



TRƯỜNG ĐẠI HỌC HÙNG VƯƠNG

GT.0000027084

TS. PHẠM THẾ QUẾ (CHỦ BIÊN) - ThS. ĐINH THÁI SƠN  
ThS. NGUYỄN ĐÌNH NHƯ - ThS. VŨ THỊ THU MINH - ThS. NÔNG THỊ LÝ

## GIÁO TRÌNH

# KỸ THUẬT LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Sách tặng



NHÀ XUẤT BẢN ĐẠI HỌC THÁI NGUYÊN



**TRƯỜNG ĐẠI HỌC HÙNG VƯƠNG**

TS. Phạm Thế Quέ (Chủ biên) - ThS. Đinh Thái Sơn  
ThS. Nguyễn Đình Như - ThS. Vũ Thị Thu Minh - ThS. Nông Thị Lý

**(GIÁO TRÌNH**

**KỸ THUẬT LẬP TRÌNH  
HƯỚNG ĐỐI TƯỢNG**

**NHÀ XUẤT BẢN ĐẠI HỌC THÁI NGUYÊN  
NĂM 2017**

MÃ SỐ: 02 - 49  
ĐHTN - 2017

## LỜI MỞ ĐẦU

Lập trình cấu trúc là phương pháp tổ chức, phân chia chương trình thành các hàm, thủ tục, chúng được dùng để xử lý dữ liệu nhưng lại tách rời các cấu trúc dữ liệu. Thông qua các ngôn ngữ Foxpro, Pascal, C đa số những người làm Tin học đã khá quen biết với phương pháp lập trình này.

Lập trình hướng đối tượng dựa trên việc tổ chức chương trình thành các lớp. Khác với hàm và thủ tục, lớp là một đơn vị bao gồm cả dữ liệu và phương thức xử lý. Vì vậy lớp có thể mô tả các thực thể một cách chân thực, đầy đủ cả phần dữ liệu và yêu cầu quản lý. Tư tưởng lập trình hướng đối tượng được áp dụng cho hầu hết các ngôn ngữ chạy trên môi trường Windows và Unix như Visual C++, Visual C#, PHP, Java.

Ngôn ngữ lập trình C ra đời năm 1973 với mục đích ban đầu là để viết hệ điều hành Unix trên máy tính mini PDP. Sau đó C đã được sử dụng rộng rãi trên nhiều loại máy tính khác nhau và đã trở thành một ngôn ngữ lập trình cấu trúc rất được ưa chuộng.

Để đưa ngôn ngữ C vào thế giới hướng đối tượng, năm 1980 nhà khoa học người Mỹ B.Stroustrup đã cho ra đời một ngôn ngữ C mới có tên ban đầu là "C có lớp", sau đó đến năm 1983 thì gọi là C++. Ngôn ngữ C++ là một sự phát triển mạnh mẽ của C. Trong C++ chẳng những đưa vào tất cả các khái niệm, công cụ của lập trình hướng đối tượng mà còn đưa vào nhiều khả năng mới mẽ cho hàm. Như vậy C++ là một ngôn ngữ lai cho phép tổ chức chương trình theo các lớp và các hàm. Có thể nói C++ đã thúc đẩy ngôn ngữ C vốn đã rất thuyết phục đi vào thế giới lập trình hướng đối tượng và C++ đã trở thành ngôn ngữ hướng đối tượng nổi bật trong những năm 90.

Trong những năm gần đây, lập trình hướng đối tượng đã trở nên gần gũi nhờ sự ra đời liên tiếp của các ngôn ngữ lập trình hướng đối tượng. Sức mạnh của phương pháp lập trình hướng đối tượng thể hiện ở chỗ khả năng mô hình hóa hệ thống dựa trên các đối tượng thực tế, khả năng đóng gói và bảo vệ an

toàn dữ liệu, khả năng sử dụng lại mã nguồn để tiếp cận chi phí và tài nguyên, đặc biệt là khả năng chia sẻ mã nguồn trong cộng đồng lập trình viên chuyên nghiệp. Những điểm mạnh này hứa hẹn sẽ thúc đẩy phát triển môi trường lập trình tiên tiến cùng với nền công nghiệp lắp ráp phần mềm với các thư viện thành phần có sẵn.

Giáo trình *Kỹ thuật lập trình hướng đối tượng* được biên soạn phục vụ môn học **Lập trình hướng đối tượng** (3 tín chỉ), dành cho đối tượng là sinh viên năm thứ 2 và năm thứ 3 của trường. Nhóm tác giả hi vọng rằng giáo trình sẽ cung cấp một cách tiếp cận tổng thể về phương pháp lập trình hướng đối tượng đồng thời cũng cung cấp các kỹ thuật cơ bản về phát triển các ứng dụng bằng ngôn ngữ lập trình C++, một ngôn ngữ lập trình hướng đối tượng được sử dụng thông dụng nhất hiện nay.

Giáo trình này là kết quả tinh hợp các kiến thức và kinh nghiệm giảng dạy nhiều năm của các thầy cô tại trường Đại học Hùng Vương về lập trình hướng đối tượng. Mặc dù rất cố gắng, song chắc chắn giáo trình không thể tránh khỏi những thiếu sót, nhóm tác giả rất mong nhận được những ý kiến đóng góp, trao đổi của các nhà khoa học, đồng nghiệp và các độc giả để cuốn sách ngày càng được hoàn thiện hơn. Mọi góp ý xin gửi về: Bộ môn Công nghệ phần mềm - Khoa Kỹ thuật công nghệ - Trường Đại học Hùng Vương.

Xin trân trọng cảm ơn!

## CÁC TÁC GIẢ

## MỤC LỤC

<b>CHƯƠNG 1. CÁC KHÁI NIỆM CƠ SỞ CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG</b> .....	9
1.1. Giới thiệu.....	10
1.1.1. Tiếp cận hướng đối tượng.....	10
1.1.2. Những nhược điểm của lập trình hướng thủ tục .....	10
1.1.3. Đặc điểm lập trình hướng đối tượng.....	11
1.2. Các khái niệm cơ bản của lập trình hướng đối tượng .....	12
1.2.1. Đối tượng.....	12
1.2.2. Lớp.....	12
1.2.3. Truật tự hóa dữ liệu và bao gói thông tin .....	13
1.2.4. Kế thừa.....	14
1.2.5. Tương ứng bội .....	15
1.2.6. Liên kết động .....	16
1.2.7. Truyền thông báo .....	16
1.3. Các bước cần thiết để thiết kế chương trình theo hướng đối tượng .....	17
1.4. Các ưu điểm của lập trình hướng đối tượng.....	17
1.5. Các ngôn ngữ hướng đối tượng .....	18
1.6. Một số ứng dụng của LTHĐT .....	20
<b>CHƯƠNG 2. CÁC MỞ RỘNG CỦA NGÔN NGỮ C++</b> .....	21
2.1. Giới thiệu chung về C++ .....	21
2.2. Một số mở rộng của C++ so với C .....	22
2.2.1. Đặt lời chú thích .....	22
2.2.2. Khai báo biến .....	22
2.2.3. Phép chuyển kiểu bắt buộc .....	23
2.2.4. Lấy địa chỉ các phần tử mảng thực 2 chiều .....	24
2.3. Vào ra trong C++ .....	26

2.3.1. Xuất dữ liệu .....	26
2.3.2. Nhập dữ liệu .....	26
2.3.3. Định dạng khi in ra màn hình .....	27
<b>2.4. Cấp phát và giải phóng bộ nhớ.....</b>	<b>31</b>
2.4.1. Toán tử new để cấp phát bộ nhớ.....	31
2.4.2. Toán tử delete .....	32
<b>2.5. Biến tham chiếu.....</b>	<b>34</b>
<b>2.6. Hằng tham chiếu.....</b>	<b>35</b>
2.7. Truyền tham số cho hàm theo tham chiếu.....	36
2.8. Hàm trả về giá trị tham chiếu .....	43
2.9. Hàm với tham số có giá trị mặc định .....	45
2.10. Các hàm nội tuyến (inline) .....	47
2.11. Hàm tài bội .....	51
<b>CHƯƠNG 3. LỚP .....</b>	<b>58</b>
3.1. Định nghĩa lớp .....	59
3.2. Tạo lập đối tượng .....	60
3.3. Truy nhập tới các thành phần của lớp .....	61
3.4. Con trỏ đối tượng .....	69
3.5. Con trỏ this .....	71
3.6. Hàm bạn .....	73
3.7. Dữ liệu thành phần tĩnh và hàm thành phần tĩnh .....	80
3.7.1. Dữ liệu thành phần tĩnh .....	80
3.7.2. Hàm thành phần tĩnh .....	83
3.8. Hàm tạo (constructor).....	86
3.9. Hàm tạo sao chép .....	95
3.9.1. Hàm tạo sao chép mặc định .....	95
3.9.2. Hàm tạo sao chép .....	98
3.10. Hàm hủy (destructor).....	105
<b>CHƯƠNG 4. TOÁN TỬ TÀI BỘI .....</b>	<b>111</b>
4.1. Định nghĩa toán tử tài bội .....	111

4.2. Một số lưu ý khi xây dựng toán tử tài bội .....	112
4.3. Một số ví dụ .....	112
4.4. Định nghĩa chồng các toán tử <code>++</code> , <code>--</code> .....	124
4.5. Định nghĩa chồng toán tử <code>&lt;&lt; và &gt;&gt;</code> .....	128
<b>CHƯƠNG 5. KẾ THỪA</b> .....	131
5.1. Giới thiệu .....	131
5.2. Đơn kế thừa .....	132
5.2.1. Định nghĩa lớp dẫn xuất từ một lớp cơ sở .....	132
5.2.2. Truy nhập các thành phần trong lớp dẫn xuất .....	133
5.2.3. Định nghĩa lại các hàm thành phần của lớp cơ sở trong lớp dẫn xuất .....	134
5.2.4. Hàm tạo đối với tính kế thừa .....	140
5.2.5. Hàm hủy đối với tính kế thừa .....	143
5.2.6. Khai báo <code>protected</code> .....	144
5.2.7. Dẫn xuất <code>protected</code> .....	145
5.3. Đa kế thừa .....	145
5.3.1. Định nghĩa lớp dẫn xuất từ nhiều lớp cơ sở .....	145
5.3.2. Một số ví dụ về đa kế thừa .....	146
5.4. Hàm ảo .....	155
5.4.1 Đặt vấn đề .....	155
5.4.2. Định nghĩa hàm ảo .....	158
5.4.3. Quy tắc gọi hàm ảo .....	161
5.4.4. Quy tắc gán địa chỉ đối tượng cho con trỏ lớp cơ sở .....	161
5.5. Lớp cơ sở ảo .....	166
5.5.1. Khai báo lớp cơ sở ảo .....	166
5.5.2. Hàm tạo và hàm hủy đối với lớp cơ sở ảo .....	169
<b>CHƯƠNG 6. KHUÔN HÌNH</b> .....	177
6.1. Khuôn hình hàm .....	177
6.1.1. Khái niệm .....	177
6.1.2. Tạo một khuôn hình hàm .....	177
6.1.3. Sử dụng khuôn hình hàm .....	179

6.1.4. Các tham số kiểu của khuôn hình hàm .....	179
6.1.5. Định nghĩa chồng các khuôn hình hàm .....	182
<b>6.2. Khuôn hình lớp .....</b>	<b>183</b>
6.2.1. Khái niệm .....	183
6.2.2. Tạo một khuôn hình lớp .....	184
6.2.3. Sử dụng khuôn hình lớp .....	185
6.2.4. Các tham số trong khuôn hình lớp .....	186
6.2.5. Tóm tắt .....	187
<b>PHỤ LỤC 1. CÁC DÒNG XUẤT NHẬP .....</b>	<b>189</b>
1.1. Các lớp stream .....	189
1.2. Dòng cin và toán tử nhập >> .....	190
1.2.1 Dòng cin .....	190
1.2.2. Toán tử trích >> .....	190
1.3. Nhập ký tự và chuỗi ký tự .....	190
1.3.1. Phương thức get() có 3 dạng .....	190
1.3.2. Phương thức getline() .....	192
1.3.3. Phương thức ignore .....	193
1.4. Dòng cout và toán tử xuất << .....	193
1.4.1. Dòng cout .....	193
1.4.2. Toán tử xuất << .....	193
1.4.3. Các phương thức định dạng .....	193
1.4.4. Cờ định dạng .....	195
1.4.5. Các phương thức bật tắt cờ .....	200
1.4.6. Các bộ phận định dạng .....	200
1.4.7. Các hàm định dạng .....	201
1.5. Các dòng chuẩn .....	203
1.6. Xuất ra máy in .....	204
<b>PHỤ LỤC 2. THỨ TỰ ƯU TIÊN CỦA CÁC PHÉP TOÁN .....</b>	<b>207</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>211</b>

# Chương I

## CÁC KHÁI NIỆM CƠ SỞ CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

*Chương I trình bày những vấn đề sau:*

- Thảo luận về cách tiếp cận hướng đối tượng, những nhược điểm của lập trình truyền thống và các đặc điểm của lập trình hướng đối tượng.
- Các khái niệm cơ sở của phương pháp hướng đối tượng:
  - *Đối tượng*
  - *Lớp*
  - *Trình tượng hóa dữ liệu và bao gói thông tin*
  - *Kết thừa*
  - *Tương ứng bội*
  - *Liên kết động*
  - *Truyền thông báo*
- Các bước cần thiết để thiết kế chương trình theo hướng đối tượng
- Các ưu điểm của lập trình hướng đối tượng
- Các ngôn ngữ hướng đối tượng
- Một số ứng dụng của lập trình hướng đối tượng

## 1.1. Giới thiệu

### 1.1.1. Tiếp cận hướng đối tượng

Trong thế giới thực, xung quanh chúng ta là những đối tượng, đó là các thực thể có mối quan hệ với nhau. Ví dụ các *phòng* trong một công ty kinh doanh được xem như những đối tượng. Các *phòng* ở đây có thể là: phòng quản lý, phòng bán hàng, phòng kế toán, phòng tiếp thị,... Mỗi *phòng* ngoài những cán bộ đảm nhiệm những công việc cụ thể, còn có những dữ liệu riêng như thông tin về nhân viên, doanh số bán hàng, hoặc các dữ liệu khác có liên quan đến bộ phận đó. Việc phân chia các phòng chức năng trong công ty sẽ tạo điều kiện dễ dàng cho việc quản lý các hoạt động. Mỗi nhân viên trong phòng sẽ điều khiển và xử lý dữ liệu của phòng đó. Ví dụ phòng kế toán phụ trách về lương bỗng nhân viên trong công ty. Nếu bạn đang ở bộ phận tiếp thị và cần tìm thông tin chi tiết về lương của đơn vị mình thì sẽ gửi yêu cầu về phòng kế toán. Với cách làm này bạn được đảm bảo là chỉ có nhân viên của bộ phận kế toán được quyền truy cập dữ liệu và cung cấp thông tin cho bạn. Điều này cũng cho thấy rằng, không có người nào thuộc bộ phận khác có thể truy cập và thay đổi dữ liệu của bộ phận kế toán. Khái niệm như thế về đối tượng hầu như có thể được mở rộng đối với mọi lĩnh vực trong đời sống xã hội và hơn nữa - đối với việc tổ chức chương trình. Mọi ứng dụng có thể được định nghĩa như một tập các thực thể - hoặc các đối tượng, sao cho quá trình tái tạo những suy nghĩ của chúng ta là gần sát nhất về thế giới thực.

### 1.1.2. Những nhược điểm của lập trình hướng thủ tục

Cách tiếp cận lập trình truyền thống là lập trình hướng thủ tục (LTHTT). Theo cách tiếp cận này thì một hệ thống phần mềm được xem như là dãy các công việc cần thực hiện như đọc dữ liệu, tính toán, xử lý, lập báo cáo và in kết quả v.v.. Mỗi công việc đó sẽ được thực hiện bởi một số hàm nhất định. Như vậy trọng tâm của cách tiếp cận này là các hàm chức năng. LTHTT sử dụng kỹ thuật phân rã hàm chức năng theo cách tiếp cận trên xuống (top-down) để tạo ra cấu trúc phân cấp. Các ngôn ngữ lập trình bậc cao như COBOL, FORTRAN, PASCAL, C, v.v., là những ngôn ngữ lập trình hướng thủ tục. Những nhược điểm chính của LTHTT là:

➤ Chương trình khó kiểm soát và khó khăn trong việc bổ sung, nâng cấp chương trình. Chương trình được xây dựng theo cách tiếp cận hướng thủ tục thực chất là danh sách các câu lệnh mà theo đó máy tính cần thực hiện. Danh sách các lệnh đó được tổ chức thành từng nhóm theo đơn vị cấu trúc của ngôn ngữ lập trình và được gọi là hàm/thủ tục. Trong chương trình có nhiều hàm/thủ tục, thường thì có nhiều thành phần dữ liệu quan trọng sẽ được khai báo tổng thể (global) để các hàm/thủ tục có thể truy nhập, đọc và làm thay đổi giá trị của biến tổng thể. Điều này sẽ làm cho chương trình rất khó kiểm soát, nhất là đối với các chương trình lớn, phức tạp thì vấn đề càng trở nên khó khăn hơn. Khi ta muốn thay đổi, bổ sung cấu trúc dữ liệu dùng chung cho một số hàm/thủ tục thì phải thay đổi hầu như tất cả các hàm/thủ tục liên quan đến dữ liệu đó.

➤ Mô hình được xây dựng theo cách tiếp cận hướng thủ tục (TCHTT) không mô tả được đầy đủ, trung thực hệ thống trong thực tế.

➤ Phương pháp TCHTT đặt trọng tâm vào hàm là hướng tới hoạt động sẽ không thực sự tương ứng với các thực thể trong hệ thống của thế giới thực.

#### *1.1.3. Đặc điểm lập trình hướng đối tượng*

Lập trình hướng đối tượng (Object Oriented Programming - OOP) là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng thuật giải, xây dựng chương trình. Đối tượng được xây dựng trên cơ sở gắn cấu trúc dữ liệu với các phương thức (các hàm/thủ tục) sẽ thể hiện được đúng cách mà chúng ta suy nghĩ, bao quát về thế giới thực. Lập trình hướng đối tượng (LTHĐT) cho phép ta kết hợp những tri thức bao quát về các quá trình với những khái niệm trừu tượng được sử dụng trong máy tính.

