LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

C++

Bộ môn Hệ Thống Máy Tính & Truyền Thông
Khoa Công Nghệ Thông Tin và Truyền Thông
Đại Học Cần Thơ

Giới Thiệu Tổng Quan

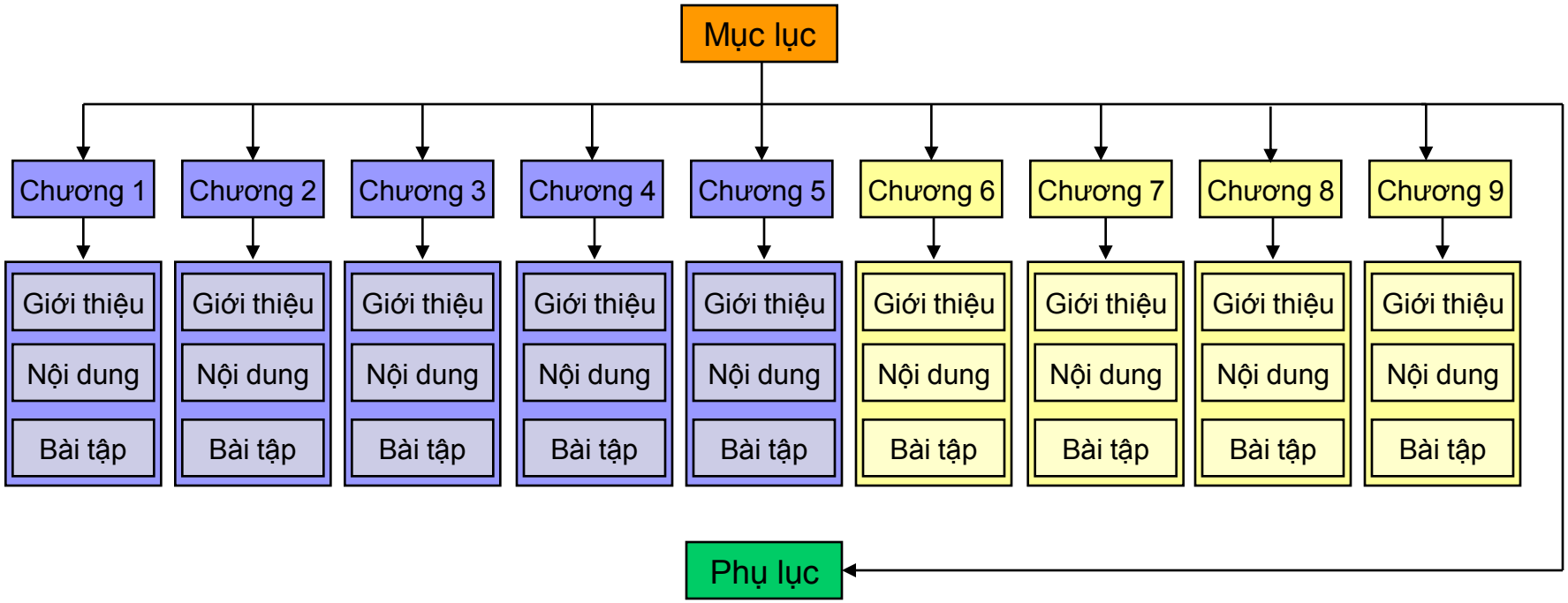
- Tên môn học: Lập Trình Hướng Đối Tượng C++
- Số đơn vị học trình: 3
- Kiến thức tiên quyết: Lập trình cơ bản
- Nội dung chính:
  - Truyền đạt những khái niệm, nguyên lý cơ bản của **Lập Trình Hướng Đối Tượng** (OOP).
  - Minh họa lập trình hướng đối tượng bằng **ngôn ngữ C++.**

Giới Thiệu Tổng Quan

- **Giáo trình chính**:

  "*Lập Trình Hướng Đối Tượng C++*"

  Thạc sĩ. Trương Văn Chí Công



Giới Thiệu Tổng Quan

Danh mục tài liệu tham khảo

1. Ali Bahrami, "Object-oriented Systems Development", McGraw-Hill Companies Inc., 1999.
2. Bruce Eckel, "Thinking in C++", Prentice Hall Inc., 2000.
3. Budd, Timothy, "An Introduction to Object-Oriented Programming", Addison-Wesley, 1997.
4. Robert Lafore, "Object-oriented Programming in C++", SAMS, 2001.
5. Phạm Văn Ất, "C++ và Lập Trình Hướng Đối Tượng", Khoa Học Kỹ Thuật, 2000.
6. Scott Robert Ladd, "C++ Kỹ Thuật và Ứng Dụng", Khoa Học Kỹ Thuật, 1992.
7. Sharam Hekmat, "C++ Programming", Pragmatix Software Pt, 2004.

Phương Pháp Học Tập

- Tìm hiểu theo chủ điểm
- Trình bày nội dung cơ bản
- Tham khảo giáo trình, tài liệu tham khảo
- Tăng cường khả năng tự học
- Thực hành trên máy
- Thảo luận nhóm
- Email, Internet

Nội dung

- Mở đầu
- Biểu thức
- Lệnh
- Hàm
- Mảng - Con trỏ - Tham chiếu
- Lập trình hướng đối tượng
- Lớp
- Tái định nghĩa
- Thừa kế

Mở Đầu

- Mục tiêu
  - Giới thiệu các khái niệm cơ bản của một chương trình C++.
- Nội dung
  - Viết và biên dịch chương trình C++
  - Biến, hằng, chú thích, kiểu dữ liệu
  - Bộ nhớ, nhập xuất

Biểu Thức

- Mục tiêu
  - Giới thiệu các toán tử cho việc soạn thảo các biểu thức.
- Nội dung
  - Toán tử toán học, quan hệ, luận lý, bit, tăng/giảm, khởi tạo, điều kiện, lấy kích thước
  - Độ ưu tiên của các toán tử
  - Chuyển kiểu

Lệnh

- Mục tiêu
  - Cung cấp cú pháp và cách sử dụng các lệnh.
- Nội dung
  - Lệnh đơn, lệnh phức
  - Lệnh rẽ nhánh: if, switch
  - Lệnh lặp: while, do..while, for
  - Lệnh nhảy: continue, break, goto

Hàm

- Mục tiêu
  - Mô tả cách khai báo, định nghĩa, và gọi hàm.
- Nội dung
  - Khai báo hàm, định nghĩa hàm
  - Tham số, đối số, phạm vi
  - Biến tự động, biến thanh ghi, biến nội tuyến
  - Đối số mặc định, đối số hàng lệnh

Mảng, Con Trỏ, Tham Chiếu

- Mục tiêu
  - Giới thiệu các cách sử dụng mảng, con trỏ, và tham chiếu.
- Nội dung
  - Mảng một chiều, nhiều chiều, bộ nhớ tĩnh
  - Con trỏ, tính toán con trỏ, bộ nhớ động
  - Con trỏ hàm, tham chiếu

Lập Trình Hướng Đối Tượng

- Mục tiêu
  - Giới thiệu những khái niệm cơ bản trong lập trình hướng đối tượng.
- Nội dung
  - Trừu tượng hóa, đối tượng, lớp.
  - Thuộc tính, phương thức, thông điệp.
  - Tính bao gói, tính kế thừa, tính đa hình

Lớp

- Mục tiêu
  - Giới thiệu cấu trúc lớp C++ để định nghĩa kiểu dữ liệu mới.
- Nội dung
  - Cấu trúc lớp, hàm xây dựng, hàm hủy
  - Hàm bạn, đối số mặc định, đối số ẩn, toán tử phạm vi, danh sách khởi tạo thành viên
  - Thành viên hằng, thành viên tĩnh, thành viên tham chiếu, thành viên đối tượng

Tái Định Nghĩa

- Mục tiêu
  - Giới thiệu cơ chế tái định nghĩa hàm và tái định nghĩa toán tử trong C++.
- Nội dung
  - Tái định nghĩa hàm, toán tử, chuyển kiểu
  - Tái định nghĩa toán tử <<, >>, [ ], ( ), new, delete, ++, --
  - Khởi tạo ngầm định, gán trị ngầm định

Thừa Kế

- Mục tiêu
  - Khai thác các đặc tính thừa kế trong C++.
- Nội dung
  - Lớp cơ sở, lớp dẫn xuất, thứ bậc lớp
  - Hàm xây dựng, hàm hủy, thành viên được bảo vệ
  - Lớp cơ sở riêng/chung/được bảo vệ, hàm ảo, đa thừa kế, sự mơ hồ, chuyển kiểu
  - Lớp cơ sở ảo, tái định nghĩa toán tử

CHƯƠNG 1:

MỞ ĐẦU
(INTRODUCTION)

Khoa Công Nghệ Thông Tin & Truyền Thông
Đại Học Cần Thơ

Lịch Sử Của C++

- C++ dựa trên ngôn ngữ lập trình C
- C được phát minh trước 1970 bởi Dennis Ritchie
  - Ngôn ngữ cài đặt hệ thống cho hệ điều hành Unix
- C++ được phát minh bởi Bijarne Stroustroup, bắt đầu năm 1979
  - Phiên bản thử nghiệm, phiên bản thương mại
- Các chuẩn ngôn ngữ C++ hiện tại được điều khiển bởi ANSI và ISO

Mở Đầu

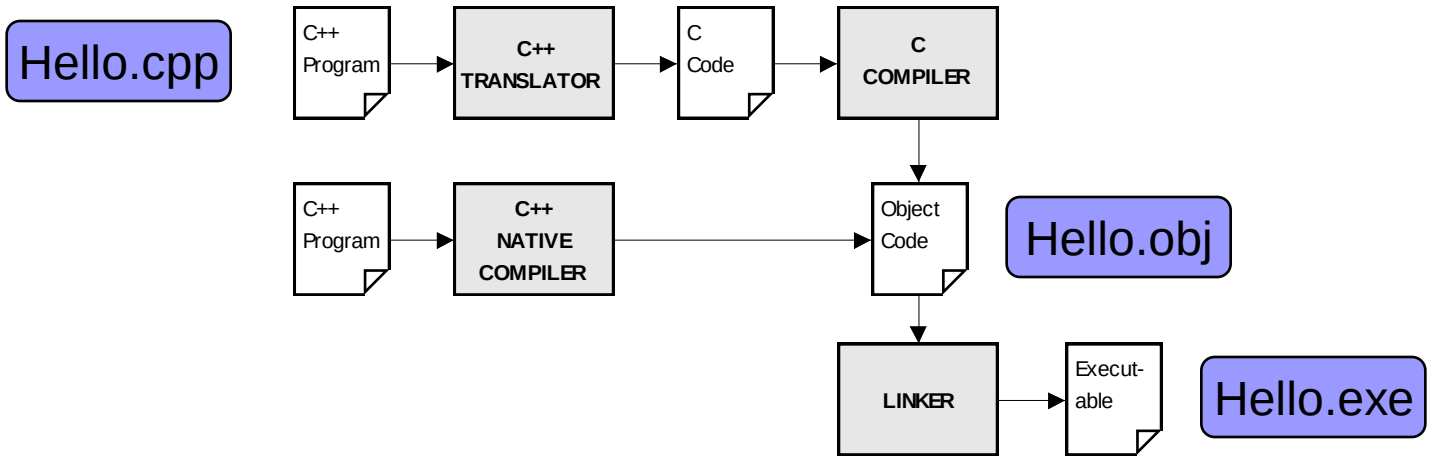
- Mục tiêu
  - Giới thiệu các khái niệm cơ bản của một chương trình C++
- Nội dung
  - Viết và biên dịch chương trình C++
  - Biến, hằng, chú thích, kiểu dữ liệu
  - Bộ nhớ, nhập xuất
  - Cách đặt tên

Chương Trình C++ Đầu Tiên

- Sử dụng bất kỳ trình soạn thảo nào
- Lưu đúng định dạng
- Biên dịch

Hello.cpp

```
#include <iostream.h>
int main (void)
{
   cout << "Hello World\n";
}
```



Biến

- Biến
  - Tên tượng trưng cho một vùng nhớ mà dữ liệu có thể được lưu trữ trên đó hay là được sử dụng lại.
- Thuộc tính của biến
  - Kiểu: được thiết lập khi các biến được định nghĩa
  - Giá trị: có thể được chuyển đổi bằng cách gán một giá trị mới cho biến

Khai Báo Biến

Danh sách 1.2

```
1   #include <iostream.h>

2   int main (void)
3   {
4     int     workDays;
5     float   workHours, payRate,
       weeklyPay;
6     workDays = 5;
7     workHours = 7.5;
8     payRate = 38.55;
9     weeklyPay = workDays  *
       workHours * payRate;
10    cout << "Weekly Pay = "<<
       weeklyPay<< '\n';
11  }
```

Khai báo biến

Khởi tạo biến

Danh sách 1.3

```
1   #include <iostream.h>

2   int main (void)
3   {
4     int     workDays = 5;
5     float   workHours = 7.5;
6     float   payRate = 38.55;
7     float   weeklyPay = workDays
       * workHours * payRate;
8   cout << "Weekly Pay = ";
9     cout << weeklyPay;
10    cout << '\n';
11  }
```

Khai báo và khởi tạo biến

Xuất Nhập Đơn Giản

Danh sách 1.4

```
1  #include <iostream.h>

2  int main (void)
3  {
4     int     workDays = 5;
5     float   workHours = 7.5;
6     float    payRate, weeklyPay;
7     cout << "What is the hourly
       pay rate? ";
8     cin >> payRate;
9     weeklyPay = workDays *
       workHours * payRate;
10    cout << "Weekly Pay = ";
11    cout << weeklyPay;
12    cout << '\n';
13 }
```

Danh sách 1.5

```
1  #include <iostream.h>

2  int main (void)
3  {
4     int     workDays = 5;
5     float   workHours, payRate,
       weeklyPay;
6     cout << "What are the work hours
       and the hourly pay rate? ";
7     cin >> workHours >> payRate;
8     weeklyPay = workDays * workHours
       * payRate;
9     cout << "Weekly Pay = " <<
       weeklyPay << '\n';
10 }
```

Chú Thích

Chú thích nhiều hàng

Danh sách 1.6

```
1  #include <iostream.h>
2  /* Chuong trinh nay tinh toan tong so tien phai tra hang tuan
3     cho mot cong nhan dua tren tong so gio lam viec va so tien
4     phai tra moi gio. */
5  int main (void)
6  {
7     int     workDays = 5;  // so ngay lam viec trong tuan
8     floatworkHours = 7.5;  // so gio lam viec trong ngay
9     floatpayRate = 33.50;  // so tien phai tra moi gio
10    floatweeklyPay;        // tong so tien phai tra moi tuan
11    weeklyPay = workDays * workHours * payRate;
12    cout << "Weekly Pay = " << weeklyPay << '\n';
13 }
```

Chú thích một hàng

Số Nguyên & Số Thực
Ký Tự & Chuỗi

- Biến số nguyên có thể được định nghĩa là short, int, hay long.
- Biến số thực có thể được định nghĩa là kiểu float hay double.
- Biến ký tự được định nghĩa là kiểu char.
- Biến chuỗi được định nghĩa kiểu char* (con trỏ ký tự).

Tên

- Tên
  - còn được gọi là định danh
  - được sử dụng để tham khảo
  - tên biến, tên hàm, tên kiểu, và tên macro
  - phải được đặt theo luật
  - không giới hạn số ký tự
  - không được đặt trùng từ khóa

CHƯƠNG 2:

BIỂU THỨC (EXPRESSION)

Khoa Công Nghệ Thông Tin & Truyền Thông
Đại Học Cần Thơ

Biểu Thức

- Mục tiêu
  - Giới thiệu các toán tử cho việc soạn thảo các biểu thức
- Nội dung
  - Toán tử toán học, quan hệ, luận lý, bit, tăng/giảm, khởi tạo, điều kiện, lấy kích thước
  - Độ ưu tiên của các toán tử
  - Chuyển kiểu

Khái Niệm Cơ Bản

- Một biểu thức là bất kỳ sự tính toán nào mà cho ra một giá trị.
- Một biểu thức ước lượng một giá trị nào đó.

