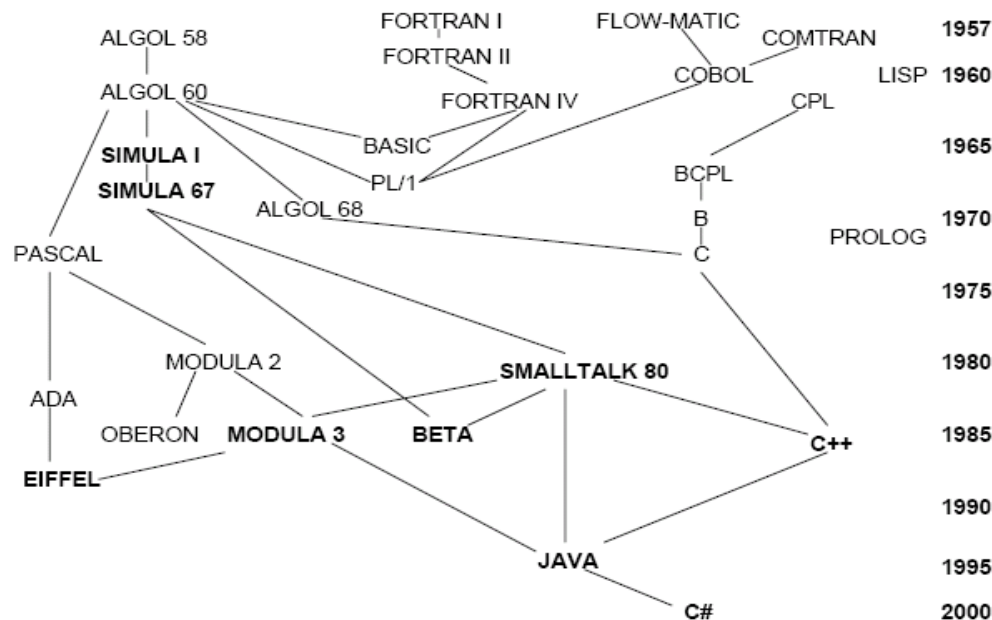


## **Chương II. NHỮNG MỞ RỘNG CỦA C++ SO VỚI C**

I. Giới thiệu ngôn ngữ C++	16
II. Mở rộng C++ so với C	16
1. Từ khóa mới	16
2. Chú thích	16
3. Dữ liệu, khai báo biến	16
3.1 Kiểu dữ liệu	16
3.2 Khai báo biến	17
3.3 Phạm vi và các khai báo	17
4. Chuyển kiểu	18
5. Khả năng vào/ra mới	19
5.1 Output	19
5.2 Input	19
6. Tham chiếu	20
6.1 Sự khác nhau giữa reference và pointer	20
6.2 Hằng tham chiếu	21
6.3 Truyền tham số cho hàm bằng con trỏ / tham chiếu	22
6.4 Tham chiếu đến con trỏ	22
6.5 Giá trị trả về của hàm là tham chiếu	23
7. Định nghĩa chồng hàm, tham số ngầm định	23
7.1 Định nghĩa chồng hàm (Overloading function)	23
7.2 Tham số ngầm định	24
8. Quản lý bộ nhớ động	25
8.1 Toán tử cấp phát bộ nhớ new	25
8.2 Toán tử giải phóng bộ nhớ delete	25
8.3 Lợi thế của new so với malloc	25
9. Hàm Inline	26
10. Const	26

## I. Giới thiệu ngôn ngữ C++



- ❖ Được đưa ra bởi Bjarne Stroustrup năm 1983
- ❖ Phát triển dựa trên ngôn ngữ C
- ❖ C++ có 2 đặc điểm mới:
  - Mở rộng so với C: tham chiếu, chồng hàm,...
  - Khả năng LTHĐT

## II. Mở rộng C++ so với C

### 1. Từ khóa mới

asm	catch	class	delete	friend	inline	new	operator	private
protected	public	template	this	throw	try	virtual		

### 2. Chú thích

- Bên cạnh chú thích kiểu C (nhiều dòng), C++ cho phép kiểu chú thích dòng đơn
- C++ cho phép kiểu chú thích /\* \*/ bao ngoài các chú thích dòng đơn.