Điểm cản bàn của phương pháp LTHĐT là thiết kế chương trình xoay quanh dữ liệu của hệ thống. Nghĩa là các thao tác xử lý của hệ thống được gắn liền với dữ liệu và như vậy khi có sự thay đổi của cấu trúc dữ liệu thì chỉ ảnh hưởng đến một số ít các phương thức xử lý liên quan.

LTHĐT không cho phép dữ liệu chuyển động tự do trong hệ thống. Dữ liệu được gắn chặt với từng phương thức thành các vùng riêng mà các phương thức đó tác động lên và nó được bảo vệ để cấm việc truy nhập tùy tiện từ bên

ngoài. LTHĐT cho phép phân tích bài toán thành tập các thực thể được gọi là các đối tượng và sau đó xây dựng các dữ liệu cùng với các phương thức xung quanh các đối tượng đó.

*Tóm lại LTHĐT có những đặc tính chủ yếu như sau:*

1. Tập trung vào dữ liệu thay cho các phương thức.
2. Chương trình được chia thành các lớp đối tượng.
3. Các cấu trúc dữ liệu được thiết kế sao cho đặc tả được các đối tượng.
4. Các phương thức xác định trên các vùng dữ liệu của đối tượng được gắn với nhau trên cấu trúc dữ liệu đó.
5. Dữ liệu được bao bọc, che giấu và không cho phép các thành phần bên ngoài truy nhập tự do.
6. Các đối tượng trao đổi với nhau thông qua các phương thức.
7. Dữ liệu và các phương thức mới có thể dễ dàng bổ sung vào đối tượng nào đó khi cần thiết.
8. Chương trình được thiết kế theo cách tiếp cận bottom-up (dưới - lên).

## **1.2. Các khái niệm cơ bản của lập trình hướng đối tượng**

Những khái niệm cơ bản trong LTHĐT bao gồm: Đối tượng; Lớp; Trừu tượng hóa dữ liệu, bao gói thông tin; Kế thừa; Tương ứng bội; Liên kết động; Truyền thông báo.

### *1.2.1. Đối tượng*

Trong thế giới thực, khái niệm đối tượng được hiểu như là một thực thể, nó có thể là người, vật hoặc một bảng dữ liệu cần xử lý trong chương trình,... Trong LTHĐT thì đối tượng là biến thể hiện của *lớp*.

### *1.2.2. Lớp*

*Lớp* là một khái niệm mới trong LTHĐT so với kỹ thuật LTHTT. Nó là một bản mẫu mô tả các thông tin cấu trúc dữ liệu và các thao tác hợp lệ của các

phản tử dữ liệu. Khi một phản tử dữ liệu được khai báo là phản tử của một lớp thì nó được gọi là *đối tượng*. Các hàm được định nghĩa hợp lệ trong một lớp được gọi là các *phương thức* (method) và chúng là các hàm duy nhất có thể xử lý dữ liệu của các đối tượng của lớp đó. Chẳng hạn, chúng ta xét lớp HINH\_CN bao gồm các thuộc tính: (x1,y1) toạ độ góc trên bên trái, d, r là chiều dài và chiều rộng của hình chữ nhật (HCN). Các phương thức nhập số liệu cho HCN, hàm tính diện tích, chu vi và hàm hiển thị. Lớp HINH\_CN có thể được mô tả như sau:

HINH_CN	
<b>Thuộc tính :</b>	x1,y1 d,r
<b>Phương thức:</b>	Nhập_sl Diện tích Chu vi Hiển thị

Hình 1.1. Mô tả lớp HINH\_CN

### 1.2.3. Trừu tượng hóa dữ liệu và bao gói thông tin

*Trùu tượng hóa* là cách biểu diễn những đặc tính chính và bỏ qua những chi tiết nhỏ hoặc những giải thích. Khi xây dựng các lớp, ta phải sử dụng khái niệm trừu tượng hóa. Ví dụ ta có thể định nghĩa một lớp để mô tả các đối tượng trong không gian hình học bao gồm các thuộc tính trừu tượng như là kích thước, hình dáng, màu sắc và các phương thức xác định trên các thuộc tính này.

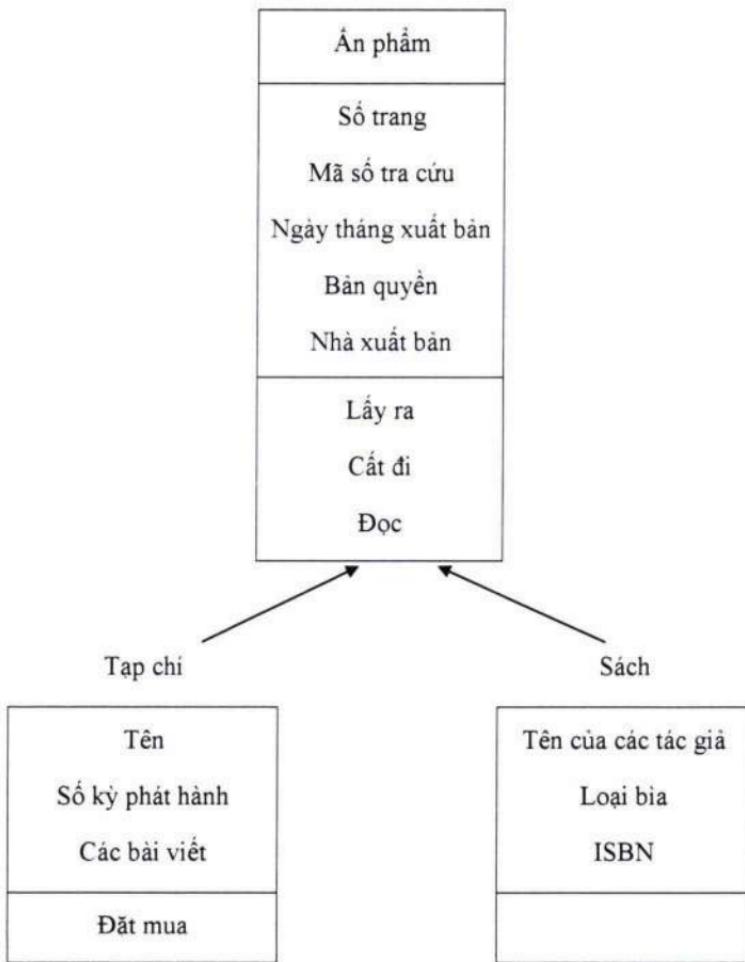
Việc đóng gói dữ liệu và các phương thức vào một đơn vị cấu trúc lớp được xem như một nguyên tắc *bao gói thông tin*. Dữ liệu được tổ chức sao cho thế giới bên ngoài (các đối tượng ở lớp khác) không truy nhập vào, mà chỉ cho

phép các phương thức trong cùng lớp hoặc trong những lớp có quan hệ kế thừa với nhau mới được quyền truy nhập. Chính các phương thức của lớp sẽ đóng vai trò như là giao diện giữa dữ liệu của đối tượng và phần còn lại của chương trình. Nguyên tắc bao gói dữ liệu để ngăn cấm sự truy nhập trực tiếp trong lập trình được gọi là sự che giấu thông tin.

#### 1.2.4. Kế thừa

*Kế thừa* là quá trình mà các đối tượng của lớp này được quyền sử dụng một số tính chất của các đối tượng của lớp khác. Sự kế thừa cho phép ta định nghĩa một lớp mới trên cơ sở các lớp đã tồn tại. Lớp mới này, ngoài những thành phần được kế thừa, sẽ có thêm những thuộc tính và các hàm mới. Nguyên lý kế thừa hỗ trợ cho việc tạo ra cấu trúc phân cấp các lớp.

Ví dụ: Xây dựng một tập các lớp mô tả cho thư viện các ấn phẩm. Có hai kiểu ấn phẩm: tạp chí và sách. Chúng ta tạo một ấn phẩm tổng quát bằng cách định nghĩa các thành phần dữ liệu tương ứng với số trang, mã số tra tra cứu, ngày tháng xuất bản, bản quyền và nhà xuất bản. Các ấn phẩm có thể được lấy ra, cắt đi và đọc. Đó là các phương thức thực hiện trên một ấn phẩm. Tiếp theo ta định nghĩa hai lớp dẫn xuất tên là tạp chí và sách. Tạp chí có tên, số kỳ phát hành và chứa nhiều bài của các tác giả khác nhau. Các thành phần dữ liệu tương ứng với các yếu tố này được đặt vào định nghĩa của lớp tạp chí. Tạp chí cũng cần có một phương thức nữa đó là đặt mua. Các thành phần dữ liệu xác định cho sách sẽ bao gồm tên của các tác giả, loại bìa và số hiệu ISBN của nó. Vậy có thể thấy, sách và tạp chí có chung các đặc trưng ấn phẩm, trong khi vẫn có các thuộc tính riêng của chúng:



Hình 1.2. Lớp *án phầm* và các lớp *dẫn xuất* của nó

Với tính kế thừa, chúng ta không phải mất công xây dựng lại từ đầu các lớp mới, chỉ cần bổ sung để có được trong các lớp dẫn xuất các đặc trưng cần thiết.

### 1.2.5. *Tương ứng bội*

*Tương ứng bội* là khả năng của một khái niệm (chẳng hạn các phép toán) có thể sử dụng với nhiều chức năng khác nhau. Ví dụ, phép + có thể biểu diễn cho phép “cộng” các số nguyên (int), số thực (float), số phức (complex) hoặc

xâu ký tự (string) v.v.. Hành vi của phép toán tương ứng bội phụ thuộc vào kiểu dữ liệu mà nó sử dụng để xử lý.

Tương ứng bội đóng vai quan trọng trong việc tạo ra các đối tượng có cấu trúc bên trong khác nhau nhưng cùng dùng chung một giao diện bên ngoài (như tên gọi).

Các bước tương ứng bội có thể tổng kết lại như sau:

1. Xây dựng lớp cơ sở trừu tượng bao gồm những thuộc tính chung nhất của các thực thể cần quản lý. Dựa vào các phương thức ảo hay thuần ảo dùng để xây dựng các nhóm phương thức ảo cho các lớp dẫn xuất sau này. Mỗi nhóm phương thức ảo sẽ thực hiện một chức năng nào đó trên các lớp.

2. Xây dựng các lớp dẫn xuất bắt đầu từ lớp cơ sở ảo. Số mức dẫn xuất là không hạn chế. Các lớp dẫn xuất sẽ mô tả các đối tượng cụ thể cần quản lý.

3. Xây dựng các phương thức ảo trong các dẫn xuất. Các phương thức này tạo thành các nhóm phương thức ảo trong sơ đồ các lớp có quan hệ thừa kế.

4. Xây dựng lớp quản lý các đối tượng. Dữ liệu của lớp này là một dãy con trỏ của lớp cơ sở trừu tượng ban đầu. Các con trỏ này có thể chứa địa chỉ đối tượng của các lớp dẫn xuất. Do vậy có thể dùng các con trỏ này để thực hiện các thao tác trên các đối tượng của bất kỳ lớp dẫn xuất nào.

#### *1.2.6. Liên kết động*

*Liên kết động* là dạng liên kết các thủ tục và hàm khi chương trình thực hiện lời gọi tới các hàm, thủ tục đó. Như vậy trong *liên kết động*, nội dung của đoạn chương trình ứng với thủ tục, hàm sẽ không được biết cho đến khi thực hiện lời gọi tới thủ tục, hàm đó.

#### *1.2.7. Truyền thông báo*

Các đối tượng gửi và nhận thông tin với nhau giống như con người trao đổi với nhau. Chính nguyên lý trao đổi thông tin bằng cách truyền thông báo cho phép ta dễ dàng xây dựng được hệ thống mô phỏng gần hơn những hệ thống trong thế giới thực. *Truyền thông báo* cho một đối tượng là yêu cầu đối

tương thực hiện một việc gì đó. Cách ứng xử của đối tượng được mô tả bên trong lớp thông qua các phương thức.

Trong chương trình, thông báo gửi đến cho một đối tượng chính là yêu cầu thực hiện một công việc cụ thể, nghĩa là sử dụng những hàm tương ứng để xử lý dữ liệu đã được khai báo trong đối tượng đó. Vì vậy, trong thông báo phải chỉ ra được hàm cần thực hiện trong đối tượng nhận thông báo. Thông báo truyền đi cũng phải xác định tên đối tượng và thông tin truyền đi.

Trong chương trình hướng đối tượng, mỗi đối tượng chỉ tồn tại trong thời gian nhất định. Đối tượng được tạo ra khi nó được khai báo và sẽ bị hủy bỏ khi chương trình ra khỏi miền xác định của đối tượng đó. Sự trao đổi thông tin chỉ có thể thực hiện trong thời gian đối tượng tồn tại.

### **1.3. Các bước cần thiết để thiết kế chương trình theo hướng đối tượng**

Chương trình theo hướng đối tượng bao gồm một tập các đối tượng và mối quan hệ giữa các đối tượng với nhau. Vì vậy, lập trình trong ngôn ngữ hướng đối tượng bao gồm các bước sau:

1. Xác định các dạng đối tượng (lớp) của bài toán (định dạng đối tượng).
2. Tìm kiếm các đặc tính chung (dữ liệu chung) trong các dạng đối tượng này, những gì chúng cùng nhau chia sẻ.
3. Xác định được lớp cơ sở dựa trên cơ sở các đặc tính chung của các dạng đối tượng.
4. Từ lớp cơ sở, sử dụng quan hệ tổng quát hóa để đặc tả trong việc đưa các lớp dẫn xuất chứa các thành phần, những đặc tính không chung còn lại của các dạng đối tượng. Ngoài ra, ta còn đưa ra các lớp có quan hệ với các lớp cơ sở và lớp dẫn xuất. Các quan hệ này có thể là quan hệ kết hợp, quan hệ tập hợp lại, quan hệ phụ thuộc.

### **1.4. Các ưu điểm của lập trình hướng đối tượng**

Cách tiếp cận hướng đối tượng giải quyết được nhiều vấn đề tồn tại trong quá trình phát triển phần mềm và tạo ra được những sản phẩm phần mềm có chất lượng cao. Những ưu điểm chính của LTHĐT là:

1. Thông qua nguyên lý kế thừa, có thể loại bỏ được những đoạn chương trình lặp lại trong quá trình mô tả các lớp và mở rộng khả năng sử dụng các lớp đã được xây dựng.

2. Chương trình được xây dựng từ những đơn thể (đối tượng) trao đổi với nhau nên việc thiết kế và lập trình sẽ được thực hiện theo quy trình nhất định chứ không phải dựa vào kinh nghiệm và kỹ thuật như trước. Điều này đảm bảo rút ngắn được thời gian xây dựng hệ thống và tăng năng suất lao động.

3. Nguyên lý che giấu thông tin giúp người lập trình tạo ra được những chương trình an toàn không bị thay bởi những đoạn chương trình khác.

4. Có thể xây dựng được ánh xạ các đối tượng của bài toán vào đối tượng của chương trình.

5. Cách tiếp cận thiết kế đặt trọng tâm vào đối tượng, giúp chúng ta xây dựng được mô hình chi tiết và gần với dạng cài đặt hơn.

6. Những hệ thống hướng đối tượng dễ mở rộng, nâng cấp thành những hệ lớn hơn.

7. Kỹ thuật truyền thông báo trong việc trao đổi thông tin giữa các đối tượng giúp cho việc mô tả giao diện với các hệ thống bên ngoài trở nên đơn giản hơn.

8. Có thể quản lý được độ phức tạp của những sản phẩm phần mềm.

Không phải trong hệ thống hướng đối tượng nào cũng có tất cả các tính chất nêu trên. Khả năng có các tính chất đó còn phụ thuộc vào lĩnh vực ứng dụng của dự án tin học và vào phương pháp thực hiện của người phát triển phần mềm.

### **1.5. Các ngôn ngữ hướng đối tượng**

Lập trình hướng đối tượng không là đặc quyền của một ngôn ngữ nào đặc biệt. Cũng giống như lập trình có cấu trúc, những khái niệm trong lập trình hướng đối tượng có thể cài đặt trong những ngôn ngữ lập trình như C hoặc Pascal,... Tuy nhiên, đối với những chương trình lớn thi vẫn đề lập trình sẽ trở

nên phức tạp. Những ngôn ngữ được thiết kế đặc biệt, hỗ trợ cho việc mô tả, cài đặt các khái niệm của phương pháp hướng đối tượng được gọi chung là ngôn ngữ đối tượng. Dựa vào khả năng đáp ứng các khái niệm về hướng đối tượng, ta có thể chia ra làm hai loại:

1. Ngôn ngữ lập trình dựa trên đối tượng

2. Ngôn ngữ lập trình hướng đối tượng

Lập trình dựa trên đối tượng là kiểu lập trình hỗ trợ chính cho việc bao gói, che giấu thông tin và định danh các đối tượng. Lập trình dựa trên đối tượng có những đặc tính sau:

- Bao gói dữ liệu
- Cơ chế che giấu và truy nhập dữ liệu
- Tự động tạo lập và xóa bỏ các đối tượng
- Phép toán tài bội

Ngôn ngữ hỗ trợ cho kiểu lập trình trên được gọi là ngôn ngữ lập trình dựa trên đối tượng. Ngôn ngữ trong lớp này không hỗ trợ cho việc thực hiện kế thừa và liên kết động, chẳng hạn Ada là ngôn ngữ lập trình dựa trên đối tượng.

Lập trình hướng đối tượng là kiểu lập trình dựa trên đối tượng và bổ sung thêm nhiều cấu trúc để cài đặt những quan hệ về kế thừa và liên kết động. Vì vậy đặc tính của LTHDT có thể viết một cách ngắn gọn như sau:

Các đặc tính dựa trên đối tượng + kế thừa + liên kết động.

Ngôn ngữ hỗ trợ cho những đặc tính trên được gọi là ngôn ngữ LTHDT, ví dụ như C++, Smalltalk, Object Pascal v.v..

Việc chọn một ngôn ngữ để cài đặt phần mềm phụ thuộc nhiều vào các đặc tính và yêu cầu của bài toán ứng dụng, vào khả năng sử dụng lại của những chương trình đã có và vào tổ chức của nhóm tham gia xây dựng phần mềm.