Toán Tử Toán Học & Luận Lý

Toán tử	Tên	Ví dụ
+	Cộng	`12 + 4.9  // cho 16.9`
-	Trừ	`3.98 - 4  // cho -0.02`
*	Nhân	`2 * 3.4  // cho 6.8`
/	Chia	`9 / 2.0  // cho 4.5`
%	Lấy phần dư	`13 % 3  // cho 1`

Toán tử	Tên	Ví dụ
==	So sánh bằng	`5 == 5  // cho 1`
!=	So sánh không bằng	`5 != 5  // cho 0`
<	So sánh nhỏ hơn	`5 < 5.5  // cho 1`
<=	So sánh nhỏ hơn hoặc bằng	`5 <= 5  // cho 1`
>	So sánh lớn hơn	`5 > 5.5  // cho 0`
>=	So sánh lớn hơn hoặc bằng	`6.3 >= 5  // cho 1`

Toán Tử Luận Lý & Trên Bit

Toán tử	Tên	Ví dụ
!	Phủ định luận lý	!(5 == 5) // được 0
&&	Và luận lý	5 < 6 && 6 < 6 // được 0
||	Hoặc luận lý	5 < 6 || 6 < 5 // được 1

0: SAI (false)

Khác 0: ĐÚNG (true)

Toán tử	Tên	Ví dụ
~	Phủ Định Bit	~'\011' // được '\366'
&	Và bit	'\011' & '\027' // được '\001'
|	Hoặc bit	'\011' | '\027' // được '\037'
^	Hoặc exclusive bit	'\011' ^ '\027' // được '\036'
<<	Dịch trái bit	'\011' << 2 // được '\044'
>>	Dịch phải bit	'\011' >> 2 // được '\002'

Toán Tử Tăng/Giảm & Khởi Tạo

Toán Tử	Tên	Ví dụ
++	Tăng một (tiền tố)	++k + 10 // được 16
++	Tăng một (hậu tố)	k++ + 10 // được 15
--	Giảm một (tiền tố)	--k + 10 // được 14
--	Giảm một (hậu tố)	k-- + 10 // được 15

Toán Tử	Ví dụ	Tương đương với
=	`n = 25`	
+=	`n += 25`	`n = n + 25`
-=	`n -= 25`	`n = n - 25`
*=	`n *= 25`	`n = n * 25`
/=	`n /= 25`	`n = n / 25`
%=	`n %= 25`	`n = n % 25`
<<=	`n <<= 4`	`n = n << 4`
>>=	`n >>= 4`	`n = n >> 4`

Toán Tử Điều Kiện, Phẩy & Lấy Kích Thước

Toán tử điều kiện

```
min = (m < n ? m++ : n++);
```

Toán tử phẩy

```
min = (m < n ? mCount++, m : nCount++, n);
```

Toán tử lấy kích thước

```
cout << "float  size = " << sizeof(float) << " bytes\n";
```

Độ Ưu Tiên Của Các Toán Tử

Mức	Toán tử						Loại	Thứ tự
Cao nhất	::						Một ngôi	Cả hai
	()	[]	->	.			Hai ngôi	Trái tới phải
	+ -	++ --	! ~	* &	new delete	sizeof ()	Một ngôi	Phải tới trái
	->*	.*					Hai ngôi	Trái tới phải
	*	/	%				Hai ngôi	Trái tới phải
	+	-					Hai ngôi	Trái tới phải
	<<	>>					Hai ngôi	Trái tới phải
	<	<=	>	>=			Hai ngôi	Trái tới phải
	==	!=					Hai ngôi	Trái tới phải
	&						Hai ngôi	Trái tới phải
	^						Hai ngôi	Trái tới phải
	|						Hai ngôi	Trái tới phải
	&&						Hai ngôi	Trái tới phải
	||						Hai ngôi	Trái tới phải
	? :						Ba ngôi	Trái tới phải
	=	+= -=	*= /=	^= %=	&= |=	<<= >>=	Hai ngôi	Phải tới trái
Thấp nhất	,						Hai ngôi	Trái tới phải

CHƯƠNG 3:

LỆNH (INSTRUCTION)

Khoa Công Nghệ Thông Tin & Truyền Thông
Đại Học Cần Thơ

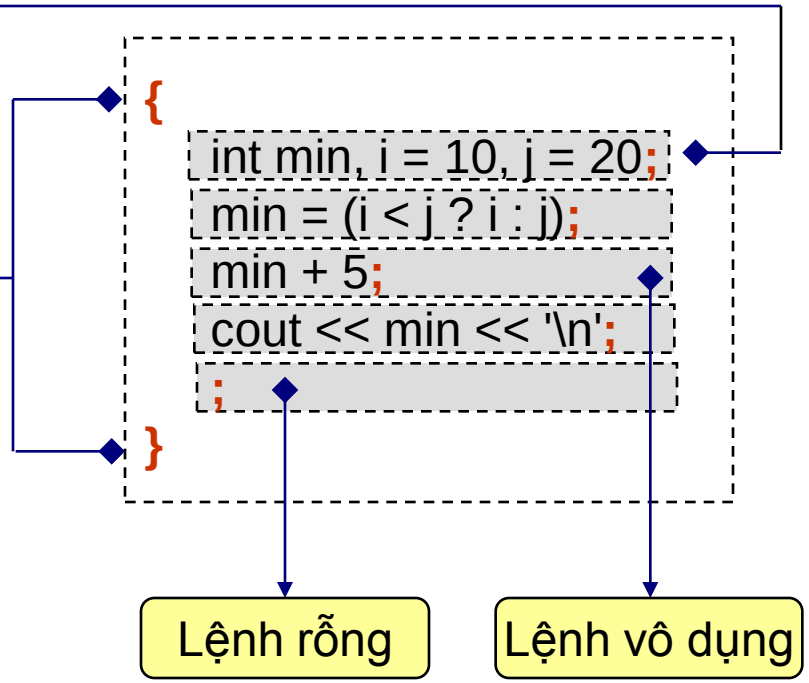
Lệnh

- Mục tiêu
  - Cung cấp cú pháp và cách sử dụng các lệnh
- Nội dung
  - Lệnh đơn, lệnh phức
  - Lệnh khai báo
  - Lệnh gán
  - Lệnh rẽ nhánh: if, switch
  - Lệnh lặp: while, do..while, for
  - Lệnh nhảy: continue, break, goto

Lệnh Đơn & Lệnh Phức

- **Lệnh đơn** là một sự tính toán được kết thúc bằng dấu *chấm phẩy*.
- Nhiều lệnh đơn có thể kết nối lại thành một **lệnh phức** bằng cách rào chúng bên trong các dấu *ngoặc xoắn*.

Ví dụ:



Lệnh Rẽ Nhánh

- Lệnh if và if-else

```
if (biểu thức)
    lệnh;
```

```
if (biểu thức)
    lệnh 1;
else
    lệnh 2;
```

- Lệnh switch

```
switch (biểu thức) {
    case hằng 1:
            các lệnh;  break;
    ...
    case hằng n:
            các lệnh;  break;
    default:
            các lệnh;
}
```

Khi nào chúng ta nên sử dụng if-else và khi nào chúng ta nên sử dụng switch?

Lệnh Lặp

- Lệnh while; do-while

```
while (biểu thức)
    lệnh;
```

```
do
    lệnh;
while (biểu thức);
```

- Lệnh for

```
for (biểu thức1; biểu thức2; biểu thức3)
    lệnh;
```

khởi tạo

điều kiện dừng

điều khiển lặp

Sử dụng từng kiểu lệnh lặp để in ra các số từ 0 đến 9 ?

Lệnh Nhảy

- Lệnh continue

  dừng lần lặp hiện tại của một vòng lặp và nhảy tới lần lặp kế tiếp

- Lệnh goto

  nhảy trực tiếp đến nhãn được chỉ định.

- Lệnh break

  nhảy ra bên ngoài những lệnh lặp hoặc switch và kết thúc chúng.

- Lệnh return

  cho phép một hàm trả về một giá trị cho thành phần gọi nó.

CHƯƠNG 4:

HÀM (FUNCTION)

Khoa Công Nghệ Thông Tin & Truyền Thông
Đại Học Cần Thơ

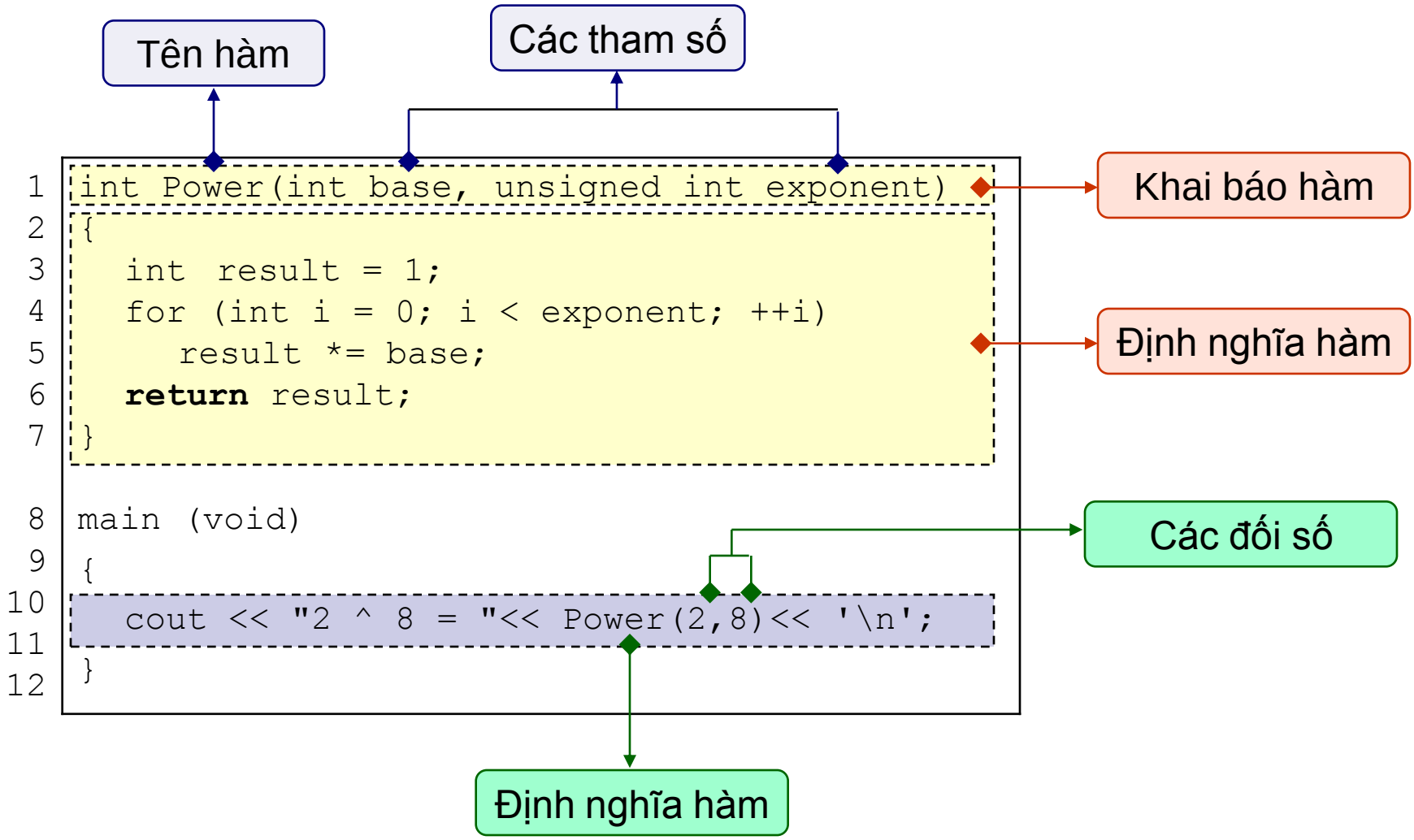
Hàm

- Mục tiêu
  - Mô tả cách khai báo, định nghĩa, và gọi hàm
- Nội dung
  - Khai báo hàm, định nghĩa hàm
  - Tham số, đối số, phạm vi
  - Biến tự động, biến thanh ghi, biến nội tuyến
  - Đối số mặc định, đối số hàng lệnh

Hàm Là Gì?

- Một phương thức để đóng gói quá trình tính toán → dễ dàng sử dụng
- Định nghĩa hàm
  - Khai báo hàm
    - Tên hàm
    - Các tham số của hàm
    - Kiểu trả về của hàm
  - Định nghĩa hàm
    - Các lệnh
- Sử dụng hàm
  - Gọi hàm
    - Tên hàm
    - ( )
    - Các đối số
  - Nhận kết quả trả về của hàm

Ví Dụ Về Hàm


Tên hàm
Các tham số
1 int Power(int base, unsigned int exponent)
2 {
3 int result = 1;
4 for (int i = 0; i < exponent; ++i)
5 result *= base;
6 return result;
7 }
8 main (void)
9 {
10 cout << "2 ^ 8 = "<< Power(2,8)<< '\n';
11 }
12
Khai báo hàm
Định nghĩa hàm
Các đối số
Định nghĩa hàm

Phạm Vi Toàn Cục – Cục Bộ

- Phạm vi toàn cục
  - Được định nghĩa ở phạm vi chương trình
- Phạm vi cục bộ
  - Được định nghĩa ở phạm vi khối hay hàm
- Toán tử phạm vi
  - ::

```
int xyz = 1;              // xyz là toàn cục
void Foo (int xyz)        // xyz là cục bộ cho thân của Foo
{
    if (xyz > 0) {
        double xyz =2;    // xyz là cục bộ cho khối này
        cout <<xyz;
    }
    else {
        cout<< ::xyz;
    }
}
```

Biến Tự Động – Thanh Ghi

- Biến tự động
  - được xác định hoàn toàn tự động
  - Ví dụ: int auto sum;
- Biến thanh ghi
  - được lưu trữ trong thanh ghi để tăng hiệu xuất của chương trình
  - Ví dụ:

```
for (register int i = 0; i < n; ++i)
    sum += i;
```

Hàm Nội Tuyến – Đệ Qui

- Sử dụng hàm
  - Thuận lợi: chương trình dễ đọc, tăng khả năng sử dụng lại, tránh các hiệu ứng phụ
  - Bất lợi: tốn chi phí cho việc gọi hàm
  - → Giải pháp: định nghĩa hàm là nội tuyến (inline)

```
inline int Abs (int n)
{
    return   n > 0 ? n : -n;
}
```

- Đệ qui
  - Một hàm gọi chính nó

```
int Factorial (unsigned int n)
{
    return   n == 0 ? 1 : n * Factorial(n-1);
}
```

Đối Số Mặc Định - Hàng Lệnh

- Đối số mặc định
  - bỏ bớt đi gánh nặng phải chỉ định các giá trị của đối số cho tất cả các tham số hàm
- Đối số hàng lệnh
  - Có thể nhận không hay nhiều đối số từ hàng lệnh
    - được tạo ra sẵn cho một chương trình C++ thông qua hàm main
    - Ví dụ: int main (int argc, const char* argv[]);

CHƯƠNG 5:

MẢNG - CON TRỎ - THAM CHIẾU

Khoa Công Nghệ Thông Tin & Truyền Thông
Đại Học Cần Thơ

Mảng, Con Trỏ, Tham Chiếu

- Mục tiêu
  - Giới thiệu các cách sử dụng mảng, con trỏ, và tham chiếu
- Nội dung
  - Mảng một chiều, nhiều chiều, bộ nhớ tĩnh
  - Con trỏ, tính toán con trỏ, bộ nhớ động
  - Con trỏ hàm, tham chiếu

Mảng

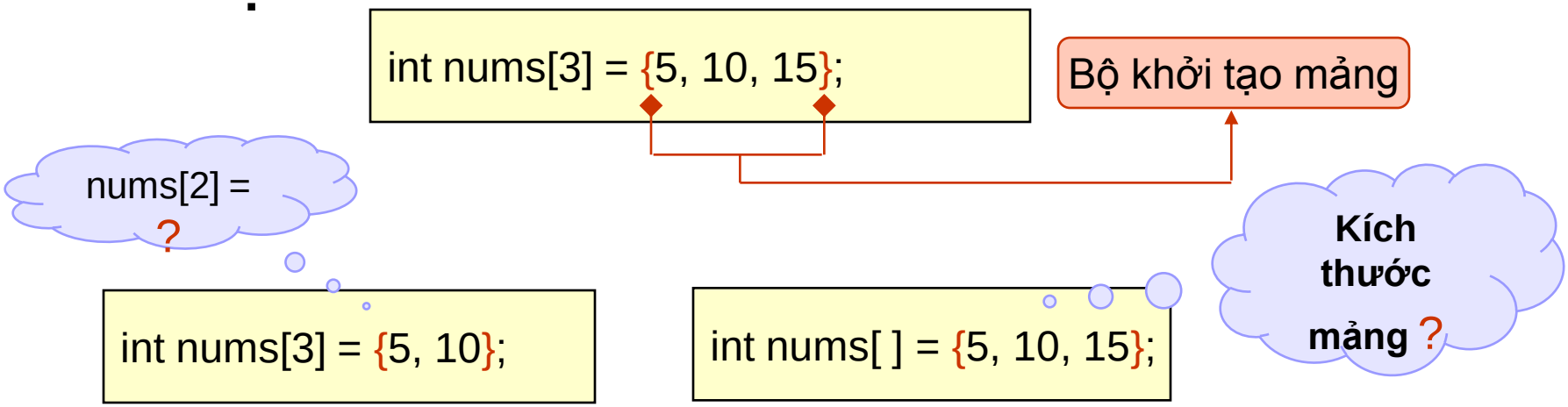
- Mảng (array)
  - Gồm một tập các đối tượng cùng kiểu và được sắp xếp liên tiếp trong bộ nhớ
  - Mỗi **phần tử mảng** được xác định bởi một **chỉ số** biểu thị vị trí của phần tử trong mảng
    - Phần tử đầu tiên của mảng luôn có chỉ số 0
  - Số lượng phần tử trong mảng được gọi là **kích thước** của mảng (cố định; xác định trước)
  - Gồm mảng một chiều và mảng đa chiều

Biến Mảng

- Được định nghĩa bằng cách đặc tả kích thước mảng và kiểu các phần tử của nó
  - Ví dụ: int heights[10];
- Truy xuất 1 phần tử qua chỉ số mảng
  - Ví dụ: heights[0]= 210; cout<< heights[3];
- Truy xuất phần tử không tồn tại → lỗi vượt ngoài biên
  - Ví dụ: cout<<heights[-1]; cout<<heights[10];

Bộ Khởi Tạo Mảng

- Mỗi mảng có một bộ khởi tạo mảng
- Ví dụ



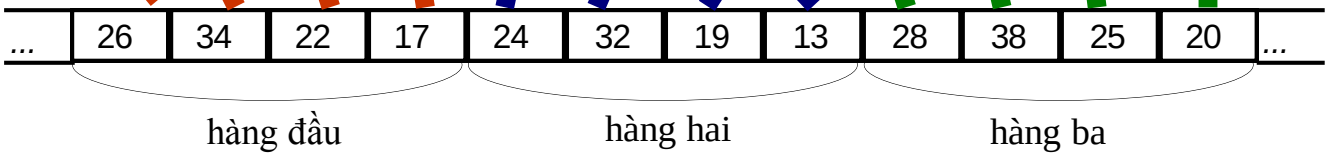
- Chuỗi là một mảng ký tự
- Ví dụ: so sánh sự khác nhau của

char str[] = "HELLO"; và char str[] = {'H', 'E', 'L', 'L', 'O'};

