

Lập trình hướng đối tượng (Object – Oriented Programming)



PGS. TS. Trần Văn Lăng

Email: lang@hcmc.netnam.vn

☎ : 0903 938 036

Mục tiêu

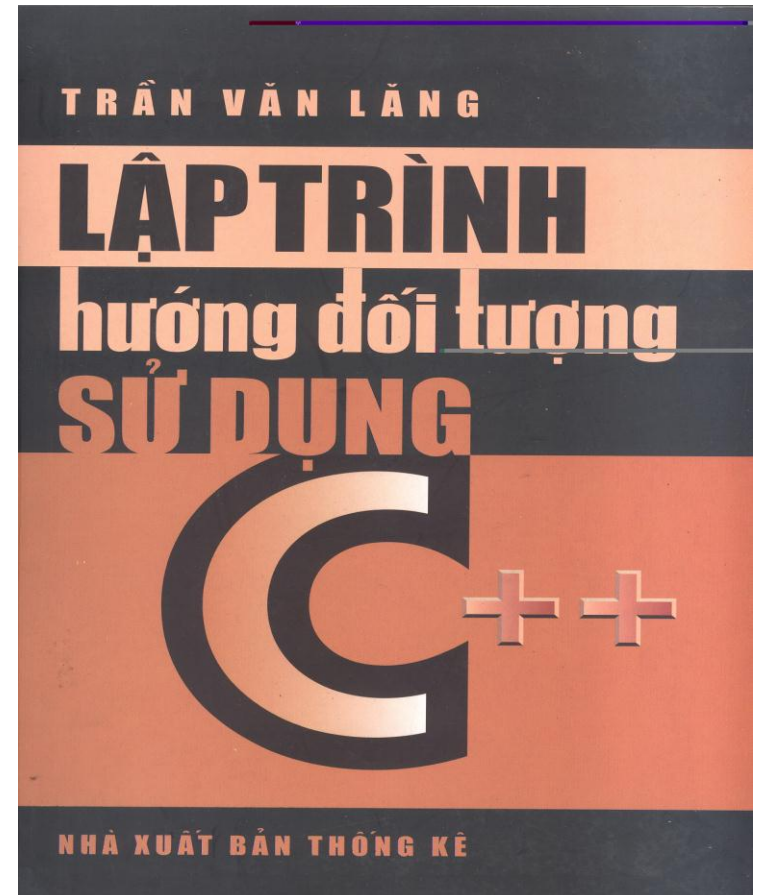
- Phương pháp viết chương trình hướng về với các đối tượng
- Sử dụng ngôn ngữ lập trình hướng đối tượng C++

Nội dung

- Một số khái niệm chung
- Ngôn ngữ C++
- Cách thức xây dựng lớp
- Vấn đề tạo đối tượng
- Kiểu dữ liệu lớp trong C++
- Tính thừa kế
- Tính đa hình

Giáo trình

- Trần Văn Lăng, *Lập trình hướng đối tượng sử dụng C++*, Nxb. Thống kê, 2004.



Tài liệu tham khảo

- John Hubbard, *Programming with C++*, McGraw-Hill, 1996.
- Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1997

Chương 1



Khái niệm về lập trình
hướng đối tượng

Lập trình hướng đối tượng là gì

- Object-Oriented Programming – OOP
- Lập trình cấu trúc: Procedural Programming
 - ◆ Phân công việc → những việc nhỏ hơn
 - ◆ Là các chương trình con
 - ◆ Thiết kế top-down

Chương trình = Dữ liệu + Thuật toán

➤ Lập trình hướng đối tượng

- ♦ Từ những đối tượng, sự vật, sự kiện, ... tạo nên chương trình
- ♦ Thiết kế bottom-up

Đối tượng = Dữ liệu + Hành vi



=

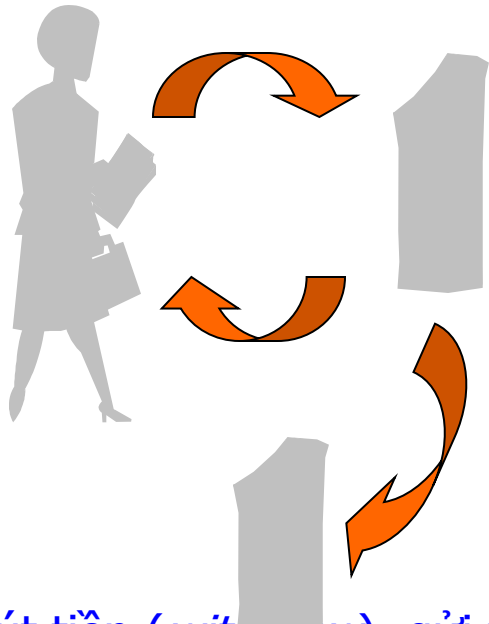


+



Sự khác biệt

➤ Theo thủ tục



Rút tiền (*withdraw*), gửi tiền
(*deposit*), chuyển tiền (*transfer*)

➤ Hướng đối tượng



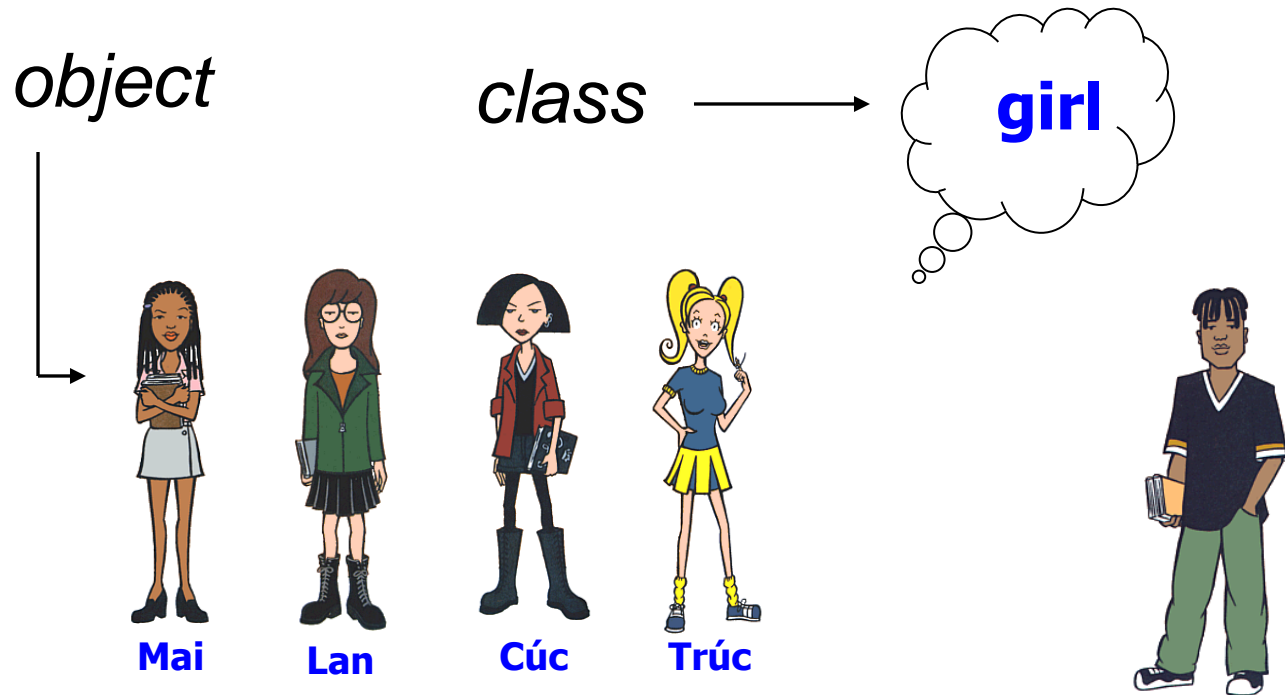
Khách hàng (*customer*), tiền
(*money*), tài khoản (*account*)

Đối tượng là gì ?



- Một đối tượng như là một hộp đen, mà chi tiết bên trong được dấu kín
- Các đối tượng giao tiếp với nhau thông qua việc truyền các thông điệp (*messages*)
- Thông điệp được nhận bởi các hành vi của đối tượng

Đối tượng



Như vậy,

➤ Trong đối tượng bao gồm:

- ◆ Hành vi (behavior), và
- ◆ Dữ liệu (data)

Hành vi của đối tượng là gì?



- Thao tác (*operation*)
- Phương thức (*method*)
- Hàm (*function*)
- Thủ tục (*procedure*)

Dữ liệu

- Thông tin (*information*)
- Tính chất (*property*)
- Thuộc tính (*attribute*)
- Trường (*field*)

Những gì là đối tượng



- Vật có thể sờ mó được
(*Tangible things*)

➤ Như là xe hơi, máy in, ...

- Vai trò (*Roles*)



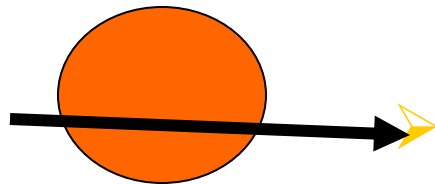
Công nhân, người chủ, ...

- Sự việc xảy ra, tình tiết
(*Incidents*)



Chuyến bay, tràn số, ...

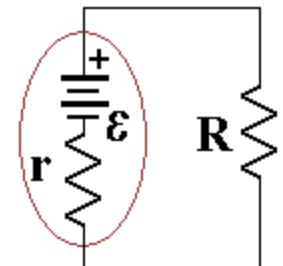
- Sự tương tác
(*Interactions*)



Ký kết thỏa ước, mua hàng, ...