## **1.6. Một số ứng dụng của LTHĐT**

LTHĐT đang được ứng dụng để phát triển phần mềm trong nhiều lĩnh vực khác nhau. Trong số đó, có ứng dụng quan trọng và nổi tiếng nhất hiện nay là hệ điều hành Windows của hãng Microsoft đã được phát triển dựa trên kỹ thuật LTHĐT. Một số những lĩnh vực ứng dụng chính của kỹ thuật LTHĐT bao gồm:

- + Những hệ thống làm việc theo thời gian thực.
- + Trong lĩnh vực mô hình hóa hoặc mô phỏng các quá trình.
- + Các cơ sở dữ liệu hướng đối tượng.
- + Những hệ siêu văn bản, multimedia
- + Lĩnh vực trí tuệ nhân tạo và các hệ chuyên gia.
- + Lập trình song song và mạng nơ-ron.
- + Những hệ tự động hóa văn phòng và trợ giúp quyết định.

## Chương 2

### CÁC MỞ RỘNG CỦA NGÔN NGỮ C++

*Chương 2 trình bày những vấn đề sau đây:*

- *Giới thiệu chung về ngôn ngữ C++*
- *Một số mở rộng của ngôn ngữ C++ so với ngôn ngữ C*
- *Các đặc tính của C++ hỗ trợ lập trình hướng đối tượng*
- *Vào ra trong C++*
- *Cấp phát và giải phóng bộ nhớ*
- *Biến tham chiếu, hằng tham chiếu*
- *Truyền tham số cho hàm theo tham chiếu*
- *Hàm trả về giá trị tham chiếu*
- *Hàm với tham số có giá trị mặc định*
- *Các hàm nội tuyến (inline)*
- *Hàm tái bội*

#### **2.1. Giới thiệu chung về C++**

C++ là ngôn ngữ lập trình hướng đối tượng và là sự mở rộng của ngôn ngữ C. Vì vậy mọi khái niệm trong C đều dùng được trong C++. Phần lớn các chương trình C đều có thể chạy được trong C++. Trong chương này chỉ tập trung giới thiệu những khái niệm, đặc tính mới của C++ hỗ trợ cho lập trình hướng đối tượng. Một số kiến thức có trong C++ nhưng đã có trong ngôn ngữ C sẽ không được trình bày lại ở đây.

## 2.2. Một số mở rộng của C++ so với C

### 2.2.1. Đặt lời chú thích

Ngoài kiểu chú thích trong C bằng `/* ... */`, C++ đưa thêm một kiểu chú thích thứ hai, đó là chú thích bắt đầu bằng `//`. Kiểu chú thích `/*...*/` được dùng cho các khối chú thích lớn gồm nhiều dòng, còn kiểu `//` được dùng cho các chú thích trên một dòng. Chương trình dịch sẽ bỏ qua tất cả các chú thích trong chương trình.

Ví dụ: /\* Đây là

câu chú thích trên nhiều dòng \*/

// Đây là chú thích trên một dòng

### 2.2.2. Khai báo biến

Trong C tất cả các câu lệnh khai báo biến, mảng cục bộ phải đặt tại đầu khối. Vì vậy vị trí khai báo và vị trí sử dụng của biến có thể ở cách khá xa nhau, điều này gây khó khăn trong việc kiểm soát chương trình. C++ đã khắc phục nhược điểm này bằng cách cho phép các lệnh khai báo biến có thể đặt bất kỳ chỗ nào trong chương trình trước khi các biến được sử dụng. Phạm vi hoạt động của các biến kiểu này là khỏi trong đó biến được khai báo.

**Ví dụ 2.1.** Chương trình sau đây nhập một dãy số thực rồi sắp xếp theo thứ tự tăng dần:

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

void main()
{
    int n;
    printf("\n So phan tu cua day N=");
    scanf("%d", &n);
```

```

float *x=(float*)malloc((n+1)*sizeof(float));
for (int i=0;i<n;i++)
{
    printf("\n X[%d]=",i);
    scanf("%f",x+i);
}
for(i=0;i<n-1;++i)
for (int j=i+1;j<n;++j)
if (*x[i]>x[j])
{
    float tg=x[i];
    x[i]=x[j];
    x[j]=tg;
}
printf("\n Day sau khi sap xep\n");
for (i=0;i<n;++i)
printf("%0.2f ",x[i]);
getch();
}

```

### 2.2.3. Phép chuyển kiểu bắt buộc

Ngoài phép chuyển kiểu bắt buộc được viết trong C theo cú pháp:

(kiểu) biến thức

C++ còn sử dụng một phép chuyển kiểu mới như sau:

Kiểu (biến thức)

Phép chuyển kiểu này có dạng như một hàm số chuyển kiểu đang được gọi. Cách chuyển kiểu này thường được sử dụng trong thực tế.

### Ví dụ 2.2. Chương trình sau đây tính tổng

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

Với n là một số nguyên dương nhập từ bàn phím.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int n;
    printf("\n So phan tu cua day N=");
    scanf("%d", &n);
    float s=0.0;
    for (int i=1;i<=n;++i)
        s+= float(1)/float(i); //chuyen kieu theo C++
    printf("S=%0.2f", s);
    getch();
}
```

### 2.2.4. Lấy địa chỉ các phần tử mảng thực 2 chiều

Trong C không cho phép dùng phép toán & để lấy địa chỉ của các phần tử mảng thực 2 chiều. Vì vậy khi nhập một ma trận thực (dùng hàm scanf()) ta phải nhập qua một biến trung gian sau đó mới gán cho các phần tử mảng.

C++ cho phép dùng phép toán & để lấy địa chỉ các phần tử mảng thực 2 chiều, do đó thê dùng hàm scanf() để nhập trực tiếp vào các phần tử mảng.

**Ví dụ 2.3.** Chương trình sau đây cho phép nhập một mảng thực cấp 20x20 và tìm các phần tử có giá trị lớn nhất.

```

#include <conio.h>
#include <stdio.h>
void main()
{
    float a[20][20],smax;
    int m,n,i,j,imax,jmax;
    clrscr();
    puts(" Cho biet so hang va so cot cua ma tran: ");
    scanf("%d%d",&m,&n);
    for (i=0;i<m;++i)
        for (j=0;j<n;++j)
        {
            printf("\n a[%d][%d]=",i,j);
            scanf("%f",&a[i][j]);
        }
    smax=a[0][0];
    imax=0;
    jmax=0;
    for (i=0;i<m;++i)
        for(j=0;j<n;++j)
            if(smax<a[i][j])
            {
                smax=a[i][j];
                imax=i;
                jmax=j;
            }
}

```

```

    puts("\n\n Ma tran");

    for (i=0;i<m;++i)

        for (j=0;j<n;++j)

    {

        if (j==0) puts("");

        printf("%6.1f",a[i][j]);

    }

    puts("\n\n Phan tu max:");

    printf("\n Co gia tri=%6.1f", smax);

    printf("\n\n Tai hang %d cot %d",imax,jmax);

    getch();

}

```

### **2.3. Vào ra trong C++**

Để xuất dữ liệu ra màn hình và nhập dữ liệu từ bàn phím, trong C++ vẫn có thể dùng hàm printf() và scanf(), ngoài ra trong C++ ta có thể dùng dòng xuất/nhập chuẩn để nhập/xuất dữ liệu thông qua hai biến đối tượng của dòng (stream object) là cout và cin.

#### *2.3.1. Xuất dữ liệu*

Cú pháp:           cout << biểu thức 1 << . . . << biểu thức N;

Trong đó cout được định nghĩa trước như một đối tượng biểu diễn cho thiết bị xuất chuẩn của C++ là màn hình, cout được sử dụng kết hợp với toán tử chèn << để hiển thị giá trị các biểu thức 1, 2, ..., N ra màn hình.

#### *2.3.2. Nhập dữ liệu*

Cú pháp:           cin >> biến 1 >> . . . >> biến N;

Toán tử *cin* được định nghĩa trước như một đối tượng biêu diễn cho thiết bị vào chuẩn của C++ là bàn phím, *cin* được sử dụng kết hợp với toán tử trích >> để nhập dữ liệu từ bàn phím cho các biến 1, 2, ..., N.

### Chú ý:

- Để nhập một chuỗi không quá n ký tự và lưu vào mảng một chiều a (kiểu char) có thể dùng hàm *cin.get* như sau: *cin.get(a,n)*;
- Toán tử nhập *cin>>* sẽ để lại ký tự chuyển dòng '\n' trong bộ đệm. Ký tự này có thể làm trôi phương thức *cin.get*. Để khắc phục tình trạng trên cần dùng phương thức *cin.ignore(1)* để bỏ qua một ký tự chuyển dòng.
- Để sử dụng các loại toán tử và phương thức nói trên cần khai báo tập tin dẫn hướng *iostream.h*

#### 2.3.3. Định dạng khi in ra màn hình

- Để quy định số thực được hiển thị ra màn hình với p chữ số sau dấu chấm thập phân, ta sử dụng đồng thời các hàm sau:

```
setiosflags(ios::showpoint); // Bật cờ hiệu showpoint(p)  
setprecision(p);
```

Các hàm này cần đặt trong toán tử xuất như sau:

```
cout<<setiosflag(ios::showpoint)<<setprecision(p);
```

Câu lệnh trên sẽ có hiệu lực đối với tất cả các toán tử xuất tiếp theo cho đến khi gặp một câu lệnh định dạng mới.

- Để quy định độ rộng tối thiểu để hiển thị là k vị trí cho giá trị (nguyên, thực, chuỗi) ta dùng hàm: *setw(k)*

Hàm này cần đặt trong toán tử xuất và nó chỉ có hiệu lực cho một giá trị được in gần nhất. Các giá trị in ra tiếp theo sẽ có độ rộng tối thiểu mặc định là 0, như vậy câu lệnh:

```
cout<<setw(6)<<"Truong "<<"DHHV"
```

sẽ in ra chuỗi "Truong DHHV".

**Ví dụ 2.4.** Chương trình sau cho phép nhập một danh sách không quá 100 thí sinh. Dữ liệu mỗi thí sinh gồm họ tên, các điểm thi môn 1, môn 2, môn 3. Sau đó in danh sách thí sinh theo thứ tự giảm dần của tổng điểm.

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void main()
{
    struct
    {
        char ht[25];
        float d1,d2,d3,td;
    }ts[100],tg;
    int n,i,j;
    clrscr();
    cout << "So thi sinh:";
    cin >> n;
    for (i=0;i<n;++i)
    {
        cout << "\n Thi sinh:"<<i;
        cout << "\n Ho ten:";
        cin.ignore(1);
        cin.get(ts[i].ht,25);
        cout << "Diem cac mon thi :";
        cin>>ts[i].d1>>ts[i].d2>>ts[i].d3;
```

```

        ts[i].td=ts[i].d1+ts[i].d2+ts[i].d3;
    }

    for (i=0;i<n-1;++i)
        for(j=i+1;j<n;++j)
            if(ts[i].td<ts[j].td)
            {

                tg=ts[i];
                ts[i]=ts[j];
                ts[j]=tg;
            }

        cout<< "\n Danh sach thi sinh sau khi sap xep :";
        for (i=0;i<n;++i)
        {

            cout<< "\n" <<
setw(25)<<ts[i].ht<<setw(5)<<ts[i].td;
        }

        getch();
    }

```

**Ví dụ 2.5.** Chương trình sau đây cho phép nhập một mảng thực cấp 50x50. Sau đó in ma trận dưới dạng bảng và tìm một phần tử lớn nhất.

```

#include <iostream.h>
#include <iomanip.h>
#include <conio.h>

void main()
{
    float a[50][50],smax;
    int m,n,i,j,imax,jmax;

```

```

clrscr();

cout<< "Nhập số hàng và cột:";

cin>>m>>n;

for (i=0;i<m;++i)

    for (j=0;j<n;++j)

    {

        cout<< "a["<<i<<","<<j<<"]=";

        cin>> a[i][j];

    }

smax= a[0][0];

imax=0;

jmax=0;

for (i=0;i<m;++i)

    for (j=0;j<n;++j)

        if (smax<a[i][j])

    {

        smax=a[i][j];

        imax=i;

        jmax=j;

    }

cout << "\n\n Mang da nhap";

cout << setiosflags(ios::showpoint)<<setprecision(1);

for (i=0;i<m;++i)

    for (j=0;j<n;++j)

    {

```

```

    if (j==0) cout<<"\n";
    cout << setw(6)<<a[i][j];
}

cout << "\n\n" << "Phan tu max:" << "\n";
cout << "co gia tri =" << setw(6)<<smax;
cout << "\nTai hang" << imax << " cot " << jmax;
getch();
}

```

## 2.4. Cấp phát và giải phóng bộ nhớ

Trong C có thể sử dụng các hàm cấp phát bộ nhớ như malloc(), calloc() và hàm free() để giải phóng bộ nhớ được cấp phát. C++ đưa thêm một cách thức mới để thực hiện việc cấp phát và giải phóng bộ nhớ bằng cách dùng hai toán tử **new** và **delete**.

### 2.4.1. Toán tử new để cấp phát bộ nhớ

Toán tử new thay cho hàm malloc() và calloc() của C có cú pháp như sau:

```

new Tên kiểu ;
hoặc      new (Tên kiểu);

```

Trong đó Tên kiểu là kiểu dữ liệu của biến con trỏ, nó có thể là: các kiểu dữ liệu chuẩn như int, float, double, char,... hoặc các kiểu do người lập trình định nghĩa như mảng, cấu trúc, lớp,...

**Chú ý:** Để cấp phát bộ nhớ cho mảng một chiều, dùng cú pháp như sau:

```
Biến con trỏ = new kiểu[n];
```

Trong đó n là số nguyên dương xác định số phần tử của mảng.

Ví dụ:

```
float *p = new float; //cấp phát bộ nhớ cho biến con trỏ p có kiểu int
```

```
int *a = new int[100]; //cấp phát bộ nhớ để lưu trữ mảng một chiều a  
gồm 100 phần tử
```

Khi sử dụng toán tử new để cấp phát bộ nhớ, nếu không đủ bộ nhớ để cấp phát, new sẽ trả lại giá trị NULL cho con trỏ. Đoạn chương trình sau minh họa cách kiểm tra lỗi cấp phát bộ nhớ:

```
double *p;  
  
int n;  
  
cout << "\n So phan tu : ";  
  
cin >> n;  
  
p = new double[n];  
  
if (p == NULL)  
{  
    cout << "Loi cap phat bo nho";  
    exit(0);  
}
```

#### 2.4.2. Toán tử delete

Toán tử delete thay cho hàm free() của C, nó có cú pháp như sau:

```
delete con trỏ;
```

**Ví dụ 2.6.** Chương trình sau minh họa cách dùng new để cấp phát bộ nhớ chứa n thí sinh. Mỗi thí sinh là một cấu trúc gồm các trường ht(họ tên), sobd(số báo danh), và td(tổng điểm). Chương trình sẽ nhập n, cấp phát bộ nhớ chứa n thí sinh, kiểm tra lỗi cấp phát bộ nhớ, nhập n thí sinh, sắp xếp thí sinh theo thứ tự giảm của tổng điểm, in danh sách thí sinh sau khi sắp xếp, giải phóng bộ nhớ đã cấp phát.

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

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

struct TS
{
    char ht[20];
    long sobd;
    float td;
};

void main(void)
{
    TS *ts;
    int n;
    cout<<"\nSo thi sinh n = ";
    cin>>n;
    ts = new TS[n+1];
    if (ts == NULL)
    {
        cout << "\n Loi cap phat vung nho";
        getch();
        exit(0);
    }
    for (int i=0;i<n;++i)
    {
        cout << "\n Thi sinh thu "<<i;
        cout<< "\n Ho ten";
```

```

    cin.ignore(1);

    cin.get (ts[i].ht,20);

    cout << "so bao danh";

    cin>> ts[i].sobd;

    cout<< "tong diem:";

    cin>>ts[i].td;

}

for (i=0;i<n-1;++i)

    for (int j=i+1;j<n;++j)

        if (ts[i].td<ts[j].td)

        {

            TS tg=ts[i];

            ts[i]=ts[j];

            ts[j]=tg;

        }

        cout <<

setiosflags(ios::showpoint)<<setprecision(1);

        for (i=0;i<n;++i)

            cout << "\n" <<

setw(20)<<ts[i].ht<<setw(6)<<ts[i].td;

        delete ts;

        getch();

}

```

## 2.5. Biến tham chiếu

Trong C có 2 loại biến là: Biến giá trị dùng để chứa dữ liệu (nguyên, thực, ký tự,...) và biến con trỏ dùng để chứa địa chỉ. Các biến này đều được

cung cấp bộ nhớ và có địa chỉ. C++ cho phép sử dụng loại biến thứ ba là biến tham chiếu. Biến tham chiếu là một tên khác (bí danh) cho biến đã định nghĩa trước đó. Cú pháp khai báo biến tham chiếu như sau:

Kiểu & Biến tham chiếu = Biến;

Biến tham chiếu có đặc điểm là nó được dùng làm bí danh cho một biến (kiểu giá trị) nào đó và sử dụng vùng nhớ của biến này.

**Ví dụ:** Với câu lệnh: int a, &tong=a; thi tong là bí danh của biến a và biến tong dùng chung vùng nhớ của biến a. Lúc này, trong mọi câu lệnh, viết a hay viết tong đều có ý nghĩa như nhau, vì đều truy nhập đến cùng một vùng nhớ. Mọi sự thay đổi đối với biến tong đều ảnh hưởng đổi với biến a và ngược lại.

**Ví dụ:**

```
int a, &tong = a;  
tong =1;           //a=1  
cout<< tong;     //in ra số 1  
tong++;          //a=2  
++a;             //a=3  
cout<<tong;      //in ra số 3
```

### Chú ý:

- Trong khai báo biến tham chiếu phải chỉ rõ tham chiếu đến biến nào.
- Biến tham chiếu có thể tham chiếu đến một phần tử mảng, nhưng không cho phép khai báo mảng tham chiếu.
- Biến tham chiếu có thể tham chiếu đến một hàng. Khi đó nó sử dụng vùng nhớ của hàng và có thể làm thay đổi giá trị chứa trong vùng nhớ này.
- Biến tham chiếu thường được sử dụng làm đối của hàm để cho phép hàm truy nhập đến các tham biến trong lời gọi hàm

### 2.6. Hàng tham chiếu

Cú pháp khai báo hàng tham chiếu như sau:

```
const Kiểu dữ liệu &Biến = Biến/Hàng,
```

### Ví dụ:

```
int n = 10;  
const int &m = n;  
const int &p = 123;
```

Hằng tham chiếu có thể tham chiếu đến một biến hoặc một hằng.