C	C++
<pre>/* This is a variable */ int x; /* This is the variable  * being given a value */ x = 5;</pre>	<pre>// This is a variable int x; // This is the variable // being given a value x = 5;</pre>

### 3. Dữ liệu, khai báo biến

#### 3.1 Kiểu dữ liệu

- + Kiểu giá trị Boolean: bool
  - .. Hai giá trị: true hoặc false
  - .. Các toán tử logic (!, &&, ...) lấy/tạo một giá trị bool
  - .. Các phép toán quan hệ (==, <, ...) tạo một giá trị bool
  - .. Các lệnh điều kiện (if, while, ...) đòi hỏi một giá trị bool
- + Để tương thích ngược với C, C++ ngầm chuyển từ int sang bool khi cần
  - .. Giá trị 0 → false
  - .. Giá trị khác 0 → true
- + C++ kiểm soát kiểu dữ liệu chặt chẽ hơn C
- + C++ đòi hỏi hàm phải được khai báo trước khi sử dụng (mọi lời gọi hàm được kiểm tra khi biên dịch)
- + C++ không cho phép gán giá trị nguyên cho các biến kiểu enum

```
enum Temperature {hot, cold};
enum Temperature t = 1; // Error in C++
```

- + C++ không cho phép các con trỏ không kiểu (void\*) sử dụng trực tiếp tại bên phải lệnh gán hoặc một lệnh khởi tạo

```
void * vp;  
int * ip = vp; // Error: Invalid conversion
```

### 3.2 Khai báo biến

- C++ cho phép khai báo biến: Mọi nơi; Trước khi sử dụng

```
int n;  
n = 2;  
cout << n << "\n";  
int m = 3;  
cout << m << "\n";
```

- Một biến chỉ có tầm tác dụng trong khối lệnh nó được khai báo.

- Do đó, C++ cung cấp toán tử định phạm vi (::) để xác định rõ biến nào được sử dụng khi xảy ra tình trạng định nghĩa chồng một tên biến trong một khối lệnh con.

#### \* Toán tử phạm vi (::)

+ Thường được dùng để truy cập các biến toàn cục trong trường hợp có biến cục bộ trùng tên

+ *Ví dụ:* `y = ::x + 3;`

```
// Using the unary scope resolution operator.  
#include <iostream>  
using std::cout;  
using std::endl;  
#include <iomanip>  
using std::setprecision;  
  
// define global constant PI  
const double PI = 3.14159265358979;  
  
int main() {  
    // define local constant PI  
    const float PI = static_cast<float> (::PI);  
    /*Access the global PI with ::PI.  
    Cast the global PI to a float for the local PI. This example will show the difference  
    between float and double.*/  
  
    // display values of local and global PI constants  
    cout << setprecision(20)  
        << "  Local float value of PI = " << PI  
        << "\nGlobal double value of PI = " << ::PI << endl;  
    return 0; // indicates successful termination  
} // end main
```

#### Kết quả:

Local float value of PI = 3.1415927410125732422

Global double value of PI = 3.14159265358979000074

### 3.3 Phạm vi và các khai báo

- Trong C, các biến phải được định nghĩa tại đầu file hoặc tại bắt đầu của một khối {...}

- C++ cho phép khai báo sau và phạm vi của các biến được giới hạn chính xác hơn

“ Các khai báo có thể đặt tại các câu lệnh lặp for và các câu lệnh điều kiện

“ Phạm vi giới hạn bên trong vòng lặp hoặc khối điều kiện

- C++ còn bổ sung hai phạm vi mới:

“ Phạm vi không gian tên - Namespace scope

“ Phạm vi lớp - Class scope

#### \* Namespace - Không gian tên

- Không gian tên được bổ sung vào C++ để biểu diễn cấu trúc logic và cung cấp khả năng quản lý phạm vi tốt hơn

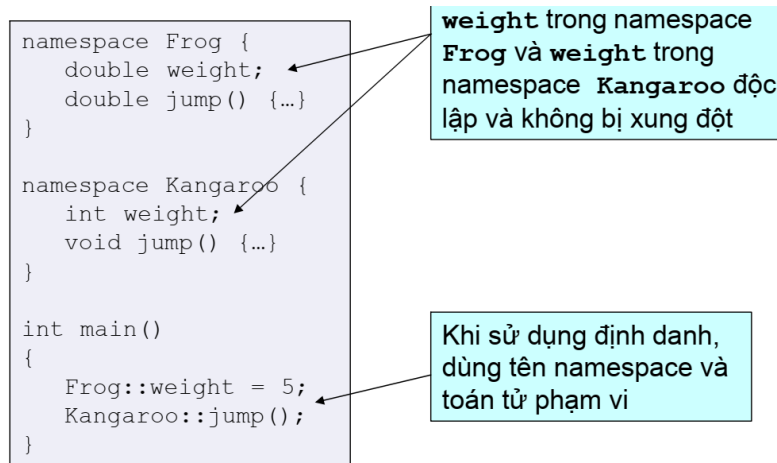
- Không gian tên cung cấp một cơ chế tường minh để tạo các vùng khai báo