- Sự mô tả (*Specifications*)

➤ Màu, hình dạng



Một số đặc tính

- All information in an object-oriented system is stored within its objects and can only be manipulated when the objects are ordered to perform operations

Ivar Jacobson

Như vậy,

➤ Tính đóng gói (*encapsulation*)

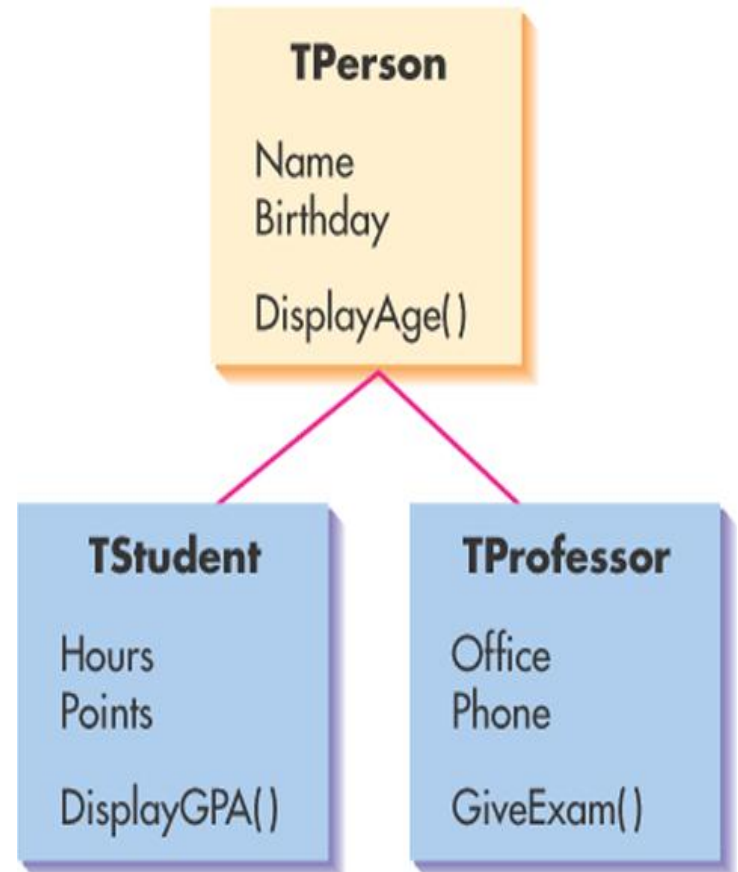
- ◆ Dữ liệu và thao tác được nhóm lại cùng nhau




Thực chất là sự
ghép chung những
hiểu biết về thể giới
thực → Có sự đồng
nhất giữa dữ liệu và
thao tác trên dữ
liệu

Tính thừa kế (inheritance)

- Tạo ra một kiểu dữ liệu mới từ kiểu đã có
- Nhằm sử dụng lại, và bổ sung những gì cần thiết
- Thực chất là sự phân lớp (*classification*) trong việc thiết kế hệ thống theo hướng đối tượng



- 
- Theo ngôn ngữ lớp, sự thừa kế có nghĩa là một lớp thừa kế các đặc tính của lớp khác.
 - Đây chính là quan hệ “là một” (“is a”)

A car *is a* vehicle

A dog *is an* animal

A teacher *is a* person

Tính đa hình (polymorphism)

- Nhiều đối tượng cùng chia sẻ đặc tính chung, nhưng có những tác động khác nhau.
- Có cùng yêu cầu, nhưng mỗi đối tượng có đáp ứng khác nhau.
- Thực chất là tính đa dạng (*many form*)
- Để hiện thực được tính đa hình, ngôn ngữ đối tượng có đặc tính như overload, override.

Overloaded và Overridden methods ?



➤ Overloaded methods:

- ◆ Nhằm cung cấp các dạng khác nhau của hành vi, nhưng vẫn có cùng tên gọi.

➤ Overridden methods:

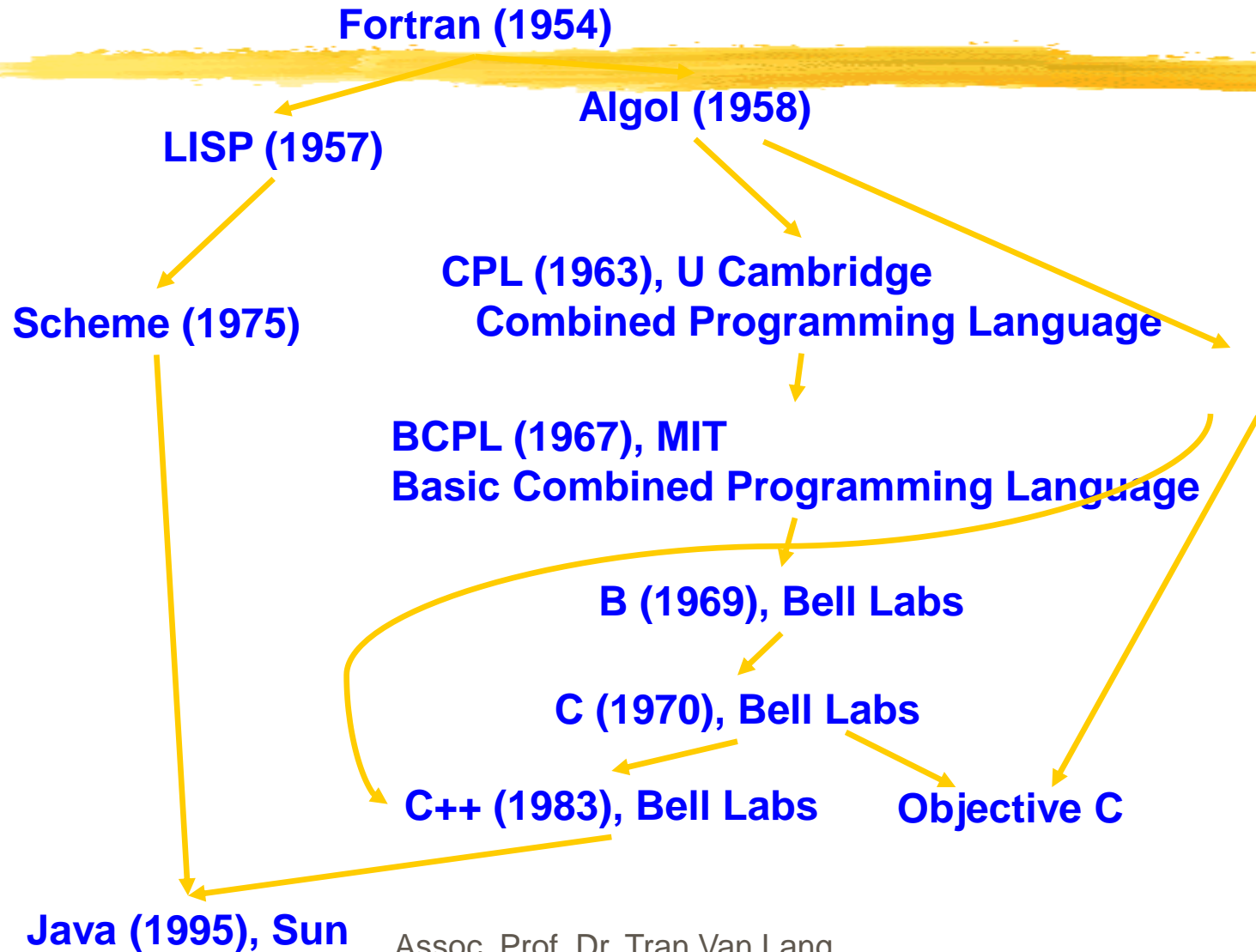
- ◆ Hiện thực lại hành vi đã có của tổ tiên
- ◆ Phải có cùng tên và trùng mọi yếu tố tạo nên hành vi này.

Ngôn ngữ lập trình hướng đối tượng

- SmallTalk (Palo Alto Research Center - PARC)
- Eiffel
- Object Pascal
- C++
- Java



Programming Languages Phylogeny



Viết chương trình

➤ Có 2 bước cơ bản để viết chương trình theo hướng đối tượng:

- ◆ *Tạo các lớp:* Tạo ra, mở rộng hoặc sử dụng lại các kiểu dữ liệu trừu tượng.
- ◆ *Tạo tương tác giữa các đối tượng:* Tạo các đối tượng từ các lớp và xác định mối quan hệ giữa chúng.

Lập trình hướng đối tượng (Object – Oriented Programming)



PGS. TS. Trần Văn Lăng


Email: lang@hcmc.netnam.vn

☎ : 0903 938 036

Chương 2



Ngôn ngữ C++

- 
- Khái quát về ngôn ngữ C++
 - Kiểu dữ liệu cơ bản, phép toán
 - Các cấu trúc điều khiển
 - Hàm
 - Mảng và mẫu tin
 - Con trỏ và tham chiếu
 - Nhập xuất và tập tin

Khái quát về ngôn ngữ C++

- 1970, Denis Ritchie (Bell Lab.) phát triển ngôn ngữ C.
 - ◆ Dạng System Implementation Language (SIL)
 - ◆ Phát triển từ ngôn ngữ CPL (Combined Programming Language), BCPL (Basic CPL) và ngôn ngữ B.
 - ◆ Brian Kernighan, D. Ritchie (1978), *The C Programming Language*, Prentice-Hall

➤ Đầu 1980, Bjarne Stroustrup phát triển ngôn ngữ C++

- ◆ Trên sở sở ngôn ngữ Simula 67
- ◆ Tương thích hoàn toàn với C
- ◆ Mở rộng C với cấu trúc OOP
- ◆ Tên gọi "C with Classes"
- ◆ Năm 1983, Ricj Mascitti đề nghị C++
- ◆ Bjarne Stroustrup (1985), *The C++ Programming Language*, Prentice-Hall

Cấu trúc chương trình C++

```
#include <iostream.h>
// Output string into display
main()
{
    cout << "Hello, world.\n"
    return 1;
}
```