### Chú ý:

➤ Biến tham chiếu và hằng tham chiếu khác nhau ở chỗ: không cho phép dùng hằng tham chiếu để làm thay đổi giá trị của vùng nhớ mà nó tham chiếu.

### Ví dụ: int y=12, z;

```
const int &p = y //Hằng tham chiếu p tham chiếu đến biến y
```

```
p = p + 1; //Sai, trình biên dịch sẽ thông báo lỗi
```

➤ Hằng tham chiếu cho phép sử dụng giá trị chứa trong một vùng nhớ, nhưng không cho phép thay đổi giá trị này.

➤ Hằng tham chiếu thường được sử dụng làm tham số của hàm để cho phép sử dụng giá trị của các tham số trong lời gọi hàm, nhưng tránh làm thay đổi giá trị tham số.

## 2.7. Truyền tham số cho hàm theo tham chiếu

Trong C chỉ có một cách truyền dữ liệu cho hàm là truyền theo giá trị. Chương trình sẽ tạo ra các bản sao của các tham số thực sự trong lời gọi hàm và sẽ thao tác trên các bản sao này chứ không xử lý trực tiếp với các tham số thực sự. Cơ chế này rất tốt nếu khi thực hiện hàm trong chương trình không cần làm thay đổi giá trị của biến gốc. Tuy nhiên, nhiều khi ta lại muốn những tham số đó thay đổi khi thực hiện hàm trong chương trình. C++ cung cấp thêm cách truyền dữ liệu cho hàm theo tham chiếu bằng cách dùng đối là tham chiếu. Cách làm này có ưu điểm là không cần tạo ra các bản sao của các tham số, do đó tiết kiệm bộ nhớ và thời gian chạy máy. Mặt khác, hàm này sẽ thao

tác trực tiếp trên vùng nhớ của các tham số, do đó dễ dàng thay đổi giá trị các tham số khi cần.

**Ví dụ 2.7.** Chương trình sau sẽ nhập dãy số thực, sắp xếp dãy theo thứ tự tăng dần và hiển thị ra màn hình.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void nhapds(double *a,int n)
{
    for(int i=0;i<n;++i)
    {
        cout<<"\n Phan tu thu "<<i<<":";
        cin>>a[i];
    }
}
void hv(double &x,double &y)
{
    double tam=x;x=y;y=tam;
}
void sapxep(double *a,int n)
{
    for(int i=0;i<n-1;++i)
        for(int j=i+1;j<n;++j)
            if(a[i]>a[j])
                hv(a[i],a[j]);
```

```

}

void main()
{
double x[100];
int i,n;
clrscr();
cout<<"\n nhap so phan tu N = ";
cin>>n;
nhapds(x,n);
sapxep(x,n);
cout<<"\nCac phan tu mang sau khi sap xep :";
for(i=0;i<n;++i)
printf("\n%6.2f",x[i]);
getch();
}

```

**Ví dụ 2.8.** Chương trình sẽ nhập dữ liệu một danh sách thí sinh bao gồm họ tên, điểm các môn 1, môn 2, môn 3 và in danh sách thí sinh:

```

#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <iomanip.h>

struct TS
{
    char ht[20];
    float d1,d2,d3,td;

```

```

};

void ints(const TS &ts)
{
    cout<<setiosflags(ios::showpoint)<<setprecision(1);
    cout<<"\n ho ten"<<setw(20)<<ts.ht<<setw(6)<<ts.td;
}

void nhapsl(TS *ts,int n)
{
    for(int i=0;i<n;++i)
    {
        cout<<"\n Thi sinh"<<i;
        cout<<"\n ho ten  ";
        cin.ignore(1);
        cin.get(ts[i].ht,25);
        cout<<"Nhap diem cac mon thi : ";
        cin>>ts[i].d1>>ts[i].d2>>ts[i].d3;
        ts[i].td=ts[i].d1+ts[i].d2+ts[i].d3;
    }
}

void hvts(TS &ts1,TS &ts2)
{
    TS tg=ts1;
    ts1=ts2;
}

```

```

    ts2=tg;
}

void sapxep(TS *ts,int n)
{
    for(int i=0;i<n-1;++i)
        for(int j=i+1;j<n;++j)
            if(ts[i].td<ts[j].td)
                hvts(ts[i],ts[j]) ;

}

void main()
{
    TS ts[100];
    int n,i;
    clrscr();
    cout<<"So thi sinh : ";
    cin>>n;
    nhapsl(ts,n);
    sapxep(ts,n);
    float dc;
    cout<<"\n\nDanh sach thi sinh \n";
    for(i=0;i<n;++i)
        if(ts[i].td>=dc)
            ints(ts[i]);
}

```

```
    else  
        break;  
  
    getch();  
}
```

**Ví dụ 2.9.** Chương trình sau sẽ nhập một mảng thực kích thước 20x20, in mảng đã nhập và in các phần tử lớn nhất và nhỏ nhất trên mỗi hàng của mảng.

```
#include <iostream.h>  
  
#include <conio.h>  
  
#include <stdio.h>  
  
#include <iomanip.h>  
  
void nhapmt(float a[20][20],int m,int n)  
{  
    for(int i=0;i<m;++i)  
        for(int j=0;j<n;++j)  
        {  
            cout<<"\n a["<<i<<","<<j<<"]=";  
            cin>> a[i][j];  
        }  
}  
  
void inmt(float a[20][20],int m,int n)  
{  
    cout<<setiosflags(ios::showpoint)<<setprecision(1);  
    cout<<"\nMang da nhap : ";  
    for(int i=0;i<m;++i)  
        for(int j=0;j<n;++j)
```

```

    {
        if(j==0) cout<<"\n";
        cout<<setw(6)<<a[i][j];
    }
}

void maxminds(float *x,int n,int &vtmax,int &vtmin)
{
    vtmax=vtmin=0;
    for(int i=1;i<n;++i)
    {
        if(x[i]>x[vtmax]) vtmax=i;
        if(x[i]<x[vtmin]) vtmin=i;
    }
}

void main()
{
    float a[20][20];
    int m,n;
    clrscr();
    cout<<"\n Nhap so hang va so cot : ";
    cin>>m>>n;
    nhapmt(a,m,n);
    inmt(a,m,n);
    float *p=(float*) a;
    int vtmax,vtmin;
}

```

```

    for(int i=0;i<m;++i)
    {
        p=((float*)a)+i*20;
        maxminds(p,n,vtmax,vtmin);
        printf("\n Hang %d phan tu max=%6.1f tai cot %d",
        i, p[vtmax], vtmax);
        printf("\n Phan tu min=%6.1f tai cot %d", p[vtmin],
        vtmin);
    }
    getch();
}

```

## 2.8. Hàm trả về giá trị tham chiếu

C++ cho phép hàm trả về giá trị là một tham chiếu, lúc này định nghĩa của hàm có dạng như sau :

Kiểu &Tên hàm(...)

```

{
    //thân hàm
    return <biến phạm vi toàn cục>;
}
```

Trong trường hợp này biểu thức được trả lại trong câu lệnh *return* phải là tên của một biến xác định từ bên ngoài ham, bởi vì khi đó mới có thể sử dụng được giá trị của hàm. Khi ta trả về một tham chiếu đến một biến cục bộ khai báo bên trong hàm, biến cục bộ này sẽ bị mất đi khi kết thúc thực hiện hàm. Do vậy tham chiếu của hàm sẽ không còn ý nghĩa nữa.

Khi giá trị trả về của hàm là tham chiếu, ta có thể gắp các câu lệnh gán hơi khác thường, trong đó về trái là một lời gọi hàm chứ không phải là tên của một biến. Điều này hoàn toàn hợp lý, bởi vì bản thân hàm đó có giá trị trả về là một tham chiếu. Nói cách khác, về trái của lệnh gán có thể là lời gọi đến một hàm có giá trị trả về là một tham chiếu. Xem các ví dụ sau:

### **Ví dụ 2.10.**

```
#include <iostream.h>
#include <conio.h>
int z;
int &f()// ham tra ve mot bi danh cua bien toan bo z
{
    return z;
}
void main()
{
    f()=50;//z=50
    cout<<"\nz="<<z;
    getch();
}
```

### **Ví dụ 2.11.**

```
#include <iostreams.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
int & max(int& a, int& b);
void main()
{
    clrscr();
    int b =10, a= 7, c= 20;
    cout << "Max a,b : "<<max(b,a) << endl;
```

```

    max(b, a)++;

    cout << "Gia tri b va a :" << b << " " << a << endl;

    max(b, c) = 5;

    cout << "Gia tri b va a va c :" << b << " " << a <<
" " << c << endl;

}

int &max(int &a, int &b)

{
    return a > b ? a : b;
}

```

Kết quả trên màn hình sẽ là:

Max a,b : 10

Gia tri cua b va a : 11 7

Gia tri cua b va a va c : 11 7 5

## 2.9. Hàm với tham số có giá trị mặc định

C++ cho phép xây dựng hàm với các tham số được khởi gán giá trị mặc định. Quy tắc xây dựng hàm với tham số mặc định như sau:

- Các đối có giá trị mặc định cần là các tham số cuối cùng tinh từ trái qua phải.
- Nếu chương trình sử dụng khai báo nguyên mẫu hàm thì các tham số mặc định cần được khởi gán trong nguyên mẫu hàm, không được khởi gán lại cho các đối măc định trong dòng đầu của định nghĩa hàm.

```

void f(int a, float x, char *st="TRUNG TAM", int b=1,
double y = 1.234);

```

```
void f(int a, float x, char *st="TRUNG TAM", int b=1,  
double y = 1.234)  
{  
    //Các câu lệnh  
}
```

- Khi xây dựng hàm, nếu không khai báo nguyên mẫu, thì các đối măc định được khởi gán trong dòng đầu của định nghĩa hàm, ví dụ:

```
void f(int a, float x, char *st="TRUNG TAM", int b=1,  
double y = 1.234)  
{  
    //Các câu lệnh  
}
```

**Chú ý:** Đối với các hàm có tham số mặc định thì lời gọi hàm cần viết theo quy định: Các tham số vắng mặt trong lời gọi hàm tương ứng với các tham số mặc định cuối cùng (tinh từ trái sang phải), ví dụ với hàm:

```
void f(int a, float x, char *st="TRUNG TAM", int b=1, double y = 1.234);
```

thì các lời gọi hàm đúng:

```
f(3,3.4,"TIN HOC",10,1.0); //Đầy đủ tham số  
f(3,3.4,"ABC"); //Thiếu 2 tham số cuối  
f(3,3.4); //Thiếu 3 tham số cuối
```

Các lời gọi hàm sai:

```
f(3);  
f(3,3.4, ,10);
```

### Ví dụ 2.12.

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

```

void ht(char *dc="TRUNG TAM",int n=5);
void ht(char *dc,int n)
{
    for(int i=0;i<n;++i)
        cout<<"\n" <<dc;
}
void main()
{
    ht(); // in dong chu "TRUNG TAM" tren 5 dong
    ht("ABC",3); // in dong chu "ABC" tren 3 dong
    ht("DEF"); // in dong chu "DEF" tren 5 dong
    getch();
}

```

## 2.10. Các hàm nội tuyến (inline)

Việc tổ chức chương trình thành các hàm có ưu điểm chương trình được chia thành các đơn vị độc lập, điều này giảm được kích thước chương trình, vì mỗi đoạn chương trình thực hiện nhiệm vụ của hàm được thay bằng lời gọi hàm. Tuy nhiên hàm cũng có nhược điểm là làm chậm tốc độ thực hiện chương trình vì phải thực hiện một số thao tác có tính thủ tục mỗi khi gọi hàm như: cấp phát vùng nhớ cho các tham số và biến cục bộ, truyền dữ liệu của các tham số cho các đối, giải phóng vùng nhớ trước khi thoát khỏi hàm.

C++ cho khả năng khắc phục được nhược điểm nói trên bằng cách dùng hàm nội tuyến. Để biến một hàm thành hàm nội tuyến ta viết thêm từ khóa **inline** vào trước khai báo nguyên mẫu hàm.

**Chú ý:** Trong mọi trường hợp, từ khóa inline phải xuất hiện trước các lời gọi hàm thì trình biên dịch mới biết cần xử lý hàm theo kiểu inline.

Ví dụ hàm f() trong chương trình sau sẽ không phải là hàm nội tuyến vì inline viết sau lời gọi hàm.

### Ví dụ 2.13.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int s ;
    s=f(5,6);
    cout<<s;
    getch();
}
inline int f(int a,int b)
{
    return a*b;
}
```

#### Chú ý:

➤ Chương trình dịch các hàm inline như tương tự như các macro, nghĩa là nó sẽ thay đổi lời gọi hàm bằng một đoạn chương trình thực hiện nhiệm vụ hàm. Cách làm này sẽ tăng tốc độ chương trình do không phải thực hiện các thao tác có tính thủ tục khi gọi hàm nhưng lại làm tăng khói lượng bộ nhớ chương trình (nhất là đối với các hàm nội tuyến có nhiều câu lệnh). Vì vậy chỉ nên dùng hàm inline đối với các hàm có nội dung đơn giản.

➤ Không phải khi gặp từ khoá inline là chương trình dịch nhất thiết phải xử lý hàm theo kiểu nội tuyến. Từ khoá inline chỉ là một từ khoá gợi ý cho chương trình dịch chứ không phải là một mệnh lệnh bắt buộc.

**Ví dụ 2.14.** Chương trình sau sử dụng hàm inline để tính chu vi và diện tích hình chữ nhật.

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

inline void dtcvhcn(int a,int b,int &dt,int &cv)
{
    dt=a*b;
    cv=2*(a+b);
}

void main()
{
    int a[20],b[20],cv[20],dt[20],n;
    cout<<"\n So hinh chu nhat";
    cin>>n;
    for(int i=0;i<n;++i)
    {
        cout<<"\n Nhap 2 canh cua hinh chu nhat"<<i<<":";
        cin>>a[i]>>b[i];
        dtcvhcn(a[i],b[i],dt[i],cv[i]);
    }
    clrscr();
    for(i=0;i<n;++i)
    {
        cout<<"\n Hinhanh chu nhat thu "<<i+1<<":";
        cout<<"\n Do dai hai canh "<<a[i]<<"va"<<b[i];
```

```

cout<<"\n dien tich "<<dt[i];
cout<<"\n chu vi "<<cv[i];
}

getch();
}

```

**Ví dụ 2.15.** Một cách viết khác của chương trình trong ví dụ 2.14

```

#include <iostream.h>
#include <conio.h>

inline void dtcvhcn(int a,int b,int &dt,int &cv);

void main()
{
    int a[20],b[20],cv[20],dt[20],n;
    cout<<"\n So hinh chu nhat";
    cin>>n;
    for(int i=0;i<n;++i)
    {
        cout<<"\n Nhap 2 canh cua hinh chu nhat"<<i<<" :";
        cin>>a[i]>>b[i];
        dtcvhcn(a[i],b[i],dt[i],cv[i]);
    }
    clrscr();
    for(i=0;i<n;++i)
    {
        cout<<"\n Hinh chu nhat thu "<<i+1<<" :";
        cout<<"\n Do dai hai canh "<<a[i]<<"va"<<b[i];
    }
}

```

```

cout<<"\n dien tich "<<dt[i];
cout<<"\n chu vi "<<cv[i];
}

getch();
}

void dtcvhcn(int a,int b,int &dt ,int &cv)
{
    dt=a*b;
    cv=2*(a+b);
}

```

## 2.11. Hàm tái bội

Các hàm tái bội là các hàm có cùng một tên và có tập đối khác nhau (về số lượng các đối hoặc kiểu). Khi gặp lời gọi các hàm tái bội thì trình biên dịch sẽ căn cứ vào số lượng và kiểu các tham số để gọi hàm có đúng tên và đúng các tham số tương ứng.

**Ví dụ 2.16.** Chương trình tìm max của một dãy số nguyên và max của một dãy số thực. Trong chương trình có 6 hàm: hai hàm dùng để nhập dãy số nguyên và dãy số thực có tên chung là nhapds, bốn hàm: tính max 2 số nguyên, tính max 2 số thực, tính max của dãy số nguyên, tính max của dãy số thực được đặt chung một tên là max.

```

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
void nhapds(int *x,int n);
void nhapds(double *x,int n);
int max(int x,int y);

```

```
double max(double x,double y);
void nhapds(int *x,int n)
{
    for(int i=0;i<n;++i)
    {
        cout<<"Phan tu "<<i<<" = ";
        cin>>x[i];
    }
}
void nhapds(double *x,int n)
{
    for (int i=0;i<n;i++)
    {
        cout<<"Phan tu "<<i<<" = ";
        cin>>x[i];
    }
}
int max(int x,int y)
{
    return x>y?x:y;
}
double max(double x,double y)
{
    return x>y?x:y;
}
```

```
int max(int *x,int n)
{
    int s=x[0];
    for(int i=1;i<n;++i)
        s=max(s,x[i]);
    return s;
}

double max(double *x,int n)
{
    double s=x[0];
    for(int i=1;i<n;++i)
        s=max(s,x[i]);
    return s;
}

void main()
{
    int a[20],n,ni,nd,maxi;
    double x[20],maxd;
    clrscr();
    cout<<"\n So phan tu nguyen n: ";
    cin>>ni;
    cout<<"\n Nhap day so nguyen: ";
    nhapds(a,ni);
    cout<<"\n So phan tu so thuc: ";
```

```

    cin>>nd;

    cout<<"\n Nhap day so thuc: ";
    nhapds(x,nd);

    maxi=max(a,ni);

    maxd=max(x,nd);

    cout<<"\n Max day so nguyen ="<<maxi;
    cout<<"\n Max day so thuc="<<maxd;
    getch();
}

```

**Chú ý:** Nếu hai hàm trùng tên và trùng đối thì trình biên dịch không thể phân biệt được. Ngay cả khi hai hàm này có cùng kiểu khác nhau thì trình biên dịch vẫn báo lỗi. Ví dụ sau xây dựng hai hàm cùng có tên là f và cùng một đối nguyên a, nhưng kiểu hàm khác nhau. Hàm thứ nhất có kiểu nguyên (trả về  $a^*a$ ), hàm thứ hai có kiểu void. Chương trình sau sẽ bị thông báo lỗi khi biên dịch.

### Ví dụ 2.17.

