;
    int ketqua;
    // goi ham de tinh tong hai so.
    ketqua = sum(a, b);
    cout << "Tong gia tri la: " << ketqua << endl;
    // goi ham mot lan nua.
    ketqua = sum(a);
    cout << "Tong gia tri la: " << ketqua << endl;
    return 0;
}
```

Chạy chương trình C/C++ trên sẽ cho kết quả sau:

```
Tong gia tri la: 300
Tong gia tri la: 120
```

## **5. Number trong C++**

Thông thường, khi chúng ta làm việc với Number (các kiểu giá trị số), chúng ta sử dụng các kiểu dữ liệu gốc như int, short, long, float và double, .... Các kiểu dữ liệu số, về giá trị có thể và dãy giá trị của chúng, đã được bàn luận trong chương Kiểu dữ liệu trong C++.

### **5.1 Định nghĩa Number trong C++**

Bạn đã thấy định nghĩa các số trong các ví dụ đa dạng ở các chương trước. Dưới đây là một ví dụ tổng hợp để định nghĩa các kiểu số đa dạng trong C++:

```
#include <iostream>
using namespace std;
```

```

int main ()
{
    // phan dinh nghia cac so:
    short s;
    int i;
    long l;
    float f;
    double d;

    // phep gan cho cac so;
    s = 10;
    i = 1000;
    l = 1000000;
    f = 230.47;
    d = 30949.374;

    // in cac so;
    cout << "short s la: " << s << endl;
    cout << "int i la: " << i << endl;
    cout << "long l la: " << l << endl;
    cout << "float f la: " << f << endl;
    cout << "double d la: " << d << endl;
    return 0;
}

```

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

```

short s la: 10
int i la: 1000
long l la: 1000000
float f la: 230.47
double d la: 30949.4

```

## 5.2 Hàm toán học trong C++

Bên cạnh các hàm đa dạng bạn có thể tạo, C++ cũng bao gồm một số hàm toán học hữu ích cho bạn sử dụng. Những hàm này có sẵn trong các thư viện C và C++ chuẩn, và được gọi là các hàm **built-in**. Đây là các hàm mà có thể được bao trong chương trình của bạn và sau đó sử dụng.

C++ có một tập hợp hàm toán học đa dạng, có thể được thực hiện trên các kiểu số khác nhau. Bảng dưới liệt kê một số hàm toán học có sẵn hữu ích trong C++. Để sử dụng các hàm này, bạn cần bao header file là **<cmath>**.

STT	Hàm & Mục đích
1	<b>double cos(double);</b>

	Hàm này trả về cosin của một góc (dạng một double)
2	<b>double sin(double);</b> Hàm này trả về sin của một góc (dạng một double)
3	<b>double tan(double);</b> Hàm này trả về tang của một góc (dạng một double)
4	<b>double log(double);</b> Hàm này trả về logarit tự nhiên (ln) của số đó
5	<b>double pow(double, double);</b> Hàm mũ với cơ số là số double đầu tiên và số mũ là double thứ hai
6	<b>double hypot(double, double);</b> Nếu bạn truyền độ dài của hai cạnh của tam giác (lần lượt là các số double), nó sẽ trả về độ dài cạnh huyền
7	<b>double sqrt(double);</b> Trả về căn bậc hai của số double
8	<b>int abs(int);</b> Trả về trị tuyệt đối của int
9	<b>double fabs(double);</b> Trả về trị tuyệt đối của bất kỳ số double nào
10	<b>double floor(double);</b> Tìm số integer mà nhỏ hơn hoặc bằng tham số đã truyền cho nó

Ví dụ sau minh họa một số hàm toán học trong C++:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    // phân định nghĩa các số:
    short s = 10;
    int i = -1000;
    long l = 100000;
    float f = 230.47;
    double d = 200.374;
```



```
// cac hoat dong toan hoc;
cout << "sin(d)   co gia tri la: " << sin(d) << endl;
cout << "abs(i)   co gia tri la: " << abs(i) << endl;
cout << "floor(d) co gia tri la: " << floor(d) << endl;
cout << "sqrt(f)   co gia tri la: " << sqrt(f) << endl;
cout << "pow( d, 2) co gia tri la: " << pow(d, 2) << endl;
return 0;
}
```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
sin(d)      co gia tri la: -0.634939
abs(i)      co gia tri la: 1000
floor(d)    co gia tri la: 200
sqrt(f)     co gia tri la: 15.1812
pow( d, 2)  co gia tri la: 40149.7
-----
```

### Số ngẫu nhiên (Random Number) trong C++

Trong một số trường hợp, bạn muốn tạo một số ngẫu nhiên (random number). Có hai hàm có thể giúp bạn thực hiện việc này. Hàm đầu tiên là **rand()** được định nghĩa trong thư viện <cstdlib>: sẽ chỉ trả về một số ngẫu nhiên giả, mà có thể không thay đổi qua các lần chạy chương trình. Để giải quyết điểm bất thường này, chúng ta sẽ sử dụng hàm **srand()** trong thư viện <ctime>.

Ví dụ sau sẽ tạo vài số ngẫu nhiên trong C++. Ví dụ có sử dụng hàm time() để lấy số giây trên System time của bạn. Giá trị trả về từ time là qua srand, bạn lưu ý là số ngẫu nhiên được tạo ra trước lời gọi rand.

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;
int main ()
{
    int i,j;
    srand( (unsigned)time( NULL ) );

    /* tao 10 so ngau nhien. */
    for( i = 0; i < 10; i++ )
    {
        j= rand();
        cout <<" So ngau nhien la : " << j << endl;
    }

    return 0;
}
```