Biến, đối tượng

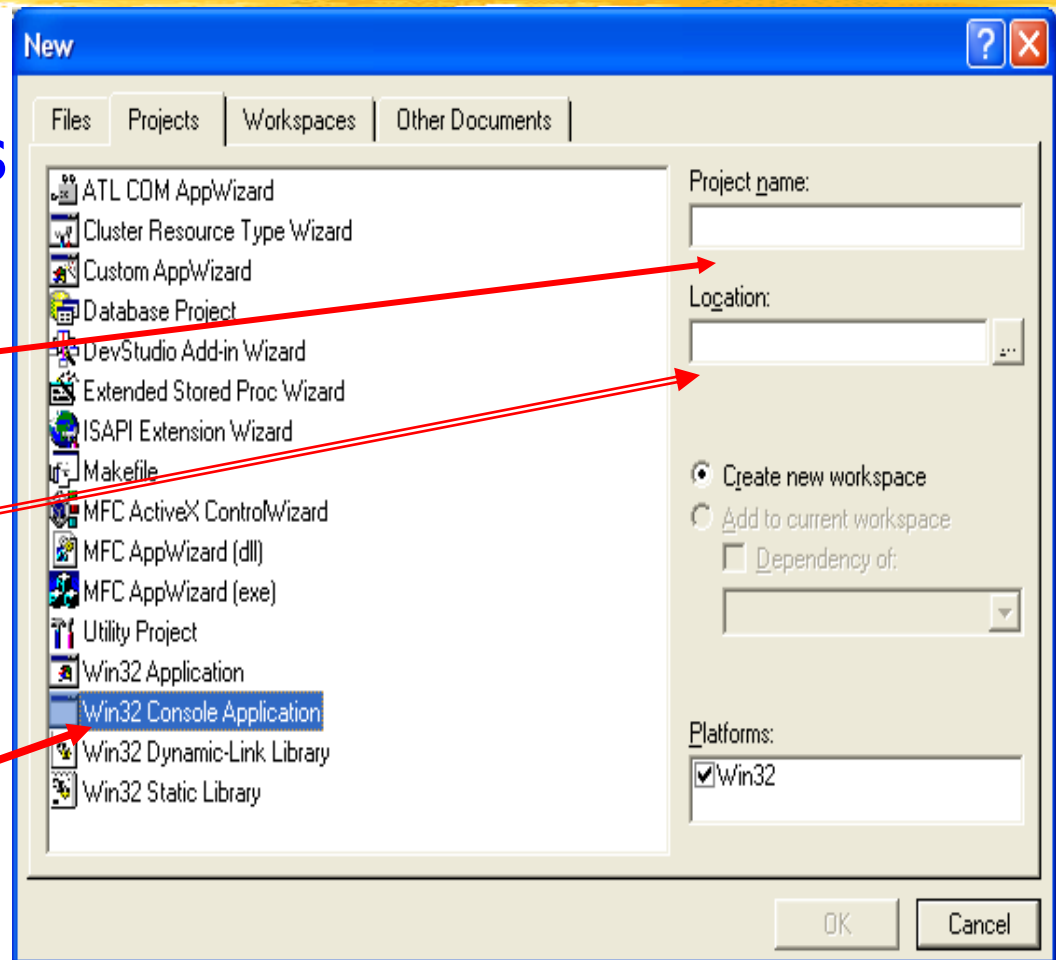
- Có thể khai báo ở bất kỳ vị trí nào (trước khi sử dụng)
 - ◆ Khai báo biến còn mang ý nghĩa thực thi câu lệnh, tạo đối tượng

Viết chương trình

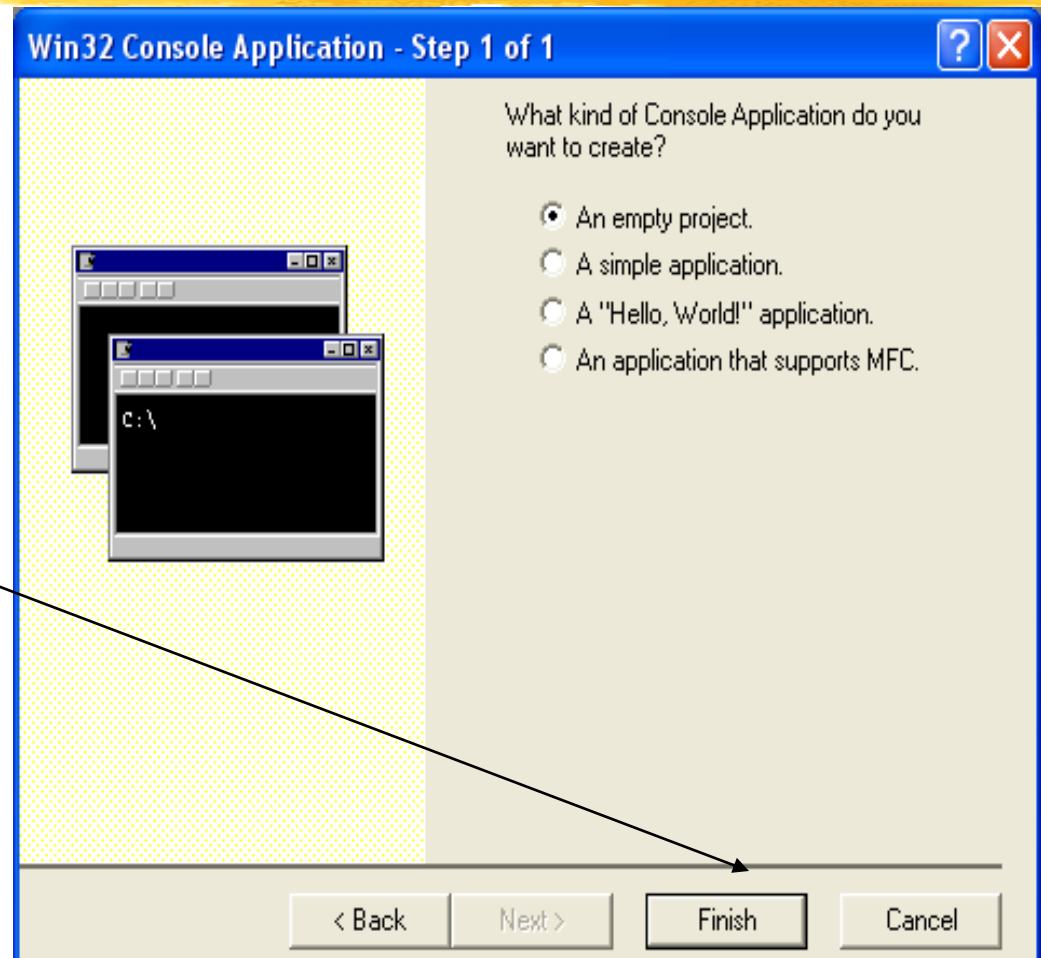
- Trên môi trường Windows, có thể sử dụng Visual C++, hoặc DJGPP, v.v...
- Trên Linux, có thể sử dụng GNU C++

Dùng Visual C++ 6.0

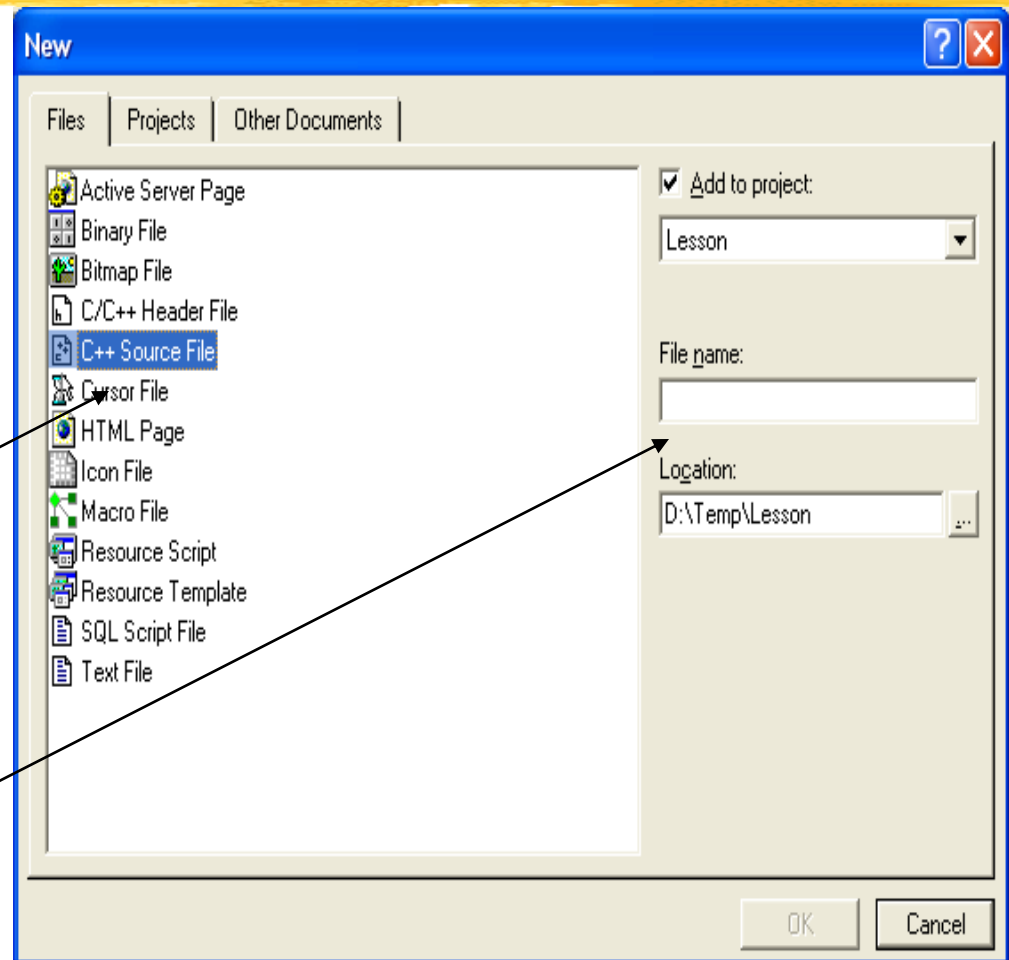
- Chọn chức năng File/New/Projects
- Đưa vào tên của Project
- Và vị trí lưu trữ trên đĩa
- Lưu ý, chọn Win32 Console Application