```

#include <iostream.h>
#include <conio.h>
int f(int a);
void f(int a);
int f(int a)
{
    return a*a;
}
void f(int a)
{
    cout<<"\n"<<a;
}

```

```

void main()
{
    int b = f(5);
    f(b);
    getch();
}

```

### Bài tập

1. Viết chương trình nhập vào số nguyên dương h ( $2 < h < 23$ ), sau đó in ra các tam giác có chiều cao là h như các hình sau:

A vertical column of asterisks forming an inverted triangle. The pattern is as follows:  
Row 1: \*  
Row 2: \* \* \*  
Row 3: \* \* \* \* \*  
Row 4: \* \* \* \* \* \* \*  
Row 5: \* \* \* \* \* \* \* \*  
A vertical double-headed arrow to the right of the pattern is labeled 'h'.

A vertical column of asterisks forming an inverted triangle. The pattern is as follows:  
Row 1: \*  
Row 2: \* \*  
Row 3: \* \* \*  
Row 4: \* \* \* \*  
Row 5: \* \* \* \* \*  
Row 6: \* \* \* \* \* \*  
Row 7: \* \* \* \* \* \* \*  
Row 8: \* \* \* \* \* \* \* \*  
A vertical double-headed arrow to the right of the pattern is labeled 'h'.

2. Viết chương trình thực hiện các yêu cầu sau đây:

- Nhập dữ liệu cho các sinh viên (dùng cấu trúc danh sách liên kết đơn), các thông tin của sinh viên bao gồm: mã sinh viên, họ tên, lớp, điểm trung bình.

- Chương trình có sử dụng toán tử new và delete.

- In ra danh sách sinh viên có sắp xếp vị thứ theo điểm trung bình.

3. Viết chương trình để sắp xếp một mảng thực hai chiều theo thứ tự tăng dần, trong chương trình có có sử dụng toán tử new và delete.

4. Viết các hàm tái bội để tính diện tích tam giác, diện tích hình chữ nhật, diện tích hình tròn.

5. Viết chương trình nhân hai ma trận  $A_{m \times n}$  và  $B_{n \times p}$ , mỗi ma trận được cấp phát động và các giá trị của chúng phát sinh ngẫu nhiên.

6. Viết chương trình hiển thị thông tin về sinh viên: id, điểm thi ba môn Toán, Ngoại ngữ, Giáo dục thể chất, tổng điểm và điểm trung bình ba môn. Với các yêu cầu sau:

- Khai báo lớp **sinhvien**

- Khai báo và định nghĩa hàm get() để lấy thông tin sinh viên.

- Khai báo lớp gdthechat

- Khai báo và định nghĩa hàm nhapdiemgdtc() để đọc điểm thể chất.

- Tạo lớp thongtinsv được kế thừa từ hai lớp là sinhvien và gdthechat

- Khai báo và định nghĩa hàm display() để tìm tổng điểm và điểm trung bình

- Khai báo đối tượng obj của lớp kế thừa, gọi hàm nhapdiemkiemtra(), nhapdiemgdtc() và display().

7. Viết chương trình C++ để hiển thị chi tiết bảng lương của nhân viên. Chương trình của bạn sẽ nhắc người dùng nhập số thứ tự nhân viên, id, tên, chức danh, lương cơ bản, phụ cấp thu hút, phụ cấp chức vụ, phụ cấp trách nhiệm, sau đó tính tổng lương và hiển thị kết quả. Với các yêu cầu sau:

- Khai báo một lớp cơ sở **nhanvien**, định nghĩa hàm get() để lấy chi tiết về nhân viên.

- Khai báo lớp kế thừa **luong**, sau đó khai báo hàm get1() để lấy chi tiết về lương nhân viên.

- Kế tiếp, định nghĩa hàm calculate() để tìm tổng lương và định nghĩa hàm display().

- Tạo đối tượng của lớp kế thừa, đọc số nhân viên.

- Sau đó gọi hàm get(), get1() và calculate() cho mỗi nhân viên và cuối cùng gọi hàm display().

8. Viết chương trình sử dụng hàm inline để trả về giá trị lớn nhất của hai số nguyên nhập vào từ bàn phím.

9. Viết một chương trình sử dụng hàm inline để nhập hai giá trị và tính tích và lập phương của chúng. Với các yêu cầu sau:

- Khai báo lớp

- Khai báo và định nghĩa hàm inline cho các hàm để tính tích và lập phương

- Khai báo đối tượng và các biến.

- Đọc hai giá trị

- Gọi hàm tính tích và lập phương bởi sử dụng các đối tượng của lớp.

- Trả về giá trị và hiển thị

## Chương 3

### LỚP

*Chương này trình bày những vấn đề sau đây:*

- *Định nghĩa lớp*
- *Tạo lập đối tượng*
- *Truy nhập đến các thành phần của lớp*
- *Con trỏ đối tượng*
- *Con trỏ this*
- *Hàm bạn*
- *Dữ liệu thành phần tĩnh, hàm thành phần tĩnh*
- *Hàm tạo, hàm hủy*
- *Hàm tạo sao chép*

Lớp là khái niệm trung tâm của lập trình hướng đối tượng, nó là sự mở rộng của các khái niệm cấu trúc (struct) của C. Ngoài các thành phần dữ liệu, lớp còn chứa các thành phần hàm, còn gọi là phương thức (method) hoặc hàm thành viên (member function). Lớp có thể xem như một kiểu dữ liệu các biến, mảng đối tượng. Từ một lớp đã định nghĩa, có thể tạo ra nhiều đối tượng khác nhau, mỗi đối tượng có vùng nhớ riêng.

Chương này sẽ trình bày cách định nghĩa lớp, cách xây dựng phương thức, giải thích về phạm vi truy nhập, sử dụng các thành phần của lớp, cách khai báo biến, mảng cấu trúc, lời gọi tới các phương thức.

### 3.1. Định nghĩa lớp

Cú pháp: Lớp được định nghĩa theo mẫu :

```
class tên_lớp  
{  
    private: [Khai báo các thuộc tính]  
            [Định nghĩa các hàm thành phần (phương thức)]  
    public : [Khai báo các thuộc tính]  
            [Định nghĩa các hàm thành phần (phương thức)]  
};
```

Thuộc tính của lớp được gọi là dữ liệu thành phần và hàm được gọi là phương thức hoặc hàm thành viên. Thuộc tính và hàm được gọi chung là các thành phần của lớp. Các thành phần của lớp được tổ chức thành hai vùng: vùng sở hữu riêng (private) và vùng dùng chung (public) để quy định phạm vi sử dụng của các thành phần. Nếu không quy định cụ thể (không dùng các từ khóa private và public) thì C++ hiểu đó là private. Các thành phần private chỉ được sử dụng bên trong lớp (trong thân của các hàm thành phần). Các thành phần public được phép sử dụng ở cả bên trong và bên ngoài lớp. Các hàm không phải là hàm thành phần của lớp thì không được phép sử dụng các thành phần này.

**Khai báo các thuộc tính của lớp:** được thực hiện y như việc khai báo biến. Thuộc tính của lớp không thể có kiểu chính của lớp đó, nhưng có thể là kiểu con trỏ của lớp này,

**Ví dụ:**

```
class A  
{  
    A *x; //Không cho phép, vì x có kiểu lớp A
```

```
A *p ; // Cho phép, vì p là con trỏ kiểu lớp A
```

```
} ;
```

**Định nghĩa các hàm thành phần:** Các hàm thành phần có thể được xây dựng bên ngoài hoặc bên trong định nghĩa lớp. Thông thường, các hàm thành phần đơn giản, có ít dòng lệnh sẽ được viết bên trong định nghĩa lớp, còn các hàm thành phần dài thì viết bên ngoài định nghĩa lớp. Các hàm thành phần viết bên trong định nghĩa lớp được viết như hàm thông thường. Khi định nghĩa hàm thành phần ở bên ngoài lớp, ta dùng cú pháp sau đây:

```
Kiểu_trả_về_của_hàm Tên_lớp::Tên_hàm(khai báo các tham số)
{
    [nội dung hàm]
}
```

Toán tử :: được gọi là toán tử phân giải miền xác định, được dùng để chỉ ra lớp mà hàm đó thuộc vào.

Trong thân hàm thành phần, có thể sử dụng các thuộc tính của lớp, các hàm thành phần khác và các hàm tự do trong chương trình.

### **Chú ý :**

- Các thành phần dữ liệu khai báo là private nhằm bảo đảm nguyên lý che giấu thông tin, bảo vệ an toàn dữ liệu của lớp, không cho phép các hàm bên ngoài xâm nhập vào dữ liệu của lớp.
- Các hàm thành phần khai báo là public có thể được gọi tới từ các hàm thành phần public khác trong chương trình.

### **3.2. Tạo lập đối tượng**

Sau khi định nghĩa lớp, ta có thể khai báo các biến thuộc kiểu lớp. Các biến này được gọi là các đối tượng. Cú pháp khai báo biến đối tượng như sau:

```
Tên_lớp Danh_sách_biến ;
```

Đối tượng cũng có thể khai báo khi định nghĩa lớp theo cú pháp sau:

```
class tên_lớp  
{  
    ...  
} <Danh_sách_biện>;
```

Mỗi đối tượng sau khi khai báo sẽ được cấp phát một vùng nhớ riêng để chứa các thuộc tính của chúng. Không có vùng nhớ riêng để chứa các hàm thành phần cho mỗi đối tượng. Các hàm thành phần sẽ được sử dụng chung cho tất cả các đối tượng cùng lớp.

### 3.3. Truy nhập tới các thành phần của lớp

- Để truy nhập đến dữ liệu thành phần của lớp, ta dùng cú pháp:

Tên\_đối\_tượng.Tên\_thuộc\_tính

Cần chú ý rằng dữ liệu thành phần riêng chỉ có thể được truy nhập bởi những hàm thành phần của cùng một lớp, đối tượng của lớp cũng không thể truy nhập.

- Để sử dụng các hàm thành phần của lớp, ta dùng cú pháp:

Tên\_đối\_tượng.Tên\_hàm( Các\_khai\_báo\_tham\_số\_thực\_sự)

#### Ví dụ 3.1.

```
#include <conio.h>  
  
#include <iostream.h>  
  
class DIEM  
{  
private :  
    int x,y ;  
public :
```

```

void nhapsl( )
{
    cout << "\n Nhap hoanh do va tung do cua diem:";

    cin >>x>>y ;

}

void hienthi( )
{ cout<<"\n x = " <<x<<" y = "<<y<<endl; }

} ;

void main()
{ clrscr();

    DIEM d1;

    d1.nhapsl();

    d1.hienthi();

    getch();

}

```

### Ví dụ 3.2.

```

#include <conio.h>

#include <iostream.h>

class A

{ int m,n;

public :

void nhap( )

{
    cout << "\n Nhap hai so nguyen : " ;

    cin>>m>>n ;

```

```

    }

int max()
{
    if(m>n)
        return m;
    else
        return n;
}

void hienthi( )
{
    cout<<"\n Thanh phan du lieu lon nhat x
= "
<<max()<<endl;
}

void main ()
{
    clrscr();
    A ob;
    ob.nhap();
    ob.hienthi();
    getch();
}

```

**Chú ý:** Các hàm tự do có thể có các đối là đối tượng nhưng trong thân hàm không thể truy nhập đến các thuộc tính của lớp. Ví dụ giả sử đã định nghĩa lớp:

```

class DIEM
{
private :
    double x,y ; // toa do cua diem
public :

```

```

void nhapsl()
{
    cout << " Toa do x,y : " ;
    cin >> x>>y ;
}

void in()
{
    cout << "x =""<<x<<"y="<<y ;
}

} ;

```

Dùng lớp DIEM, ta xây dựng hàm tự do tính độ dài của đoạn thẳng đi qua hai điểm như sau:

```

double do_dai ( DIEM d1, DIEM d2 )
{
    return sqrt(pow(d1.x-d2.x,2) + pow(d1.y-d2.y,2));
}

```

Chương trình đích sẽ báo lỗi đối với hàm này. Bởi vì trong thân hàm không cho phép sử dụng các thuộc tính d1.x,d2.x,d1.y của các đối tượng d1 và d2 thuộc lớp DIEM.

**Ví dụ 3.3.** Ví dụ sau minh họa việc sử dụng hàm thành phần với tham số mặc định:

```

#include <iostream.h>
#include <conio.h>
class Box
{

```

```

private:
    int dai;
    int rong;
    int cao;
public:
    int get_thetich(int lth,int wdh = 2,int ht = 3);
};

int Box::get_thetich(int l, int w, int h)
{
    dai = l;
    rong = w;
    cao = h;
    cout<< dai<<'t'<< rong<<'t'<<cao<<'t';
    return dai * rong * cao;
}

void main()
{
    Box ob;
    int x = 10, y = 12, z = 15;
    cout <<"Dai Rong Cao Thetich\n";
    cout << ob.get_thetich(x, y, z) << "\n";
    cout << ob.get_thetich(x, y) << "\n";
    cout << ob.get_thetich(x) << "\n";
    cout << ob.get_thetich(x, 7) << "\n";
    cout << ob.get_thetich(5, 5, 5) << "\n";
    getch();
}

```

Kết quả chương trình như sau:

Dai Rong Cao Thetich

10 12 15 1800

10 12 3 360

10 2 3 60

10 7 3 210

5 5 5 125

**Ví dụ 3.4.** Ví dụ sau minh họa việc sử dụng hàm **inline** trong lớp:

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

#include <conio.h>

class phrase
{
private:
    char dongtu[10];
    char danhtu[10];
    char cumtu[25];
public:
    phrase();
    inline void set_danhtu(char* in_danhtu);
    inline void set_dongtu(char* in_dongtu);
    inline char* get_phrase(void);
};

void phrase::phrase()
{
    strcpy(danhtu,"");
    strcpy(dongtu,"");
    strcpy(cumtu,"");
}

inline void phrase::set_danhtu(char* in_danhtu)
{
    strcpy(danhtu, in_danhtu);
}

inline void phrase::set_dongtu(char* in_dongtu)
{
    strcpy(dongtu, in_dongtu);
}

inline char* phrase::get_phrase(void)
{
    strcpy(cumtu,dongtu);
```

```

    strcat(cumtu," the ");
    strcat(cumtu,danhtu);
    return cumtu;
}

void main()
{
    phrase text;
    cout << "Cum tu la : -> " << text.get_phrase()
        << "\n";
    text.set_danhtu("file");
    cout << "Cum tu la : -> " <<
        text.get_phrase()<<"\n";
    text.set_dongtu("Save");
    cout << "Cum tu la : -> " <<
        text.get_phrase()<<"\n";
    text.set_danhtu("program");
    cout << "Cum tu la : -> " <<
        text.get_phrase()<<"\n";
}

```

Kết quả chương trình như sau:

Cum tu la : -> the

Cum tu la : -> the file

Cum tu la : -> Save the file

Cum tu la : -> Save the program

**Ví dụ 3.5.** Ví dụ sau minh họa việc sử dụng từ khóa const trong lớp:

```

#include <iostream.h>
#include <conio.h>

class constants

```

```

{
private:
    int number;
public:
    void print_it(const int data_value);
};

void constants::print_it(const int data_value)
{
    number = data_value;
    cout << number << "\n";
}

void main()
{
    constants num;
    const int START = 3;
    const int STOP = 6;
    int index;
    for (index=START; index<=STOP; index++)
    {
        cout<< "index = " ;
        num.print_it(index);
        cout<< "START = " ;
        num.print_it(START);
    }
    getch();
}

```

Kết quả chương trình như sau:

```

index = 3
START = 3
index = 4
START = 3
index = 5
START = 3

```

index = 6

START = 3

### 3.4. Con trỏ đối tượng

Con trỏ đối tượng dùng để chứa địa chỉ của biến đối tượng, được khai báo như sau :

Tên\_lớp \* Tên\_con\_trỏ ;

**Ví dụ:** Dùng lớp DIEM, ta có thể khai báo:

DIEM \*p1, \*p2, \*p3 ; // Khai báo 3 con trỏ p1, p2, p3

DIEM d1, d2 ; //Khai báo hai đối tượng d1, d2

DIEM d [20] ; // Khai báo mảng đối tượng

Có thể thực hiện câu lệnh :

p1 = &d2 ; //p1 chứa địa chỉ của d2, p1 trỏ tới d2

p2 = d ; // p2 trỏ tới đâu mảng d

p3 = new DIEM //tạo một đối tượng và chứa địa chỉ của nó vào p3

Để truy xuất các thành phần của lớp từ con trỏ đối tượng, ta viết như sau:

Tên\_con\_trỏ -> Tên\_thuộc\_tính

Tên\_con\_trỎ -> Tên\_hàm(các tham số thực sự)

Nếu con trỏ chứa địa chỉ của mảng, có thể dùng con trỏ như tên mảng.