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:

```
So ngau nhien la : 5039
So ngau nhien la : 4448
So ngau nhien la : 6317
So ngau nhien la : 24475
So ngau nhien la : 6790
So ngau nhien la : 2070
So ngau nhien la : 15390
So ngau nhien la : 25290
So ngau nhien la : 15276
So ngau nhien la : 19143
```

---

## CHUYÊN ĐỀ 5: MẢNG MỘT CHIỀU (ARRAY) TRONG C/C++

Ngôn ngữ lập trình C/C++ cung cấp cấu trúc dữ liệu gọi là **mảng**, được lưu trữ trong một tập hợp các dữ liệu cùng kiểu với độ dài cố định. Một mảng được sử dụng để lưu trữ tập hợp dữ liệu, nhưng nó rất hữu dụng nếu bạn nghĩ về một mảng các biến với cùng một kiểu.

Thay vì khai báo biến một cách rời rạc, như biến `so0`, `so1`,... và `so99`, bạn có thể khai báo một mảng các giá trị như `so[0]`, `so[1]` và ... `so[99]` để biểu diễn các giá trị riêng biệt. Một thành viên cụ thể của mảng có thể được truy cập qua index (chỉ số).

Tất cả mảng đều bao gồm các vị trí nhớ liên kề nhau. Địa chỉ thấp nhất tương ứng với thành viên đầu tiên và địa chỉ cao nhất tương ứng với thành viên cuối cùng của mảng.

### 1. Khai báo mảng trong C/C++

Để khai báo một mảng trong ngôn ngữ C/C++, bạn xác định kiểu của biến và số lượng các phần tử được yêu cầu bởi biến đó như sau:

```
Kieu Ten_mang [ Kich_co_mang ];
```

Đây là mảng một chiều. **Kich\_co\_mang** phải là một số nguyên lớn hơn 0 và **Kieu** phải hợp lệ trong ngôn ngữ C/C++. Ví dụ, khai báo một mảng 10 phần tử gọi là `balance` với kiểu `double`, sử dụng câu lệnh sau đây:

```
char sinhvien[10];
```

### 2. Khởi tạo mảng trong C/C++

Bạn có thể khởi tạo mảng trong C/C++ hoặc từng phần tử một hoặc sử dụng một câu lệnh như dưới đây:

```
int hanghoa[5] = {45, 34, 29, 67, 49};
```

Số lượng các giá trị trong dấu ngoặc kép `{ }` không được lớn hơn số lượng phần tử khai báo trong dấu ngoặc vuông `[ ]`.

Nếu bạn bỏ sót kích cỡ mảng thì mảng đó đủ lớn để giữ các giá trị được khởi tạo: Bạn sẽ tạo chính xác một chuỗi có giá trị giống hệt chuỗi bên trên bằng cách gán từng phần tử một. Dưới đây là một ví dụ khi gán giá trị cho một phần tử của mảng:

```
int hanghoa[] = {45, 34, 29, 67, 49};
```

Bạn có thể tạo ra cùng một mảng giống như đã làm trong ví dụ trước.

```
hanghoa[4] = 50;
```

Câu lệnh bên trên gán giá trị thứ 5 của mảng giá trị 50.0. Tất cả các mảng đều có chỉ số (index) đầu tiên bằng 0, đây được gọi là chỉ số cơ bản và phần tử cuối

cùng của mảng có chỉ số bằng độ lớn của mảng trừ đi 1. Dưới đây là cách biểu diễn hình họa cho chuỗi khai báo bên trên thông qua chỉ số:

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

### 3. Truy cập các phần tử mảng trong C/C++

Một mảng được truy cập bởi cách đánh chỉ số trong tên của mảng. Dưới đây là một cách truy cập một giá trị của mảng:

```
int hocphi = hocphik60[55];
```