- Các tên khai báo trong một không gian tên

“ không xung đột với các tên được khai báo trong các không gian tên khác

“ Tránh xung đột tên biến, tên hàm

“ Nghiễm nhiên có thể được liên kết ra ngoài (external linkage)

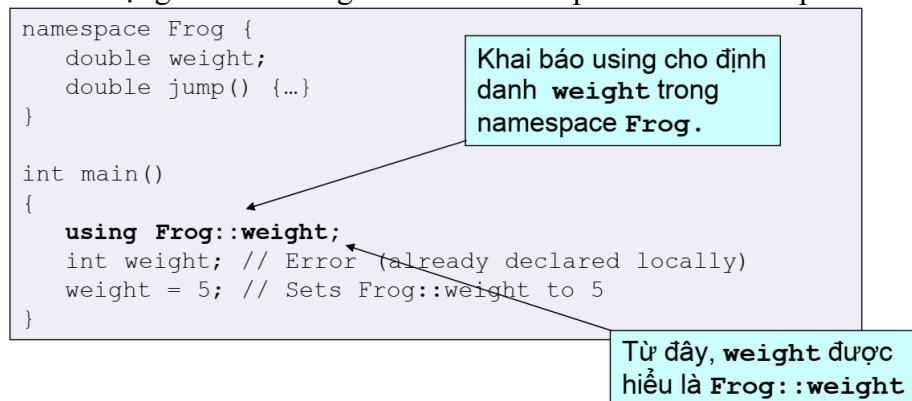


- C++ cung cấp hai cơ chế để đơn giản hóa việc sử dụng các namespaces: các khai báo using và các định hướng using.

- Khai báo using (using-declaration) cho phép truy nhập một định danh cụ thể trong vùng khai báo tạm thời

**using <namespace>::**name**>;**

Từ đây, ta có thể sử dụng tên mà không cần mỗi lần đều phải chỉ rõ namespace chứa nó.



- Khai báo using dành cho 1 tên, định hướng using cho phép truy nhập mọi định danh trong namespace

**using namespace <namespace>;**

- Định hướng using thường được đặt tại mức toàn cục.

```

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

```

#### 4. Chuyển kiểu

- C++ cho phép người dùng đổi kiểu dữ liệu một cách khá rộng rãi

- Trình biên dịch tự động thực hiện nhiều chuyển đổi dễ thấy:

- .. Gán một giá trị thuộc kiểu số học này cho một biến thuộc kiểu khác
- .. Các kiểu số học khác nhau cùng có trong các biểu thức
- .. Truyền đổi số cho các hàm

- Nếu hiểu rõ khi nào các chuyển đổi này xảy ra và trình biên dịch đang làm gì, ta có thể giải thích được các kết quả không mong đợi

- Tự động chuyển đổi từ các đối tượng nhỏ thành các đối tượng lớn thì không có vấn đề gì, chiều ngược lại có thể có vấn đề

.. short → long (~16 bits → ~32 bits) không có vấn đề

.. long → short (~32 bits → ~16 bits) có thể mất dữ liệu

.. Khi chuyển từ các kiểu chấm động sang các kiểu nguyên có thể làm giảm độ chính xác của dữ liệu

- Trình biên dịch sẽ sinh cảnh báo (warning) đối với các chuyển đổi tự động có thể gây mất dữ liệu.

- C++ cho phép người dùng ép kiểu một cách tường minh bằng nhiều cách

.. Ép kiểu kiểu C: **myInt = (int) myFloat;**

.. Ép kiểu kiểu hàm C++: **myInt = int(myFloat);**

```

float m = 2.5;
int n = (int)m; //cách cũ
int n = int(m); //cách mới

```

- Để hạn chế ép kiểu quá mức và loại trừ các lỗi do ép kiểu, C++ cung cấp một cách mới sử dụng 4 loại ép kiểu tường minh: static\_cast; const\_cast; reinterpret\_cast; dynamic\_cast

- Cú pháp ***myInt = static\_cast<int>(myFloat)***

## 5. Khả năng vào/ra mới

- C++ sử dụng stream để thực hiện thao tác vào/ra
- cin, cout, cerr là object của lớp tương ứng istream, ostream