#### Ví dụ 3.6.

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

class mang
{
    int maso;
    float gia;
```

```

public: void getdata(int a, float b)
    {maso= a; gia= b;}
void show()
{ cout << "maso" << maso<< endl;
    cout << "gia" << gia<< endl;
}
};

const int k=5;

void main()
{ clrscr();
    mhang *p = new mhang[k];
    mhang *d = p;
    int x,i;
    float y;
    cout<<"\nNhap vao du lieu 5 mat hang :";
    for (i = 0; i <k; i++)
    { cout <<"\nNhap ma hang va don gia cho mat hang
thu " <<i+1;
        cin>>x>>y;
        p -> getdata(x,y);
        p++;
    }
    for (i = 0; i <k; i++)
    { cout <<"\nMat hang thu : " << i + 1 <<
        endl;
        d -> show();
        d++;
    }
}

```

```
    }  
  
    getch();  
}
```

### 3.5. Con trỏ this

Mỗi hàm thành phần của lớp có một tham số ẩn, đó là con trỏ **this**. Con trỏ this trỏ đến từng đối tượng cụ thể. Ta hãy xem lại hàm nhapsl() của lớp DIEM trong ví dụ ở trên:

```
void nhapsl( )  
{  
    cout << "\n Nhập hoành do và tung do của điểm : ";  
    cin >>x>>y;  
}
```

Trong hàm này ta sử dụng tên các thuộc tính x,y một cách đơn độc. Điều này thường như mâu thuẫn với quy tắc sử dụng thuộc tính. Tuy nhiên nó được lý giải như sau: C++ sử dụng một con trỏ đặc biệt trong các hàm thành phần. Các thuộc tính viết trong hàm thành phần được hiểu là thuộc một đối tượng do con trỏ **this** trỏ tới. Như vậy, hàm nhapsl() có thể viết một cách tường minh như sau:

```
void nhapsl()  
{  
    cout << "\n Nhập hoành do và tung do của điểm: ";  
    cin >>this->x>>this->y;  
}
```

Con trỏ **this** là đối thứ nhất của hàm thành phần. Khi một lời gọi hàm thành phần được phát ra bởi một đối tượng thì tham số truyền cho con trỏ **this** chính là địa chỉ của đối tượng đó.

**Ví dụ:** Xét một lời gọi tới hàm nhapsl() :

```
DIEM d1 ;
```

```
d1.nhapsl();
```

Trong trường hợp này của d1 thì `this = &d1`. Do đó `this -> x` chính là `d1.x` và `this-> y` chính là `d1.y`

**Chú ý:** Ngoài tham số đặc biệt `this` không xuất hiện một cách tường minh, hàm thành phần lớp có thể có các tham số khác được khai báo như trong các hàm thông thường.

### **Ví dụ 3.7.**

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

class time
{
    int h,m;
public :
    void nhap(int h1, int m1)
    {
        h= h1; m = m1;
    }
    void hienthi(void)
    {
        cout <<h << " gio "<<m << " phut"<<endl;
    }
    void tong(time, time);
};

void time::tong(time t1, time t2)
{
    m= t1.m+ t2.m; //this->m = t1.m+ t2.m;
    h= m/60;           //this->h = this->m/60;
    m= m%60;           //this->m = this->m%60;
    h = h+t1.h+t2.h;   //this->h = this->h + t1.h+t2.h;
```

```

}

void main()
{
    clrscr();

    time ob1, ob2, ob3;

    ob1.nhap(2, 45);

    ob2.nhap(5, 40);

    ob3.tong(ob1, ob2);

    cout << "object 1 = "; ob1.hienthi();

    cout << "object 2 = "; ob2. hienthi();

    cout << "object 3 = "; ob3. hienthi();

    getch();
}

```

Chương trình cho kết quả như sau :

object 1 = 2 gio 45 phut

object 2 = 5 gio 40 phut

object 3 = 8 gio 25 phut

### **3.6. Hàm bạn**

Trong thực tế thường xảy ra trường hợp có một số lớp cần sử dụng chung một hàm. C++ giải quyết vấn đề này bằng cách dùng hàm bạn. Để một hàm trở thành bạn của một lớp, có 2 cách viết:

**Cách 1:** Dùng từ khóa *friend* để khai báo hàm trong lớp và xây dựng hàm bên ngoài như các hàm thông thường (không dùng từ khóa friend). Mẫu viết như sau:

```

class A
{
    private :
        // Khai báo các thuộc tính
    public :
        ...
        // Khai báo các hàm bạn của lớp A
        friend void f1 (...);
        friend double f2 (...);
        ...
    } ;
// Xây dựng các hàm f1, f2, f3
void f1 (...)

{
    ...
}

double f2 (...)

{
    ...
}

```

**Cách 2:** Dùng từ khóa friend để xây dựng hàm trong định nghĩa lớp.  
Mẫu viết như sau :

```

class A
{
    private :

```

```

// Khai báo các thuộc tính
public :

...
// Khai báo các hàm bạn của lớp A
void f1 (...)

{
    ...
}

double f2 (...)

{
    ...
}

}
;

```

### **Hàm bạn có những tính chất sau:**

- Hàm bạn không phải là hàm thành phần của lớp.
- Việc truy nhập tới hàm bạn được thực hiện như hàm thông thường.
- Trong thân hàm bạn của một lớp có thể truy nhập tới các thuộc tính của đối tượng thuộc lớp này. Đây là sự khác nhau duy nhất giữa hàm bạn và hàm thông thường.
- Một hàm có thể là bạn của nhiều lớp. Lúc đó nó có quyền truy nhập tới tất cả các thuộc tính của các đối tượng trong các lớp này. Để làm cho hàm f trở thành bạn của các lớp A, B và C ta sử dụng mẫu viết sau :

class B ; //Khai báo trước lớp A

class B ; // Khai báo trước lớp B

```
class C ; // Khai báo trước lớp C  
// Định nghĩa lớp A  
class A  
{  
    // Khai báo f là bạn của A  
    friend void f(... )  
};  
// Định nghĩa lớp B  
class B  
{  
    // Khai báo f là bạn của B  
    friend void f(... )  
};  
// Định nghĩa lớp C  
class C  
{  
    // Khai báo f là bạn của C  
    friend void f(... )  
};  
// Xây dựng hàm f  
void f(... )  
{  
    ...  
};
```

### Ví dụ 3.8.

```
#include <iostream.h>
#include <conio.h>
class sophuc
{float a,b;
public :    sophuc() {}
            sophuc(float x, float y)
            {a=x; b=y;}
friend sophuc tong(sophuc,sophuc);
friend void hienthi(sophuc);
};

sophuc tong(sophuc c1,sophuc c2)
{sophuc c3;
c3.a=c1.a + c2.a ;
c3.b=c1.b + c2.b ;
return (c3);
}

void hienthi(sophuc c)
{cout<<c.a<<" + "<<c.b<<"i"<<endl; }

void main()
{ clrscr();
sophuc d1 (2.1,3.4);
sophuc d2 (1.2,2.3) ;
sophuc d3 ;
```

```

d3 = tong(d1,d2);

cout<<"d1= ";hienthi(d1);

cout<<"d2= ";hienthi(d2);

cout<<"d3= ";hienthi(d3);

getch();

}

```

Chương trình cho kết quả như sau :

d1= 2.1 + 3.4i

d2= 1.2 + 2.3i

d3= 3.3 + 5.7i

### Ví dụ 3.9.

```

#include <iostream.h>

#include <conio.h>

class LOP1;

class LOP2

{
    int v2;

public:
    void nhap(int a)
    { v2=a; }

    void hienthi(void)
    { cout<<v2<<"\n"; }

    friend void traodoi(LOP1 &, LOP2 &);

};

```

```
class LOP1
{
    int v1;
public:
    void nhap(int a)
    { v1=a; }

    void hienthi(void)
    { cout<<v1<<"\n"; }

    friend void traodoi(LOP1 &, LOP2 &);

};

void traodoi(LOP1 &x, LOP2 &y)
{
    int t = x.v1;
    x.v1 = y.v2;
    y.v2 = t;
}

void main()
{
    clrscr();
    LOP1 ob1;
    LOP2 ob2;
    ob1.nhap(150);
    ob2.nhap(200);
    cout << "Gia tri ban dau :" << "\n";
    ob1.hienthi();
```

```
    ob2.hienthi();

    traodoi(ob1, ob2); //Thuc hien hoan doi

    cout << "Gia tri sau khi thay doi:" << "\n";

    ob1.hienthi();

    ob2.hienthi();

    getch();

}
```

Chương trình cho kết quả như sau:

Gia tri ban dau:

150

200

Gia tri sau khi thay doi:

200

150

### **3.7. Dữ liệu thành phần tĩnh và hàm thành phần tĩnh**

#### *3.7.1. Dữ liệu thành phần tĩnh*

Dữ liệu thành phần tĩnh được khai báo bằng từ khoá static và được cấp phát một vùng nhớ cố định, nó tồn tại ngay cả khi lớp chưa có một đối tượng nào cả. Dữ liệu thành phần tĩnh là chung cho cả lớp, nó không phải là riêng của mỗi đối tượng, ví dụ:

```
class A

{

private:

    static int ts; // Thành phần tĩnh

    int x;

    ...

}
```

```
};  
A u, v; // Khai báo 2 đối tượng
```

Giữa các thành phần x và ts có sự khác nhau như sau: u.x và v.x có 2 vùng nhớ khác nhau, trong khi u.ts và v.ts chỉ là một, chúng cùng biểu thị một vùng nhớ, thành phần ts tồn tại ngay khi u và v chưa khai báo.

Để biểu thị thành phần tĩnh, ta có thể dùng tên lớp, ví dụ: A::ts

**Khai báo và khởi gán giá trị cho thành phần tĩnh:** Thành phần tĩnh sẽ được cấp phát bộ nhớ và khởi gán giá trị đầu bằng một câu lệnh khai báo đặt sau định nghĩa lớp theo mẫu như sau:

```
int A::ts; // Khởi gán cho ts giá trị 0
```

```
int A::ts = 1234; // Khởi gán cho ts giá trị 1234
```

**Chú ý:** Khi chưa khai báo thì thành phần tĩnh chưa tồn tại. Hãy xem chương trình sau:

### Ví dụ 3.10.

```
#include <conio.h>  
#include <iostream.h>  
  
class HDBH  
{  
  
private:  
    char *tenhang;  
    double tienban;  
    static int tshd;  
    static double tstienban;  
  
public:  
    static void in()  
    {
```

```

cout <<"\n" << tshd;
cout <<"\n" << tstienban;
}
};

void main ()
{
HDBH::in();
getch();
}

```

Các thành phần tĩnh tshd và tstienban chưa khai báo, nên chưa tồn tại. Vì vậy các câu lệnh in giá trị các thành phần này trong hàm in() là không thể được. Khi dịch chương trình, sẽ nhận được các thông báo lỗi. Có thể sửa chương trình trên bằng cách đưa vào các lệnh khai báo các thành phần tĩnh tshd và tstienban như sau:

### **Ví dụ 3.11.**

```

#include <conio.h>
#include <iostream.h>

class HDBH
{
private:
    int shd;
    char *tenhang;
    double tienban;
    static int tshd;
    static double tstienban;
public:

```

```

static void in()
{
    cout <<"\n" <<tshd;
    cout <<"\n" <<tstienban;
}
};

int HDBH::tshd=5
double HDBH::tstienban=20000.0;
void main()
{
    HDBH::in();
    getch();
}

```

### *3.7.2. Hàm thành phần tĩnh*

Hàm thành phần tĩnh được viết theo một trong hai cách:

- Dùng từ khoá static đặt trước định nghĩa hàm thành phần viết bên trong định nghĩa lớp.
- Nếu hàm thành phần xây dựng bên ngoài định nghĩa lớp, thì dùng từ khoá static đặt trước khai báo hàm thành phần bên trong định nghĩa lớp. Không cho phép dùng từ khoá static đặt trước định nghĩa hàm thành phần viết bên ngoài định nghĩa lớp.

#### *Các đặc tính của hàm thành phần tĩnh:*

- Hàm thành phần tĩnh là chung cho toàn bộ lớp và không lệ thuộc vào một đối tượng cụ thể, nó tồn tại ngay khi lớp chưa có đối tượng nào.
- Lời gọi hàm thành phần tĩnh như sau:

Tên lớp: Tên hàm thành phần tĩnh(các tham số thực sự)

- Vì hàm thành phần tĩnh là độc lập với các đối tượng, nên không thể dùng hàm thành phần tĩnh để xử lý dữ liệu của các đối tượng trong lời gọi phương thức tĩnh. Nói cách khác không cho phép truy nhập các thuộc tính (trừ thuộc tính tĩnh) trong thân hàm thành phần tĩnh. Đoạn chương trình sau minh họa điều này:

```
class HDBH
{
private:
    int shd;
    char *tenhang;
    double tienban;
    static int tshd;
    static double tstienban;

public:
static void in()
{
    cout << "\n" << tshd;
    cout << "\n" << tstienban;
    cout << "\n" << tenhang           //loi
    cout << "\n" << tienban;   //loi
}
};
```

### Ví dụ 3.12.

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

```
class A
{
    int m;

    static int n; //n la bien tinh

    public: void set_m(void) { m= ++n; }

    void show_m(void)
    {
        cout << "\n Doi tuong thu:" << m << endl;
    }

    static void show_n(void)
    {
        cout << " m = " << n << endl;
    }
};

int A::n=1; //khai gan gia tri ban dau 1 cho bien
tinh n

void main()
{
    clrscr();
    A t1, t2;
    t1.set_m();
    t2.set_m();
    A::show_n();
    A t3;
    t3.set_m();
```

```
A::show_n();  
t1.show_m();  
t2.show_m();  
t3.show_m();  
getch();  
}
```

Kết quả chương trình trên là:

m = 3

m = 4

Doi tuong thu : 2

Doi tuong thu : 3

Doi tuong thu : 4

### 3.8. Hàm tạo (constructor)

Hàm tạo là một hàm thành phần đặc biệt của lớp làm nhiệm vụ tạo lập một đối tượng mới. Chương trình dịch sẽ cấp phát bộ nhớ cho đối tượng, sau đó sẽ gọi đến hàm tạo. Hàm tạo sẽ khởi gán giá trị cho các thuộc tính của đối tượng và có thể thực hiện một số công việc khác nhằm chuẩn bị cho đối tượng mới. Khi xây dựng hàm tạo cần lưu ý những đặc tính sau của hàm tạo:

- Tên hàm tạo trùng với tên của lớp.
- Hàm tạo không có kiểu trả về.
- Hàm tạo phải được khai báo trong vùng public.
- Hàm tạo có thể được xây dựng bên trong hoặc bên ngoài định nghĩa lớp.
- Hàm tạo có thể có tham số hoặc không có tham số.
- Trong một lớp có thể có nhiều hàm tạo (cùng tên nhưng khác các tham số).

### Ví dụ 3.13.

```
class DIEM

{
    private:
        int x,y;

public:
    DIEM()           //Ham tao khong tham so
    {
        x = y = 0;
    }

    DIEM(int x1, int y1) //Ham tao co tham so
    {
        x = x1; y=y1;
    }

    //Cac thanh phan khac
};


```

**Chú ý 1:** Nếu lớp không có hàm tạo, chương trình dịch sẽ cung cấp một hàm tạo mặc định không đổi, hàm này thực chất không làm gì cả. Như vậy một đối tượng tạo ra chỉ được cấp phát bộ nhớ, còn các thuộc tính của nó chưa được xác định.

### Ví dụ 3.14.

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

class DIEM
{
```

```

private:
    int x,y;
public:
    void in()
    {
        cout <<"\n" << y << " " << m;
    }
};

void main()
{
    DIEM d;
    d.in();
    DIEM *p;
    p= new DIEM [10];
    clrscr();
    d.in();
    for (int i=0;i<10;++i)
        (p+i)->in();
    getch();
}

```

### **Chú ý 2:**

- Khi một đối tượng được khai báo thì hàm tạo của lớp tương ứng sẽ tự động thực hiện và khởi gán giá trị cho các thuộc tính của đối tượng. Dựa vào các tham số trong khai báo mà chương trình dịch sẽ biết cần gọi đến hàm tạo nào.
- Khi khai báo mảng đối tượng không cho phép dùng các tham số để khởi gán cho các thuộc tính của các đối tượng mảng.

- Câu lệnh khai báo một biến đối tượng sẽ gọi tới hàm tạo một lần.
- Câu lệnh khai báo một mảng  $n$  đối tượng sẽ gọi tới hàm tạo mặc định  $n$  lần.
  - Với các hàm có các tham số kiểu lớp, thì chúng chỉ xem là các tham số hình thức, vì vậy khai báo tham số trong dòng đầu của hàm sẽ không tạo ra đối tượng mới và do đó không gọi tới các hàm tạo.

### Ví dụ 3.15.

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

class DIEM
{
private:
    int x,y;
public:
    DIEM()
    {
        x = y = 0;
    }
    DIEM(int x1, int y1)
    {
        x = x1; y = y1;
    }
    friend void in(DIEM d)
    {

```

```

    cout <<"\n" << d.x << " " << d.y;
}

void in()
{
    cout <<"\n" << x << " " << y ;
}

};

void main()
{
    DIEM d1;
    DIEM d2(2,3);
    DIEM *d;
    d = new DIEM (5,6);
    clrscr();
    in(d1); // Gọi ham ban in()
    d2.in(); // Gọi ham thanh phan in()
    in(*d); // Gọi ham ban in()
    DIEM(2,2).in(); // Gọi ham thanh phan in()
    DIEM t[3]; // 3 lan goi ham tao khong doi
    DIEM *q; // Gọi ham tao khong doi
    int n;
    cout << "\n N = ";
    cin >> n;
    q = new DIEM [n+1]; // n+1 lan goi ham tao khong doi
    for (int i=0;i<=n;++i)

```

```

q[i]=DIEM (3+i,4+i);//n+1 lan goi ham tao co doi
for (i=0;i<=n;++i)
    q[i].in();      // Gọi hàm thanh phan in()
for (i=0;i<=n;++i)
    DIEM(5+i,6+i).in(); //Gọi hàm thanh phan in()
getch();
}

```

**Chú ý 3:** Nếu trong lớp đã có ít nhất một hàm tạo, thì hàm tạo mặc định sẽ không được phát sinh nữa. Khi đó mọi câu lệnh xây dựng đối tượng mới đều sẽ gọi đến một hàm tạo của lớp. Nếu không tìm thấy hàm tạo cần gọi thì chương trình dịch sẽ báo lỗi. Điều này thường xảy ra khi chúng ta không xây dựng hàm tạo không đổi, nhưng lại sử dụng các khai báo không tham số như ví dụ sau:

### Ví dụ 3.16.

```

#include <conio.h>
#include <iostream.h>
class DIEM
{
private:
    int x,y;
public:
    DIEM(int x1, int y1)
    {
        x=x1; y=y1;
    }
}

```

```

void in()
{
    cout << "\n" << x << " " << y << " " << m;
}

};

void main()
{
    DIEM d1(200,200); // Gọi hàm tạo có doi
    DIEM d2; // Lỗi, gọi hàm tạo không doi
    d2= DIEM_DH (3,5); // Gọi hàm tạo có doi
    d1.in();
    d2.in();
    getch();
}

```

Trong ví dụ này, câu lệnh DIEM d2; trong hàm main() sẽ bị chương trình dịch báo lỗi. Bởi vì lệnh này sẽ gọi tới hàm tạo không đối, mà hàm tạo này chưa được xây dựng. Có thể khắc phục điều này bằng cách chọn một trong hai giải pháp sau:

- Xây dựng thêm hàm tạo không đối.