Câu lệnh trên lấy phần tử thứ 56 của mảng và gán giá trị này cho biến hocphi. Dưới đây là một ví dụ về việc sử dụng với tất cả mô tả bên trên:

```
#include <iostream>
using namespace std;

#include <iomanip>
using std::setw;

int main ()
{
    int n[ 10 ]; // n là một mảng gồm 10 số nguyên

    // khởi tạo giá trị các phần tử của mảng n là 0
    for ( int i = 0; i < 10; i++ )
    {
        n[ i ] = i + 100; // thiết lập phần tử tại vị trí i là i + 100
    }
    cout << "Phần tử thu:" << setw( 13 ) << "Giá trị là:" << endl;

    // hiển thị giá trị của mọi phần tử
    for ( int j = 0; j < 10; j++ )
    {
        cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
    }

    return 0;
}
```

Chương trình này sử dụng hàm **setw(so\_nguyên)** trong C/C++ để định dạng output. Tại đây, tham số **so\_nguyên** là một số chỉ độ rộng của kết quả mà bạn muốn hiển thị. Chẳng hạn, với so\_nguyên là 3 tức là bạn dành 3 vị trí để in kết quả, nếu kết quả cần hiển thị là thừa thì nó sẽ bị cắt bớt, nếu thiếu thì chèn thêm khoảng trống vào. Hàm setw() được dùng cho cả **cout** và **cin**.

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

```
Phan tu thu: Gia tri la:
0          100
1          101
2          102
3          103
4          104
5          105
6          106
7          107
8          108
9          109
```

Chi tiết về mảng trong C/C++

Mảng là một phần rất quan trọng trong ngôn ngữ C/C++. Dưới đây là những định nghĩa quan trọng liên quan đến một mảng cụ thể mà được trình bày rõ ràng hơn cho các lập trình viên C/C++:

Khái niệm	Miêu tả
<b>Mảng đa chiều trong C/C++</b>	C/C++ hỗ trợ các mảng đa chiều. Mẫu đơn giản nhất của mảng này là mảng hai chiều
<b>Con trỏ tới một mảng trong C/C++</b>	Bạn có thể trỏ tới phần tử đầu tiên của mảng một cách đơn giản chỉ bởi xác định tên mảng đó, chứ không phải một chỉ số
<b>Truyền mảng tới hàm như là tham số trong C/C++</b>	Bạn có thể truyền tới hàm một điểm trỏ chỉ tới một mảng bởi xác định tên mảng chứ không phải là một chỉ số
<b>Trả về mảng từ hàm trong C/C++</b>	C/C++ cho phép một hàm có thể trả về một mảng

## CHUYÊN ĐỀ 6: CHUỖI (XÂU KÍ TỰ) TRONG C/C++

### 1. Chuỗi (String) trong C/C++

C++ cung cấp hai kiểu biểu diễn chuỗi như sau:

- Chuỗi theo phong cách của ngôn ngữ C (C-style),
- Lớp Chuỗi (String) được giới thiệu trong C/C++ chuẩn.

Chuỗi theo phong cách C

Dạng chuỗi này bắt nguồn từ ngôn ngữ C và tiếp tục được hỗ trợ trong C/C++. Chuỗi trong ngôn ngữ lập trình C thực chất là mảng một chiều của các ký tự mà kết thúc bởi một ký tự **null** '\0'.

Phần khai báo và khởi tạo dưới đây tạo ra một chuỗi bao gồm một từ "Hello". Để giữ các giá trị null tại cuối của mảng, cỡ của mảng các ký tự bao gồm một chuỗi phải nhiều hơn số lượng các ký tự trong từ khóa "Hello".

```
char loiChao[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

Nếu bạn theo quy tắc khởi tạo các chuỗi, bạn có thể viết lệnh như sau:

```
char loiChao[] = "Hello";
```

Dưới đây là phần biểu diễn ô nhớ cho đoạn chuỗi trên trong ngôn ngữ C/C++:

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

Thực tế, bạn không đặt ký tự null tại vị trí cuối cùng của biến hằng số. Bộ biên dịch C tự động thêm '\0' tại vị trí cuối cùng của chuỗi khi nó khởi tạo chuỗi. Cùng thử ví dụ in ra chuỗi sau đây:

```
#include <iostream>

using namespace std;

int main ()
{
    char loiChao[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

    cout << "Khi gặp nhau, chúng ta nói: ";
    cout << loiChao << endl;

    return 0;
```

```
}
```

Khi đoạn code trên được biên dịch và thực hiện, kết quả in ra sẽ như sau:

Khi gap nhau, chúng ta nói: Hello

Ngôn ngữ C/C++ hỗ trợ nhiều hàm đa dạng để thao tác các chuỗi kết thúc là null:

STT	Hàm & Mục đích
1	<b>strcpy(s1, s2);</b> Sao chép chuỗi s2 cho chuỗi s1.
2	<b>strcat(s1, s2);</b> Nối chuỗi s2 vào cuối chuỗi s1.
3	<b>strlen(s1);</b> Trả về độ dài của chuỗi s1.
4	<b>strcmp(s1, s2);</b> Trả về 0 nếu s1 và s2 là như nhau; nhỏ hơn 0 nếu s1<s2; lớn hơn 0 nếu s1>s2.
5	<b>strchr(s1, ch);</b> Trả về con trỏ tới vị trí đầu tiên của ch trong s1.
6	<b>strstr(s1, s2);</b> Trả về con trỏ tới vị trí đầu tiên của chuỗi s2 trong chuỗi s1.

Dưới đây là ví dụ cho việc sử dụng một vài hàm bên trên:

```
#include <iostream>
#include <cstring>

using namespace std;

int main ()
{
    char chuoi1[10] = "Hello";
    char chuoi2[10] = "Christmas";
    char chuoi3[10];
    int len ;

    // sao chép chuoi1 vào trong chuoi3
    strcpy( chuoi3, chuoi1);
    cout << "strcpy( chuoi3, chuoi1) : " << chuoi3 << endl;

    // nói hai chuoi: chuoi1 và chuoi2
```

```

strcat( chuoi1, chuoi2);
cout << "strcat( chuoi1, chuoi2): " << chuoi1 << endl;

// tong do dai cua chuoi1 mot sau khi thuc hien noi chuoi
len = strlen(chuoi1);
cout << "Dung ham strlen(chuoi1) de tinh do dai chuoi1: " << len << endl;

return 0;
}

```

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

```

strcpy( chuoi3, chuoi1) : Hello
strcat( chuoi1, chuoi2): HelloChristmas
Dung ham strlen(chuoi1) de tinh do dai chuoi1: 14
=====

```

## 2. Lớp String trong C/C++

Thư viện chuẩn C/C++ cung cấp một kiểu lớp **String** mà hỗ trợ tất cả hoạt động liên quan tới chuỗi đã đề cập ở trên, và bổ sung thêm nhiều tính năng nữa. Chúng ta sẽ học lớp này trong Thư viện chuẩn C/C++ (C++ Standard Library), nhưng lúc này, chúng ta xem xét ví dụ sau:

Lúc này, có thể bạn không hiểu ví dụ này, bởi vì chúng ta chưa bàn luận về **Lớp và Đối tượng** trong C/C++. Vì thế, bạn quan sát và ghi nhớ chúng tới khi bạn đã hiểu các khái niệm về Hướng đối tượng được trình bày ở chương sau đó.

```

#include <iostream>
#include <string>

using namespace std;

int main ()
{
    string chuoi1 = "Hello";
    string chuoi2 = "Christmas";
    string chuoi3;
    int len ;

    // sao chép chuoi1 vào trong chuoi3
    chuoi3 = chuoi1;
    cout << "Bây giờ chuoi3 là: " << chuoi3 << endl;

    // nối hai chuỗi: chuoi1 và chuoi2
    chuoi3 = chuoi1 + chuoi2;
    cout << "chuoi1 + chuoi2 có kết quả là: " << chuoi3 << endl;

    // tổng độ dài của chuoi3 một sau khi thực hiện nối chuỗi
    len = chuoi3.size();
}

```



```

cout << "Tính độ dài voi ham chuo3.size() : " << len << endl;

return 0;
}

```

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

```

Bây giờ chuo3 là: Hello
chuo1 + chuo2 có kết quả là: HelloChristmas
Tính độ dài voi ham chuo3.size() : 14

```

### 3. Các phương thức, phép toán tiện ích của kiểu string

Kiểu string của STL hỗ trợ các nhóm phương thức và phép toán tiện ích sau đây.

#### a) Các phép toán và phương thức cơ bản

- Các toán tử +, += dùng để ghép hai chuỗi và cũng để ghép một ký tự vào chuỗi;
- Các phép so sánh theo thứ tự từ điển: == (bằng nhau), != (khác nhau), > (lớn hơn), >= (lớn hơn hay bằng), < (nhỏ hơn), <= (nhỏ hơn hay bằng);
- Hàm length() và phép lấy chỉ số [] để duyệt từng ký tự của chuỗi: nếu s là biến kiểu string thì s[i] là ký tự thứ i của s với  $0 \leq i < s.length()$ ;
- Phép gán = dùng để gán biến kiểu string bằng một chuỗi, hoặc bằng string khác, chẳng hạn: string s="ABCDEF"; hay s1=s2; mà không cần copy xâu.

Những constructor thường sử dụng nhất: string();

string(const char \*str); // char\* là kiểu dữ liệu xâu của C

string(const string & str);

Có thể dùng toán tử << với cout để xuất một chuỗi ra màn hình hoặc dùng toán tử >> với cin để nhập một chuỗi ký tự đến khi gặp một khoảng trống thì dừng.