### 5.1 Output

```
cout << "Hello";  
cout << 20;  
cout << x;
```

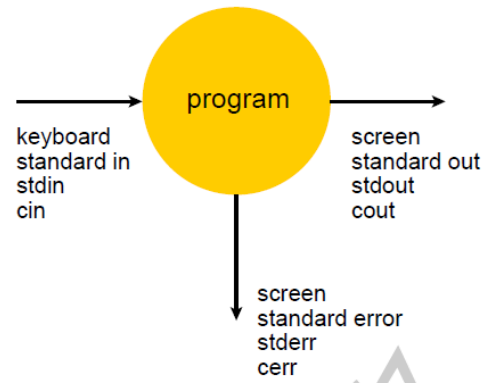
- Toán tử "<<" có thể sử dụng nhiều lần trên 1 dòng lệnh

```
int age = 25;  
cout << "Hello, I am " << age << " years old " << endl;
```

- Chuyển sang dòng mới: endl, '\n'

```
int age = 25;  
cout << "First sentence.\n";  
cout << "Second sentence" << endl;
```

- Toán tử "<<" có thể sử dụng để xuất ra màn hình giá trị thuộc các kiểu: Hằng; Kiểu dữ liệu cơ bản(char, int, float, double); Xâu ký tự; Con trỏ.



### 5.2 Input

```
int age;  
cin >> age;
```

- Toán tử ">>" có thể sử dụng nhiều lần trên 1 dòng lệnh

```
int n;  
float m;  
char c;  
cin >> n >> m >> c;
```

- Nhập chuỗi

```
#include <iostream>  
using namespace std;  
int main()  
{  
    string s;  
    cout << "Nhap chuoi: " << endl;  
    cin >> s;  
    cout << "Chuoi da nhap: " << s;  
}
```

**Kết quả:**  
Nhap chuoi:  
Hello you  
Chuoi da nhap: Hello

- ***istream& getline (char\* s, streamsize n);***  
***istream& getline (char\* s, streamsize n, char delim);***

```
#include <iostream>  
using namespace std;  
  
int main() {  
    char name[256], title[256];  
  
    cout << "Enter your name: ";  
    cin.getline(name, 256);  
  
    cout << "Enter your favourite movie: ";  
    cin.getline(title, 256);  
  
    cout << name << "'s favourite movie is " << title;  
  
    return 0;  
}
```

**Kết quả:**  
Enter your name: Harry  
Enter your favourite movie: Harry Potter  
Harry's favourite movie is Harry Potter

- Hàm xóa bộ đệm: **`fflush(stdin); cin.clear();`**

```
#include <iostream>
using namespace std;
int main()
{
    int x;
    cout << "enter choice: ";
    cin >> x;
    cin.clear();
    char c;
    cin >> c;
    return 0;
}
```

- Toán tử ">>" để nhập dữ liệu thuộc các kiểu:
  - + Kiểu dữ liệu cơ bản(char, int, float, double)
  - + Xâu kí tự char\*

## 6. Tham chiếu

- Tham chiếu là địa chỉ vùng nhớ được cấp phát cho một biến.
- Là "bí danh" của biến khác
- Kí hiệu & đặt trước biến hoặc hàm để xác định tham chiếu của chúng

```
int n = 3;
int& r = n; // r là biến tham chiếu
cout << r << endl;
```

- Tham chiếu có thể được sử dụng là biến, tham số của hàm, giá trị trả về của hàm
- Có thể coi mọi thao tác trên tham chiếu đều được thực hiện trên chính đối tượng nguồn.

```
#include <iostream>
using namespace std;
int main() {
    int x = 5;
    int& y = x;
    cout << "x = " << x << " y = " << y << ".\n"; // x = 5 y = 5
    x = x + 1;
    cout << "x = " << x << " y = " << y << ".\n"; // x = 6 y = 6
    y = y + 1;
    cout << "x = " << x << " y = " << y << ".\n"; // x = 7 y = 7
    int* p = &y;
    *p = 9;
    cout << "x = " << x << " y = " << y << ".\n"; // x = 9 y = 9
}
```

### Kết quả:

x = 5 y = 5.  
x = 6 y = 6.  
x = 7 y = 7.  
x = 9 y = 9.