- Gán giá trị mặc định cho tất cả các đối x1, y1 của hàm tạo đã xây dựng ở trên.

Theo giải pháp thứ 2, chương trình trên có thể sửa lại như sau:

### **Ví dụ 3.17.**

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

```
class DIEM
{
private:
    int x,y;
public:
    DIEM(int x1=0, int y1=0)
    {
        x = x1; y = y1;
    }
    void in()
    {
        cout << "\n" << x << " " << y << " " << m;
    }
};

void main()
{
    DIEM d1(2,3); //Goi ham tao, khong dung tham so mac
dinh
    DIEM d2; //Goi ham tao, dung tham so mac dinh
    d2= DIEM(6,7); //Goi ham tao, khong dung tham so mac
dinh
    d1.in();
    d2.in();
    getch(); }
```

### Ví dụ 3.18.

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

class rectangle
{
private:
    int dai;
    int rong;
    static int extra_data;
public:
    rectangle();
    void set(int new_dai, int new_rong);
    int get_area();
    int get_extra();
};

int rectangle::extra_data;
rectangle::rectangle()
{
    dai = 8;
    rong = 8;
    extra_data = 1;
}
void rectangle::set(int new_dai,int new_rong)
{
    dai = new_dai;
    rong = new_rong;
}
int rectangle::get_area()
{
    return (dai * rong);
}
int rectangle::get_extra()
{
    return extra_data++; }
```

```

void main()
{
    rectangle small, medium, large;
    small.set(5, 7);
    large.set(15, 20);
    cout<<"Dien tich la : "<<small.get_area()<<"\n";
    cout<<"Dien tich la : "<<
        medium.get_area()<<"\n";
    cout<<"Dien tich la : "<<large.get_area()<<"\n";
    cout <<"Gia tri du lieu tinh la : "<<
        small.get_extra()<<"\n";
    cout <<"Gia tri du lieu tinh la : "<<
        medium.get_extra()<<"\n";
    cout <<"Gia tri du lieu tinh la : "<<
        large.get_extra()<<"\n";
}

```

### **Chương trình cho kết quả như sau :**

Dien tich la : 35  
 Dien tich la : 64  
 Dien tich la : 300  
 Gia tri du lieu tinh la : 1  
 Gia tri du lieu tinh la : 2  
 Gia tri du lieu tinh la : 3

### **3.9. Hàm tạo sao chép**

#### *3.9.1. Hàm tạo sao chép mặc định*

Giả sử đã định nghĩa một lớp ABC nào đó. Khi đó:

- Ta có thể dùng câu lệnh khai báo hoặc cấp phát bộ nhớ để tạo các đối tượng mới, ví dụ:

ABC p1, p2;

```
ABC *p = new ABC;
```

- Ta cũng có thể dùng lệnh khai báo để tạo một đối tượng mới từ một đối tượng đã tồn tại, ví dụ:

```
ABC u;
```

```
ABC v(u); // Tạo v theo u
```

Câu lệnh này có ý nghĩa như sau:

- Nếu trong lớp ABC chưa xây dựng hàm tạo sao chép, thì câu lệnh này sẽ gọi tới một hàm tạo sao chép mặc định của C++. Hàm này sẽ sao chép nội dung từng bit của u vào các bit tương ứng của v. Như vậy các vùng nhớ của u và v sẽ có nội dung như nhau.

- Nếu trong lớp ABC đã có hàm tạo sao chép thì câu lệnh:

PS v(u); sẽ tạo ra đối tượng mới v, sau đó gọi tới hàm tạo sao chép để khởi gán v theo u.

Ví dụ sau minh họa cách dùng hàm tạo sao chép mặc định:

Trong chương trình đưa vào lớp PS (phân số):

- + Các thuộc tính gồm: t (tử số) và m (mẫu).

- + Trong lớp mà không có phương thức nào cả mà chỉ có hai hàm bạn là các hàm toán tử nhập (>>) và xuất (<<).

- + Nội dung chương trình là: Dùng lệnh khai báo để tạo một đối tượng u (kiểu PS) có nội dung như đối tượng đã có d.

### **Ví dụ 3.19.**

```
#include <conio.h>
#include <iostream.h>
class PS
{
```

```

private: int t,m;
public:
friend ostream& operator<< (ostream& os,const PS &p)
{
    os<<" = "<<p.t<<"/"<<p.m;
    return os;
}
friend istream& operator>> (istream& is, PS &p)
{
    cout <<" Nhap tu va mau : ";
    is>>p.t>>p.m;
    return is;
}
};

void main()
{
    PS d;
    cout <<"\n Nhap phan so d "; cin>>d;
    cout<<"\n PS d "<<d;
    PS u(d);
    cout<<"\n PS u "<<u;
    getch();
}

```

### 3.9.2. Hàm tạo sao chép

Hàm tạo sao chép sử dụng một đối tượng tham chiếu đối tượng để khởi gán cho đối tượng mới và được viết theo mẫu sau:

Tên\_lớp (const Tên\_lớp &ob)

{

// Các câu lệnh dùng các thuộc tính của đối tượng ob để khởi gán

// cho các thuộc tính của đối tượng mới

}

**Ví dụ:**

```
class PS
{
    private: int t, m;
public:
    PS(const PS &p)
    {
        t= p.t;
        m= p.m;
    }
    ...
};
```

Hàm tạo sao chép trong ví dụ trên không khác gì hàm tạo sao chép mặc định.

**Chú ý:**

- Nếu lớp không có các thuộc tính kiểu con trỏ hoặc tham chiếu thì dùng hàm tạo sao chép mặc định là đủ.

- Nếu lớp có các thuộc tính con trỏ hoặc tham chiếu, thì hàm tạo sao chép mặc định chưa đáp ứng được yêu cầu.

```
class DT

{
    private:
        int n; // Bac da thuc
        double *a; // Tro toi vung nho
        chua cac he so da thuc a0, a1, ...
    public:
        DT()
    {
        n = 0; a = NULL;
    }
    DT(int n1)
    {
        n = n1;
        a = new double[n1+1];
    }
    friend ostream& operator<<
    (ostream& os, const DT &d);
    friend istream& operator>>
    (istream& is, DT &d);
    ...
};
```

Bây giờ chúng ta hãy theo dõi xem việc dùng hàm tạo mặc định trong đoạn chương trình sau sẽ dẫn đến sai lầm như thế nào:

```

DT d;
cin >> d;

/* Nhập đối tượng d gồm: nhập một số nguyên dương và gán cho d.n, cấp
phát vùng nhớ cho d.n, nhập các hệ số của đa thức và chứa vào vùng nhớ được
cấp phát

*/
DT u(d);

/* Dùng hàm tạo mặc định để xây dựng đối tượng u theo d. Kết quả: u.n
= d.n và u.a = d.a. Như vậy hai con trỏ u.a và d.a cùng trỏ đến một vùng nhớ.

*/

```

**Nhận xét:** Mục đích của ta là tạo ra một đối tượng u giống như d, nhưng độc lập với d. Nghĩa là khi d thay đổi thì u không bị ảnh hưởng gì. Thế nhưng mục tiêu này không đạt được, vì u và d có chung một vùng nhớ chứa hệ số của đa thức, nên khi sửa đổi các hệ số của đa thức trong d thì các hệ số của đa thức trong u cũng thay đổi theo. Còn một trường hợp nữa cũng dẫn đến lỗi là khi một trong hai đối tượng u và d bị giải phóng (thu hồi vùng nhớ chứa đa thức) thì đối tượng còn lại cũng sẽ không còn vùng nhớ nữa.

Ví dụ sau sẽ minh họa nhận xét trên: Khi d thay đổi thì u cũng thay đổi và ngược lại khi u thay đổi thì d cũng thay đổi theo.

### Ví dụ 3.20.

```

#include <conio.h>
#include <iostream.h>
#include <math.h>
class DT
{
private:
    int n; // Bac da thuc

```

```

double *a; // Tro toi vung nho chua cac he
           // so da thuc a[0],a[1],...
public:
    DT()
    {
        this->n=0; this->a=NULL;
    }
    DT(int n1)
    {
        this->n=n1;
        this->a= new double[n1+1];
    }
friend ostream& operator<< (ostream& os,const DT &d);
friend istream& operator>> (istream& is,DT &d);
};

ostream& operator<< (ostream& os,const DT &d)
{
    os <<" Cac he so ";
    for (int i=0; i<=d.n; ++i)
        os << d.a[i]<<" ";
    return os;
}
istream& operator>> (istream& is,DT &d)
{
    if (d.a != NULL) delete d.a;
}

```

```

cout << " \nBac da thuc:";

cin >>d.n;

d.a = new double[d.n+1];

cout<<"Nhap cac he so da thuc:\n";

for (int i=0 ; i<=d.n ; ++i)

{

cout<<"He so bac"<< i << "=";

is >> d.a[i];

}

return is;

}

void main()

{

DT d;

clrscr();

cout<<"\nNhap da thuc d" ; cin >> d;

DT u(d);

cout <<"\nDa thuc d" << d ;

cout <<"\nDa thuc u" << u ;

cout <<"\nNhap da thuc d" << d ; cin >> d;

cout <<"\nDa thuc d" << d ;

cout <<"\nDa thuc u" << u ;

cout <<"\nDa thuc u" << u ; cin >> u;

cout <<"\nDa thuc d" << d ;

cout <<"\nDa thuc u" << u ;

```

```
    getch();
```

```
}
```

**Ví dụ 3.21.** Ví dụ sau minh họa về hàm tạo sao chép:

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

class DT

{
private:
    int n; // Bac da thuc

    double *a; // Tro toi vung nho chua cac da
    thuc

    // a0, a1, ...

public:
    DT()
    {
        this->n=0; this->a=NULL;
    }

    DT(int n1)
    {
        this->n=n1;
        this->a= new double[n1+1];
    }

    DT(const DT &d);

friend ostream& operator<< (ostream& os,const DT &d);
```

```

friend istream& operator>> (istream& is,DT &d);
};

DT::DT(const DT &d)
{
    this->n = d.n;
    this->a = new double[d.n+1];
    for (int i=0;i<=d.n;++i)
        this->a[i] = d.a[i];
}

ostream& operator<< (ostream& os,const DT &d)
{
    os<<"-Cac he so (tu ao): ";
    for (int i=0 ; i<=d.n ; ++i)
        os << d.a[i] << " ";
    return os;
}

istream& operator>> (istream& is,DT &d)
{
    if (d.a != NULL) delete d.a;
    cout << "\n Bac da thuc:";

    cin >> d.n;
    d.a = new double[d.n+1];
    cout << "Nhap cac he so da thuc:\n";
    for (int i=0 ; i<= d.n ; ++i)
    {

```

```

cout << "He so bac " << i << "=";
is >> d.a[i];

}

return is;
}

void main()
{
    DT d;

    clrscr();

    cout << "\nNhập dữ liệu d " ; cin >> d;

    DT u(d);

    cout << "\nĐã nhập d " << d;
    cout << "\nĐã nhập u " << u;

    cout << "\nNhập dữ liệu d " ; cin >> d;
    cout << "\nĐã nhập d " << d;
    cout << "\nĐã nhập u " << u;
    cout << "\nNhập dữ liệu u " ; cin >> u;
    cout << "\nĐã nhập d " << d;
    cout << "\nĐã nhập u " << u;

    getch();
}

```

### **3.10. Hàm hủy (destructor)**

Hàm hủy là một hàm thành phần của lớp, có chức năng ngược với hàm tạo. Hàm hủy được gọi trước khi giải phóng một đối tượng để thực hiện một số công việc có tính “dọn dẹp” trước khi đối tượng được hủy bỏ, ví dụ giải

phóng một vùng nhớ mà đối tượng đang quản lý, xoá đối tượng khỏi màn hình nếu như nó đang hiển thị... Việc hủy bỏ đối tượng thường xảy ra trong 2 trường hợp sau:

- Trong các toán tử và hàm giải phóng bộ nhớ như delete, free...
- Giải phóng các biến, mảng cục bộ khi thoát khỏi hàm, phương thức.

Nếu trong lớp không định nghĩa hàm huỷ thì một hàm huỷ mặc định không làm gì cả được phát sinh. Đối với nhiều lớp thì hàm huỷ mặc định là đủ, không cần đưa vào một hàm huỷ mới. Trường hợp cần xây dựng hàm huỷ thì tuân theo quy tắc sau:

- Mỗi lớp chỉ có một hàm huỷ.
- Hàm huỷ không có kiểu, không có giá trị trả về và không có tham số.
- Tên hàm hàm huỷ có một dấu ngã ngay trước tên lớp.

### Ví dụ:

```
class A
{
private:
    int n;
    double *a;
public:
    ~A()
    {
        n = 0;
        delete a;
    }
};
```

### **Vi dụ 3.22.**

```
#include <iostream.h>
class Count{
private:
    static int counter;
    int obj_id;
public:
    Count();
    static void display_total();
    void display();
    ~Count();
};
int Count::counter;
Count::Count()
{
    counter++;
    obj_id = counter;
}
Count::~Count()
{
    counter--;
    cout<<"Doi tuong "<<obj_id<<" duoc huy bo\n";
}
void Count::display_total()
{
    cout <<"So cac doi tuong duoc tao ra la = "<< counter <<
endl;
}
void Count::display()
{
    cout << "Object ID la "<<obj_id<<endl;
}
void main()
{
    Count a1;
```

```
Count::display_total();
Count a2, a3;
Count::display_total();
a1.display();
a2.display();
a3.display();
}
```

**Kết quả chương trình như sau:**

So cac doi tuong duoc tao ra la = 1

So cac doi tuong duoc tao ra la = 3

Object ID la 1

Object ID la 2

Object ID la 3

Doi tuong 3 duoc huy bo

Doi tuong 2 duoc huy bo

Doi tuong 1 duoc huy bo

## Bài tập

1. Xây dựng lớp thời gian **Time**. Dữ liệu thành phần bao gồm giờ, phút giây. Các hàm thành phần bao gồm: hàm tạo, hàm truy cập dữ liệu, hàm normalize() để chuẩn hóa dữ liệu nằm trong khoảng quy định của giờ ( $0 \leq$  giờ  $< 24$ ), phút ( $0 \leq$  phút  $< 60$ ), giây ( $0 \leq$  giây  $< 60$ ), hàm advance(int h, int m, int s) để tăng thời gian hiện hành của đối tượng đang tồn tại, hàm reset(int h, int m, int s) để chỉnh lại thời gian hiện hành của một đối tượng đang tồn tại và một hàm print() để hiển thị dữ liệu.

2. Xây dựng lớp **Date**. Dữ liệu thành phần bao gồm ngày, tháng, năm. Các hàm thành phần bao gồm: hàm tạo, hàm truy cập dữ liệu, hàm normalize() để chuẩn hóa dữ liệu nằm trong khoảng quy định của ngày ( $1 \leq$  ngày  $<$  daysIn(tháng)), tháng ( $1 \leq$  tháng  $< 12$ ), năm ( $năm \geq 1$ ), hàm daysIn(int) trả về số ngày trong tháng, hàm advance(int y, int m, int d) để tăng ngày hiện lên các năm y, tháng m, ngày d của đối tượng đang tồn tại, hàm reset(int y, int m, int d) để đặt lại ngày cho một đối tượng đang tồn tại và một hàm print() để hiển thị dữ liệu.

3. Thực hiện một lớp **String**. Mỗi đối tượng của lớp sẽ đại diện một chuỗi ký tự. Những thành phần dữ liệu là chiều dài chuỗi, và chuỗi ký tự. Các hàm thành phần bao gồm: hàm tạo, hàm truy cập, hàm hiển thị, hàm character(int i) trả về một ký tự trong chuỗi được chỉ định bằng tham số i.

4. Xây dựng lớp ma trận có tên là **Matrix** cho các ma trận, các hàm thành phần bao gồm: hàm tạo mặc định, hàm nhập xuất ma trận, cộng, trừ, nhân hai ma trận.

5. Xây dựng lớp ma trận có tên là **Matrix** cho các ma trận vuông, các hàm thành phần bao gồm: hàm tạo mặc định, hàm nhập xuất ma trận, tính định thức và tính ma trận nghịch đảo.

6. Xây dựng lớp **Stack** cho ngăn xếp kiểu int. Các hàm thành phần bao gồm: Hàm tạo mặc định, hàm hủy, hàm isEmpty() kiểm tra stack có rỗng

không, hàm isFull() kiểm tra stack có đầy không, hàm push(), pop(), hàm in nội dung ngăn xếp. Sử dụng một mảng để thực hiện.

7. Xây dựng lớp hàng đợi **Queue** chứa phần tử kiểu int. Các hàm thành phần bao gồm: hàm tạo, hàm hủy và những toán tử hàng đợi thông thường: hàm insert() để thêm phần tử vào hàng đợi, hàm remove() để loại bỏ phần tử, hàm isEmpty() kiểm tra hàng đợi có rỗng không, hàm isFull() kiểm tra hàng đợi có đầy không. Sử dụng một mảng để thực hiện.

8. Xây dựng lớp **Sinhvien** để quản lý hộ tên sinh viên, năm sinh, điểm thi 9 môn học của các sinh viên. Cho biết sinh viên nào được làm khóa luận tốt nghiệp, bao nhiêu sinh viên thi tốt nghiệp, bao nhiêu sinh viên thi lại, tên môn thi lại, Tiêu chuẩn để xét như sau:

- Sinh viên làm khóa luận phải có điểm trung bình từ 7 trở lên, trong đó không có môn nào dưới 5.
- Sinh viên thi tốt nghiệp khi điểm trung bình nhỏ hơn 7 và điểm các môn không dưới 5.
- Sinh viên thi lại môn dưới 5.

9. Xây dựng lớp **vector** để lưu trữ các vectơ gồm các số thực. Các thành phần dữ liệu bao gồm:

- Kích thước vectơ.
- Một mảng động chứa các thành phần của vectơ.