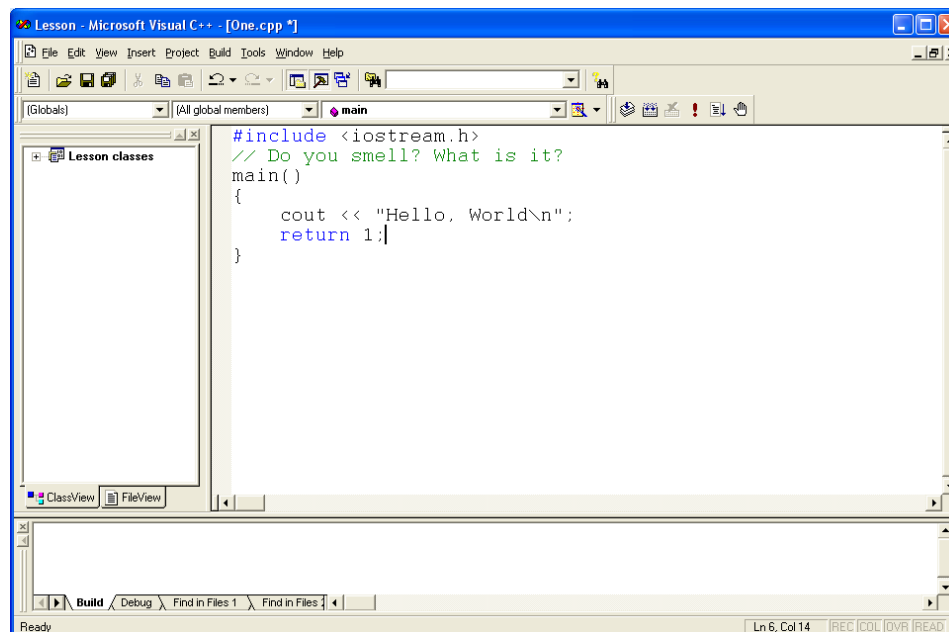
- Sau khi đưa vào Project Name và Location, hộp hội thoại xuất hiện
- Chọn Finish



- Đưa các tập tin nguồn vào để sử dụng, bằng cách
- Project/Add to Project/New/Files.
- Chọn C++ Source file
- Đưa vào tên file



- Sau khi chọn, có màn hình để soạn thảo tập tin One.cpp.
- Vào menu Build để biên dịch và thực thi chương trình.



Một số phép toán



➤ Phép toán nhập

```
#include <iostream.h>
```

```
main(){
```

```
    int age;
```

```
    cout << "When were you born? ";
```

```
    cin >> age;
```

```
    cout << "After 10 years, you will be "
```

```
        << 2008 - age + 10 << "years old\n"
```

```
    return 1;
```

```
}
```

➤ Phép toán gán

```
int y, x;
```

```
y = (x = 100);
```

Hay

```
y = x = 100;
```

➤ Phép toán tăng, giảm

```
int n, m = 10;  
n = m++;  
cout << n++ << endl;
```

```
int n, m = 10;  
n = m;  
m = m + 1;  
cout << n << endl;  
n = n + 1;
```

```
int n, m = 10;  
n = ++m;  
cout << ++n << endl;
```

```
int n, m = 10;  
m = m + 1;  
n = m;  
n = n + 1;  
cout << n << endl;
```

➤ Phép toán điều kiện

```
if ( a > 100.0 )  
    m = 5;  
else  
    m = a + 10;
```

```
m = a > 100.0 ? 5 : a + 10;
```

```
if ( a > b )  
    max = a;  
else  
    max = b;
```

```
max = a > b ? a : b;
```


Độ ưu tiên phép toán

Phép toán	Mô tả	Ưu tiên	Thứ tự trong biểu thức	Toán hạng	Ví dụ
::	Truy cập biến toàn cục	17	Phải sang	1	::x
::	Phân định thành phần của lớp	17	Trái sang	2	NAME::x
.	Truy cập thành phần của đối tượng hay mẫu tin	16	- nt -	- nt -	obj.n
->	Truy cập đến thành phần của con trỏ đối tượng (hoặc mẫu tin)	16	- nt -	- nt -	obj->n
[]	Truy cập chỉ số	16	- nt -	- nt -	a[i]
()	Gọi hàm	16	- nt -		
()	Chuyển đổi kiểu	16	- nt -		int(ch)
++	Tăng sau	16	Phải sang	1	n++
--	Giảm sau	16	- nt -	- nt -	n--

Phép toán	Mô tả	Ưu tiên	Thứ tự trong biểu thức	Toán hạng	Ví dụ
sizeof	Kích thước của đối tượng hoặc của kiểu dữ liệu	15	- nt -	- nt -	sizeof(a)
++	Tăng trước	15	- nt -	- nt -	++n
--	Giảm trước	15	- nt -	- nt -	--n
~	Bitwise NOT	15	- nt -	- nt -	~s
!	Phủ định	15	- nt -	1	!q
+	Chuyển thành dương	15	- nt -	- nt -	+n
-	Chuyển thành âm	15	- nt -	- nt -	-n
*	Lấy giá trị tại địa chỉ	15	Phải sang	- nt -	*ptr
&	Truy cập địa chỉ	15	- nt -	- nt -	&x

Phép toán	Mô tả	Ưu tiên	Thứ tự trong biểu thức	Toán hạng	Ví dụ
new	Cấp phát bộ nhớ	15	- nt -	- nt -	new p
delete	Thu hồi bộ nhớ	15	- nt -	- nt -	delete p
()	Chuyển đổi kiểu	15	- nt -	- nt -	(int) ch
. *	Truy cập đến thành phần của đối tượng hay của mẫu tin	14	Trái sang	2	x.*ptr
->*	Truy cập đến thành phần của con trỏ đối tượng (hoặc mẫu tin)	14	- nt -	- nt -	p->*ptr
*	Nhân	13	- nt -	- nt -	a*b
/	Chia	13	- nt -	- nt -	m/n
%	Chia lấy phần dư	13	- nt -	- nt -	m%n
+	Cộng	12	- nt -	- nt -	m+n
-	Trừ	12	- nt -	- nt -	m-n

Phép toán	Mô tả	Ưu tiên	Thứ tự trong biểu thức	Toán hạng	Ví dụ
<<	Bit shift left	11	- nt -	- nt -	
>>	Bit shift right	11	- nt -	- nt -	
<	Nhỏ hơn	10	- nt -	- nt -	$x < y$
<=	Nhỏ hơn hay bằng	10	- nt -	- nt -	$x \leq y$
>	Lớn hơn	10	- nt -	- nt -	$x > y$
>=	Lớn hơn hay bằng	10	- nt -	- nt -	$x \geq y$
==	So sánh bằng	9	- nt -	- nt -	$x == y$
!=	Không bằng	9	- nt -	2	$x != y$
&	Bitwise AND	8	- nt -	- nt -	$m \& n$
^	Bitwise XOR	7	- nt -	- nt -	$m \wedge n$
	Bitwise OR	6	- nt -	- nt -	$m n$
&&	Phép toán AND	5	- nt -	- nt -	$p \&\& q$
	Phép toán OR	4	- nt -	- nt -	$p q$

Assoc. Prof. Dr. Tran Van Lang

Phép toán	Mô tả	Ưu tiên	Thứ tự trong biểu thức	Toán hạng	Ví dụ
? :	Điều kiện	3	Phải sang	- nt -	q ? x : y
=	Gán	2	- nt -	- nt -	n = 10
+=	Gán cộng dồn	2	- nt -	- nt -	n += 10
-=	Gán trừ dồn	2	- nt -	- nt -	n -= 10
*=	Gán nhân	2	- nt -	- nt -	n *= 2
/=	Gán chia	2	- nt -	- nt -	n /= 5
%=	Gán modulo	2	- nt -	- nt -	n %= 2
&=	Gán Bitwise AND	2	- nt -	- nt -	n &= mask
^=	Gán Bitwise XOR	2	- nt -	- nt -	n ^= mask
=	Gán Bitwise OR	2	- nt -	- nt -	n = mask
<<=	Gán Bit shift left	2	- nt -	- nt -	n <<= 1
>>=	Gán Bit shift right	2	- nt -	- nt -	n >>= 1
throw	Throw exception	1	- nt -	1	throw(20)
,	Dấu phẩy	0	Trái sang	2	++m, --n


Cấu trúc điều khiển




- Tuần tự
- Cấu trúc điều kiện
- Cấu trúc lặp

➤ Cấu trúc điều kiện, dùng câu lệnh:

- ♦ `if (condition) st1
else st2`
- ♦ `switch (expression){
case value1: st1;
case value2: st2;
...
default: stn;
}`

- 
- Lưu ý, trong C/C++ không có kiểu luận lý, nên các biểu thức có giá trị nguyên.
 - Giá trị khác không, mang ý nghĩa đúng. Ngược lại, mang ý nghĩa sai

- 
- Cấu trúc lặp có các dạng thể hiện:
 - ◆ `while (condition) st;`
 - ◆ `do st while (condition);`
 - ◆ `for (init; condition; update) st;`
 - Ngoài ra,
 - ◆ `break`: để thoát khỏi chu trình lặp
 - ◆ `continue`: chuyển qua lần lặp kế tiếp

Một vài ví dụ



➤ Tìm các số nguyên tố

Yêu cầu

- Sử dụng được một trình biên dịch C/C++ nào đó để viết chương trình.
- Viết được chương trình cơ bản dùng các cấu trúc điều khiển trên các kiểu dữ liệu cơ bản