### 6.1 Sự khác nhau giữa reference và pointer

- + Không tồn tại tham chiếu NULL

```
int &n = NULL; //error
int *n = NULL; //OK
```

- + Tham chiếu cần phải được khởi tạo

```
string& rs; //error
string s = "abc";
string& rs = s; //OK

string* rs; //OK
```

+ Tham chiếu chỉ có thể trỏ đến 1 vùng nhớ duy nhất

```
#include <iostream>
using namespace std;
int main()
{
    char s[5] = "lena";
    int n = 6;
    int k = 5;
    cout << "n = " << n << ", k = " << k << endl;
    int& rn = n;
    int* pn = &n;
    cout << "rn = " << rn << ", *pn = " << *pn << endl;
    rn = k;
    pn = &k;
    cout << "n = " << n << ", k = " << k << endl;
    cout << "rn = " << rn << ", *pn = " << *pn << endl;
}
```

**Kết quả:**

n = 6, k = 5  
rn = 6, \*pn = 6  
n = 5, k = 5  
rn = 5, \*pn = 5

- + Giá trị của tham chiếu không được thay đổi sau khi đã khởi tạo
  - “ không thể “chiếu” lại một tham chiếu tới đối tượng khác
  - “ chú ý phân biệt giữa khởi tạo tham chiếu và gán trị cho tham chiếu

+ Tham chiếu chứa *giá trị*, con trỏ chứa *địa chỉ* của vùng nhớ trỏ đến

```
#include <iostream>
using namespace std;
int main()
{
    char c = 'c';
    char& rn = c;
    char* pn = &c;
    cout << "sizeof (rn) = " << sizeof(rn) << ", sizeof (pn) = " << sizeof(pn) << endl;
}
```

**Kết quả:** sizeof (rn) = 1, sizeof (pn) = 8

- Tại sao dùng tham chiếu thay cho con trỏ?

- + Tham chiếu sạch hơn, không dễ gây lỗi như con trỏ, đặc biệt khi dùng trong hàm: tham chiếu đảm bảo không chiếu tới null
- + Sử dụng tham chiếu trong nguyên mẫu hàm giúp cho việc gọi hàm dễ hiểu hơn: không cần dùng toán tử địa chỉ

## 6.2 Hằng tham chiếu

```
int& n = 4; //error
const int& n = 4;

int m = 5;
const int& n = m;
```

- **Hằng tham chiếu làm tham số hàm:**

+ Có hai lý do để truyền tham biến, nhưng nếu hàm được truyền không cần sửa đối tượng được truyền thì ta nên khai báo tham biến là const. Có hai ích lợi:

1. Nếu trong hàm, ta lỡ sửa đổi tham số thì trình biên dịch sẽ bắt lỗi, ngăn chặn được một số lỗi lập trình viên có thể phạm
2. Ta có thể truyền các đối số là hằng hoặc không phải hằng cho hàm có hằng tham biến. Ngược lại, đối với các hàm có tham biến không phải là hằng, ta không thể truyền hằng làm đối số.

```
void non_constRef(LargeObj& Lo) { Lo.height += 10; } // Fine
void constRef(const LargeObj& Lo) { Lo.height += 10; }
// Error: Lo là hằng nên không được sửa đổi
```

```

void non_constRef(LargeObj& Lo) { cout << Lo.height; }
void constRef(const LargeObj& Lo) { cout << Lo.height; }
int main{
    LargeObj dinosaur; const LargeObj rocket;
    non_constRef(dinosaur);
    constRef(dinosaur);
    non_constRef(rocket);
    // Error: rocket là hằng, nên không thể làm đối số không phải hằng
    constRef(rocket);
}

```

### 6.3 Truyền tham số cho hàm bằng con trỏ / tham chiếu

<pre> void swap(int* a, int* b) {     int temp = *a;     *a = *b;     *b = temp; } </pre>	<pre> void swap_v2(int&amp; a, int&amp; b) {     int temp = a;     a = b;     b = temp; } </pre>
---	--

- Tham chiếu có thể được dùng độc lập nhưng thường hay được dùng làm tham số cho hàm
- C truyền mọi đối số cho hàm bằng giá trị (truyền trị - call by value)
  - “ Khi cần, ta có thể truyền một con trỏ tới đối tượng (chính nó cũng được truyền bằng giá trị)
- C++ cho phép các đối số hàm được truyền bằng tham chiếu (call by reference)
  - “ void myFunction(myObj\* obj) {...}
  - “ void myFunction(myObj& obj) {...}
  - “ myObj& có nghĩa “tham chiếu tới myObj”
  - “ đối với các đối số là các đối tượng lớn, truyền bằng tham chiếu đỡ tốn kém hơn truyền bằng giá trị (do chỉ truyền địa chỉ bộ nhớ)
- **Ví dụ:** tham chiếu làm đối số cho hàm

```

int f(int &i) { ++i; return i; }
int main() {
    int j = 7; cout << f(j) << endl; cout << j << endl;
}