Mảng Đa Chiều

	Mùa xuân	Mùa hè	Mùa thu	Mùa đông
Sydney	26	34	22	17
Melbourne	24	32	19	13
Brisbane	28	38	25	20

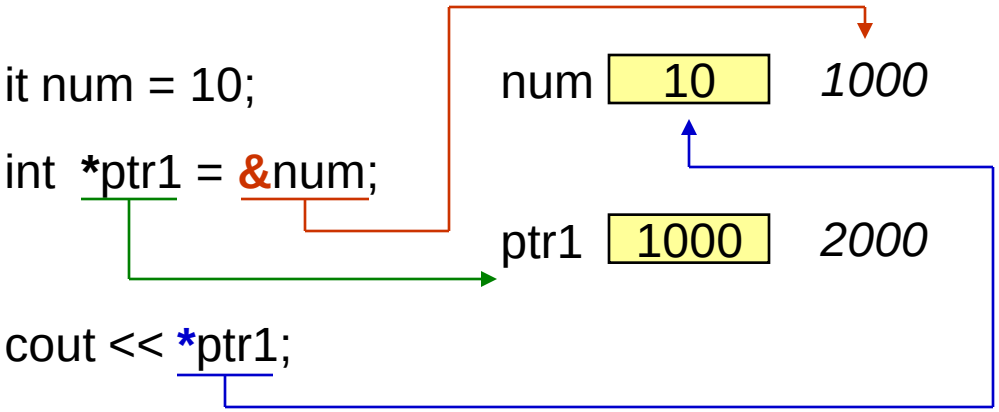
```
int seasonTemp[3][4];
```



Cách tổ chức trong bộ nhớ

Con Trỏ

- Con trỏ đơn giản chỉ là *địa chỉ* của một vị trí bộ nhớ và cung cấp cách gián tiếp để truy xuất dữ liệu trong bộ nhớ
- Ví dụ



Bộ Nhớ Động - Tĩnh

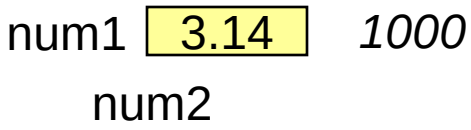
- Bộ nhớ động (heap)
  - Vùng nhớ được cấp phát động trong thời gian thực thi
- Bộ nhớ tính (stack)
  - Vùng nhớ được sử dụng để lưu trữ các biến toàn cục và lời gọi hàm
- Hai toán tử được sử dụng
  - **new**: cấp phát
  - **delete**: thu hồi

```
void Foo (void)
{
    int     *ptr = new int;
    char  *str = new char[10];
    //...
   delete  ptr;
   delete  [ ]str;
}
```

Tham Chiếu

- Một tham chiếu (reference) là một biệt hiệu (alias) cho một đối tượng.
- Ví dụ

```
double  num1 = 3.14;
double  &num2 = num1;
```



- Ghi chú
  - Một tham chiếu phải luôn được khởi tạo khi nó được định nghĩa
  - Có thể khởi tạo tham chiếu tới một hằng

Truyền Bằng Trị - Con Trỏ - Tham Chiếu

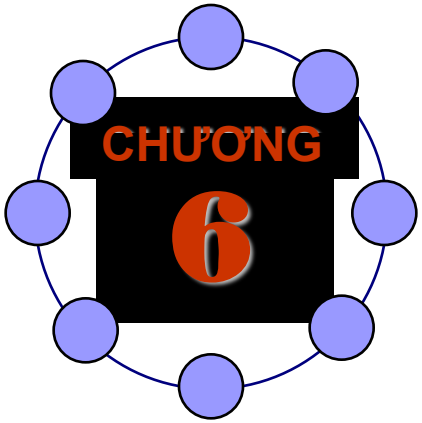
```
   // Truyền bằng trị (đối tượng)
1  void Swap1 (int x, int y)
2  {
3     int temp = x;
4     x = y;
5     y = temp;
6  }
   // Truyền bằng địa chỉ (con trỏ)
7  void Swap2 (int *x, int *y)
8  {
9     int temp = *x;
10    *x = *y;
11    *y = temp;
   }
12 // Truyền bằng tham chiếu
13 void Swap3 (int &x, int &y)
14 {
15    int temp = x;
16    x = y;
17    y = temp;
18 }
```

```
int main (void)
{
   int i = 10, j = 20;
   Swap1(i, j);      cout << i << ", " << j << '\n';
   Swap2(&i, &j); cout << i << ", " << j << '\n';
   Swap3(i, j);      cout << i << ", " << j << '\n';
}
```

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

C++

Đại Học Cần Thơ
Khoa Công Nghệ Thông Tin & Truyền Thông
Bộ môn Hệ thống máy tính & Truyền Thông



Lập Trình Hướng Đối Tượng (Object-oriented Programming)

- Mục tiêu
  - Giới thiệu những khái niệm cơ bản trong lập trình hướng đối tượng
- Nội dung
  - Trừu tượng hóa, đối tượng, lớp
  - Thuộc tính và phương thức
  - Thông điệp và truyền thông điệp
  - Tính bao gói, tính kế thừa, tính đa hình

Khái Niệm

- **Lập trình hướng đối tượng** (OOP- Object-Oriented Programming)
  - một cách tư duy mới, tiếp cận hướng đối tượng để giải quyết vấn đề bằng máy tính.
  - một phương pháp thiết kế và phát triển phần mềm dựa trên kiến trúc lớp và đối tượng.
- Quá trình tiến hóa của OOP
  1. Lập trình tuyến tính
  2. Lập trình có cấu trúc
  3. Sự trừu tượng hóa dữ liệu
  4. Lập trình hướng đối tượng

Tại Sao Tiếp Cận Hướng Đối Tượng?

- Loại bỏ những thiếu sót của tiếp cận theo thủ tục
- Trong OOP
  - Dữ liệu được xem như một phần tử chính yếu và được bảo vệ
  - Hàm gắn kết với dữ liệu, thao tác trên dữ liệu
  - Phân tách bài toán thành nhiều thực thể (đối tượng) → xây dựng dữ liệu + hàm cho các đối tượng này.
- Tăng cường khả năng sử dụng lại

Đặc Điểm Quan Trọng

- Nhấn mạnh trên dữ liệu hơn là thủ tục
- Các chương trình được chia thành các đối tượng
- Dữ liệu được che giấu và không thể được truy xuất từ các hàm bên ngoài
- Các đối tượng có thể giao tiếp với nhau thông qua các hàm
- Dữ liệu hay các hàm mới có thể được thêm vào khi cần
- Theo tiếp cận từ dưới lên

Thuận Lợi

- So với các tiếp cận cổ điển thì OOP có những thuận lợi sau:
  - OOP cung cấp một cấu trúc module rõ ràng
    - Giao diện được định nghĩa tốt
    - Những chi tiết cài đặt được ẩn
  - OOP giúp lập trình viên duy trì mã và sửa đổi mã tồn tại dễ dàng (các đối tượng được tạo ra với những khác nhau nhỏ so với những đối tượng tồn tại).
  - OOP cung cấp một framework tốt với các thư viện mã mà các thành phần có thể được chọn và sửa đổi bởi lập trình viên.

Trừu Tượng Hóa (Abstraction)

- Trừu tượng hóa
  - Phân biệt **cần thiết** với **chi tiết**
    - Giao diện – Cài đặt
    - Cái gì – Thế nào
    - Phân tích – Thiết kế
- Các kỹ thuật trừu tượng
  - Đóng gói (encapsulation)
  - Ẩn thông tin (information hiding)
  - Thừa kế (inheritance)
  - Đa hình (polymorphism)

Đối Tượng (Object)

- **Đối tượng** là chìa khóa để hiểu được kỹ thuật hướng đối tượng
- Trong hệ thống hướng đối tượng, mọi thứ đều là đối tượng

Viết một chương trình hướng đối tượng nghĩa là đang xây dựng một mô hình của một vài bộ phận trong thế giới thực

Đối Tượng Thế Giới Thực (Real Object)

- Một **đối tượng thế giới thực** là một thực thể cụ thể mà thông thường bạn có thể *sờ*, *nhìn thấy* hay *cảm nhận* được.

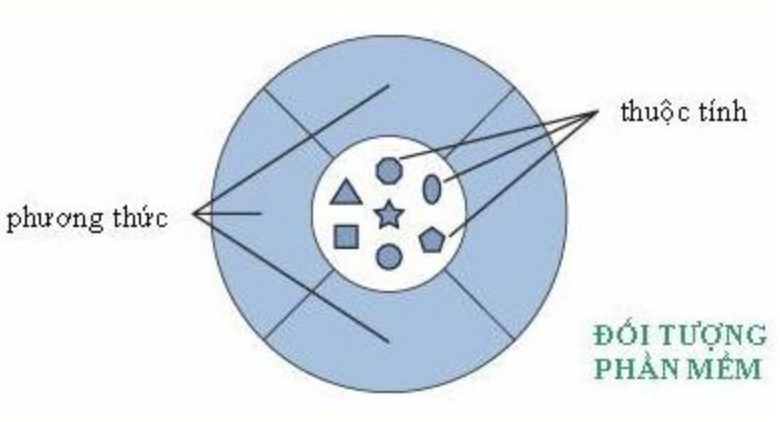
- Tất cả có trạng thái (state) và hành động (behaviour)

	Trạng thái	Hành động	
Con chó	Tên Màu Giống Vui sướng	Sủa Vẫy tai Chạy Ăn	
Xe đạp	Bánh răng Bàn đạp Dây xích Bánh xe	Tăng tốc Giảm tốc Chuyển bánh răng ...	

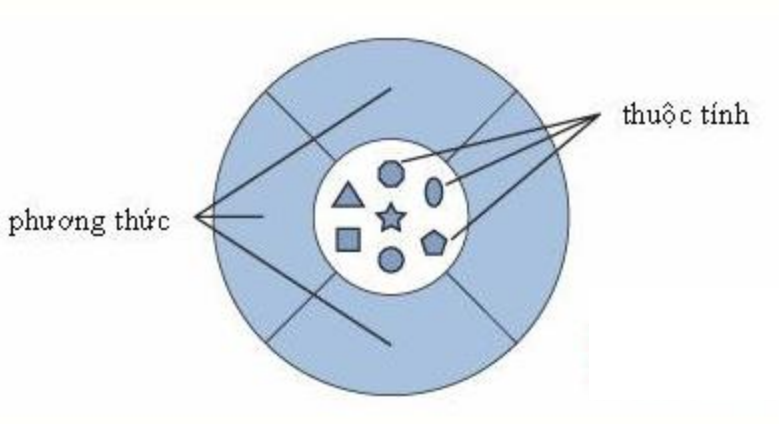
Đối Tượng Phần Mềm (Software Object)

- Các **đối tượng phần mềm** có thể được dùng để *biểu diễn* các đối tượng thế giới thực.

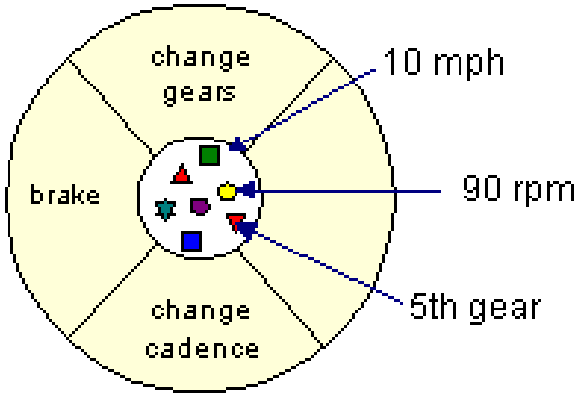
- Cũng có trạng thái và hành động
  - Trạng thái: thuộc tính (attribute; property)
  - Hành động: phương thức (method)



Đối Tượng



Đối tượng phần mềm



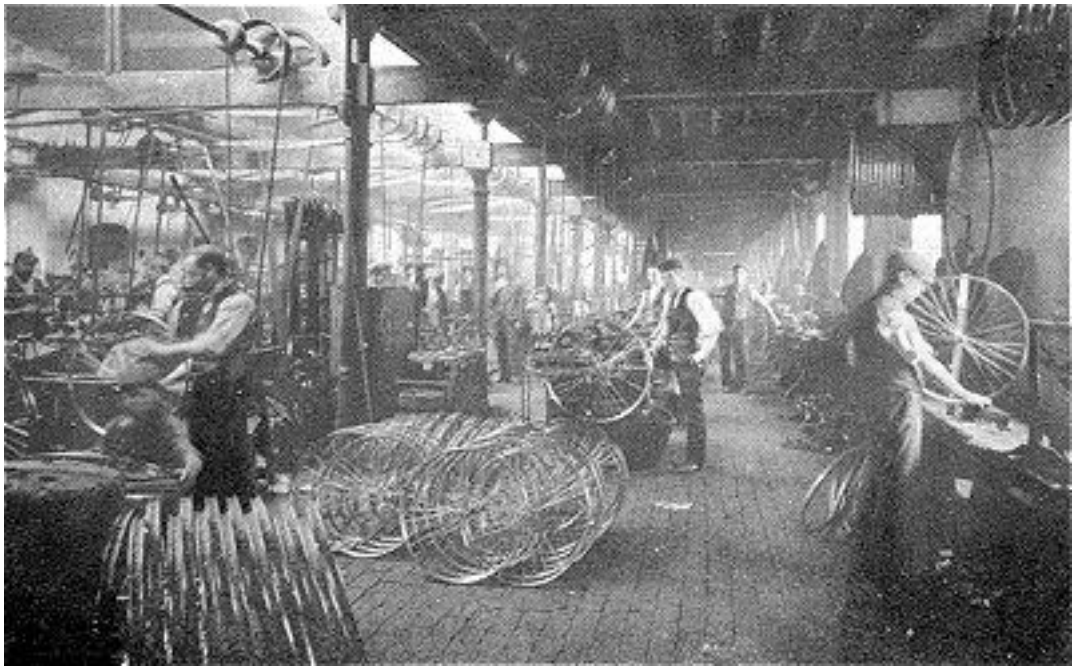
Đối tượng phần mềm **Xe Đạp**

Đối tượng (object) là một thực thể phần mềm bao bọc các **thuộc tính** và các **phương thức** liên quan.

Thuộc tính được xác định bởi giá trị cụ thể gọi là **thuộc tính thể hiện**. Một đối tượng cụ thể được gọi là một **thể hiện**.

Lớp (Class)

- Trong thế giới thực có nhiều đối tượng cùng loại.
- Chương trình hướng đối tượng có nhiều đối tượng cùng loại chia sẻ những đặc điểm chung.
- Ví dụ

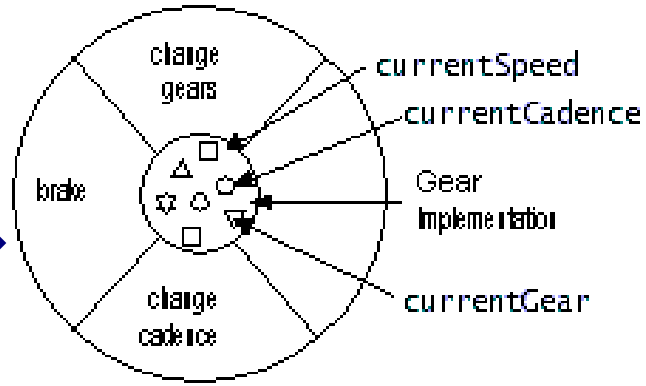


Lớp Là Gì?

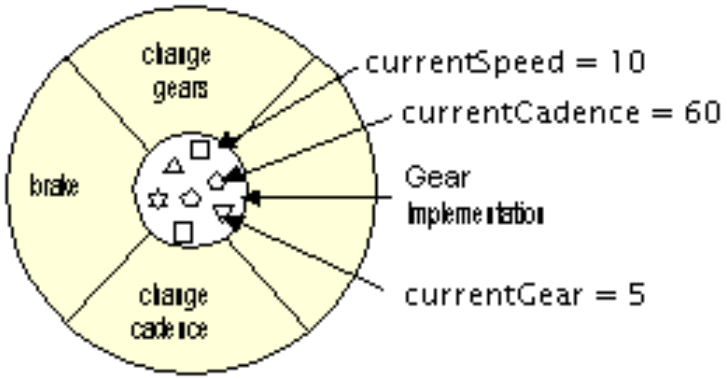
- Một **lớp** là một thiết kế (blueprint) hay mẫu (prototype) cho các đối tượng cùng kiểu
  - Ví dụ: lớp XeDap là một thiết kế chung cho nhiều đối tượng xe đạp được tạo ra
- Lớp định nghĩa các thuộc tính và các phương thức chung cho tất cả các đối tượng của cùng một loại nào đó
- Một đối tượng là một **thể hiện** cụ thể của một lớp.
  - Ví dụ: mỗi đối tượng xe đạp là một thể hiện của lớp XeDap
- Mỗi thể hiện có thể có những thuộc tính thể hiện khác nhau
  - Ví dụ: một xe đạp có thể đang ở bánh răng thứ 5 trong khi một xe khác có thể là đang ở bánh răng thứ 3.