Object – Oriented Programming



PGS. TS. Trần Văn Lăng

☎: 0903 938 036

Email: lang@hcmc.netnam.vn, langtv@gmail.com

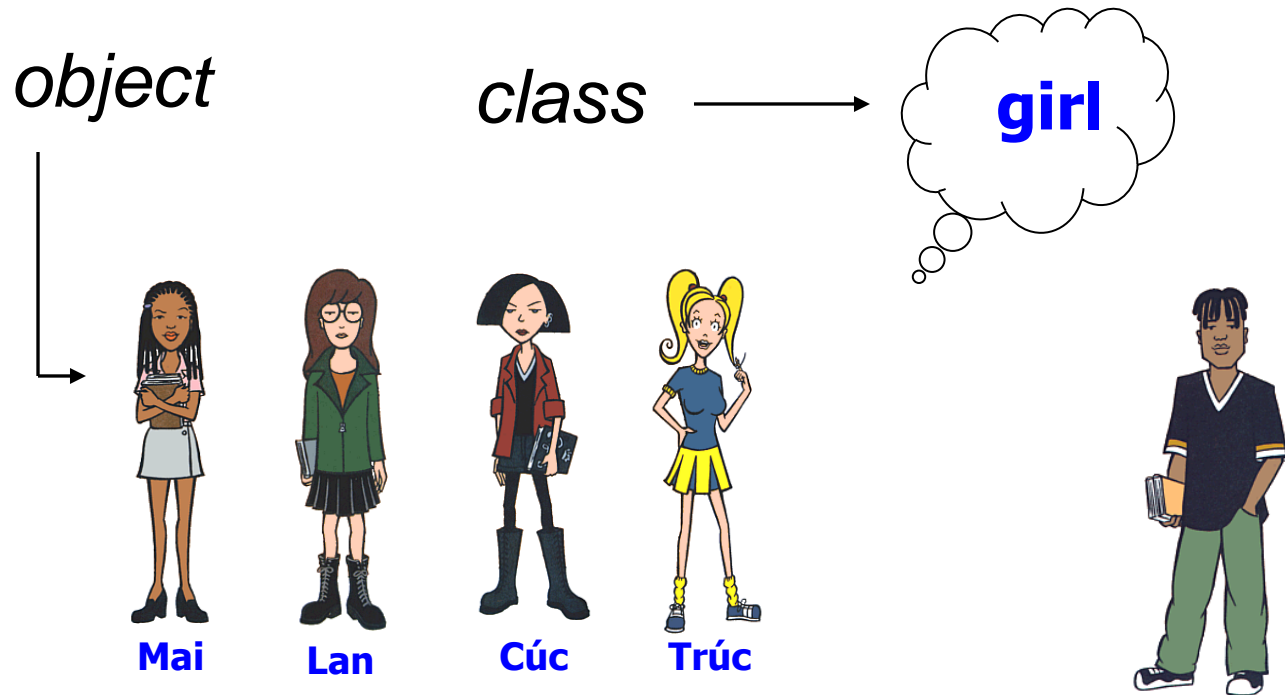
Chương 3



Cách thức xây dựng lớp


Lớp là gì ?

- Khi một số các đối tượng cùng tính chất được nhóm lại, tạo nên lớp



Như vậy,

- Lớp được dùng để mô tả tất cả các đối tượng có hành vi và dữ liệu tương tự nhau.
- Một lớp là một mẫu (template) hay một khuôn dạng (mold) để từ đó có thể tạo ra những đối tượng mới.

- 
- Có thể xem lớp như một dạng đối tượng (object's type)
 - Hay kiểu dữ liệu (data type)

Lớp là sự biểu diễn của một mẫu các đối tượng và mô tả cách mà những đối tượng này được cấu tạo bên trong

Hiện thực lớp trong C++


```
class NAME{  
    //members  
};
```

- Như vậy, lớp được bao bọc bởi từ khóa class ở bên ngoài.
- Bên trong là các thành phần, bao gồm dữ liệu và hành vi.

Ví dụ, lớp về người

```
class PERSON{  
    char name[40];  
    int birthYear;  
public:  
    void getData();  
    int age()  
};
```

Để đơn giản,
trước mắt chỉ
quan tâm đến
vài dữ liệu




➤ Trong đó,

```
void getData(){  
    cout << "Name: ";  
    cin.getline(name, 39);  
    cout << "Year of birth: ";  
    cin >> birthYear;  
}
```



Do muốn nhập được cả họ và tên



```
int age(){  
    cout << name << " is "  
    << 2008 - birthyear  
    << " years old\n"  
}
```

So với struct trong C/C++

- **class** có thêm các hàm chứa bên trong
- Các hàm phải chỉ định thêm thuộc tính truy cập (*access attributes*), chẳng hạn
 - ◆ **public**
 - ◆ **private**
 - ◆ **protected**
- Các dữ liệu thành viên cũng phải có thuộc tính truy cập

➤ Chẳng hạn,
`struct PERSON{
 char name[40];
 int birthYear;
};`

➤ Tương đương với
`class PERSON{
public:
 char name[40];
 int birthYear;
};`

Khi đó mới có thể truy cập được **name**,
birthYear như đã làm trong kiểu **struct**.

Hoàn thiện lớp

- Lớp **PERSON** ở trên mới chỉ mang tính mô tả, chưa sử dụng được
- Để hoàn thiện, có 2 cách viết khác nhau
 - ◆ Đơn giản
 - ◆ Phức tạp

Cách đơn giản

```
class PERSON{
    char name[40];
    int birthYear;
public:
    void getData(){
        cout << "Name: ";
        cin.getline(name, 39);
        cout << "Year of birth: ";
        cin >> birthYear;
    }
    int age(){
        cout << name << " is " << 2008 - birthYear
        << " years old\n"
    }
};
```


Phức tạp và chuyên nghiệp hơn

```
class PERSON{
    char name[40];
    int birthYear;
public:
    void getData();
    int age();
};

void PERSON::getData(){
    cout << "Name: ";
    cin.getline(name, 39);
    cout << "Year of birth: ";
    cin >> birthYear;
}

int PERSON::age(){
    cout << name << " is " << 2008 - birthYear << "
    years old\n"
}



```

Lưu ý,

- Trong lớp `PERSON` cũng có một vài điều nhỏ nhỏ cần quan tâm về việc dùng ngôn ngữ C++.
 - ◆ Do phép toán trích (`>>`) chỉ nhận chuỗi ký tự không chứa ký tự blank (space, tab, new line), nên phải dùng hàm `getline()`.
 - ◆ Hàm `getline()` có trong lớp chứa đối tượng `cin`.


Ví dụ

```
#include <iostream.h>
main() {
    char name[4][40];
    int n = 0;
    cout << "Name of people loved me\n";
    while(cin.getline(name[n++],40));
    --n;
    cout << "They are as follow\n";
    for ( int i = 0; i < n; i++ )
        cout << name[i] << "\n"; return 1;
}
```

- 
- Sau khi có lớp, có thể tạo đối tượng bằng cách khai báo biến thuộc kiểu lớp

PERSON a, b;

- **a, b** là 2 đối tượng.
- Khi đó **a** và **b** có thể coi là 2 instance (là ví dụ, là trường hợp, là cái có thực) của lớp **PERSON**.

- 
- Trong ví dụ trên, có thể tạo các đối tượng như sau:

```
main(){  
    PERSON a;  
    a.getData();  
    a.age()  
}
```

Thuộc tính truy cập là gì ?

- Hay còn gọi hình thức truy cập, hay tầm nhìn.
- Có 3 thuộc tính, nhằm để quy định khả năng truy cập đến các thành viên của lớp, có "trông thấy" được thành viên nào đó không:
 - ◆ private
 - ◆ protected
 - ◆ public

public

- Từ bất kỳ hàm nào có chứa đối tượng thuộc lớp, đều truy cập được các thành viên có thuộc tính này.
 - ◆ Vì vậy những thành viên mang thuộc tính **public** còn được xem là thành viên có khả năng giao tiếp với môi trường bên ngoài.

private

- **private**: ngược lại với **public**, các thành viên mang thuộc tính **private** chỉ được truy cập từ những hành vi thuộc lớp và từ những hành vi bè bạn (**friend**), từ những lớp là bè bạn của nó.
 - ◆ **private** là thuộc tính chuẩn của ngôn ngữ C++, để thông báo rằng đây là những thành viên riêng tư của lớp, nội bộ của lớp mới nhìn thấy, mới nói chuyện được, chúng không giao tiếp với thế giới bên ngoài lớp.

protected

- **protected**: để cho phép các thành viên trong những lớp hậu duệ được quyền truy cập đến.
 - ◆ Nói cách khác, ngoài việc nói chuyện với các thành viên của lớp, những thành viên có thuộc tính truy cập **protected** còn có thể giao tiếp với các thành viên trong lớp con cháu.

Giả sử có lớp sau đây

```
#include <iostream.h>
class POINT{
    int x, y;
public:
    void set( int xx, int yy ){
        x = xx;
        y = yy;
    }
    int get( int& xx, int& yy ){
        xx = x;
        yy = y;
        return 1;
    }
};
```



```
main()
```

```
{
```

```
    POINT p;
```

```
    p.set( 10, 5 );
```

```
    cout << p.x << " " << p.y <<
```

```
    endl; return 1;
```

```
}
```



Không
được

Sửa lại

```
main()
{
    POINT p;
    p.set( 10, 5 );
    int x, y;
    p.get( x,y );
    cout << "Coordinate (" << x
    << "," << y << ")\n";
    return 1;
}
```

Một vài ví dụ

➤ Lớp về ngăn xếp

```
class STACK{
    int top;
    char data[100];
public:
    int push( char );
    int pop( char& );
    void init();
private:
    int empty();
    int full();
};
```

```
void STACK::init(){
    top = -1;
}
int STACK::empty(){
    return top == -1 ? 1: 0;
}
int STACK::full(){
    return top == 99 ? 1: 0;
}
int STACK::push( char c ){
    if ( full() )
        return 0;
    else{
        data[++top] = c;
        return 1;
    }
}
int STACK::pop( char& c ){
    if ( empty() )
        return 0;
    else{
        c = data[top--];
        return 1;
    }
}
```

➤ Lớp hàng đợi

```
class QUEUE{
    int rear, front;
    char data[1000];
    int full();
    int empty();
public:
    void init();
    int add( char );
    int remove( char&
);
};
```

```
#define MAX 32
void QUEUE::init(){
    rear = front = 0;
}
int QUEUE::add( char c ){
    if ( full() )
        return 0;
    else{
        data[rear] = ch;
        rear = (rear+1) % MAX;
        return 1;
    }
}
int QUEUE::remove( char& c ){
    if ( empty() )
        return 0;
    else{
        ch = data[front];
        front = (front+1) % MAX;
        return 1;
    }
}
int QUEUE::full(){
    return (rear+1) % MAX == front ? 1:0;
}
int QUEUE::empty(){
    return rear == front ? 1:0;
}
```

Loại hành vi của lớp

- Tự động thực hiện
- Nhận biết và thay đổi giá trị dữ liệu
- Phép toán
- Đặc thù của lớp

Mỗi hành vi có thể có thuộc tính truy cập khác nhau. Những hành vi chỉ phục vụ cho hành vi khác của lớp sẽ có thuộc tính là **private**.