```

- + Biến i là một biến địa phương của hàm f. i thuộc kiểu tham chiếu int và được tạo khi f được gọi.
- + Trong lời gọi f(j), i được tạo tương tự như trong lệnh int &i = j;
- + Do đó trong hàm f, i sẽ là một tên khác của biến j và sẽ luôn như vậy trong suốt thời gian tồn tại của i

### 6.4 Tham chiếu đến con trỏ

<pre> #include &lt;iostream&gt; using namespace std; void function_a(int*&amp; a) {     *a += 5;     int k = 7;     a = &amp;k; }  void function_b(int* a) {     *a += 5;     int k = 7;     a = &amp;k; }  int main() {     int* myInt = new int(5);     int* myInt2 = new int(5);     function_a(myInt); //myInt = ?     function_b(myInt2); //myInt2=?     cout &lt;&lt; "myInt = " &lt;&lt; myInt &lt;&lt; "; myInt2 = " &lt;&lt; myInt2;     return 0; } </pre>	<p><b>Kết quả:</b> myInt = -858993460; myInt2 = 10</p> <p>* Trong hàm function_a, ta truyền vào một tham chiếu đến con trỏ myInt. Khi thay đổi giá trị của a trong hàm này (a = &amp;k;), ta thực sự đang thay đổi con trỏ myInt mà hàm main đang giữ. <i>Tuy nhiên, k là một biến cục bộ của hàm function_a, vì vậy sau khi hàm này kết thúc, k không còn tồn tại nữa và con trỏ myInt đang trỏ đến một vùng nhớ không hợp lệ.</i> Điều này có thể dẫn đến các lỗi không xác định khi cố gắng truy cập vào nó (*myInt).</p>
--	--



```

#include <iostream>
using namespace std;
void function_a(int*& a) {
    *a += 5;
}

void function_b(int* a) {
    *a += 5;
}

int main() {
    int* myInt = new int(5);
    int* myInt2 = new int(5);
    function_a(myInt); //myInt = ?
    function_b(myInt2); //myInt2 = ?
    cout << "myInt = " << *myInt << "; myInt2 = " << *myInt2;
    delete myInt;
    delete myInt2;
    return 0;
}

```

**Kết quả:**  
myInt = 10; myInt2 = 10

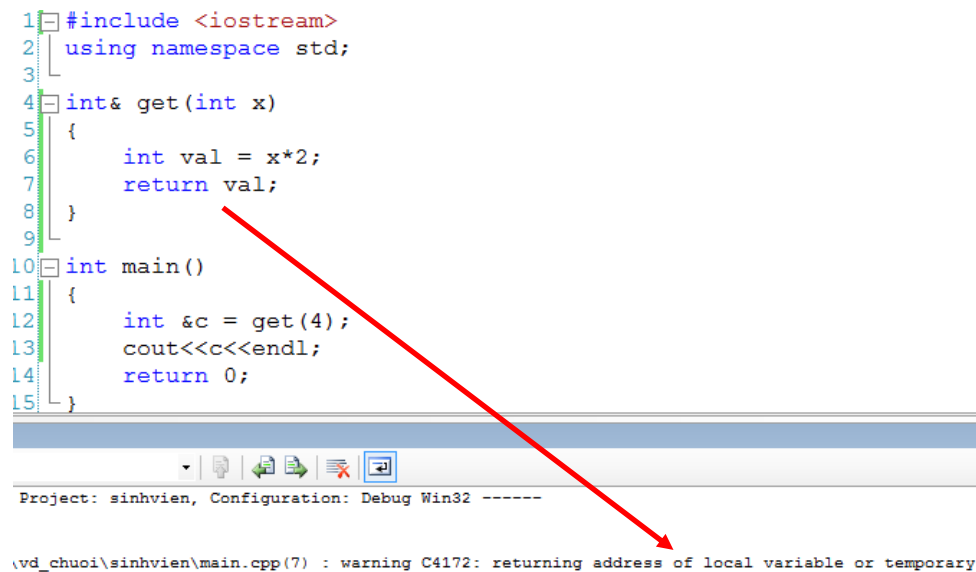
## 6.5 Giá trị trả về của hàm là tham chiếu

- Khai báo

```

type& func(...)
{
    ...
    return <biến_phạm_vi_toàn_cục>
}

```



```

1 #include <iostream>
2 using namespace std;
3
4 int& get(int x)
5 {
6     int val = x*2;
7     return val;
8 }
9
10 int main()
11 {
12     int &c = get(4);
13     cout<<c<<endl;
14     return 0;
15 }

```

Project: sinhvien, Configuration: Debug Win32 -----

\\vd\_chuoi\sinhvien\main.cpp(7) : warning C4172: returning address of local variable or temporary

A red arrow points from the `return val;` line in the `get` function to the warning message.