Ví Dụ Lớp Xe Đạp

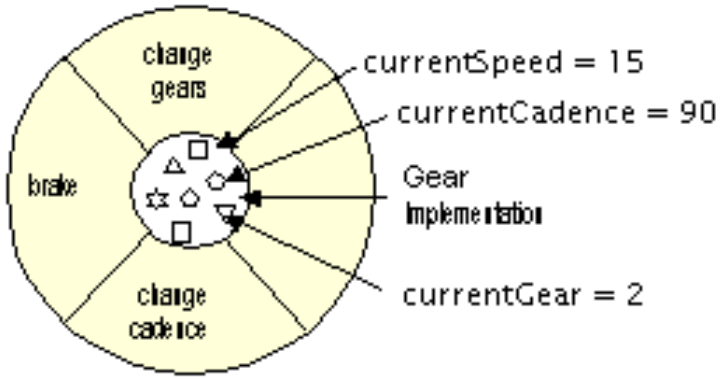
Khai báo cho lớp XeDap



Đối tượng của lớp XeDap

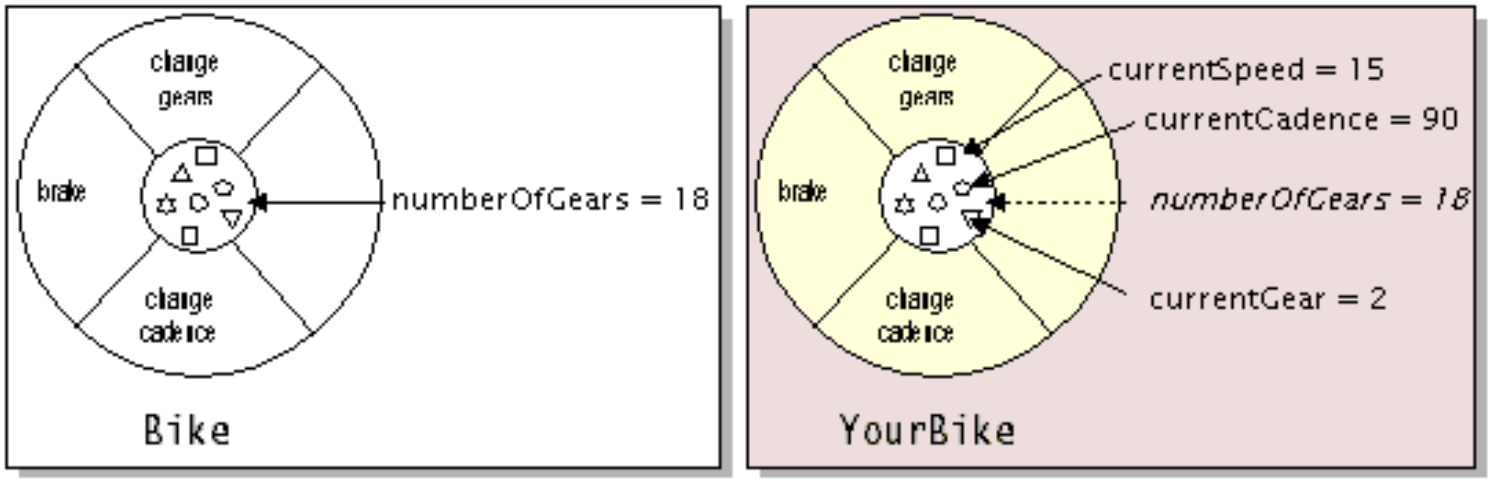


MyBike



YourBike

Thuộc Tính Lớp & Phương Thức Lớp



Class

Instance of a Class

Thuộc Tính Lớp & Phương Thức Lớp

- **Thuộc tính lớp** (class attribute) là một hạng mục dữ liệu liên kết với một lớp cụ thể mà không liên kết với các thể hiện của lớp. Nó được định nghĩa bên trong định nghĩa lớp và được chia sẻ bởi tất cả các thể hiện của lớp.
- **Phương thức lớp** (class method) là một phương thức được triệu gọi mà không tham khảo tới bất kỳ một đối tượng nào. Tất cả các phương thức lớp ảnh hưởng đến toàn bộ lớp chứ không ảnh hưởng đến một lớp riêng rẽ nào.

Thuộc Tính & Phương Thức

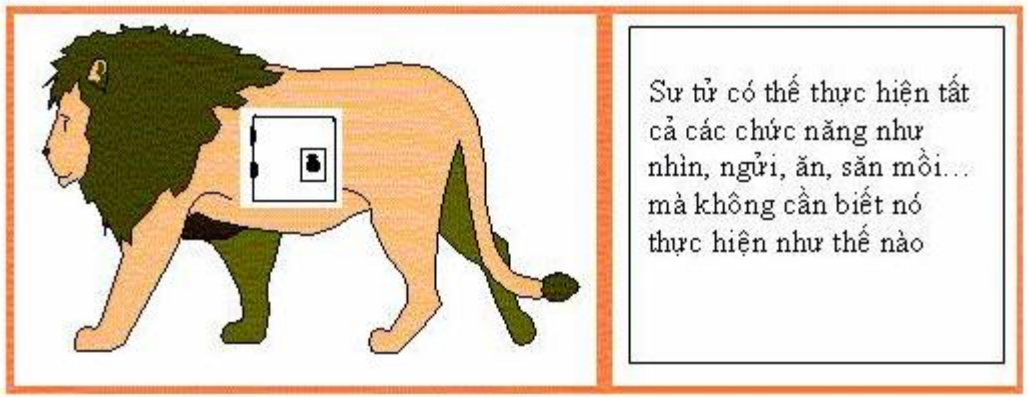
- **Thuộc tính** (attribute) là dữ liệu trình bày các đặc điểm về một đối tượng.
- **Phương thức** (method) có liên quan tới những thứ mà đối tượng có thể làm. Một phương thức đáp ứng một chức năng tác động lên dữ liệu của đối tượng (thuộc tính).

Thông Điệp & Truyền Thông Điệp

- **Thông điệp** (message) là một lời yêu cầu một hoạt động. Gồm có:
  - Đối tượng nhận thông điệp
  - Tên của phương thức thực hiện
  - Các tham số mà phương thức cần
- **Truyền thông điệp**: một đối tượng triệu gọi một hay nhiều phương thức của đối tượng khác để yêu cầu thông tin.

Tính Bao Gói (Encapsulation)

- **Đóng gói** (encapsulation) là tiến trình che giấu việc thực thi chi tiết của một đối tượng.



Ẩn Thông Tin (Information Hiding)

- **Đóng gói → Thuộc tính được lưu trữ hay phương thức được cài đặt như thế nào → được che giấu đi từ các đối tượng khác**

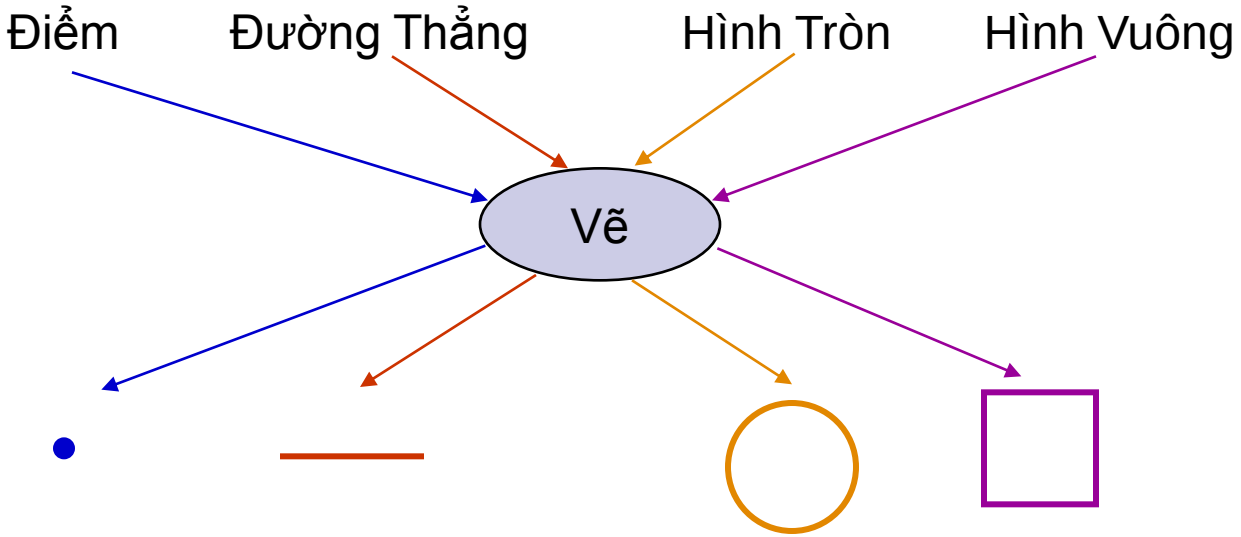
Việc che giấu những chi tiết thiết kế và cài đặt từ những đối tượng khác được gọi là **ẩn thông tin**

Tính Thừa Kế (Inheritance)

- Hệ thống hướng đối tượng cho phép các lớp được định nghĩa kế thừa từ các lớp khác
  - Ví dụ, lớp xe đạp leo núi và xe đạp đua là những lớp con (subclass) của lớp xe đạp.
- **Thừa kế** nghĩa là các phương thức và các thuộc tính được định nghĩa trong một lớp có thể được thừa kế hoặc được sử dụng lại bởi lớp khác.

Tính Đa Hình (Polymorphism)

- **Đa hình**: "nhiều hình thức", hành động cùng tên có thể được thực hiện khác nhau đối với các đối tượng/các lớp khác nhau.
- Ngữ cảnh khác → kết quả khác



CHƯƠNG 7: LỚP (CLASS)

Bộ môn Hệ Thống Máy Tính và Truyền Thông
Khoa Công Nghệ Thông Tin và Truyền Thông
Đại học Cần Thơ

Nội dung

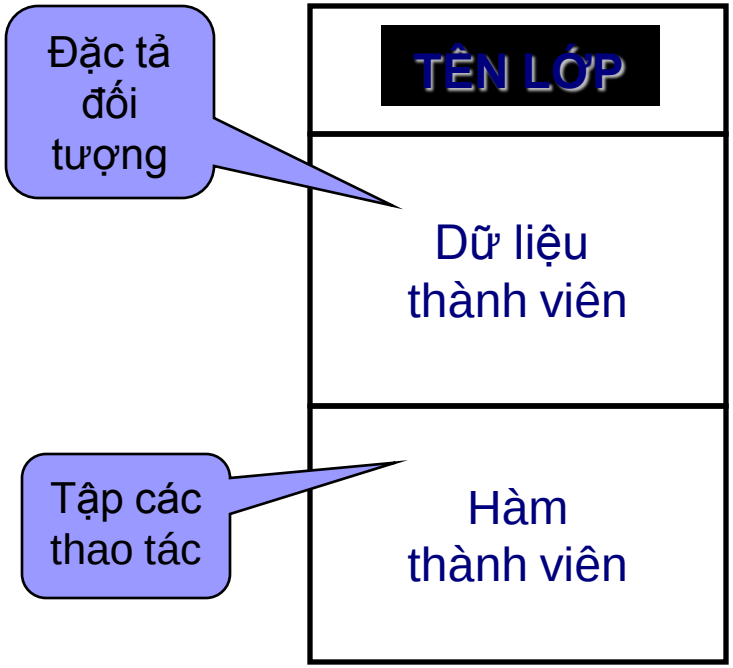
- Lớp – Quyền truy xuất
- Khai báo, định nghĩa 1 lớp đơn giản
- Hàm thành viên nội tuyến (inline)
- Hàm xây dựng (constructor)
- Hàm hủy (destructor)
- Hàm bạn (friend) – Lớp bạn
- Đối số mặc định
- Đối số thành viên ẩn (con trỏ this)

Nội dung (tt)

- Toán tử phạm vi
- Danh sách khởi tạo thành viên
- Thành viên hằng - Thành viên tĩnh
- Thành viên tham chiếu
- Thành viên là đối tượng của 1 lớp
- Mảng các đối tượng
- Phạm vi lớp
- Cấu trúc (structure) và hợp (union)
- Các trường bit

Khái niệm lớp

- **Lớp**: kiểu dữ liệu trừu tượng.

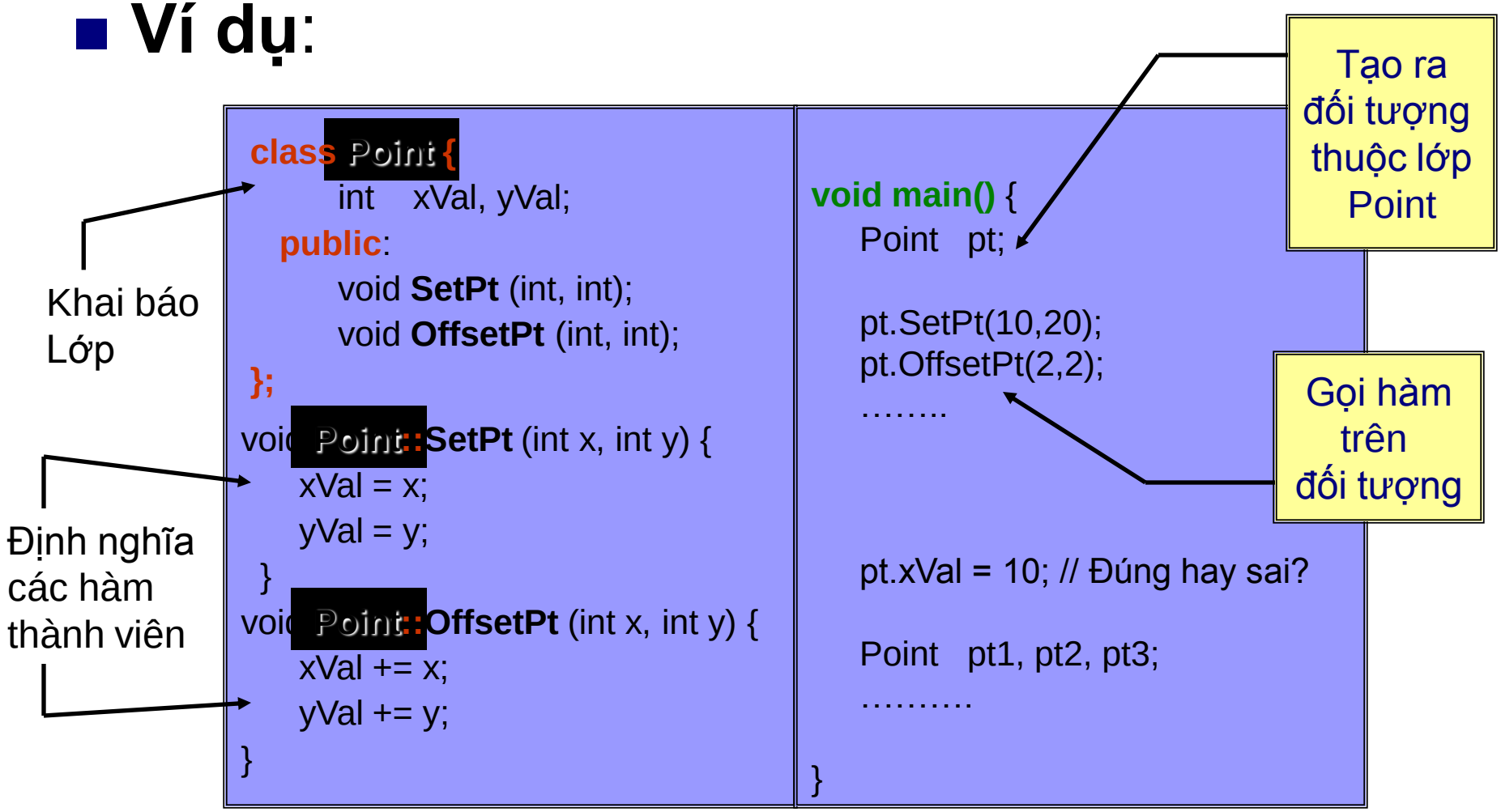


private
protected
public

```
class TÊNLỚP
   [: <Quyền truy xuất> LỚPCHA
 {   <Quyền truy xuất > :
          DataType1  memberdata1;
          DataType2  memberdata2;
          ……..
     < Quyền truy xuất > :
          memberFunction1();
          memberFunction2();
          ……..
 };
```

Lớp đơn giản

- **Ví dụ**:



Hàm thành viên nội tuyến

- Hàm **inline**
  - Cải thiện tốc độ thực thi
  - Tốn bộ nhớ (dành cho mã lệnh) khi thực thi.