```
char st[]="ABCDEF";
```

```
string s;
```

```
s="XYZ";
```

```
cout << s << endl;
```

```
s=st;
```

```
cout << s.length() << " : " << s << endl;
```

Một vấn đề thường nảy sinh trong các ứng dụng có sử dụng C-string: một C-String chưa khởi tạo cần được gán NULL. Tuy nhiên, rất nhiều hàm thư viện của C-String sẽ gặp sự cố trong thời gian chạy khi gặp đối tượng C-String là NULL.

Chẳng hạn, lệnh

```
char* x = NULL;
```

```
cout << strlen(x);
```

được một số trình biên dịch chấp nhận, nhưng với nhiều hiện thực khác của thư viện C-String, thì gặp lỗi trong thời gian chạy. string không gặp vấn đề này, ta hoàn toàn có thể cho 1 xâu là rỗng mà không gặp bất cứ lỗi nào: string s="";

String thực chất là một `vector<char>` có bổ sung thêm một số hàm và thuộc tính, do đó, nó có toàn bộ các tính chất của 1 vector, như hàm `size()`, `push_back()`, toán tử `[]`, ...

Các hàm từ vector:

`v.size()`: Số lượng phần tử

`v.empty()`: Trả về 1 nếu chuỗi rỗng, 0 nếu ngược lại.

`v.max_size()`: Trả về số lượng phần tử tối đa đã được cấp phát

`v1 == v2`: Trả về 1 nếu hai chuỗi giống nhau

`v1 != v2`: Trả về 1 nếu hai chuỗi khác nhau

`v.begin()`: Trả về iterator đầu tiên của chuỗi

`v.end()`: Trả về iterator cuối cùng của chuỗi (trở vào sau ký tự cuối cùng)

`v.front()`: Trả về phần tử đầu tiên của chuỗi

`v.back()`: Trả về phần tử cuối cùng của chuỗi

`v1.swap(v2)`: Hoán đổi 2 chuỗi với nhau (giống việc hoán đổi giá trị của 2 biến)

```
#include <iostream>
```

```
#include <conio.h>
```

```
#include <string>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    string s = "Hello string"; // Khai báo biến kiểu string
```

```
    cout << "Nội dung string: " << s << endl; // In nội dung string ra màn hình
```

```
    cout << "Chiều dài của string: " << s.size() << endl;
```

```
    // Chiều dài
```

```
    cout << "Ký tự 0: " << s[0] << endl; // In ký tự đầu tiên của chuỗi
```

```
    cout << "Ký tự 1: " << s[1] << endl; // In ký tự thứ 2
```

```
    cout << "Ký tự 2: " << s[2] << endl; // In ký tự thứ 3
```

```
    getch();
```

```
    return 0;
```

```
}
```

Nhập một string trên 1 dòng (chú ý cin sẽ chỉ đọc đến dấu cách hoặc xuống dòng đầu tiên): `istream& getline ( istream& in, string& str, char delimiter = '\n')`;

Đọc 1 dòng văn bản từ `istream in` (có thể là file hay đối tượng chuẩn cin) từng ký tự đến khi ký tự `delimiter` được nhập vào (mặc định là `\n`)

```
// getline with strings
```

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
int main ()
```

```
{
```

```
    string str;
```

```

short age;
cout << "Please enter full name and age"<< endl;
getline( cin, str) >> age;
cout << "Thank you " << str << "!\n";
return 0;
}

```

### **b) Các phương thức chèn, xóa, lấy chuỗi con:**

Phương thức substr(int pos, int nchar) trích ra chuỗi con của một chuỗi cho trước, ví dụ str.substr(2,4) trả về chuỗi con gồm 4 ký tự của chuỗi str kể từ ký tự ở vị trí thứ 2 (ký tự đầu tiên của chuỗi ở vị trí 0).

```

//get substring
#include <iostream>
#include <string>
#include <conio.h>
using namespace std;
int main ()
{
    string s="ConCho chạy qua rảo";
    cout << s.substr(2,4) << endl;
    // cout << new string(str.begin()+2, str.begin()+2+4);
    getch();
    return 0;
}

```

Phương thức insert() chèn thêm ký tự hay chuỗi vào một vị trí nào đó của chuỗi str cho trước. Có nhiều cách dùng phương thức này:

str.insert(int pos, char\* s); chèn s (mảng ký tự kết thúc \0) vào vị trí pos của str;  
 str.insert(int pos, string s); chèn chuỗi s (kiểu string) vào vị trí pos của chuỗi str;  
 str.insert(int pos, int n, int ch); chèn n lần ký tự ch vào vị trí pos của chuỗi str;