## 7. Định nghĩa chồng hàm, tham số ngầm định

### 7.1 Định nghĩa chồng hàm (Overloading function)

- Sử dụng một tên cho nhiều hàm khác nhau
- Trình dịch gọi các hàm này dựa vào: Số lượng tham số; Kiểu của tham số

**Chú ý:** Không dựa vào kiểu trả về của hàm

```

2 | using namespace std;
3 |
4 | int get(int x)
5 | {
6 |     x = x*2;
7 |     return x;
8 | }
9 | float get(int x)
10 | {
11 |     float f = 2*x;
12 |     return f;
13 | }
14 |
15 | int main()
16 | {

```

Project: sinhvien, Configuration: Debug Win32 -----

\vd\_chuoi\sinhvien\main.cpp(10) : error C2556: 'float get(int)' : overloaded function differs only by return type from 'int get(int)'

```

#include <iostream>
using namespace std;

int operate(int a, int b)
{
    return (a * b);
}

float operate(float a, float b)
{
    return (a / b);
}

int main()
{
    int x = 5, y = 2;
    float n = 5.0, m = 2.0;
    cout << operate(x, y);
    cout << "\n";
    cout << operate(n, m);
    cout << "\n";
    return 0;
}

```

**Kết quả:**

10  
2.5

## 7.2 Tham số ngầm định

- Là tham số của hàm có giá trị mặc định
- Mục đích: Gán các giá trị mặc nhiên cho các tham số của hàm.
- Khai báo tham số mặc nhiên:
  - + Tham số với giá trị ngầm định *phải nằm cuối cùng trong danh sách tham số*
  - + Chỉ cần đưa vào khai báo, không cần trong định nghĩa.
- Gọi hàm có tham số mặc nhiên:
  - + Nếu cung cấp đủ tham số → dùng tham số truyền vào.
  - + Nếu không đủ tham số → dùng tham số mặc nhiên.
- Ưu điểm:
  - + Không cần hiểu rõ ý nghĩa tất cả các tham số
  - + Có thể giảm được số lượng hàm cần định nghĩa
- Quy tắc tái định nghĩa: Các hàm trùng tên phải khác nhau về tham số: Số lượng, thứ tự, kiểu
- Quy tắc gọi hàm?
  - + Tìm hàm có kiểu tham số phù hợp
  - + Dùng phép ép kiểu tự động
  - + Tìm hàm gần đúng (phù hợp) nhất

**Ví dụ:**

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

void print(int val1, int val2 = 20)
{
    cout << "value1 = " << val1 << endl;
    cout << "value2 = " << val2 << endl;
}

int main()
{
    print(1);
    print(3,4);
    return 0;
}
```

**Kết quả:**

```
value1 = 1
value2 = 20
value1 = 3
value2 = 4
```

## 8. Quản lý bộ nhớ động

- Cấp phát bộ nhớ động trong C trông rối rắm và dễ lỗi

```
myObj* obj = (myObj*)malloc(sizeof(myObj));
```

- Các nhà thiết kế C++ thấy rằng:

.. Một ngôn ngữ sử dụng class sẽ hay phải sử dụng bộ nhớ động.

.. Không có lý do gì để tách cấp phát bộ nhớ động ra khỏi việc khởi tạo đối tượng (hay tách thu hồi bộ nhớ động ra khỏi việc hủy đối tượng)

- malloc và free đã được thay bằng new và delete

### 8.1 Toán tử cấp phát bộ nhớ new

```
type *pointer = new type[n]
```

- Cấp phát vùng nhớ với kích thước n\*sizeof(type)

**Ví dụ 1:**

```
int *a = new int;
int *arr = new int[10];
```

**Ví dụ 2:**

```
Student* student = NULL;
int n = 10;
student = new Student [n];
```

### 8.2 Toán tử giải phóng bộ nhớ delete

```
delete [ ] pointer;
```

- Toán tử delete giải phóng vùng nhớ được cấp phát bởi toán tử new

**Ví dụ 1:**