Cách 1: thêm Từ khóa **inline**

```
class Point {
        int    xVal, yVal;
    public:
        void SetPt (int, int);
        void OffsetPt (int, int);
 };
inline void Point::SetPt (int x, int y) {
     xVal = x;
     yVal = y;
 }
……….
```

Cách 2: Định nghĩa bên trong lớp

```
class Point {
        int    xVal, yVal;
public:
        void SetPt (int  x, int  y) {
             xVal = x;
             yVal = y;
         }
        void OffsetPt (int  x, int  y) {
             xVal += x;
             yVal += y;
         }
};
```

Ví dụ - Lớp Set (tập hợp)

```
#include <iostream.h>
const  maxCard = 100;
enum   Bool {false, true};
class  Set {
    private:
        int    elems[maxCard];
        int    card;
    public:
        void    EmptySet(){ card = 0; }
        Bool    IsMember (const int);
        void    AddElem (const int);
        void    RmvElem (const int);
        void    Copy (Set&);
        Bool    Equal (Set&);
        void    Intersect (Set&, Set&);
        void    Union (Set&, Set&);
        void    Print ();
};
```

```
Bool Set::IsMember (const int elem) {
    for (register i = 0; i < card; ++i)
        if (elems[i] == elem)
            return true;
    return false;
}
void Set::AddElem (const int elem) {
    if (IsMember(elem))
        return;
    if (card < maxCard)
        elems[card++] = elem;
    else
        cout << "Set overflow“<<endl;
}
void Set::RmvElem (const int elem) {
    for (register i = 0; i < card; ++i)
        if (elems[i] == elem) {
            for (; i < card-1; ++i)  // Dịch
                elems[i] = elems[i+1];
            --card;
        }
}
```

Ví dụ - Lớp Set (tt)

```
void Set::Copy (Set &set) {
    for (register i = 0; i < card; ++i)
        set.elems[i] = elems[i];
    set.card = card;
}
Bool Set::Equal (Set &set) {
    if (card != set.card)
        return false;
    for (register i = 0; i < card; ++i)
        if (!set.IsMember(elems[i]))
            return false;
    return true;
}
void Set::Print (void) {
    cout << "{";
    for (int i = 0; i < card-1; ++i)
        cout << elems[i] << ",";
    if (card > 0)
        cout << elems[card-1];
    cout << "}"<<endl;
}
```

```
……
int main (void) {
    Set    s1, s2;
    s1.EmptySet();  s2.EmptySet();
    s1.AddElem(10); s1.AddElem(20);
    s1.AddElem(30); s1.AddElem(40);
    s2.AddElem(30); s2.AddElem(50);
    s2.AddElem(10); s2.AddElem(60);
    cout << "s1 = ";   s1.Print();
    cout << "s2 = ";   s2.Print();
    s2.RmvElem(50);
    cout << "s2 - {50} = ";
    s2.Print();
    if (s1.IsMember(20))
        cout << "20 is in s1\n";
    if (!s1.Equal(s2))
        cout << "s1 <> s2\n";
    return 0;
}
```

Hàm xây dựng

- Dùng để định nghĩa và khởi tạo đối tượng cùng 1 lúc.
- Có tên trùng với tên lớp, không có kiểu trả về.
- Không gọi trực tiếp, sẽ được tự động gọi khi khởi tạo đt.
- Gán giá trị, cấp vùng nhớ cho các dữ liệu thành viên

```
class Point {
        int    xVal, yVal;
    public:
        Point (int  x, int  y) {
            xVal = x;   yVal = y;
         }
        void OffsetPt (int  x, int  y) {
            xVal += x;  yVal += y;
         }
};
```

```
void main() {
    Point   pt1(10,20);
    pt1.OffsetPt(2,2);
    ……..
  // Khai báo nào là sai ?
    Point  pt2;
    Point  pt3();
    Point  pt4 = Point(5,5);
    Point  pt5 = new Point(5,5);
    …….
 }
```

Hàm xây dựng (tt)

```
class Point {
        int    xVal, yVal;
    public:
        Point ()  // Hàm xây dựng mặc nhiên
          {  xVal = 0;  yVal = 0;  }
        Point (int  x, int  y) {
              xVal = x;   yVal = y;
          }
        Point (float  len, float  angle) {
              xVal = (int) (len * cos(angle));
              yVal = (int) (len * sin(angle));
          }
        void OffsetPt (int  , int  ); …
};
void main() {
      Point  p1;
     Point  p2(10,20);
     Point  p3(60.3, 3.14);
}
```

```
class Set {
    private:
        int    *elems;
        int    maxCard;
        int    card;
    public:
        Set(const int size) {
              elems = new int[size];
              maxCard = size;
              card = 0;
          }
        ……
 };
void main() {
     Set   s1(100);
     Set   s2(20);
     Set   s3(1000); …
}
```

Mềm dẻo hơn

Không cần phải nhớ gọi hàm EmptySet() khi khởi tạo

Hàm hủy

- Dọn dẹp 1 đối tượng *trước khi* nó được thu hồi.
- Cú pháp: ~TenLop() { …….. }
- Không gọi trực tiếp, sẽ được tự động gọi khi hủy bỏ đt.
- **Thu hồi vùng nhớ** cho các *dữ liệu thành viên* *con trỏ*

```
class Set {
  private:
      int   *elems;
      int   maxCard;
      int   card;
  public:
      Set(const int size) { …… }
     ~Set() {  delete[]   elems;  }
      ….
};
```

```
Set  TestFunct1(Set  s1) {
      Set  *s = new Set(50);
      return *s;
  }

void main() {
    Set  s1(40), s2(50);
    s2 = TestFunct1(s1);
}
```

Tổng cộng có bao nhiêu lần hàm hủy được gọi ?

Bạn (Friend) – Đặt vấn đề

Tập Các Số Nguyên

```
class IntSet
    public:
        //...
    private:
        int elems[maxCard];
        int card;
};
class RealSet
    public:
        //...
    private:
        float elems[maxCard];
        int   card;
};
```

Tập Các Số Thực

Hàm SetToReal dùng để chuyển tập số nguyên thành tập số thực

```
void IntSet::SetToReal (RealSet &set) {
    set.card = card;
    for (register i = 0; i < card; ++i)
        set.elems[i] = (float) elems[i];
}
```

Làm thế nào để thực hiện được việc truy xuất đến thành viên **Private**

Hàm bạn (Friend)

- **Cách 1**: Khai báo hàm thành viên của lớp IntSet là bạn (friend) của lớp RealSet.

Giữ nguyên định nghĩa của lớp IntSet

Thêm dòng khai báo **Friend** cho hàm thành viên **SetToReal**

```
class IntSet
{
    public:
        //...
    private:
        int elems[maxCard];
        int card;
};
class RealSet
{
    public:
        //...
        friend void IntSet::SetToReal(RealSet&);
    private:
        float elems[maxCard];
        int   card;
};
```

Hàm bạn (Friend)

- **Cách 2**:
  - Chuyển hàm SetToReal ra ngoài (độc lập).
  - Khai báo hàm đó là bạn của cả 2 lớp.

```
class IntSet
{
    public:
        //...
        friend void SetToReal (IntSet &, RealSet&);
    private:
        int elems[maxCard];
        int card;
};
class RealSet
{
    public:
        //...
        friend void SetToReal (IntSet &, RealSet&);
    private:
        float elems[maxCard];
        int   card;
};
```

```
void SetToReal (IntSet&  iSet,
                RealSet& rSet )
{
   rSet.card = iSet.card;
   for (int i = 0; i < iSet.card; ++i)
     rSet.elems[i] =
           (float) iSet.elems[i];
}
```

Hàm độc lập là bạn(friend) của cả 2 lớp.

Bạn (Friend)

- Hàm bạn
  - Có quyền truy xuất đến tất cả các *dữ liệu* và *hàm* thành viên (protected + private) của 1 lớp.
  - Lý do:
    - Cách định nghĩa hàm chính xác.
    - Hàm cài đặt không hiệu quả.
- Lớp bạn:
  - Tất cả các hàm trong lớp bạn: là hàm bạn.

```
class A;
class B { // ……
      friend class A;
};
```

```
class IntSet …….. }
class RealSet // …….
      friend class IntSet;
};
```

Đối số mặc định

- Đối số mặc định tính từ bên phải.

```
class Point
        int   xVal, yVal;
    public:
        Point (int x = 0, int y = 0);
        //...
};

void main() {
    Point   p1;          // như là ???
    Point   p2(10);      // như là ???
    Point   p3(10,20);
    Point   p4(, 20);  // ?????
   …..
}
```

```
class Point
        int   xVal, yVal;
    public:
        Point (int x = 0,  int y = 0);
        Point (float x=0, float y=0);
        //...
};

void main() {
    Point   p2(1.6, 5.0);     // như là ???
    Point   p3(10,20);        // như là ???
    Point   p4;               // ?????
   …..
}
```

Tối nghĩa
Mơ hồ

Đối số thành viên ẩn

Con trỏ *this

- Là 1 thành viên ẩn, có thuộc tính là private.
- Trỏ tới chính bản thân đối tượng.

```
void Point::OffsetPt (int x, int y) {
        xVal += x;
        yVal += y;
}
```

⇔

```
void Point::OffsetPt (int x, int y) {
        this->xVal += x;
        this->yVal += y;
}
```

⇒

- Có những trường hợp sử dụng *this là dư thừa (Ví dụ trên)
- Tuy nhiên, có những trường hợp phải sử dụng con trỏ ***this**

Toán tử phạm vi

- **Toán tử** **::** dùng để xác định chính xác hàm (thuộc tính) được truy xuất thuộc lớp nào.
- Câu lệnh: pt.OffsetPt(2,2);

  <=> pt.**Point::**OffsetPt(2,2);
- Cần thiết trong một số trường hợp:
  - Cách gọi hàm trong thừa kế.
  - Tên thành viên bị che bởi biến cục bộ.

Ví dụ:

```
Point(int  xVal, int yVal) {
    Point::xVal = xVal;
    Point::yVal = yVal;
}
```

Danh sách khởi tạo thành viên

- Tương đương việc gán giá trị dữ liệu thành viên.

```
class Point {
        int    xVal, yVal;
    public:
        Point (int  x, int  y) {
            xVal = x;
            yVal = y;
         }
        // ……….……
};
```

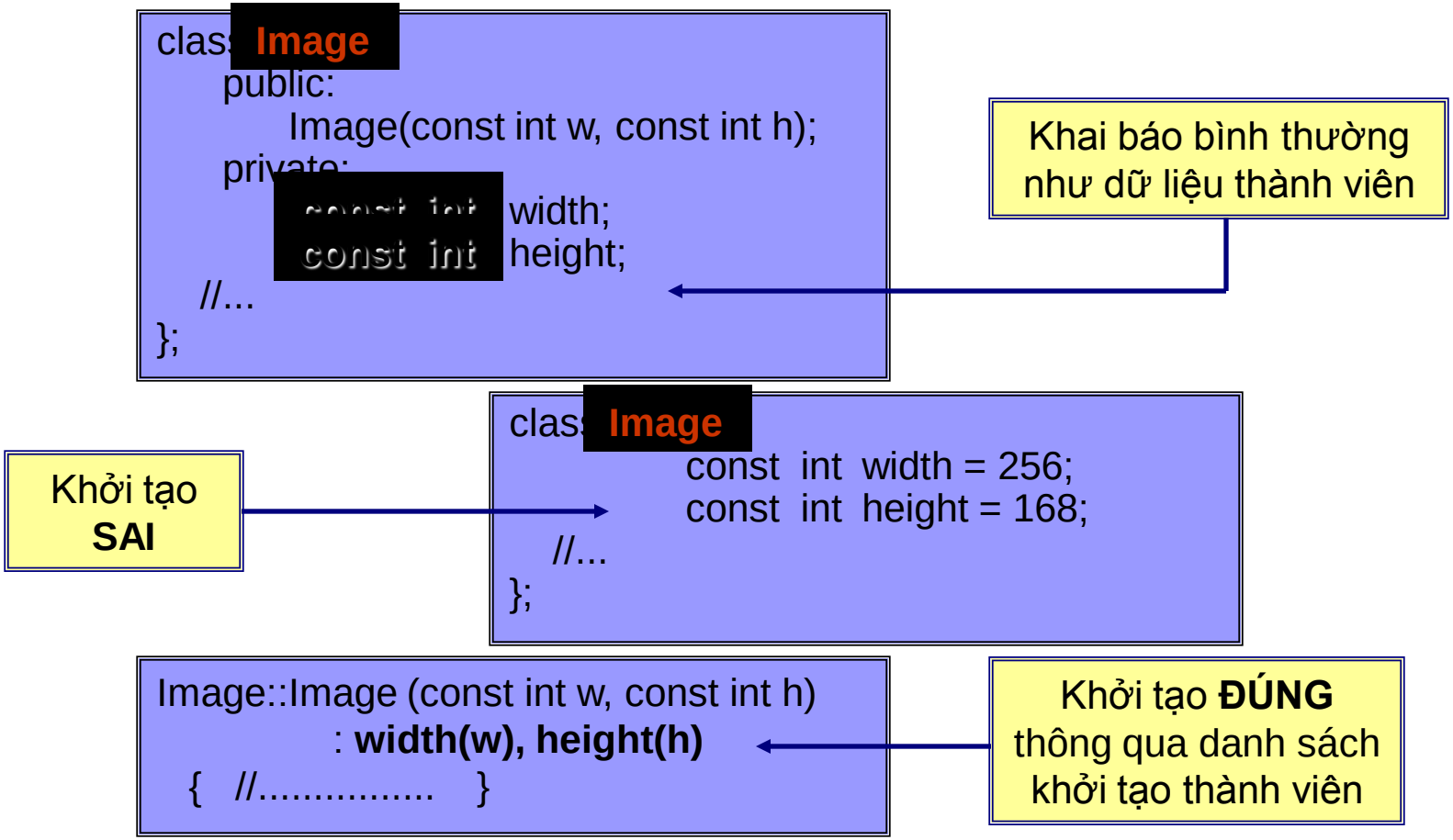
```
Point::Point (int  x, int  y)
          : xVal(x), yVal(y)
   { }
```

```
class Image
        public:
                Image(const int w, const int h);
        private:
                int   width;
                int   height;
                 //...
 };
Image::Image(const int w, const int h) {
                width = w;
                height = h;
                //.....................
 }
```

```
Image::Image (const int w, const int h)
   : width(w), height(h)
 {   //.............   }
```

Thành viên hằng

- Hằng dữ liệu thành viên:



Thành viên hằng

- Hằng đối tượng: không được thay đổi giá trị.
- Hàm thành viên hằng:
  - Được phép gọi trên hằng đối tượng.
  - Không được thay đổi giá trị dữ liệu thành viên.

```
class Set
    public:
        Set(void){ card = 0; }
        Bool    Member(const int) const;
        void    AddElem(const int);
        //...
};
Bool Set::Member (const int elem) const
 {      //...
 }
```

```
void main() {
    const Set s;
    s.AddElem(10);   // SAI
    s.Member(10);    // ok

 }
```

Thành viên tĩnh

- **Dữ liệu thành viên tĩnh**:
  - Dùng chung 1 bản sao chép (1 vùng nhớ) chia sẻ cho tất cả đối tượng của lớp đó.
  - Sử dụng: <TênLớp>::<TênDữLiệuThànhViên>
  - Thường dùng để đếm số lượng đối tượng.

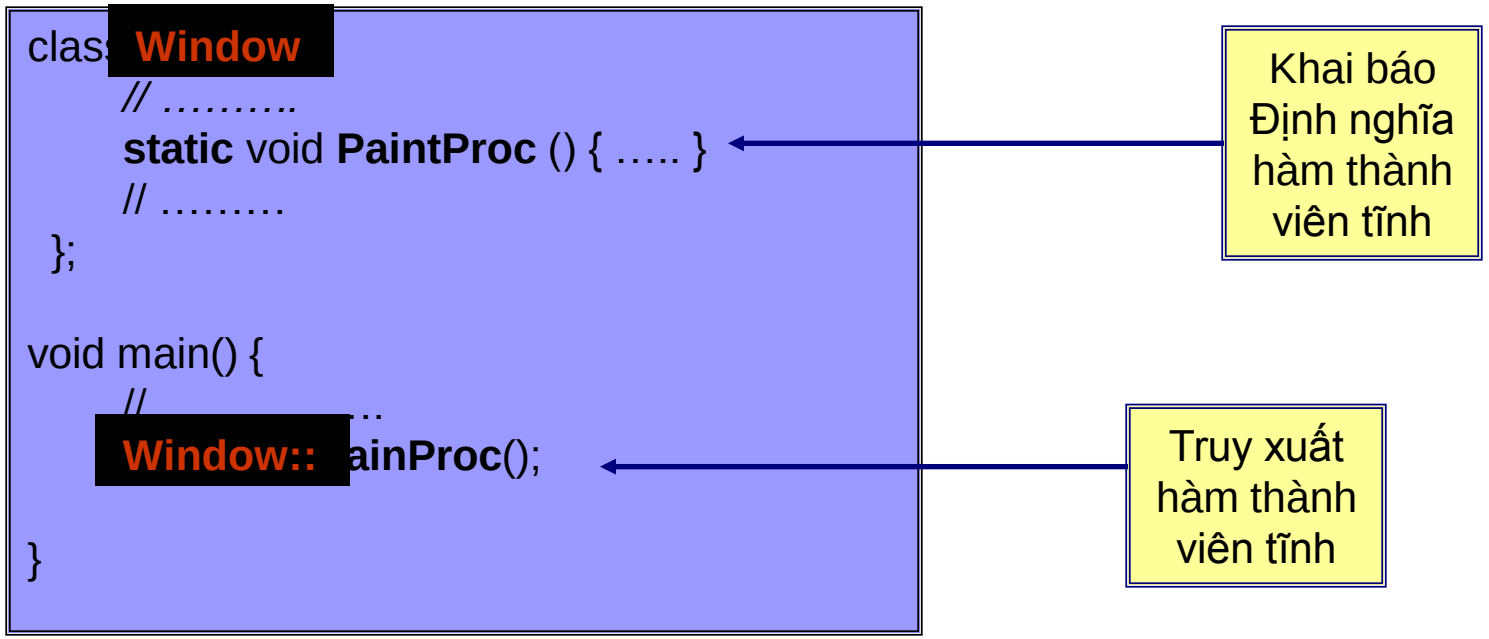
```
class Window
{
        // danh sách liên kết tất cả Window
    static Window *first;
        // con trỏ tới window kế tiếp
        Window *next;
        //...
};

Window Window::rst = &myWindow;
// ……
```

Khai báo

Khởi tạo dữ liệu thành viên tĩnh

Thành viên tĩnh

- **Hàm thành viên tĩnh**:
  - Tương đương với hàm toàn cục.
  - Gọi thông qua: <TênLớp>**::**<TênHàm>



Thành viên tham chiếu

- **Tham chiếu dữ liệu thành viên**:

```
class Image {
        int   width;
        int   height;
        int   &widthRef;
   //...
};
```

Khai báo bình thường như dữ liệu thành viên

Khởi tạo **SAI**

```
class Image {
        int   width;
        int   height;
        int   &widthRef = width;
   //...
};
```

```
Image::Image (const int w, const int h)
        : widthRef(width)
{  //…….. }
```

Khởi tạo **ĐÚNG** thông qua danh sách khởi tạo thành viên

Thành viên là đối tượng của 1 lớp

- **Dữ liệu thành viên** có thể có kiểu:
  - Dữ liệu (lớp) chuẩn của ngôn ngữ.
  - Lớp do người dùng định nghĩa (có thể là chính lớp đó).

```
class Point { };
class Rectangle
    public:
        Rectangle (int left, int top, int right, int bottom);
        //...
    private:
        Point    topLeft;
        Point    botRight;
};
Rectangle::Rectangle (int left, int top, int right, int bottom)
        : topLeft(left,top), botRight(right,bottom)
{ }
```

Khởi tạo cho các dữ liệu thành viên qua danh sách khởi tạo thành viên

Mảng các đối tượng

- Sử dụng **hàm xây dựng không đối số** (hàm xây dựng mặc nhiên - default constructor).