Yêu cầu

- Xây dựng lớp cơ bản dùng C/C++
- Phân biệt được public, private
- Cách viết chương trình chính sử dụng lớp, cách truy cập đến thành viên của lớp

Object – Oriented Programming



PGS. TS. Trần Văn Lăng

Email: lang@hcmc.netnam.vn

Chương 4



Con trỏ và tham chiếu

Truy cập địa chỉ


- Trong hầu hết các ngôn ngữ máy tính, khi một biến được khai báo, ba thuộc tính cơ bản sau đây được liên kết với nó:
 - ◆ Tên định danh của biến.
 - ◆ Kiểu dữ liệu liên quan.
 - ◆ Địa chỉ trong bộ nhớ.



➤ Chẳng hạn, với khai báo:

`int n;`

- Thì `n` là tên định danh của biến, có kiểu là số nguyên dạng `int`, và được lưu trữ đâu đó trong bộ nhớ máy tính.
- Khi đó để truy cập đến biến chúng ta sử dụng tên định danh của nó.

- 
- Ví dụ: Để xuất kết quả ra màn hình, chúng ta viết bằng câu lệnh quen thuộc

`cout << n;`

- Khi cần truy cập địa chỉ của n trong bộ nhớ, chẳng hạn


`cout << &n;`

Trong C/C++, dùng phép toán & để truy cập đến địa chỉ của một đối tượng

Biến tham chiếu

- Khi cần dùng tên gọi khác để truy cập đến cùng một địa chỉ với biến đã có, chúng ta sử dụng biến tham chiếu (*references*).
- Biến tham chiếu là một bí danh (*alias*) của biến khác.

Biến tham chiếu cũng là biến, nên được dùng như bình thường.

- 
- Để mô tả biến tham chiếu, chúng ta viết thêm phép toán tham chiếu (*reference operator*) trước tên của biến.

`type& alias = name;`

- Trong đó,
 - ♦ *type* là kiểu dữ liệu.
 - ♦ *alias* là tên biến tham chiếu.
 - ♦ *name* là tên định danh của biến mà biến *alias* tham chiếu đến.


Biến con trỏ

- Dùng phép toán & để truy cập đến địa chỉ của một biến, một đối tượng.
- Biến con trỏ là biến được dùng để lưu trữ địa chỉ bộ nhớ.
- Trong C/C++ với DOS, biến con trỏ chiếm 2 byte bộ nhớ.
- Trong môi trường Win32, UNIX, biến con trỏ chiếm 4 byte.

Ví dụ

```
#include <iostream.h>
main()
{
    int n = 19;
    int* p = &n;
    cout << "&n = " << &n << ", p = "
    << p << endl;
    cout << "&p = " << &p << endl;
    return 1;
}
```

**&n = 0xffff4, p = 0xffff4
&p = 0xffff2**

- 
- Để khai báo biến con trỏ chúng ta thêm dấu * phía sau kiểu dữ liệu.


`type* name;`

- Hoặc

`type *name;`

- Hoặc

`type * name;`



```
int n = 19;  
int* p = &n;  
int& r = *p;
```

n, *p, r

19

0xffff4

p

0xffff4

0xffff2

Kiểu dữ liệu mảng và con trỏ

➤ Con trỏ được dùng để quản lý những dữ liệu như mảng – gồm nhiều phần tử có kiểu giống nhau, đặt kề nhau.

➤ Khai báo mảng

```
element_type name[];
```

➤ Hoặc

```
element_type* name;
```

➤ Vì vậy, có thể truy cập đến 1 phần tử của mảng bằng cách viết:

$* (a+i)$ hoặc $a[i]$


$* (* (b+i) + j)$ hoặc $b[i][j]$

Sự khác nhau là mảng phải chỉ định số phần tử trước, còn con trỏ sẽ cấp phát sau

Tham số của hàm

- Có 2 hình thức chuyển tham số cho hàm
 - ◆ Chuyển giá trị (passing by value)
 - ◆ Chuyển tham chiếu (passing by reference)
- Cách thức chuyển tham chiếu
`function_name(type&)`
- Khi đó tham số hình thức là biến tham chiếu của tham số thực.


Tham số thực và hình thức có cùng một giá trị

- 
- Sử dụng con trỏ trong tham số chẳng qua cũng chỉ là việc chuyển tham số giá trị.
 - Nhưng khi đó, giá trị được chuyển là con trỏ - là địa chỉ.



➤ Ví dụ: Hoán vị 2 số

```
void swap( float* a, float* b ){  
    float tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

➤ Điều này cũng có kết quả tương tự

```
void swap( float& a, float& b ){  
    float tmp = a;  
    a = b;  
    b = tmp;  
}
```

Hàm trả về tham chiếu

- Thay vì truy cập đến một biến nào đó, chúng ta sử dụng tên hàm.
 - ◆ Từ đó tên hàm được xem như là tham chiếu đến biến.

➤ Ví dụ:

```
int x;  
int& valuex(){  
    return x;  
}
```



➤ Khi đó có thể viết

`valueX() = 100;`

➤ Để thay cho

`x = 100;`

➤ Xem ví dụ phức tạp hơn như sau

```
const int MAX = 20;
float vector[MAX];
float& v( int i ){
    return vector[i-1];
}
void sort()
{
    float tmp;
    for( int i = 2; i <= MAX ; i++ ){
        tmp = v(i);
        for( int j = i; j > 1 && v(j-1) > tmp; j-- )
            v(j) = v(j-1);
        v(j) = tmp;
    }
}
```

Cấp phát bộ nhớ

➤ Trong ngôn ngữ C sử dụng hàm để cấp phát bộ nhớ. Trong C++ có thể sử dụng phép toán `new`.

➤ Sử dụng

`new type`

➤ Hoặc

`new type[size]`



➤ Ví dụ:

```
float *e;  
e = new float;  
*e = 2.71828;
```

➤ Hoặc

```
float *p;  
p = new float (3.14159);
```

Thu hồi bộ nhớ

➤ Dùng phép toán delete dạng:

`delete name`

➤ Hoặc

`delete []name`

Cấp phát mảng 1 chiều

➤ Tạo mảng các số thực có n phần tử

```
double *a;
```

```
a = new double[n];
```

➤ Thu hồi vùng bộ nhớ đã tạo ra

```
delete []a;
```


Tạo mảng hai chiều $n \times m$ phần tử

➤ Cách thứ nhất:

```
double **a;  
a = new double*[n];  
for ( int i = 0; i < n; i++ )  
    a[i] = new double[m];
```

Thu hồi

```
for ( i = 0; i < n; i++ )  
    delete []a[i];  
delete []a;
```

➤ Cách thứ hai

```
double **a = new double*[n];  
double *tmp = new double[n*m];  
for( int i=0; i<n; i++, tmp+=m )  
    a[i] = tmp;
```

```
Thu hồi  
delete [] *a;  
delete [] a;
```

Con trỏ this

- Con trỏ this dùng để lưu trữ địa chỉ của đối tượng đang sử dụng của lớp.
- Chẳng hạn, với lớp có dữ liệu thành viên là `int size`.
- Trong một hàm nào đó của lớp này có thể truy cập đến `size` bằng cách



➤ Chẳng hạn,

size = MAX;

➤ Hay

this->size = MAX;

➤ Hay

(*this).size = MAX;

Yêu cầu

- Hiểu con trỏ một cách rõ ràng và chính xác nhất.
- Phân biệt biến dạng tham trị và dạng tham chiếu
- Biết cách dùng hàm trả về tham chiếu

Object – Oriented Programming



PGS. TS. Trần Văn Lăng

lang@hcmc.netnam.vn

Chương 5



Phương thức tự thực hiện

Phương thức tự động thực hiện


- Trong C++ có 2 phương thức thuộc loại này:
 - ◆ Phương thức thiết lập (constructor)
 - ◆ Phương thức hủy bỏ (destructor)
- Chương trình mang đúng nghĩa hướng về với đối tượng:
 - ◆ Khi tạo ra đối tượng, một số hành vi sẽ thực thi vào thời điểm đó.


Khi đó,

- Đối tượng không chỉ đơn thuần là dữ liệu có cấu trúc đã được tạo ra.
- Mà còn,
 - ◆ Mang tính hành động: một hoặc một số hành vi nào đó của nó được thi hành.
- Và ngược lại,
 - ◆ Khi đối tượng mất đi, sẽ có một số hành động được thực thi.

Phương thức thiết lập

- Được thực hiện một cách tự động ngay sau khi đối tượng được tạo ra.
- Nhằm thực hiện một số công việc ban đầu như:
 - ◆ Tạo ra vùng bộ nhớ
 - ◆ Sao chép, khởi tạo giá trị ban đầu cho dữ liệu
 - ◆ V.V...

- 
- Lớp trong C++ có thể có hoặc không có phương thức thiết lập
 - Khi không có, một số hành động sau được thực hiện:
 - ◆ Dành bộ nhớ cho các dữ liệu
 - ◆ Khởi tạo giá trị không cho tất cả các byte của dữ liệu

- 
- Trong C++, phương thức thiết lập có tên trùng với tên của lớp, không có kiểu trả về. Chẳng hạn,

```
class STACK{  
    //...  
public:  
    STACK();    //Constructor  
};
```

Ví dụ, tìm các số nguyên tố

```
class PrimeNumber{  
    unsigned int N  
public:  
    PrimeNumber();// See next page  
};
```

Có thể dùng Visual C++ để minh họa ví dụ này, qua đó biết cách tạo ra đối tượng và để đối tượng thi hành


```
PrimeNumber::PrimeNumber(){
    cout << "N = ";      cin >> N;
    unsigned i, j;
    for ( i = 3; i <= N; i++ ){
        for(j=2;j<sqrt(i) && i%j!=0;j++);
        if(j>sqrt(i)) cout << i << ", ";
    }
    cout << endl;
}
```

Ví dụ, sử dụng ngăn xếp

```
class STACK{  
    int top;  
    unsigned int *data;  
    int empty();  
    int full();  
public:  
    STACK();  
    int push( unsigned );  
    int pop( unsigned& );  
};
```


- Cần đổi số nguyên dương sang hệ đếm nhị phân,
- Khi đó, ngoài các phương thức đã có, phương thức thiết lập có thể viết như bên cạnh:

```
STACK::STACK() {  
    unsigned int N; int re;  
    cout << "N = ";  
    cin >> N;  
    top = -1;  
    data = new unsigned[16];  
    do{  
        re = N % 2;  
        if(!push(re)) exit(0);  
    } while(N/=2);  
    while(pop(re)) cout << re;  
    cout << endl;  
}
```


- 
- Một lớp có thể có nhiều phương thức thiết lập.
 - Chúng khác nhau qua danh sách tham số.
 - Đây chính là khả năng định nghĩa chồng lên nhau (overloading) của các hành vi trong lớp.

Chẳng hạn, cũng với lớp STACK

```
class STACK{
    int top;
    unsigned int *data;
    int empty();
    int full();
public;
    STACK();
    STACK(unsigned int); //second constructor
    int push( unsigned );
    int pop( unsigned& );
};
```

- 
- Khi tạo đối tượng, nếu không chỉ định thêm bất kỳ điều gì. Chẳng hạn,

STACK S ;

- Thì phương thức thiết lập chuẩn được gọi thực hiện một cách tự động.

Vậy, thế nào là constructor chuẩn

- Phương thức thiết lập chuẩn là phương thức thiết lập không có tham số.
- Hoặc phương thức thiết lập với tất cả các tham số được gán giá trị đầu. Chẳng hạn,

```
STACK( unsigned int = 1237 );  
VECTOR( int = 2; double = 3.5 );
```

Phương thức thiết lập có thuộc tính truy cập là **public**.

Phương thức hủy bỏ

- Phương thức hủy bỏ (destructor) được thực hiện trước khi đối tượng bị mất đi (trước khi vùng bộ nhớ dành cho đối tượng bị thu hồi).
- Sử dụng mang tính dọn dẹp, hoặc thông báo về sự kết thúc hoạt động.

- 
- Trong C++, phương thức hủy bỏ được viết như sau:

`~ClassName()`

- Một lớp chỉ có 1 phương thức hủy bỏ

Ví dụ

```
class PrimeNumber{
    unsigned int N
public:
    PrimeNumber();
    ~PrimeNumber(){
        cout << "Finished!\n"
    }
};
```

Hoặc

```
class STACK{
    int top;
    unsigned int *data;
    int empty();
    int full();
public;
    STACK();
    ~STACK(){ delete []data; }
    int push( unsigned );
    int pop( unsigned& );
};
```


- Ví dụ tạo kiểu chuỗi ký tự với tên gọi STRING để dùng trong chương trình

```
class STRING{  
    char* data;  
public:  
    STRING( char * );  
    ~STRING();  
    void outStr();  
};
```

Sự phức tạp của lớp này khi thực thi sẽ được khắc phục trong copy constructor

Phương thức thiết lập tạo bản sao

➤ Còn gọi là Copy Constructor.

➤ Mục tiêu

- ◆ Nhằm để tạo ra bản sao của đối tượng, trong đó quản lý chặt chẽ những gì được làm, được sao chép
- ◆ Quản lý bản sao của đối tượng được tạo ra

Đây là phương thức có trong C++

Trường hợp tạo bản sao

- Khi cần tạo bản sao, có thể viết như sau:

EXAMPLE A;

EXAMPLE B = A;

- Khi đó **B** là bản sao của đối tượng **A**, những gì được làm khi sao chép sẽ được quy định trong phương thức thiết lập tạo bản sao của lớp **EXAMPLE**.

- 
- Với cách viết **EXAMPLE B = A**, chẳng qua để dễ sử dụng – đồng nhất việc gán với việc sao chép. Thực chất, câu lệnh này là:

EXAMPLE B (A)

- Cũng cần lưu ý thêm, câu lệnh gán chỉ gán giá trị

B = A;

- Hoàn toàn khác câu lệnh – tạo đối tượng

EXAMPLE B = A;


Cách thức viết copy constructor

- Phương thức thiết lập tạo bản sao là một phương thức như mọi phương thức khác, nên chứa các câu lệnh cần thực hiện.
- Tuy nhiên, do đặc thù là được điều khiển một cách tự động, nên tên gọi và tham số được quy ước:

ClassName (ClassName&)

Ví dụ

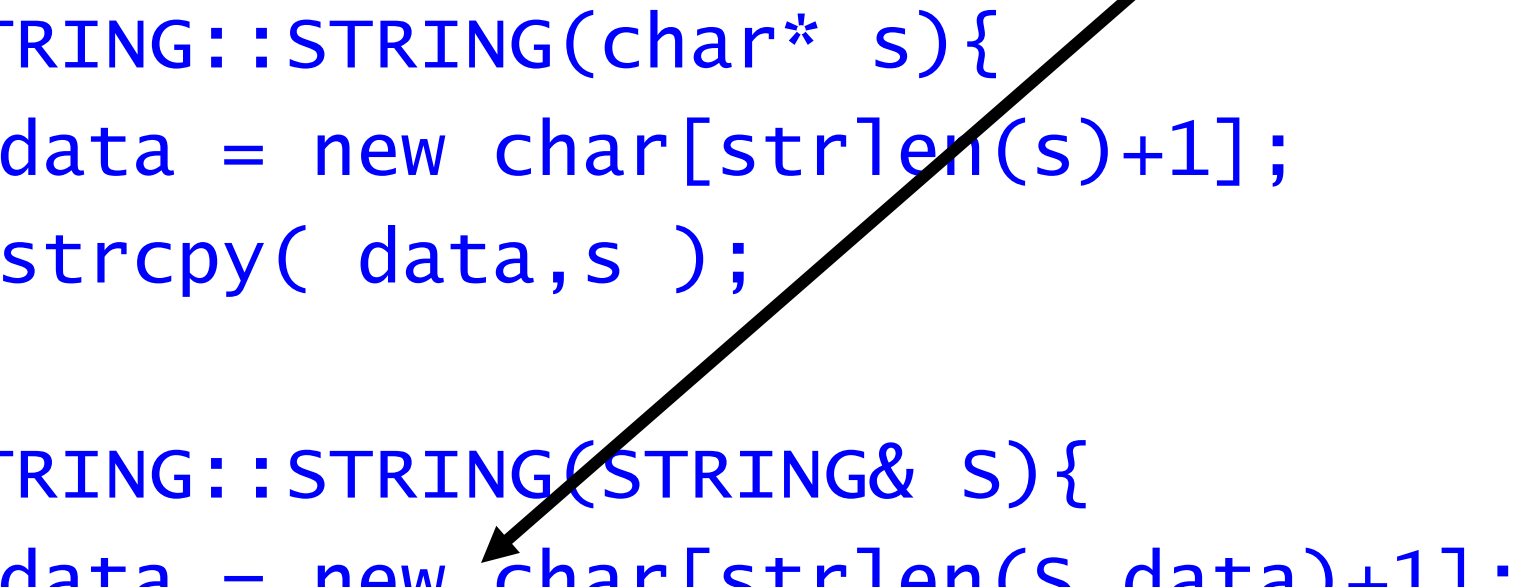
```
class EXAMPLE{
    int n;
    float r;
public:
    EXAMPLE( EXAMPLE& Obj ){
        n = Obj.n;
        cout << "The copy of Obj has just
        created\n";
    }
    // other constructors
}
```

- 
- Trong lớp này, khi một bảo sao của đối tượng (được truyền qua tham số) được tạo ra, chỉ thành phần dữ liệu n của lớp mới có giá trị giống giá trị n của bản gốc
 - Nói cách khác, nó chỉ "bắt chước" thành phần dữ liệu n .


Một ví dụ phức tạp hơn

```
class STRING{  
    char* data;  
public:  
    STRING( char * );  
    STRING( STRING& );  
    ~STRING();  
    void outStr();  
};
```


data = S.data



```
STRING::STRING(char* s){  
    data = new char[strlen(s)+1];  
    strcpy( data,s );  
}  
  
STRING::STRING(STRING& S){  
    data = new char[strlen(S.data)+1];  
    strcpy( data,S.data );  
}
```

- 
- Một lớp luôn luôn có 1 phương thức thiết lập tạo bản sao.
 - Phương thức đó có thể hiện thực hay không hiện thực.
 - Khi không hiện thực, một phương thức tạo bản sao chuẩn sẽ âm thầm tồn tại.

Nguy hiểm trong lập trình là khi mọi thứ diễn ra một cách âm thầm, người lập trình không hay biết – side effect

➤ Phương thức thiết lập tạo bản sao được thi hành khi:

- ◆ Khởi tạo đối tượng bởi đối tượng đã có
- ◆ Tham số thực được truyền cho tham số giá trị của một phương thức nào đó.
- ◆ Phương thức trả đối tượng của lớp trở về thông qua tên gọi (**return Obj**)

Lưu ý

- Vấn đề chỉ nảy sinh phức tạp khi việc cấp phát và thu hồi bộ nhớ được thực thi.
- Bởi khi đó, có thể vô tình thu hồi vùng bộ nhớ đang được sử dụng bởi một bản sao nào đó.


Xét lớp VECTOR như sau

```
class VECTOR{
    int size;
    double *data;
public:
    VECTOR( int = 2 );
    VECTOR( VECTOR& );//copy constructor
    ~VECTOR();
    void setData( double = 0.0 );
    void outData();
}
```

Khởi tạo đối tượng




```
void main(){  
    VECTOR u;  
    u.outData();  
    VECTOR v = u;  
    v.outData();  
}
```

- 
- Nếu không có phương thức thiết lập tạo bản sao, hoặc viết không đúng yêu cầu. Một vùng bộ nhớ bị thu hồi hai lần
 - Bởi thực chất có 2 đối tượng, nhưng trong trường hợp này cả hai đối tượng này có chung một vùng bộ nhớ.

Chúng ta xem xét cụ thể hơn

```
VECTOR::VECTOR( int n ){
    size = n;
    data = new double[size];
    setData();
    cout << "Object at " << data << endl;
}
VECTOR::~~VECTOR(){
    cout << "Memory location << data
        << "has been destroyed\n";
    delete []data;
}
```