```
delete a;
a = NULL;
delete [ ] arr;
arr = NULL; //tùy ý
```

**Ví dụ 2:**

```
delete [ ] student; student = NULL; //tùy ý
```

\* Nếu quên [ ] → chỉ giải phóng phần tử đầu tiên của mảng, các phần tử còn lại chưa được giải phóng nhưng không thể truy cập được

\* Gán arr = NULL để đảm bảo arr không trỏ đến vùng nhớ nào

### 8.3 Lợi thế của new so với malloc

.. Không cần chỉ ra lượng bộ nhớ cần cấp phát

.. Không cần đổi kiểu

.. Không cần dùng lệnh if để kiểm tra xem bộ nhớ đã hết chưa

.. Nếu bộ nhớ đang được cấp cho một đối tượng, hàm khởi tạo (constructor) của đối tượng sẽ được gọi tự động (tương tự, delete sẽ tự động gọi hàm hủy (destructor) của đối tượng)

## 9. Hàm Inline

- Hàm inline hay còn gọi là hàm nội tuyến.
- Sử dụng từ khóa inline
- Yêu cầu trình biên dịch copy code vào trong chương trình thay vì thực hiện lời gọi hàm:
  - + Giảm thời gian thực thi chương trình
  - + Tăng kích thước của mã lệnh thực thi
  - + Chỉ nên định nghĩa inline khi hàm có kích thước nhỏ

```
inline float sqr(float x) {           inline int  Max(int a, int b) {
    return (x*x);                      return ((a>b) ? a : b) ;
}
```

## 10. Const

- Trong C, hằng được định nghĩa bằng định hướng tiền xử lý #define

```
#define PI 3.14
```

  - .. Biên dịch chậm hơn (trình tiền xử lý tìm và thay thế)
  - .. Trình debug không biết đến các tên hằng
  - .. Sử dụng #define không gắn được kiểu dữ liệu với giá trị hằng (v.d. '15' là int hay float?)
- Const của ANSI-C ít dùng hơn và có nghĩa hơi khác: ANSI-C const không được trình biên dịch chấp nhận là hằng, còn với C++ thì được.
- Hằng của ANSI-C và C++ có các quy tắc phạm vi khác nhau
  - + Các giá trị #define có phạm vi file (do trình tiền xử lý chỉ thực hiện tìm và thay thế tại file đó)
  - + const của ANSI-C được coi là các biến có giá trị không đổi và có phạm vi chương trình. Do đó ta không thể có các biến trùng tên tại hai file khác nhau
  - + Trong C++, các định danh const tuân theo các quy tắc phạm vi như các biến khác. Do đó, chúng có thể có phạm vi toàn cục, phạm vi không gian tên, hoặc phạm vi khối

### - const và con trỏ

- + Ba loại:
  1. const int \* pi; // con trỏ tới hằng
  2. int \* const ri = &i; // hằng con trỏ
  3. const int \* const ri = &i; //hằng con trỏ tới hằng
- + Nếu một đối tượng là hằng
  - .. Không thể sửa đổi đối tượng đó
  - .. Chỉ có con trỏ tới hằng mới được dùng để trỏ tới hằng, con trỏ thường không dùng được
- + Nếu PI được khai báo là con trỏ tới hằng:
  - .. Có thể thay đổi PI, nhưng \*PI không thể bị thay đổi.
  - .. PI có thể trỏ đến hằng hoặc biến thường
- + Khi sử dụng một con trỏ, có hai đối tượng có liên quan: chính con trỏ đó và đối tượng nó trỏ tới
- + Cú pháp cho con trỏ tới hằng và hằng con trỏ rất dễ nhầm lẫn
- + Quy tắc: trong lệnh khai báo, từ khoá const bên trái dấu \* có nghĩa đối tượng được trỏ tới là hằng, từ khoá const bên phải dấu \* có nghĩa con trỏ là hằng
- + Cách dễ hơn: đọc các khai báo từ phải sang trái

```
char c = 'Y'; // c là char
char *const cpc = &c; //cpc là hằng con trỏ tới char
const char *pcc; // pcc là con trỏ tới hằng char
const char *const cpcc = &c; // cpcc là hằng con trỏ tới hằng char
```

### - const: Hàm thành viên

- + Đối với hàm thành viên không sửa dữ liệu của đối tượng chủ, ta nên khai báo hàm đó là hằng hàm
- + Đối với các đối tượng được khai báo là hằng, C++ chỉ cho phép gọi các hàm thành viên là hằng mà không cho phép gọi các hàm thành viên không phải là hằng của đối tượng đó.

```
class Date
{
    int year, month, day;
public:
    int getDay() const { return day; }
    int getMonth() const { return month; }
    void addYear(int y); // Non-const function
};
```