  VD: Point pentagon[5];

- Sử dụng bộ khởi tạo mảng:

  VD: Point triangle[3] =

  { Point(4,8), Point(10,20), Point(35,15) };

  Ngắn gọn:

  Set s[4] = { 10, 20, 30, 40 };

  tương đương với:

  Set s[4] = { Set(10), Set(20), Set(30), Set(40) };

Mảng các đối tượng

- **Sử dụng dạng con trỏ**:
  - Cấp vùng nhớ:

    VD: Point *pentagon = new Point[5];
  - Thu hồi vùng nhớ:

    delete[] pentagon;

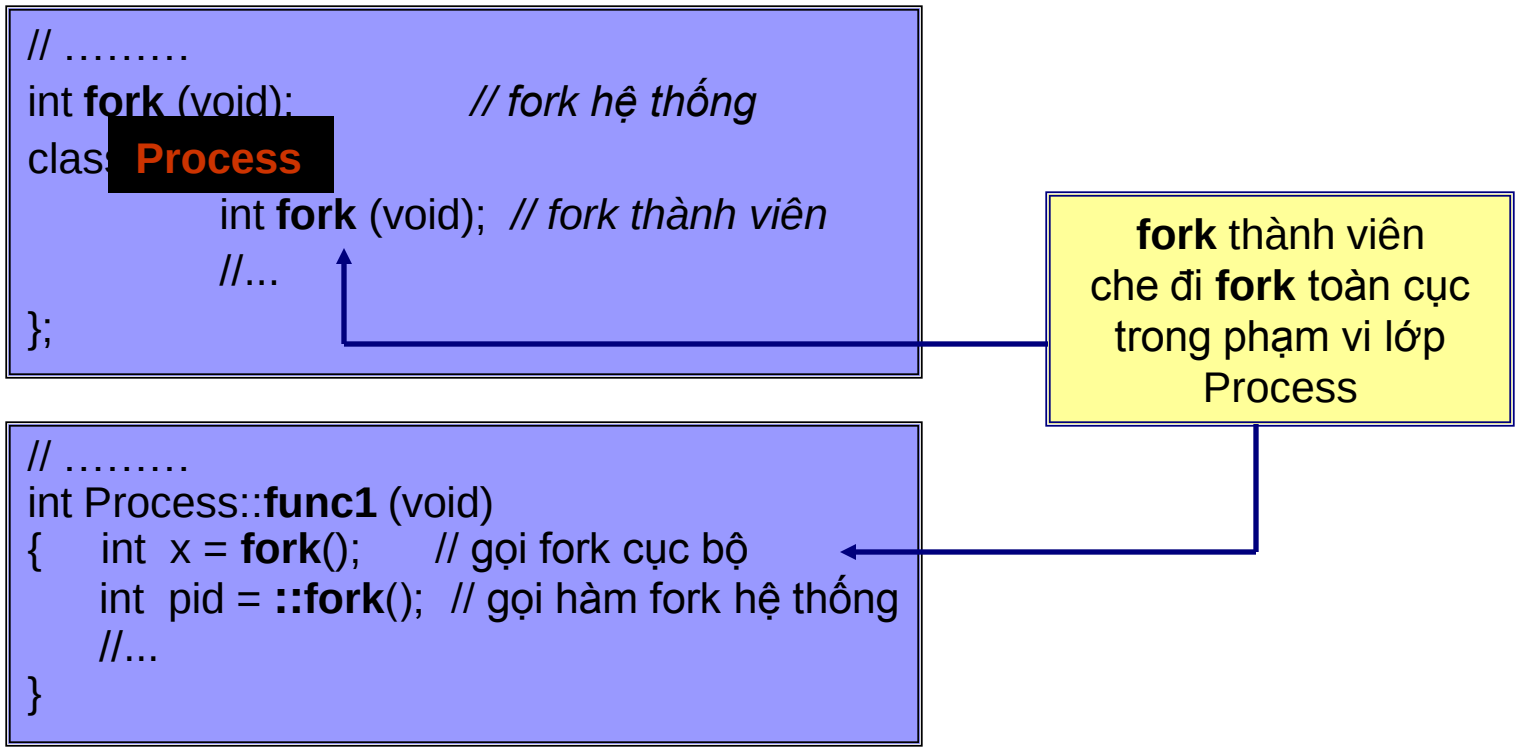
    delete pentagon; *// Thu hồi vùng nhớ đầu*

```
class Polygon
    public:
        //...
    private:
        Point   *vertices;    // các đỉnh
        int     nVertices;    // số các đỉnh
};
```

Không cần biết kích thước mảng.

Phạm vi lớp

- **Thành viên trong 1 lớp:**
  - Che các thực thể trùng tên trong phạm vi.



Phạm vi lớp

- Lớp toàn cục: đại đa số lớp trong C++.
- Lớp lồng nhau: lớp chứa đựng lớp.
- Lớp cục bộ: trong 1 hàm hoặc 1 khối.

```
class Rectangle      // Lớp lồng nhau
    public:
        Rectangle (int, int, int, int);
        //..
    private:
        class Point {
            public:
                Point(int  a, int  b) { … }
            private:
                int x, y;
         };
        Point   topLeft, botRight;
};
Rectangle::Point   pt(1,1);  // sd ở ngoài
```

```
void Render (Image   &i)
 {
    class ColorTable
        public:
            ColorTable () { /* ... */ }
            AddEntry (int r, int g, int b)
                { /* ... */ }
          //...
      };
    ColorTable   colors;
    //...
 }
ColorTable   ct;  // SAI
```

Cấu trúc và hợp

- **Cấu trúc (structure):**
  - Bắt nguồn từ ngôn ngữ C.
  - Tương đương với **class** với các thuộc tính là public.
  - Sử dụng như **class**.

```
struc Point
        Point (int, int);
        void OffsetPt(int, int);
        int  x, y;
};
```

```
clas Point
        public:
            Point(int, int);
            void OffsetPt(int, int);
            int          x, y;
};
```

```
Point   p = { 10, 20 };
```

Có thể khởi tạo dạng này nếu không có định nghĩa hàm xây dựng

Cấu trúc và hợp

- **Hợp (union)**:
  - Tất cả thành viên ánh xạ đến cùng 1 địa chỉ bên trong đối tượng chính nó (không liên tiếp).
  - Kích thước = kích thước của dữ liệu lớn nhất.

```
union Value
{
    long    integer;
    double  real;
    char    *string;
    Pair    list;
    //...
};
```

```
class Pair
{
    Value   *head;
    Value   *tail;
    //...
};
```

```
class Object
{
  private:
    enum ObjType {intObj, realObj,
                  strObj, listObj};
    ObjType type; // kiểu đối tượng
    Value   val;  // giá trị của đối tượng
    //...
};
```

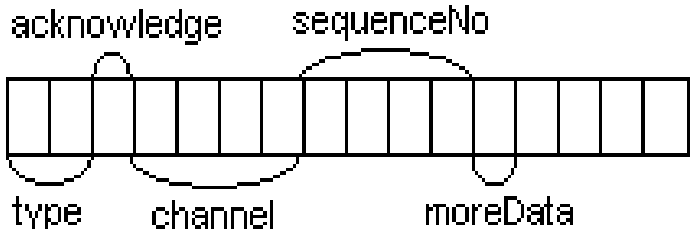
Kích thước của Value là 8 bytes = sizeof(double)

Các trường bit

- Điều khiển đối tượng ở mức bit.

  VD: Truy xuất các bit trong header của gói tin.

```
typedef unsigned int Bit
class Packet
{
        Bit   type          : 2;   // rộng 2 bit
        Bit   acknowledge   : 1;   // rộng 1 bit
        Bit   channel       : 4;   // rộng 4 bit
        Bit   sequenceNo    : 4;   // rộng 4 bit
        Bit   moreData      : 1;   // rộng 1 bit
        //...
};
enum PacketType { dataPack, controlPack,
                  supervisoryPack };
enum Bool { false, true };
```




```
//
Packet p;
p.type = controlPack;
p.acknowledge = true;
```

CHƯƠNG 8:

TÁI ĐỊNH NGHĨA (OVERLOADING)

Bộ môn Hệ Thống Máy Tính và Truyền Thông
Khoa Công Nghệ Thông Tin và Truyền Thông
Đại học Cần Thơ

Nội dung

- Tái định nghĩa hàm.
- Tái định nghĩa toán tử.
- Chuyển đổi kiểu.
- Tái định nghĩa toán tử xuất (<<)– nhập (>>)
- Tái định nghĩa toán tử [], toán tử ()
- Khởi tạo ngầm định - Gán ngầm định.
- Tái định nghĩa toán tử ++ và --
- Tái định nghĩa new và delete

Tái định nghĩa hàm

- Định nghĩa các hàm cùng tên
- Đối số phải khác nhau:
  - Số lượng
  - Kiểu
  - Thứ tự

```
class Time
{
    //...
    long GetTime (void); // số giây tính từ nửa đêm
    void GetTime (int &hours,
                  int &minutes,
                  int &seconds);
};
```

```
void main() {
  int h, m, s;
  long t = GetTime();  // Gọi hàm ???
  GetTime(h, m, s);    // Gọi hàm ???
}
```

- Có thể dùng đối số mặc định.

Tái định nghĩa toán tử

- Định nghĩa các phép toán trên đối tượng.
- Các phép toán có thể tái định nghĩa:

Đơn hạng	+	-	*	!	~	&	++	--	()	->	->*
	new	delete									
Nhị hạng	+	-	*	/	%	&	|	^	<<	>>	
	=	+=	-=	/=	%=	&=	|=	^=	<<=	>>=	
	==	!=	<	>	<=	>=	&&	||	[]	()	,

- Các phép toán không thể tái định nghĩa:

. .* :: ?: sizeof

Tái định nghĩa toán tử (tt)

- **Bằng hàm thành viên**:

```
class Point
    public:
        Point (int x, int y)      { Point::x = x; Point::y = y; }
        Point  operator + (Point &p)   { return Point(x + p.x,y + p.y); }
        Point  operator -  (Point &p)   { return Point(x - p.x, y - p.y); }
    private:
        int x, y;
};
```

Có 1 tham số
(Nếu là toán tử nhị hạng)

```
void main() {
    Point  p1(10,20),  p2(10,20);
    Point  p3 = p1 + p2;                Point  p4 = p1 - p2;
    Point  p5 = p3.operator + (p4);   Point  p6 = p3.operator – (p4);
};
```

Tái định nghĩa toán tử (tt)

- **Bằng hàm độc lập**: thường khai báo **friend**

```
class Point
    public:
        Point (int x, int y)      { Point::x = x; Point::y = y; }
        friend  Point  operator + (Point &p,  Point &q)
                    {return  Point(p.x + q.x,p.y + q.y); }
        friend  Point  operator - (Point &p,  Point &q)
                    {return  Point(p.x - q.x,p.y - q.y); }
     private:
        int x, y;
};
```

Có 2 tham số
(Nếu là toán tử nhị hạng)

```
void main() {
    Point  p1(10,20),  p2(10,20);
    Point  p3 = p1 + p2;                Point  p4 = p1 - p2;
    Point  p5 =operator + (p3, p4);   Point  p6 = operator – (p3, p4);
};
```

Tái định nghĩa toán tử (tt)

- Cải tiến lớp tập hợp (Set):

```
#include <iostream.h>
const       maxCard = 100;
enum        Bool {false, true};
class Set
{
  public:
    Set(void) { card = 0; }
    friend Bool operator &  (const int, Set&);// thanh vien ?
    friend Bool operator == (Set&, Set&);     // bang ?
    friend Bool operator != (Set&, Set&);     // khong bang ?
    friend Set  operator *  (Set&, Set&);     // giao
    friend Set  operator +  (Set&, Set&);     // hop
    //...
    void   AddElem(const int elem);
    void   Copy   (Set &set);
    void   Print  (void);
  private:
    int   elems[maxCard];
    int   card;
};
```

```
// Định nghĩa các toán tử
……….
……….
int main (void)
{  Set        s1, s2, s3;
   s1.AddElem(10); s1.AddElem(20);
   s1.AddElem(30); s1.AddElem(40);
   s2.AddElem(30); s2.AddElem(50);
   s2.AddElem(10); s2.AddElem(60);
   cout << "s1 = ";        s1.Print();
   cout << "s2 = ";        s2.Print();
   if (20 & s1) cout << "20 thuoc s1\n";
   cout << "s1 giao s2 = "; (s1 * s2).Print();
   cout << "s1 hop s2 = ";  (s1 + s2).Print();
   if (s1 != s2) cout << "s1 /= s2\n";
   return 0;
}
```

Chuyển kiểu

- Muốn thực hiện các phép cộng:

```
void main() {
        Point  p1(10,20),  p2(30,40), p3, p4, p5;
        p3 = p1 + p2;
        p4 = p1 + 5;  p5 = 5  + p1;
};
```

⇒ Có thể định nghĩa thêm 2 toán tử:

```
clas Point
        //...
        friend Point operator + (Point, Point);
        friend Point operator + (int,     Point);
        friend Point operator + (Point, int);
};
```

Chuyển kiểu (tt)

- Chuyển đổi kiểu: ngôn ngữ định nghĩa sẵn.

```
void main() {
        Point  p1(10,20),  p2(30,40), p3, p4, p5;
        p3 = p1 + p2;
        p4 = p1 + 5;    // tương đương  p1 + Point(5)
        p5 = 5  + p1;   // tương đương  Point(5) + p1
 }
```

⇒ Định nghĩa phép chuyển đổi kiểu

```
class Point
        //...
        Point (int x) { Point::x = Point::y = x; }
        friend Point  operator + (Point, Point);
 };
```

Chuyển kiểu
5 ⇔ Point(5)

Tái định nghĩa toán tử xuất <<

- Định nghĩa hàm toàn cục:

  ostream& **operator <<** (ostream&, **Class**&);

```
class Point
    public:
        Point (int x=0, int y=0)
            { Point::x = x; Point::y = y; }
        friend   ostream&  operator <<
                    (ostream&  os,  Point&  p)
            {   os<< "(" << p.x << "," << p.y << ")";  }
        // .....
    private:
        int x, y;
};
```

```
void main() {
    Point  p1(10,20), p2;
    cout<<"Diem P1: "<< p1 << endl;
    cout<<"Diem P2: "<< p2 << endl;
}
```

Kết quả trên màn hình ?

Tái định nghĩa toán tử nhập >>

- Định nghĩa hàm toàn cục:

  istream& **operator >>** (istream&, **Class**&);

```
class Point
    public:
        Point (int x=0, int y=0)
            { Point::x = x; Point::y = y; }
        friend   istream&  operator >>
                    (istream&  is,  Point&  p)
            {   cout<<"Nhap x: "; is>>p.x;
                cout<<"Nhap y: "; is>>p.y;
            }
        // .....
    private:
        int x, y;
};
```

```
void main() {
    Point  p1, p2;
    cout<<"Nhap thong tin cho P1: \n";
    cin>>p1;
    cout<<"Nhap thong tin cho P2: \n";
    cin>>p2;
 }
```

Tái định nghĩa toán tử [ ]

- Thông thường để xuất ra giá trị của 1 phần tử tại vị trí cho trước trong đối tượng.
- Định nghĩa là hàm thành viên.

```
class StringVec
    public:
        StringVec (const int dim);
        ~StringVec ();
        char*   operator [] (int);
        int   add(char* );
        // ……..
    private:
        char   **elems;    // cac phan tu
        int        dim;       // kich thuoc cua vecto
        int        used;      // vi tri hien tai
};
```

```
char*  StringVec::operator [] (int  i) {
      if ( i>=0 && i<used)  return elems[i];
      return “”;
  }

void main() {
   StringVec   sv1(100);
   sv1.add(“PTPhi”);sv1.add(“BQThai”);
   sv1.add(“LVLam”); sv1.add(“NCHuy”);
   cout<< sv1[2]<<endl;
   cout<<sv1[0];
}
```

Tái định nghĩa toán tử ()

- Định nghĩa là hàm thành viên.

```
class Matrix
  public:
    Matrix (const short rows, const short cols);
    ~Matrix (void) {delete elems;}
    double&  operator () (const short row,
                          const short col);
    friend  ostream& operator << (ostream&, Matrix&);
    friend  Matrix   operator  + (Matrix&, Matrix&);
    friend  Matrix   operator  - (Matrix&, Matrix&);
    friend  Matrix   operator  * (Matrix&, Matrix&);
  private:
    const short     rows;       // số hàng
    const short     cols;       // số cột
    double          *elems;     // các phần tử
};
```

```
double& Matrix::operator ()
   (const short row, const short col)
{
   static  double dummy = 0.0;
   return (row >= 1 && row <= rows
          && col >= 1 && col <= cols)
           ? elems[(row - 1)*cols
                   + (col - 1)]
           : dummy;
}
void main() {
   Matrix   m(3,2);
   m(1,1) = 10; m(1,2) = 20;
   m(2,1) = 30; m(2,2) = 40;
   m(3,1) = 50; m(3,2) = 60;
   cout<<m<<endl;
}
```