// inserting into a string

```

#include <iostream>
#include <string>
#include <conio.h>
using namespace std;
int main ()
{
    string str="day la .. xau thu";
    string istr = "them";
    str.insert(8, istr);
    cout << str << endl;
    getch();
    return 0;
}

```

Phương thức `str.erase(int pos, int n)` xóa `n` ký tự của chuỗi `str` kể từ vị trí `pos`; nếu không quy định giá trị `n` thì tất cả các ký tự của `str` từ vị trí `pos` trở đi sẽ bị xóa

```
// erase from a string
#include <iostream>
#include <string>
#include <conio.h>
using namespace std;
int main ()
{
    string str="day cung la xau thu";
    str.erase(0, 3); // " cung la xau thu"
    cout << str << endl;
    str.erase(6, 2);
    cout << str << endl; // " cung xau thu"
    getch();
    return 0;
}
```

### c) So sánh

Bạn có thể đơn giản là sử dụng những toán tử quan hệ (`==`, `!=`, `<`, `<=`, `>=`) được định nghĩa sẵn. Tuy nhiên, nếu muốn so sánh một phần của một chuỗi thì sẽ cần sử dụng phương thức `compare()`:

```
int compare ( const string& str ) const;
int compare ( const char* s ) const;
int compare ( size_t pos1, size_t n1, const string& str ) const;
int compare ( size_t pos1, size_t n1, const char* s ) const;
int compare ( size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2 )
const;
int compare ( size_t pos1, size_t n1, const char* s, size_t n2 ) const;
Hàm trả về 0 khi hai chuỗi bằng nhau và lớn hơn hoặc nhỏ hơn 0 cho trường hợp khác Ví dụ:
```

```
// comparing apples with apples
#include <iostream>
#include <string>
using namespace std;
int main ()
{
    string str1 ("green apple");
    string str2 ("red apple");
    if (str1.compare(str2) != 0)
        cout << str1 << " is not " << str2 << "\n";
    if (str1.compare(6,5,"apple") == 0)
```

```

cout << "still, " << str1 << " is an apple\n";
if (str2.compare(str2.size()-5,5,"apple") == 0)
cout << "and " << str2 << " is also an apple\n";
if (str1.compare(6,5,str2,4,5) == 0)
cout << "therefore, both are apples\n";
return 0;
}

```

#### **d) Các phương thức tìm kiếm và thay thế**

Phương thức find() tìm kiếm xem một ký tự hay một chuỗi nào đó có xuất hiện trong một chuỗi str cho trước hay không. Có nhiều cách dùng phương thức này:

- str.find(int ch, int pos = 0); tìm ký tự ch kể từ vị trí pos đến cuối chuỗi str
- str.find(string& s, int pos = 0); tìm chuỗi s kể từ vị trí pos đến cuối chuỗi.

Nếu không quy định giá trị pos thì hiểu mặc nhiên là 0; nếu tìm có thì phương thức trả về vị trí xuất hiện đầu tiên, ngược lại trả về giá trị -1.

```

//find substring
#include <iostream>
#include <string>
#include <conio.h>
using namespace std;
int main ()
{
    string str="ConCho chạy qua rào";
    cout << str.find("chạy") << endl; // 7
    cout << (int)str.find("Chạy") << endl; // -1
    getch();
    return 0;
}

```

#### **• Hàm tìm kiếm ngược (rfind)**

```

//find from back
#include <iostream>
#include <string>
#include <conio.h>
using namespace std;
int main ()
{
    string str="ConCho chạy qua chạy qua rào";
    cout << str.find("chạy") << endl; // 7
    cout << (int)str.rfind("chạy") << endl; // 16
    getch();
    return 0;
}

```

```
}
```

- Phương thức `replace()` thay thế một đoạn con trong chuỗi `str` cho trước (đoạn con kể từ một vị trí `pos` và đếm tới `nchar` ký tự ký tự về phía cuối chuỗi) bởi một chuỗi `s` nào đó, hoặc bởi `n` ký tự `ch` nào đó. Có nhiều cách dùng, thứ tự tham số như sau:

```
str.replace(int pos, int nchar, char *s);
    str.replace(int pos, int nchar, string s);
    str.replace(int pos, int nchar, int n, int ch);
string str="con cho la con cho con. Con meo ko phai la con cho";
    str.replace(4, 3, "CHO"); // "con CHO la con cho con. Con meo ko phai la
con cho";
    cout << str << endl;
getchar();
```

### e) Tách xâu

Trong việc xử lý xâu ký tự, không thể thiếu được các thao tác tách xâu ký tự thành nhiều xâu ký tự con thông qua các ký tự ngăn cách. Các hàm này có sẵn trong các ngôn ngữ khác như Visual Basic, Java, hay thậm chí là trong `<string.h>`. Với STL, các bạn có thể dễ dàng làm điều này với `stringstream`:

```
string S = "Xin chao tat ca cac ban"; // Khởi tạo giá trị của xâu
stringstream ss(S); // Khởi tạo stringstream từ xâu S
```

```
while (ss >> token) { // Đọc lần lượt các phần của xâu. Các phần tách nhau bởi
dấu cách hoặc xuống dòng.
```

```
    cout << token << endl;
```

```
}
```

### Output:

*Xin*

*chao*

*tat*

*ca*

*cac*

*ban*

Chú ý rằng, cách này cũng có thể dễ áp dụng nếu bạn muốn chuyển số thành xâu (hoặc ngược lại), tách 1 xâu thành nhiều số.