```
VECTOR::VECTOR( VECTOR& V ){
    size = V.size;
    data = new double[size];
    setData();
}
void VECTOR::setData( double a ){
    for ( int i = 0; i < size; i++ )
        data[i] = a;
}
void VECTOR::outData(){
    for ( int i = 0; i < size; i++ )
        cout << data[i] << ", ";
    cout << endl;
}
```

Lưu ý

- Với phương thức thiết lập của lớp **VECTOR** như trên, chúng ta có thể viết

VECTOR u = 5 ;

- Trường hợp này đồng nghĩa với


VECTOR u (5) ;

- Nên phương thức thiết lập tạo bản sao không được gọi đến

Tham số giá trị

- Khi truyền tham số thực cho tham số giá trị của một phương thức, thì phương thức thiết lập tạo bản sao sẽ được gọi đến.
- Chẳng hạn, để tính tích vô hướng của hai vector


$$(u, v) = \sum_{i=1}^n u_i v_i$$




```
double VECTOR::scalar(VECTOR v){  
    double t = 0.0;  
    for(int i = 0; i<size; i++ )  
        t += data[i]*v.data[i];  
    return t;  
}
```

➤ Chương trình gọi có thể viết

```
VECTOR u(5), v(5);  
u.setData(2.0);  
v.setData(3.0);  
cout << "(u,v) = "  
      << u.scalar(v) << endl;
```

- 
- Để kiểm tra sự hoạt động của trường hợp này, chúng ta không hiện thực phương thức thiết lập tạo bản sao.
 - Một bản sao của v được tạo ra; khi hàm `scalar()` thực hiện xong, địa chỉ `data` của bản sao này được thu hồi (do có phương thức hủy bỏ)

- 
- Đến lượt kết thúc, một lần nữa địa chỉ `data` của `v` lại bị thu hồi, trong khi đó 2 địa chỉ này lại giống nhau – do không có phương thức thiết lập tạo bản sao.

Để khắc phục tình trạng này, sử dụng tham số dạng tham chiếu:

`scalar (VECTOR& v)`

Hàm trả về đối tượng

- Phương thức thiết lập tạo bản sao sẽ được gọi để tạo ra bản sao khi hàm trả về một đối tượng của lớp.




➤ Chẳng hạn, tổng của 2 vector:

```
VECTOR VECTOR::add(VECTOR v){  
    VECTOR t(size);  
    for(int i=0; i<size; i++ )  
        t.data[i] = data[i] + v.data[i];  
    return t;  
}
```

➤ Chương trình gọi có thể viết

```
void main(){  
    VECTOR u(3), v(3);  
    u.setData(1.0);  
    v.setData(4.0);  
    (u.add(v)).outData();  
}
```

Chúng ta thử bỏ copy constructor trong lớp **VECTOR** này để theo dõi kết quả

- 
- Chúng ta cũng có thể lưu lại kết quả tổng 2 vector bằng cách bổ sung

```
VECTOR t = u.add(v) ;  
t.outData() ;
```

- Kết quả hoàn toàn tương tự

Phép toán gán

➤ Nhưng khi dùng phép toán gán

```
VECTOR t(3);
```

```
t = u.add(v);
```


```
t.outData();
```

➤ Kết quả không như mong đợi

Lý do

- Phép toán gán chuẩn sẽ gán từng byte của đối tượng này cho đối tượng kia, khi đó các biến liên quan đến địa chỉ cũng hoàn toàn giống nhau.

Phương thức thiết lập tạo bản sao sẽ tạo ra một đối tượng mới; còn phép toán gán chỉ làm thay đổi giá trị của đối tượng

- 
- Có thể bổ sung thêm hàm `assign()` để gán giá trị:

```
VECTOR VECTOR::assign(VECTOR v){  
    size = v.size();  
    for(int i = 0; i < size; i++ )  
        data[i] = v.data[i];  
    return *this;  
}
```

Để hoàn thiện

- Xây dựng lớp vector, matrix với đầy đủ một số hàm cần thiết:
- Xem sách tham khảo **Trần Văn Lăng, Lập trình hướng đối tượng sử dụng C++, trang 231, 233**

Yêu cầu

- Hiểu rõ phương thức thiết lập, huỷ bỏ và thiết lập sao chép.
- Xây dựng lớp có các phương thức tự động thực hiện.
- Sử dụng được các lớp theo nghĩa hướng về với đối tượng (tạo đối tượng, thì đối tượng tự giải quyết vấn đề nào đó)

Object – Oriented Programming



Assoc. Prof. Dr. Tran Van Lang

Email: lang@hcmc.netnam.vn

Chương 6



Các cách tạo đối tượng

Tạo bằng cách khai báo biến

➤ Dùng phương thức thiết lập chuẩn

Ví dụ:

VECTOR a ;

➤ Sử dụng phương thức thiết lập có tham số

Ví dụ:

VECTOR a (10) , b (10 , 3.5) ;

- 
- Khi dùng phương thức thiết lập có một tham số, cũng có thể viết

VECTOR a = 10 ;

- Điều này có ý nghĩa

VECTOR a (10) ;



➤ Tạo đối tượng từ đối tượng đã có

Ví dụ:

VECTOR a;

VECTOR b = a;

➤ Tương đương

Ví dụ:

VECTOR a;

VECTOR b(a)

- 
- Trường hợp tạo nhiều đối tượng, sử dụng phương thức thiết lập chuẩn

Ví dụ:

VECTOR a[5] ;

- Sử dụng phương thức thiết lập có một tham số

Ví dụ:

VECTOR a[2]={10,20} ;

- 
- Sử dụng phương thức thiết lập nhiều tham số

Ví dụ:

VECTOR

a[2]={VECTOR(10,3.5),VECTOR(20,1.6)} ;

- Minh họa (xem ví dụ trang 244,245)

Tạo bằng cách cấp phát bộ nhớ

➤ Dùng phương thức thiết lập chuẩn

Ví dụ:

```
VECTOR *pa = new VECTOR;
```

➤ Với phương thức thiết lập có tham số

Ví dụ:

```
VECTOR *pa = new VECTOR(10);
```

```
VECTOR *pb = new VECTOR(10, 1.5);
```


➤ Tạo nhiều đối tượng

Ví dụ:

```
VECTOR *a = new VECTOR[2];
```

```
VECTOR *b = new VECTOR[2] = {10,20};
```

```
VECTOR *c = new VECTOR[2] =  
    {VECTOR(10,3.5), VECTOR(20,1.5)};
```



➤ Trường hợp cần tạo mảng các con trỏ

Ví dụ:

```
VECTOR *a[2] = {new VECTOR(10) ,  
                new VECTOR(20) } ;
```


Ví dụ

```
ELEMENT *x[2]={new ELEMENT(2.4),new  
ELEMENT(1.5)};  
ELEMENT *y[5]  
void main() {  
    for( int i=0; i<2; i++ )  
        x[i]->showValue();  
    for( i=0; i<3; i++ ){  
        y[i] = new ELEMENT;  
        y[i]->inputValue();  
        delete y[i];  
    }  
}
```

Đối tượng là thành phần của lớp


Ví dụ:

```
class MERGE{
    int n1, n2;
    VECTOR u, v;
public:
    MERGE(int N1, int N2):u(N1),v(N2)    {
        n1 = N1;
        n2 = N2;
    }
}
```



➤ Khi đó trình tự thực hiện của các phương thức thiết lập và phương thức hủy bỏ theo quy tắc sau:

- ◆ Phương thức thiết lập của các lớp thành phần được thực hiện trước phương thức thiết lập của lớp,

- 
- ♦ Phương thức hủy bỏ của lớp thành phần thực hiện sau phương thức hủy bỏ của lớp,
 - ♦ Trong các thành phần của lớp, thành phần nào được khai báo trước, phương thức thiết lập sẽ thực hiện trước.
 - ♦ Trong các thành phần của lớp, thành phần nào được khai báo trước, phương thức hủy bỏ sẽ thực hiện sau.

Thao tác trên mảng các đối tượng


- Sử dụng mảng các đối tượng là một cách tiếp cận truyền thống.
- Tuy nhiên,
 - ◆ Có thể sử dụng mảng các đối tượng của lớp ngay chính trong lớp

Ví dụ

```
class SEQUENCE{  
    double data;  
public:  
    SEQUENCE();  
    SEQUENCE( SEQUENCE*, int );  
    void reorder( SEQUENCE*, int );  
    void out( SEQUENCE*, int );  
};
```


Từ đó


```
class MAIN{
    SEQUENCE *u;
public:
    MAIN( int = 2 );
    ~MAIN();
};
MAIN::MAIN( int size ){
    u = new SEQUENCE[size];
    SEQUENCE a(u,size);
    a.reorder( u, size );
    a.out( u, size );
}
MAIN::~~MAIN(){
    delete []u;
}
```




```
void main()  
{  
    MAIN object(5);  
}
```

Lớp có dữ liệu `static`

- Nhằm để các đối tượng của lớp cùng chia sẻ vùng bộ nhớ
- Dữ liệu `static` còn gọi là *thành viên tĩnh* của lớp.
- Trong ngôn ngữ lập trình hướng đối tượng, loại thành viên này thường được gọi là biến lớp (*class variables*)

- 
- Từ đó có thể sử dụng nó mà không cần tạo đối tượng thuộc lớp.
 - Do các đối tượng cùng nhau chia sẻ biến static này, nên nó phải được khai báo như biến toàn cục



```
class PERSON{
    static int count;//class variable
    char name[30];
    char code[5];
public:
    PERSON( char*, char* ):
    ~PERSON();
    void numberPerson();
};
```

Khi sử dụng

```
int PERSON::count = 0;
void main()
{
    PERSON a("Hung", "TH05");
    PERSON b("Hoang", "TH03");
    PERSON c("Lang", "TH04");
    PERSON d("Bao", "TH01");
    PERSON e("Thoai", "TH02");
    a.numberPerson();
}
```

Lớp có phương thức `static`

- Phương thức `static` là phương thức có thể gọi thực hiện ngay cả khi chưa tạo đối tượng thuộc lớp.
- Phương thức `static` là phương thức để cho các đối tượng của lớp cùng chia sẻ.
- Chẳng hạn, như trong ví dụ trên, phương thức `numberPerson()`.

Yêu cầu

- **Nắm rõ hơn về các trường hợp tạo đối tượng.**
- **Hiểu được khái niệm `static`, viết một vài chương trình sử dụng biến và phương thức `static`.**