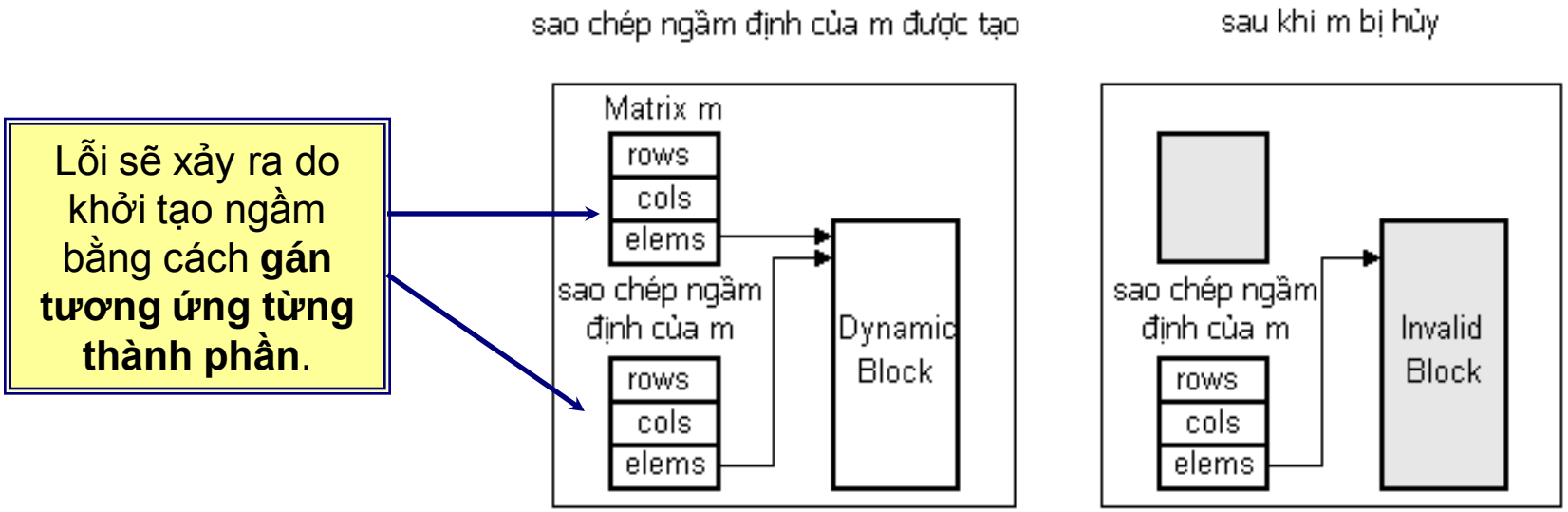
Khởi tạo ngầm định

- Được định nghĩa sẵn trong ngôn ngữ:

  VD: Point p1(10,20); Point p2 = p1;

- Sẽ gây ra lỗi (kết quả SAI) khi bên trong đối tượng có *thành phần dữ liệu là con trỏ*

  VD: Matrix m(5,6); Matrix n = m;



Khởi tạo ngầm định (tt)

Khi lớp có *thành phần dữ liệu con trỏ*
phải định nghĩa hàm **xây dựng sao chép**

Point

```
class Point
{
    int x, y;
  public:
    Point (int =0; int =0 );
    // Khong can thiet DN
    Point (const Point& p) {
        x= p.x;
        y = p.y;
      }
    // ……..
};
// ……
```

Matrix

```
class Matrix
{
    //….
    Matrix(const Matrix&);
};
Matrix::Matrix (const Matrix &m)
    : rows(m.rows), cols(m.cols)
 {
     int n = rows * cols;
     elems = new double[n];          // cùng kích thước
     for (register i = 0; i < n; ++i)   // sao chép phần tử
          elems[i] = m.elems[i];
  }
```

Gán ngầm định

- Được định nghĩa sẵn trong ngôn ngữ:
  - Gán tương ứng từng thành phần.
  - **Đúng** khi đối tượng không có dữ liệu con trỏ.

  VD: Point p1(10,20); Point p2; p2 = p1;
- Khi thành phần dữ liệu có con trỏ, bắt buộc phải định nghĩa phép gán = cho lớp.

Hàm thành viên

```
class Matrix
    //….
    Matrix&  operator = (const Matrix &m) {
        if (rows == m.rows && cols == m.cols) {        // phải khớp
            int n = rows * cols;
            for (register i = 0; i < n; ++i)  // sao chép các phần tử
                elems[i] = m.elems[i];
        }
        return *this;
    }
};
```

Tái định nghĩa toán tử ++ & --

- Toán tử ++ (hoặc toán tử --) có 2 loại:
  - Tiền tố: ++n
  - Hậu tố: n++

```
class PhanSo
{
    int tuso, mau so;
  public:
    // …..
    PhanSo(int=0 , int =1);
    friend PhanSo operator ++ (PhanSo&);
    friend PhanSo operator ++ (PhanSo&, int);
};
PhanSo operator ++ (PhanSo& p) {
    return (p = PhanSo(tuso+mauso, mauso));
}
PhanSo operator ++ (PhanSo& p, int  x) {
    PhanSo  p2 = PhanSo(tuso+mauso, mauso);
    return  p2;
}
```

```
void main() {
    PhanSo  p1(3,4), p2;
    cout<< p1++;
    cout<<++p2;
    cout<<++(p1++) + (++p2)++;
}
```

Kết quả trên màn hình ?

Tái định nghĩa new & delete

- Hàm **new** và **delete** mặc định của ngôn ngữ:
  - Nếu đối tượng kích thước nhỏ, có thể sẽ gây ra quá nhiều khối nhỏ => chậm.
  - Không đáng kể khi đối tượng có kích thước lớn.

  => Toán tử new và delete ít được tái định nghĩa.
- Định nghĩa theo dạng hàm thành viên:

```
class Point
    public:
        //...
        void* operator new (size_t bytes);
        void  operator delete (void *ptr, size_t bytes);
    private:
        int xVal, yVal;
};
```

```
void main() {
    Point  *p = new Point(10,20);
    Point  *ds = new Point[30];
    //……….
    delete p;
    delete []ds;
}
```

CHƯƠNG 9:

THỪA KẾ (INHERITANCE)

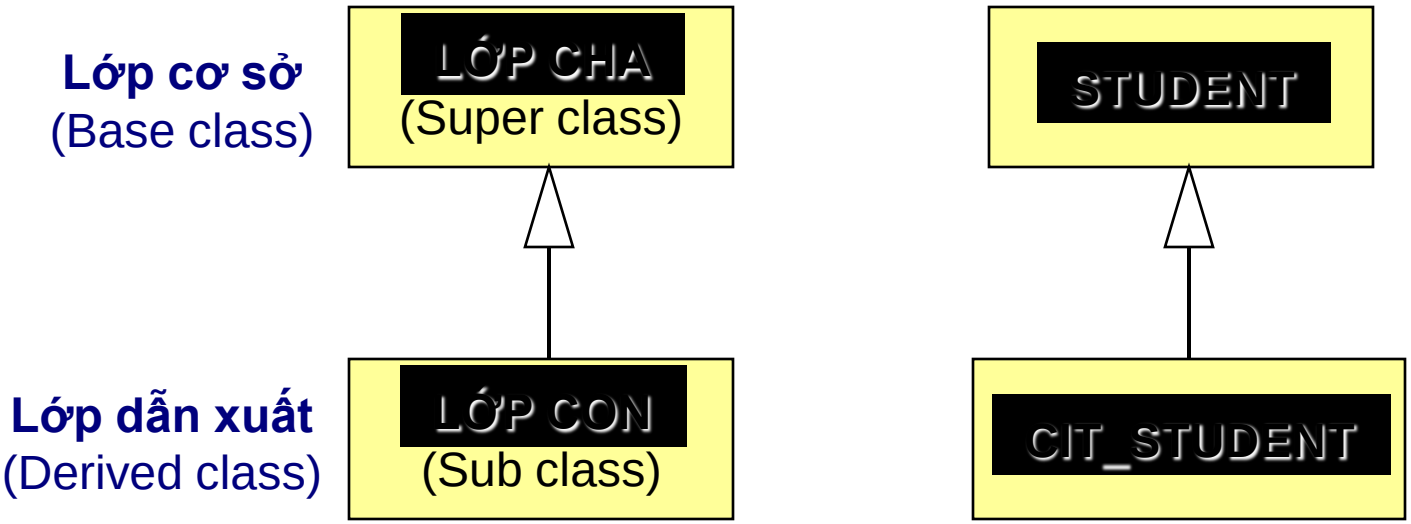
Bộ môn Hệ Thống Máy Tính và Truyền Thông
Khoa Công Nghệ Thông Tin và Truyền Thông
Đại học Cần Thơ

Nội dung

- Khái niệm
- Lớp dẫn xuất đơn giản
- Ký hiệu các thứ bậc
- Hàm xây dựng và hàm hủy
- Thành viên lớp được bảo vệ
- Lớp cơ sở riêng, chung và được bảo vệ
- Đa thừa kế - Sự mơ hồ
- Hàm ảo - Lớp cơ sở ảo
- Chuyển kiểu
- Các toán tử được tái định nghĩa

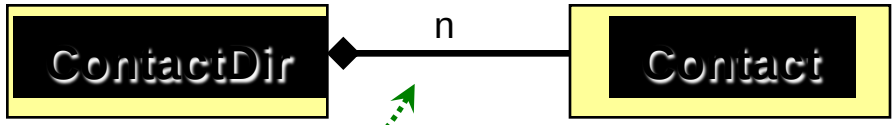
Khái niệm

- Kế thừa từ các lớp có từ trước.
- Ích lợi: có thể tận dụng lại
  - Các thuộc tính chung
  - Các hàm có thao tác tương tự



Ví dụ minh họa

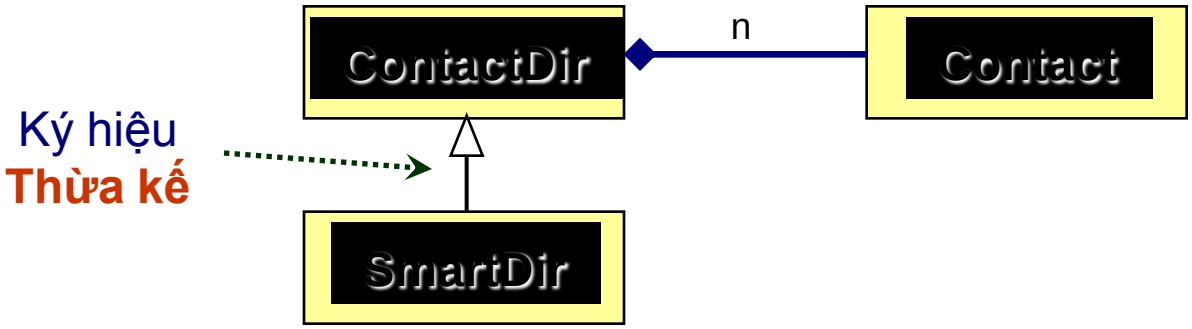
Ký hiệu **composition**




```
#include <iostream.h>
#include <string.h>
class Contact
   private:
      char   *name;      // ten doi tac
      char   *address;   // dia chi doi tac
      char    *tel;      // so dien thoai
   public:
      Contact (const char *name,
               const char *address, const char *tel);
      ~Contact ();
      const char*  Name () const     { return name;}
      const char*  Address() const { return address;}
      const char*  Tel() const     { return tel;}
      friend ostream&  operator <<
                        ( ostream&,  Contact& );
};
```

```
class ContactDir
 private:
      int    Lookup(const char *name);
      Contact   **contacts;    // ds cac doi tac
      int    dirSize;    // kich thuoc thu muc hien tai
      int    maxSize;  // kich thuoc thu muc toi da
public:
      ContactDir (const int maxSize);
      ~ContactDir();
      void  Insert(const Contact&);
      void  Delete(const char *name);
      Contact* Find(const char *name);
      friend  ostream&  operator <<
                        (ostream&, ContactDir&);
      // ……
};
```

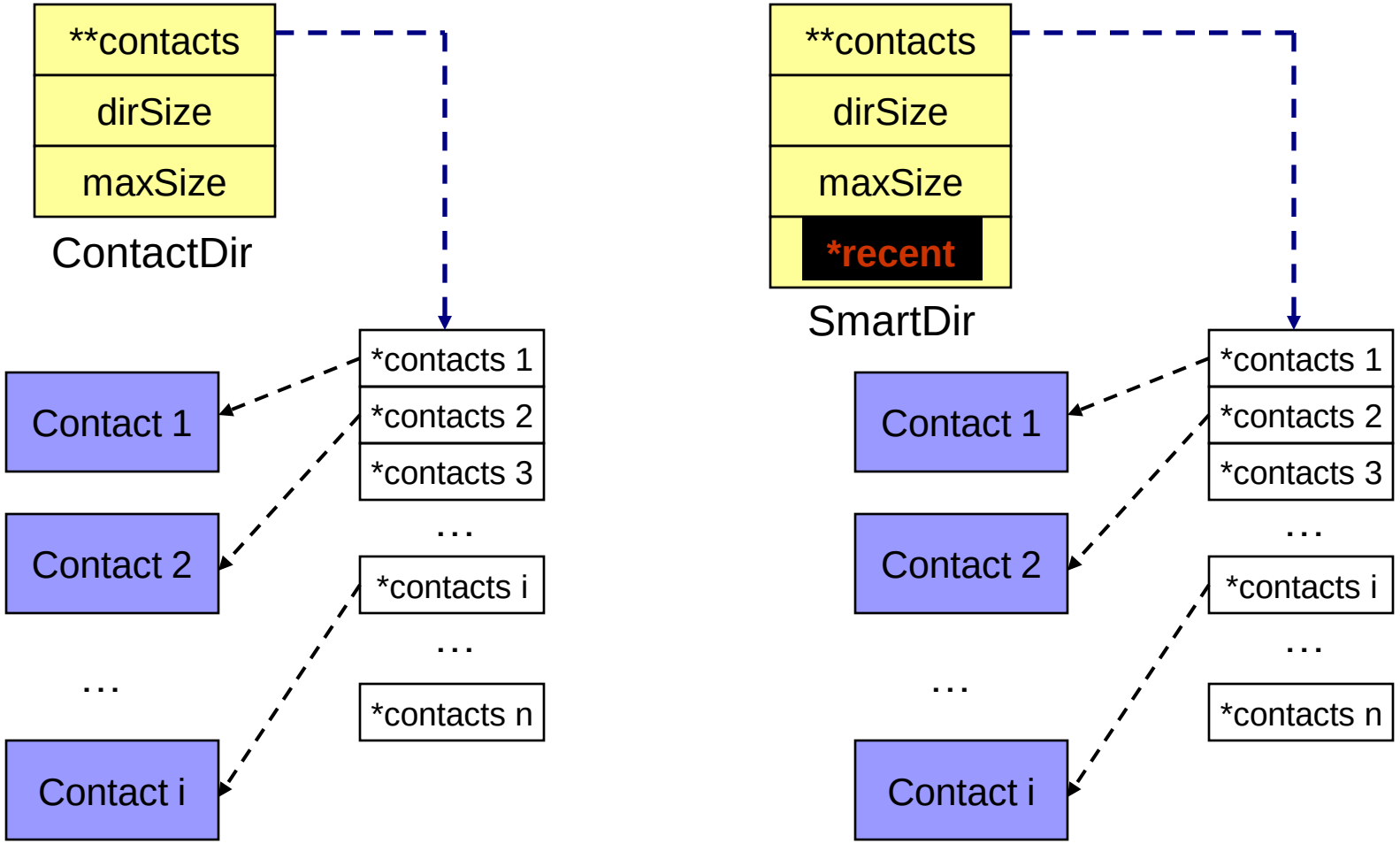
Ví dụ minh họa (tt)




```
class SmartDir public ContactDir {
    private:
        char     *recent;    // ten duoc tim gan nhat
    public:
        SmartDir(const int max) : ContactDir(max)
            { recent = 0; }
        Contact* Recent (void);
        Contact*  Find  (const char *name);
        // ……..
};
```

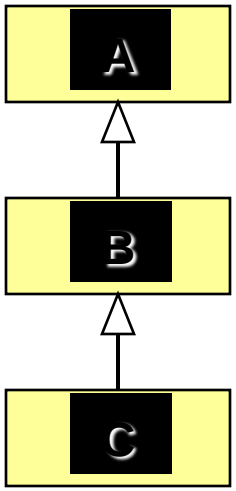
```
Contact* SmartDir::Recent (void) {
      return recent == 0 ? 0 :
                  ContactDir::Find(recent);
}
Contact* SmartDir::Find (const char *name) {
     Contact *c = ContactDir::Find(name);
     if (c != 0)
         recent = (char*) c->Name();
     return c;
}
```

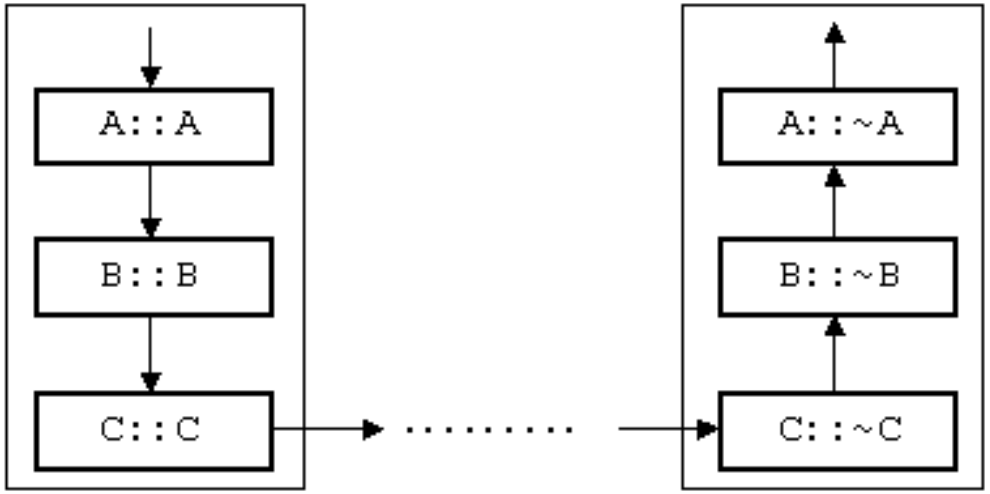
Ví dụ (tt) - mô tả trong bộ nhớ



Hàm xây dựng và hàm hủy

- Trong thừa kế, khi khởi tạo đối tượng:
  - Hàm xây dựng của **lớp cha** sẽ được gọi trước
  - Sau đó mới là hàm xây dựng của **lớp con**.
- Trong thừa kế, khi hủy bỏ đối tượng:
  - Hàm hủy của **lớp con** sẽ được gọi trước
  - Sau đó mới là hàm hủy của **lớp cha**.