Nếu không muốn sử dụng `stringstream`, các bạn cũng có thể tự xây dựng hàm tách xâu như sau:

```
string S = "Xin chao tat ca cac ban"; // Khởi tạo giá trị của xâu
```

```
string::iterator t, t2; // Các biến lặp
```

```
vector<string> split; // Mảng các xâu (lưu kết quả tách)
```

```
for (t=S.begin(); t<S.end();)
```

```

{
    // Lặp từ vị trí bắt đầu
    t2=find(t, S.end(), ' '); // Tìm ký tự space ' ' đầu tiên
    // kể từ vị trí t
    if (t!=t2) split.push_back(string(t, t2)); // Lấy xâu ký tự giữa 2 vị trí
    t = t2+1; // Chuyển sang vị trí sau
}
for (int i=0; i<split.size(); i++)
cout << split[i] << endl; // In mảng các xâu ký tự
getchar();

```

### Output:

```

Xin
chao
tat
ca
cac
ban

```

Đoạn chương trình sử dụng các kỹ thuật sau:

- Phương thức find(vị\_trí\_đầu, vị\_trí\_cuối, ký\_tự\_tìm) dùng để tìm vị trí đầu tiên của ký\_tự\_tìm bắt đầu từ vị\_trí\_đầu. Hàm này trả về vị trí của ký tự tìm được (nếu tìm thấy) hoặc vị\_trí\_cuối (nếu không tìm thấy)
- string có thể khởi tạo từ một đoạn ký tự con của một xâu ký tự khác với cú pháp string(vị\_trí\_đầu, vị\_trí\_cuối)
- Đoạn chương trình thực hiện tách các xâu ký tự kể cả trong trường hợp có nhiều ký tự space nằm liên tiếp nhau. Một cách đơn giản hơn là bạn có thể gọi hàm strtok() trong string.h để làm việc này, nhưng không may là hàm này thao tác trên char\* chứ không phải string. Hàm thành viên c\_str() sẽ giúp bạn chuyển từ string thành dạng const charT\* c\_str() const;
- Hàm này cũng tự động sinh ra ký tự null chèn vào cuối xâu.

Từ prototype ta cũng thấy được hàm trả về một hằng chuỗi, điều này đồng nghĩa với việc ta không thể thay đổi chuỗi trả về. Gọi phương thức c\_str();

```

string s = "some_string";
cout << s.c_str() << endl;
cout << strlen(s.c_str()) << endl;

```

Sau đây là ví dụ bên trên được viết lại dùng hàm thành viên c\_str() và các hàm trong <string.h>

```

// strings vs c-strings
#include <iostream>
#include <string.h>
#include <string>
using std::string;
int main ()

```

```

{
    char* cstr;
    char* p;
    string str ("Xin chao tat ca cac ban");
    cstr = new char [str.size()+1];
    strcpy (cstr, str.c_str());
    // cstr là 1 bản sao c-string của str
    p=strtok (cstr," ");
    while (p!=NULL)
    {
        cout << p << endl;
        p=strtok(NULL," ");
    }
    delete[] cstr;
    return 0;
}

```

**Output:**

Xin  
 chao  
 tat  
 ca  
 cac  
 ban



## CHUYÊN ĐỀ 6: ĐỌC VÀ GHI RA FILE TEXT TRONG C/C++

### Đọc và ghi file text trong C++

Trong C++ việc đọc và ghi file thường dùng để đưa các bài toán ở dạng đầu vào và xuất ra kết quả. Điều này giúp cho người ta dễ dàng kiểm tra xem chương trình của bạn có thực sự đúng không khi cho chương trình chạy với nhiều đầu vào và kiểm tra các kết quả ở đầu ra chương trình có đúng với kết quả chuẩn. Bài viết này sẽ trình bày các vấn đề về đọc và ghi file với các ký tự ASCII sử dụng thư viện ifstream.