Hàm xây dựng và hàm hủy (tt)

```
class SmartDir public ContactDir {
    private:
        char  *recent;  // ten duoc tim gan nhat
    public:
        SmartDir(const int max) : ContactDir(max)
            { recent = 0; }
        SmartDir(const SmartDir& sd): ContactDir(sd)
            { recent = 0; }
        ~SmartDir() {
            delete recent;
          }
        // ……
};
```

Gọi hàm xây dựng của lớp cha

Thu hồi vùng nhớ của con trỏ thành viên của lớp con nếu đã cấp vùng nhớ trong hàm xây dựng.

Thành viên lớp được bảo vệ

- Thừa kế:
  - Có tất cả các dữ liệu và hàm thành viên.
  - Không được truy xuất đến thành viên **private**.
- Thuộc tính truy cập **protected**
  - Cho phép lớp con truy xuất.

```
class ContactDir
    //...
      protected:
          int     Lookup (const char *name);
          Contact  **contacts; // ds cac doi tac
          int     dirSize;     // kich thuoc hien tai
          int  maxSize;    // kich thuoc toi da
};
```

```
class Foo {
      public:
    // cac thanh vien chung...
      private:
    // cac thanh vien rieng...
      protected:
    // cac thanh vien duoc bao ve...
      public:
    // cac thanh vien chung nua...
      protected:
    // cac thanh vien duoc bao ve nua...
};
```

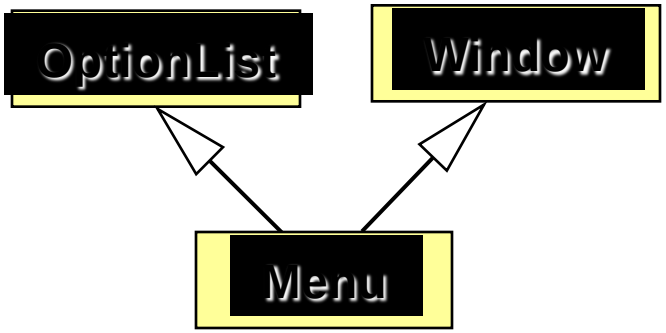
Lớp cơ sở riêng, chung và được bảo vệ

Lớp cơ sở	Thừa kế public	Thừa kế private	Thừa kế protected
private	_	_	_
public	public	private	protected
protected	protected	private	protected

```
class A
        private:
                        int x;
                        void Fx (void);
        public:
                        int y;
                        void Fy (void);
        protected:
                        int z;
                        void Fz (void);
};
```

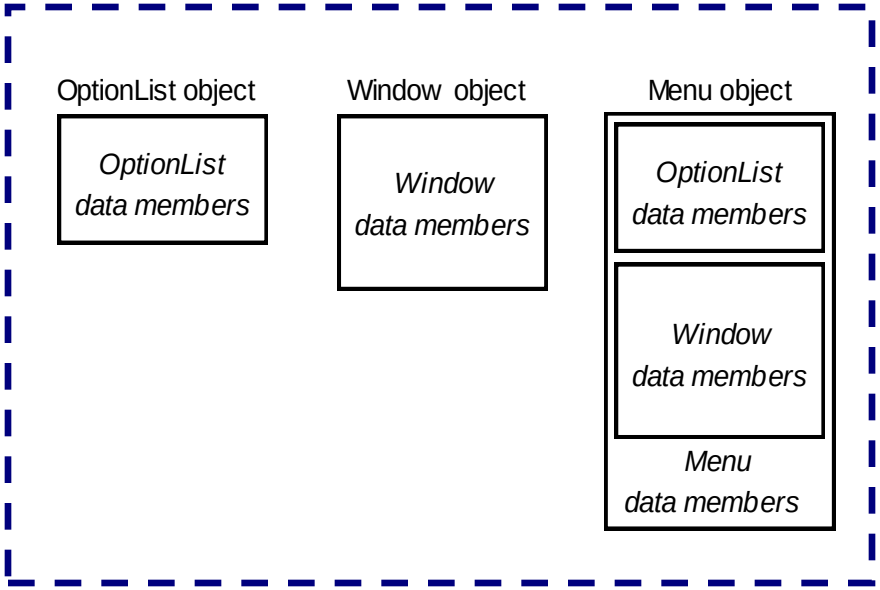
```
class B A {  // Thừa kế dạng private
        .......
 };
class C private  A {  // A là lớp cơ sở riêng của B
        .........
 };
class D public  A {  // A là lớp cơ sở chung của C
        .........
 };
class E protected  A {  // A: lớp cơ sở được bảo vệ
        .........
 };
```

Đa thừa kế




```
class OptionList
    public:
        OptionList (int n);
        ~OptionList ();
        //...
};
```

```
class Window
    public:
        Window (Rect &);
        ~Window (void);
        //...
};
```




```
class Menu
        : public OptionList, public Window {
    public:
        Menu (int n, Rect &bounds);
        ~Menu (void);
        //...
};

Menu::Menu (int n, Rect &bounds) :
    OptionList(n), Window(bounds)
 { /* ... */ }
```

Sự mơ hồ trong đa thừa kế

```
class OptionList
    public:
        // ……
        void Highlight (int part);
};
```

```
class Window
    public:
        // ……
        void Highlight (int part);
};
```

```
class Menu : public OptionList,
             public Window
    { ……. };
```

Hàm cùng tên

Chỉ rõ hàm của lớp nào

Gọi hàm của lớp nào ?

```
void main() {
    Menu  m1(….);
    m1.Highlight(10);
    ….
}
```

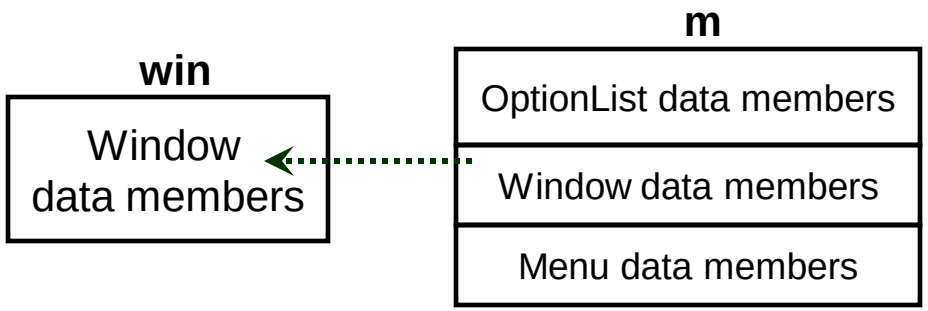
xử lý

```
void main() {
    Menu  m1(….);
    m1.OptionList::Highlight(10);
    m1.Window::Highlight(20);
    ….
}
```

Chuyển kiểu

- Có sẵn 1 phép chuyển kiểu không tường minh:
  - Đối tượng lớp cha = Đối tượng lớp con;
  - Áp dụng cho cả đối tượng, tham chiếu và con trỏ.

```
Menu     m(n, bounds);
Window  win = m;
Window  &wRef = m;
Window  *wPtr = &menu;
```



- Không được thực hiện phép gán ngược:
  - Đối tượng lớp con = Đối tượng lớp cha; // SAI

Nếu muốn thực hiện phải tự định nghĩa phép ép kiểu

```
class Menu : public OptionList, public Window {
    public:
        //...
        Menu (Window&);
};
```

Hàm ảo

- Liên kết tĩnh (*static binding*):
  - Xác định khi biên dịch chương trình.
  - Dùng **hàm thành viên**.
  - Gọi hàm của **lớp cơ sở (lớp cha)**.

```
class ContactDir
        //...
    public:
        int Lookup (const char *name);
        //...
};
```

```
class SortedDir public ContactDir {
    public:
        SortedDir(const int max) : ContactDir(max) {}
         int Lookup(const char *name);
};
```

```
void main() {
    ContactDir  c1(10);
    SortedDir  *p; p = &c1;
    cout<<p->Lookup("ABC");
    ….
}
```

Hàm ảo (tt)

- Liên kết động (*dynamic binding*)
  - Xác định khi thực thi chương trình.
  - Dùng **hàm ảo** (virtual function).
  - Gọi hàm của **lớp dẫn xuất (lớp con)**.
  - Thể hiện tính **đa hình** của OOP.

```
class ContactDir
{
    //...
  public:
    virtual int Lookup (const char *name);
};
```

```
class SortedDir public ContactDir {
    //....
  public:
    int Lookup(const char *name);
};
```

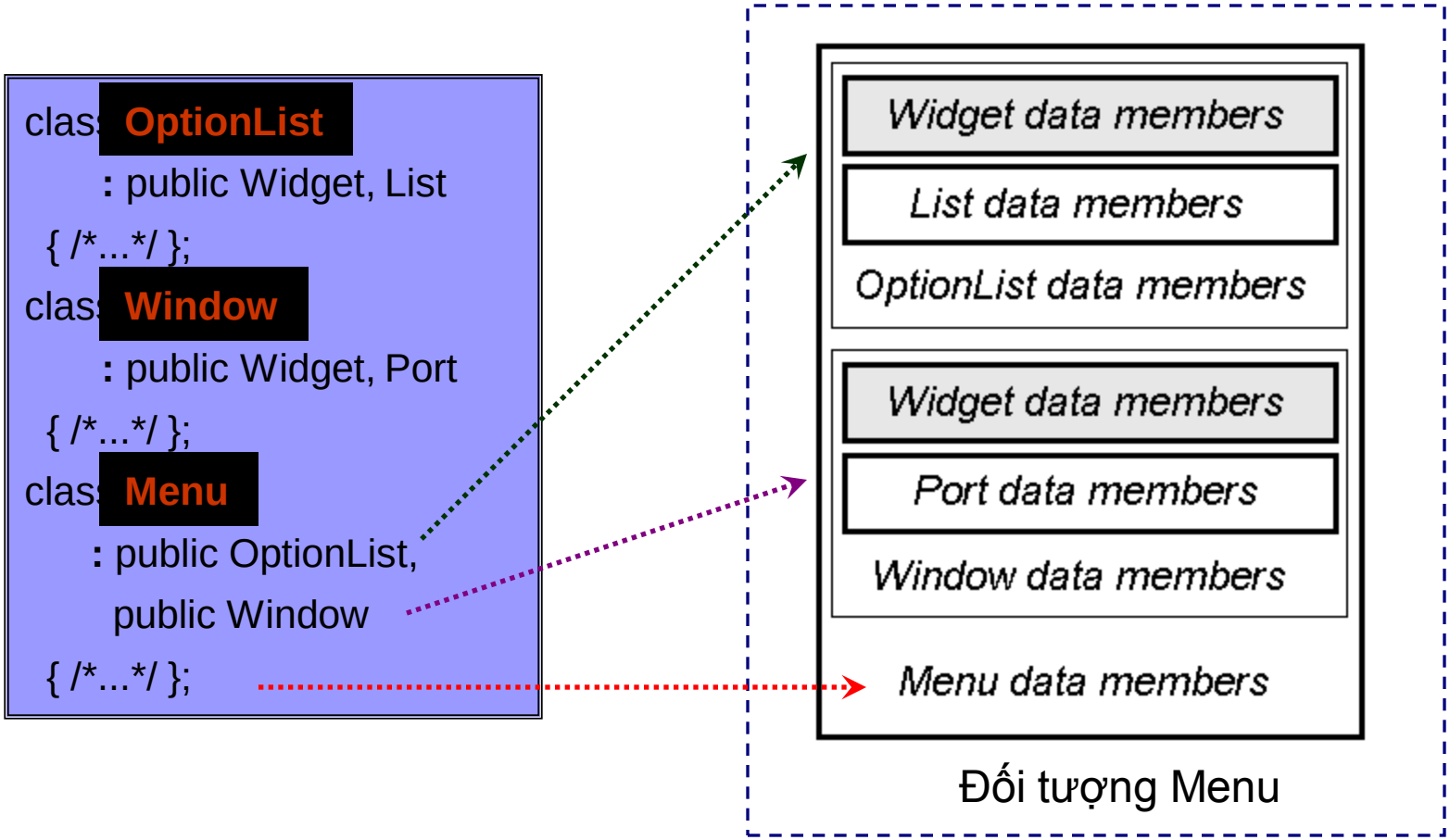
```
void main() {
    ContactDir  c1(10);
    SortedDir  *p1; p1 = &c1;
    cout<<p->Lookup("ABC");

    SortedDir  c2(20);
    ContactDir  *p2; p2 = &c2;
    cout<<p->Lookup("ABC");
}
```

Gọi hàm của lớp nào ?

Kết quả trên màn hình là gì?

Lớp cơ sở ảo

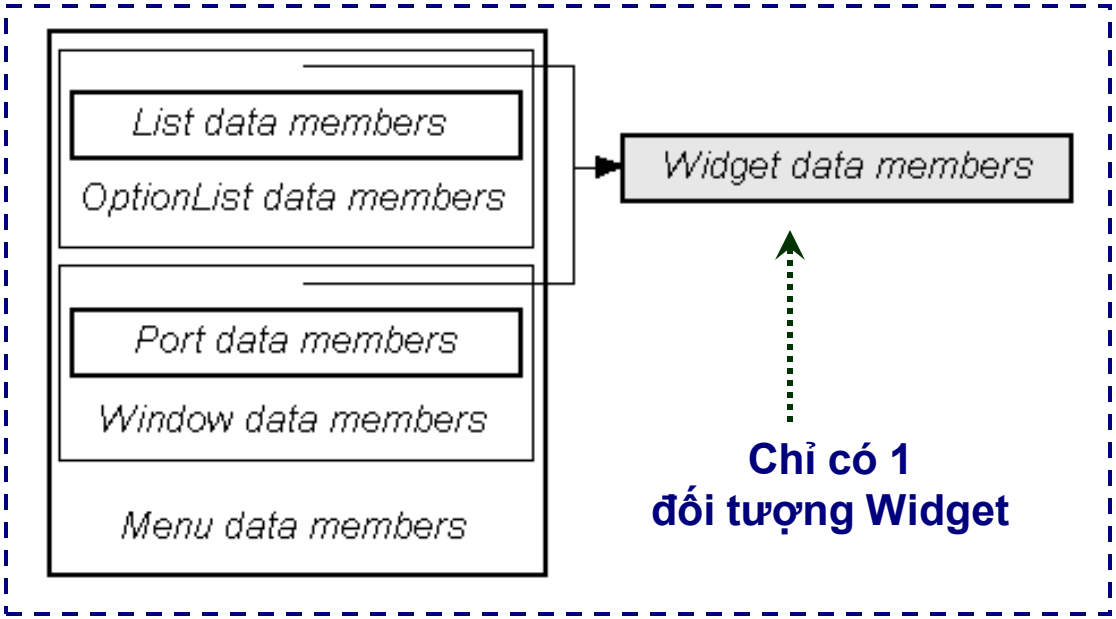
- Sự mơ hồ - dư thừa dữ liệu



Lớp cơ sở ảo (tt)

- Cách xử lý: dùng lớp cơ sở ảo.

```
class OptionList
      : virtual public Widget,
        public List
  { /*...*/ };
class Window
      : virtual public Widget,
        public Port
  { /*...*/ };
class Menu
     : public OptionList,
       public Window
  { /*...*/ };
```




```
Menu::Menu (int n, Rect &bounds) :
     Widget(bounds), OptionList(n), Window(bounds)
  { //... }
```

Các toán tử được tái định nghĩa

- Tương tự như tái định nghĩa hàm thành viên:
  - Che giấu đi toán tử của lớp cơ sở.
  - Hàm xây dựng sao chép:

    Y **Y (const Y&)**
  - Phép gán:

    Y& Y **operator =** const Y&)
- Nếu không định nghĩa, sẽ tự động có hàm xây dựng sao chép và phép gán do ngôn ngữ tạo ra.

  => **SAI** hi có con trỏ thành viên.
- Cẩn thận với toán tử **new** và **delete**.

BỔ SUNG

MỘT SỐ VẤN ĐỀ KHÁC CẦN TỰ TÌM HIỂU THÊM

Stream

- Nhập xuất thông qua Stream (Dòng):
  - Tổng quan về Stream.
  - Đối tượng nhập xuất chuẩn.
  - File.
  - Nhập xuất với File đối tượng.
  - Tái định nghĩa toán tử << với File.
  - Tái định nghĩa toán tử >> với File.

Template

- Thiết kế lớp theo Template (mẫu):
  - Thế nào là Template.
  - Hàm template.
  - Lớp template.
  - Giới thiệu về STL (**S**tandard **T**emplate **L**ibrary).

Exception

- **Exception** (Ngoại lệ):
  - Ngoại lệ là gì?
  - Đề xuất ngoại lệ - Nắm bắt ngoại lệ.
  - Ngoại lệ trong C++.
  - Cú pháp bắt ngoại lệ trong C++.
  - Sử dụng ngoại lệ với Template.

Container

- **Container** (Lớp vật chứa):
  - Container là gì?
  - Tại sao lại cần Container.
  - Container trong C++.
  - Một số Container trong 1 phiên bản của C++.

CÂU HỎI?