#### 1. Sử dụng ofstream trong thư viện fstream để ghi file

- Cách ghi ra file sử dụng thư viện fstream cũng tương đối đơn giản, gần giống với cout<< trong c++.
- Dưới đây là đoạn code đơn giản: tạo ra một file .txt rồi ghi vào đó một đoạn text mà mình thích.

```
#include <iostream>
#include <fstream>
#include <iostream>
using namespace std;
```

```
int main()
{
    ofstream FileDemo ("File Demo.txt");
    FileDemo<<"Day la file demo su dung cach doc va ghi file su dung fstream";
    FileDemo.close();
    return 0;
}
```

Khi chạy đoạn chương trình trên, chương trình sẽ tạo ra một file tên File Demo.txt nằm trong cùng thư mục với project chứa code của bạn. Khi mở file này lên, bạn sẽ thấy đoạn text mà mình muốn ghi trong đó.

- Và đây là một ví dụ về ghi ra các số chẵn từ 1 đến 1000 vào file “So Chan.txt”
- ```
#include <iostream> #include <fstream> #include <iostream> using namespace std;
```

```
int main()
{
    ofstream SoChan ("So Chan.txt");
    SoChan<<"Day so chan tu 1 -> 1000 \n";
    for(int a = 1; a <= 1000; a++)
    {
        if(a%2 == 0)
        {
```

```

        SoChan<<a;
        SoChan<<"\n";
    }
}
SoChan.close();
return 0;
}

```

File “So Chan.txt ” sau khi chạy chương trình trên sẽ như hình bên dưới.

- Và tiếp tục một ví dụ ghi ra file “Ghi So Chan Le.txt” các số chẵn và lẻ từ 0 -> 100

(lưu ý: ví dụ này được update sau theo yêu cầu một số bạn nên code không có màu).

```

#include <iostream>
#include <fstream>
#include <iostream>
using namespace std;
void ghiSoChan(ofstream &GhiSo)
{
    int dem = 0;
    GhiSo<<"Day so chan tu 1 -> 100 \n";
    for(int a = 1; a <= 100; a++)
    {
        if(a%2 == 0)
        {
            dem ++;
            GhiSo<<a;
            if(dem % 5 == 0)
            {
                GhiSo<<"\n";
            }
            if(dem % 5 != 0)
            {
                GhiSo<<"\t";
            }
        }
    }
    cout<<"\n So Chan: " << dem;
}
void ghiSoLe(ofstream &GhiSo)
{
    int dem = 0;

```

```

GhiSo<<"\nDay so le tu 1 -> 100 \n";
for(int i = 1; i <= 100; i++)
{
    if(i%2 != 0)
    {
        dem ++;
        GhiSo<<i;
        if(dem % 5 == 0)
        {
            GhiSo<<"\n";
        }
        if(dem % 5 != 0)
        {
            GhiSo<<"\t";
        }
    }
}
cout<<"\n So Le: "<<dem;
}
int main()
{
    ofstream GhiSo("Ghi So Chan Le.txt");
    ghiSoChan(GhiSo);
    ghiSoLe(GhiSo);
    GhiSo<<"\n ThanhCuong.wordpress.com";
    GhiSo.close();
    system("pause");
    return 0;
}

```

## 2. Sử dụng Hàm freopen() trong C/C++

Hàm **FILE \*freopen(const char \*filename, const char \*mode, FILE \*stream)** trong Thư viện C chuẩn gắn kết một filename mới với Stream đã cho và cùng lúc đó đóng FILE cũ trong Stream.

Khai báo hàm freopen() trong C

Dưới đây là phần khai báo cho hàm freopen() trong C:

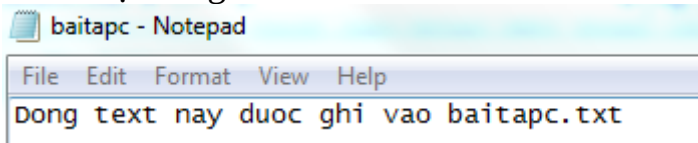
```
FILE *freopen(const char *filename, const char *mode, FILE *stream)
```

Tham số

- **filename** – Đây là chuỗi chứa tên file để được mở.
- **mode** – Đây là chuỗi chứa chế độ truy cập file. Bao gồm:



Sau lời gọi tới hàm **freopen()**, nó gắn kết STDOUT tới baitapc.txt, vì thế bất cứ cái gì chúng ta ghi tại STDOUT sẽ đi vào trong baitapc.txt. Vì thế, baitapc.txt sẽ có nội dung sau:



Bây giờ bạn theo dõi nội dung của file trên bởi sử dụng chương trình C sau:

```
#include <stdio.h>
```

```
int main ()
{
    FILE *fp;
    int c;

    fp = fopen("baitapc.txt","r");
    while(1)
    {
        c = fgetc(fp);
        if( feof(fp) )
        {
            break ;
        }
        printf("%c", c);
    }
    fclose(fp);
    return(0);
}
```

```
Dong text nay duoc ghi vao baitapc.txt
```