Các hàm thành phần bao gồm hàm tạo, hàm hủy, hàm tính tích vô hướng hai vectơ, tính chuẩn của vectơ (theo chuẩn bất kỳ nào đó).

10. Xây dựng lớp **Phanso** với dữ liệu thành phần là tử và mẫu số. Các hàm thành phần bao gồm:

- Cộng hai phân số, kết quả phải được tối giản
- Trừ hai phân số, kết quả phải được tối giản
- Nhân hai phân số, kết quả phải được tối giản
- Chia hai phân số, kết quả phải được tối giản

## Chương 4

### TOÁN TỬ TÀI BỘI

*Chương 4 trình bày các vấn đề sau:*

- *Định nghĩa toán tử tài bội*
- *Một số lưu ý khi xây dựng toán tử tài bội*
- *Một số ví dụ minh họa*

#### **4.1. Định nghĩa toán tử tài bội**

Các toán tử cùng tên thực hiện nhiều chức năng khác nhau được gọi là toán tử tài bội. Dạng định nghĩa tổng quát của toán tử tài bội như sau:

Kiểu\_trả\_về operator op(danh sách tham số)

{//thân toán tử}

Trong đó: Kiểu\_trả\_về là kiểu kết quả thực hiện của toán tử.

op là tên toán tử tài bội

operator op (danh sách tham số) gọi là hàm toán tử tài bội. nó có thể là hàm thành phần hoặc là hàm bạn, nhưng không thể là hàm tĩnh. Danh sách tham số được khai báo tương tự khai báo biến nhưng phải tuân theo những quy định sau:

- Nếu toán tử tài bội là hàm thành phần thì: không có tham số cho toán tử một ngôi và một tham số cho toán tử hai ngôi. Cũng giống như hàm thành phần thông thường, hàm thành phần toán tử có đối đầu tiên (không tường minh) là con trỏ this.

- Nếu toán tử tài bội là hàm bạn thì: có một tham số cho toán tử một ngôi và hai tham số cho toán tử hai ngôi.

## **Quá trình xây dựng toán tử tài bối được thực hiện như sau:**

- Định nghĩa lớp để xác định kiểu dữ liệu sẽ được sử dụng trong các toán tử tài bối

- Khai báo hàm toán tử tài bối trong vùng public của lớp

- Định nghĩa nội dung cần thực hiện

### **4.2. Một số lưu ý khi xây dựng toán tử tài bối**

1. Trong C++ ta có thể đưa ra nhiều định nghĩa mới cho hầu hết các toán tử trong C++, ngoại trừ những toán tử sau đây:

- Toán tử xác định thành phần của lớp (‘.’)

- Toán tử phân giải miền xác định (‘::’)

- Toán tử xác định kích thước (‘sizeof’)

- Toán tử điều kiện 3 ngôi (‘?:’)

2. Mặc dù ngữ nghĩa của toán tử được mở rộng nhưng cú pháp, các quy tắc văn phạm như số toán hạng, quyền ưu tiên và thứ tự kết hợp thực hiện của các toán tử vẫn không có gì thay đổi.

3. Không thể thay đổi ý nghĩa cơ bản của các toán tử đã định nghĩa trước, ví dụ không thể định nghĩa lại các phép toán +, - đối với các số kiểu int, float.

4. Các toán tử =, ( ), [ ], -> yêu cầu hàm toán tử phải là hàm thành phần của lớp, không thể dùng hàm bạn để định nghĩa toán tử tài bối.

### **4.3. Một số ví dụ**

#### **Ví dụ 4.1.** Toán tử tài bối một ngôi, dùng hàm bạn

```
#include <iostream.h>
#include <conio.h>
class Diem
{
```

```

private:
    float x,y,z;

public:
    Diem() {}

    Diem(float x1,float y1,float z1)
    { x = x1; y = y1; z=z1; }

    friend Diem operator -(Diem d)
    {
        Diem d1;
        d1.x = -d.x; d1.y = -d.y;d1.z=-d.z;
        return d1;
    }

    void hienthi()
    {
        cout<<"Toa do: "<<x<<" "<<y <<" "
        <<z<<endl;
    }
};

void main()
{
    clrscr();
    Diem p(2,3,-4),q;
    q = -p;
    p.hienthi();
    q.hienthi();
    getch();
}

```

**Ví dụ 4.2.** Toán tử tài bội hai ngôi, dùng hàm bạn

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

class Diem

{
private:
    float x,y,z;

public:
    Diem() {}

    Diem(float x1,float y1,float z1)
    { x = x1; y = y1; z=z1; }

    friend Diem operator +(Diem d1, Diem d2)

    {
        Diem tam;
        tam.x = d1.x + d2.x;
        tam.y = d1.y + d2.y;
        tam.z = d1.z + d2.z;
        return tam;
    }

    void hienthi()
    { cout<<x<<" "<<y <<" " <<z<<endl; }

};

void main()
{
    clrscr();
```

```

Diem d1(3,-6,8),d2(4,3,7),d3;
d3=d1+d2;
d1.hienthi();
d2.hienthi();
cout<<"\n Tong hai diem co toa do la :";
d3.hienthi();
getch();
}

```

**Ví dụ 4.3.** Toán tử tài bội hai ngôi, dùng hàm thành phần

```

#include <iostream.h>
#include <conio.h>
class Diem
{
private:
    float x,y;
public:
    Diem() {}
    Diem(float x1,float y1)
    { x = x1; y = y1;}
    Diem operator -()
    { x = -x; y = -y; z = -z;
        return (*this); }
    void hienthi()
    { cout<<"Toa do: "<<x<<" "<<y <<" z = "<<z
        <<" \n"; }

```

```

};

void main()
{
    clrscr();
    Diem p(2,3,-4),q;
    p.hienthi();
    q = -p;
    q.hienthi();
    getch();
}

```

**Ví dụ 4.4.** Toán tử tài bội hai ngôi, dùng hàm thành phần

```

#include <iostream.h>

#include <conio.h>

class Diem

{
private:
    float x,y,z;

public:
    Diem() {}

    Diem(float x1,float y1,float z1)
    { x = x1; y = y1; z=z1; }

    Diem operator +(Diem d2)

    {
        x = x + d2.x;
        y = y + d2.y;
    }
}

```

```

        z = z + d2.z;
        return (*this);
    }

void hienthi()
{
    cout<<"\n x="<<x<<" y= "<<y<<" z = " << z
        <<endl;
}

void main()
{
    clrscr();
    Diem d1(3,-6,8),d2(4,3,7),d3;
    d1.hienthi();
    d2.hienthi();
    d3=d1+d2;
    cout<<"\n Tong hai diem co toa do la :";
    d3.hienthi();
    getch();
}

```

#### **Ví dụ 4.5.**

```

#include <iostream.h>
#include <conio.h>

class Diem
{
private:
    int x,y;

```

```

public:

    Diem() {}

    Diem(int x1,int y1)
    { x = x1; y = y1; }

    void operator -()
    {
        x = -x; y = -y;
    }

    void hienthi()
    { cout<<"Toa do: "<<x<<" "<<y <<" \n"; }

};

void main()
{
    clrscr();
    Diem p(2,3),q;
    p.hienthi();
    -p;
    p.hienthi();
    getch();
}

```

**Ví dụ 4.6.** Toán tử tài bội hai ngôi

```

#include <iostream.h>

#include <conio.h>

class sophuc

{float a,b;

```

```

public : sophuc() {}

sophuc(float x, float y)

{a=x; b=y; }

sophuc operator +(sophuc c2)

{      sophuc c3;

c3.a= a + c2.a ;

c3.b= b + c2.b ;

return (c3);

}

void hienthi(sophuc c)

{ cout<<c.a<<" + "<<c.b<<"i"<<endl; }

};

void main()

{ clrscr();

sophuc d1 (2.1,3.4);

sophuc d2 (1.2,2.3) ;

sophuc d3 ;

d3 = d1+d2; //d3=d1.operator +(d2);

cout<<"d1= ";d1.hienthi(d1);

cout<<"d2= ";d2.hienthi(d2);

cout<<"d3= ";d3.hienthi(d3);

getch();

}

```

**Chú ý:** Trong các hàm toán tử thành phần hai ngôi (có hai toán hạng) thì con trỏ this ứng với toán hạng thứ nhất, vì vậy trong tham số của toán tử chỉ cần dùng một tham số tường minh để biểu thị toán hạng thứ hai.

#### Ví dụ 4.7. Phiên bản 2 của ví dụ 4.6

```
#include <iostream.h>
#include <conio.h>
class sophuc
{float a,b;
public : sophuc() {}
    sophuc(float x, float y)
    {a=x; b=y;}
    sophuc operator +(sophuc c2)
    {
        a= a + c2.a ;
        b= b + c2.b ;
        return (*this);
    }
    void hienthi(sophuc c)
    { cout<<c.a<<" + "<<c.b<<"i"<<endl; }
};

void main()
{ clrscr();
    sophuc d1 (2.1,3.4);
    sophuc d2 (1.2,2.3) ;
    sophuc d3 ;
```

```

cout<<"d1= ";d1.hienthi(d1);
cout<<"d2= ";d2.hienthi(d2);
d3 = d1+d2; //d3=d1.operator +(d2);
cout<<"d3= ";d3.hienthi(d3);
getch();
}

```

**Ví dụ 4.8.** Phiên bản 3 của ví dụ 4.6

```

#include <iostream.h>
#include <conio.h>
class sophuc
{
float a,b;
public : sophuc() {}
sophuc(float x, float y)
{
a=x; b=y;}
friend sophuc operator +(sophuc c1,sophuc c2)
{
sophuc c;
c.a= c1.a + c2.a ;
c.b= c1.b + c2.b ;
return (c);
}
void hienthi(sophuc c)
{
cout<<c.a<<" + "<<c.b<<"i"<<endl; }
};

void main()
{ clrscr();

```

```

sophuc d1 (2.1,3.4);

sophuc d2 (1.2,2.3) ;

sophuc d3 ;

cout<<"d1= ";d1.hienthi(d1);

cout<<"d2= ";d2.hienthi(d2);

d3 = d1+d2; //d3=operator +(d1,d2);

cout<<"d3= ";d3.hienthi(d3);

getch();
}

```

**Ví dụ 4.9.** Toán tử tái bội trên lớp chuỗi ký tự

```

#include <iostream.h>

#include <string.h>

#include <conio.h>

class string

{ char s[80];

public:

    string() { *s='\0'; }

    string(char *p) { strcpy(s,p); }

    char *get() { return s; }

    string operator + (string s2);

    string operator = (string s2);

    int operator < (string s2);

    int operator > (string s2);

    int operator == (string s2);

};

```

```
string string::operator +(string s2)
{
    strcat(s,s2.s);
    return *this ;
}

string string::operator =(string s2)
{
    strcpy(s,s2.s) ;
    return *this;
}

int string::operator <(string s2)
{
    return strcmp(s,s2.s)<0 ;
}

int string::operator >(string s2)
{
    return strcmp(s,s2.s)>0 ;
}

int string::operator ==(string s2)
{
    return strcmp(s,s2.s)==0 ;
}

void main()
{ clrscr();
```

```

string o1 ("Trung Tam "), o2 (" Tin hoc"), o3;
cout<<"o1 = "<<o1.get()<<'\n';
cout<<"o2 = "<<o2.get()<<'\n';
if (o1 > o2)
    cout << "o1 > o2 \n";
if (o1 < o2)
    cout << "o1 < o2 \n";
if (o1 == o2)
    cout << "o1 bang o3 \n";
o3=o1+o2;
cout<<"o3 ="<<o3.get()<<'\n'; //Trung tam tin hoc
o3=o2;
cout<<"o2 = "<<o2.get()<<'\n'; //Tin hoc
cout<<"o3 = "<<o3.get()<<'\n'; //Tin hoc
if (o2 == o3)
    cout << "o2 bang o3 \n";
getch();
}

```

#### **4.4. Định nghĩa chồng các toán tử ++ , --**

Ta có thể định nghĩa chồng cho các toán tử ++/-- theo quy định sau:

- Toán tử ++/-- dạng tiền tố trả về một tham chiếu đến đối tượng thuộc lớp.
- Toán tử ++/-- dạng tiền tố trả về một đối tượng thuộc lớp.

#### **Ví dụ 4.10.**

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

```
class Diem
{
    private:
        int x,y;
    public:
        Diem() {x = y = 0;}
        Diem(int x1, int y1)
        {x = x1;
         y = y1;}
        Diem & operator ++(); //qua tai toan tu ++ tien to
        Diem operator ++(int); //qua tai toan tu ++ hau to
        Diem & operator --(); //qua tai toan tu -- tien to
        Diem operator --(int); //qua tai toan tu -- hau to
        void hienthi()
        {
            cout<<" x = "<<x<<" y = "<<y;
        }
    };
    Diem & Diem::operator ++()
    {
        x++;
        y++;
        return (*this);
    }
    Diem Diem::operator ++(int)
```

```

{
    Diem temp = *this;
    ++*this;
    return temp;
}

Diem & Diem::operator --()

{
    x--;
    y--;
    return (*this);
}

Diem Diem::operator --(int)

{
    Diem temp = *this;
    --*this;
    return temp;
}

void main()
{
    clrscr();
    Diem d1(5,10),d2(20,25),d3(30,40),d4(50,60);
    cout<<"\n d1 : ";d1.hienthi();
    ++d1;
    cout<<"\n Sau khi tac dong cac toan tu tang truoc :";
    cout<<"\n d1 : ";d1.hienthi();
}

```

```

cout<<"\nd2 : ";d2.hienthi();

d2++;

cout<<"\n Sau khi tac dong cac toan tu tang sau";

cout<<"\nd2 : ";d2.hienthi();

cout<<"\nd3 : ";d3.hienthi();

--d3;

cout<<"\n Sau khi tac dong cac toan tu giam truoc : ";

cout<<"\nd3 : ";d3.hienthi();

cout<<"\nd4 : ";d4.hienthi();

d4--;

cout<<"\n Sau khi tac dong cac toan tu giam sau : ";

cout<<"\nd4 : ";d4.hienthi();

getch();

}

```

Chương trình cho kết quả như sau:

d1 : x = 5 y = 10

Sau khi tac dong cac toan tu tang truoc :

d1 : x = 6 y = 11

d2 : x = 20 y = 25

Sau khi tac dong cac toan tu tang sau

d2 : x = 21 y = 26

d3 : x = 30 y = 40

Sau khi tac dong cac toan tu giam truoc :

d3 : x = 29 y = 39

d4 : x = 50 y = 60

Sau khi tac dong cac toan tu giam sau :

d4 : x = 49 y = 59

**Chú ý:** Đối số int trong dạng hậu tố là bắt buộc, dùng để phân biệt với dạng tiền tố, thường nó mang trị mặc định là 0.

#### 4.5. Định nghĩa chồng toán tử << và >>

Ta có thể định nghĩa chồng cho hai toán tử vào/ra << và >> kết hợp với cout và cin (cout<< và cin>>), cho phép các đối tượng đứng bên phải chúng. Lúc đó ta có thể thực hiện các thao tác vào ra như nhập dữ liệu từ bàn phím cho các đối tượng, hiển thị giá trị thành phần dữ liệu của các đối tượng ra màn hình. Hai hàm toán tử << và >> phải là hàm tự do và khai báo là hàm bạn của lớp.

#### Ví dụ 4.11.

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

class SO
{
private:
    int giatri;
public:
    SO(int x=0)
    {
        giatri = x;
    }
    SO (SO &tso)
    {
```

```

giatri = tso.giatri;
}

friend istream& operator>>(istream&, SO&);

friend ostream& operator<<(ostream&, SO&);

};

void main(){

    clrscr();

    SO so1,so2;

    cout<<"Nhập dữ liệu cho số 1 và số 2 " << endl;

    cin>>so1;

    cin>>so2;

    cout<<"Giá trị số 1 là : " << so1

        <<" số 2 là : "<< so2 << endl;

    getch();

}

istream& operator>>(istream& nhap,SO& so)

{

    cout << "Nhập giá trị số :";

    nhap>> so.giatri;

    return nhap; }

ostream& operator<<(ostream& xuat,SO& so)

{

    xuat<< so.giatri;

    return xuat;

}

```

## Bài tập

1. Định nghĩa các phép toán tài bối `=, ==, ++, --, +=, <<, >>` trên lớp `Time` (bài tập 1 chương 3).
2. Định nghĩa các phép toán tài bối `=, ==, ++, --, +=, <<, >>` trên lớp `Date` (bài tập 2 chương 3).
3. Định nghĩa các phép toán tài bối `+, -, *, =, ==, !=` trên lớp các ma trận vuông.
4. Định nghĩa các phép toán tài bối `+, -, *` trên lớp đa thức.
5. Định nghĩa các phép toán tài bối `+, -, *, /, =, ==, +=, -=, *=, /=, <, >, <=, >=, !=, ++, --` trên lớp `Phanso` (bài tập 10 chương 3).
6. Ma trận được xem là một vectơ mà mỗi thành phần của nó là một vectơ. Theo nghĩa đó, hãy định nghĩa lớp **Matran** dựa trên vectơ. Tìm cách để chương trình dịch hiều được phép truy nhập `m[i][j]`, trong đó `m` là một đối tượng thuộc lớp **Matran**.

## Chương 5

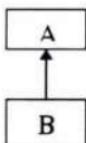
### KẾ THỪA

*Chương 5 trình bày các vấn đề sau:*

- *Đơn kế thừa, đa kế thừa*
- *Hàm tạo và hàm hủy đối với sự kế thừa*
- *Hàm ảo, lớp cơ sở ảo*

#### **5.1. Giới thiệu**

Kế thừa là một trong các khái niệm cơ sở của phương pháp lập trình hướng đối tượng. Tính kế thừa cho phép định nghĩa các lớp mới từ các lớp đã có. Một lớp có thể là lớp cơ sở cho nhiều lớp dẫn xuất khác nhau. Lớp dẫn xuất sẽ kế thừa một số thành phần (dữ liệu và hàm) của lớp cơ sở, đồng thời có thêm những thành phần mới. Có hai loại kế thừa là: đơn kế thừa và đa kế thừa, có thể minh họa qua các hình vẽ sau đây:

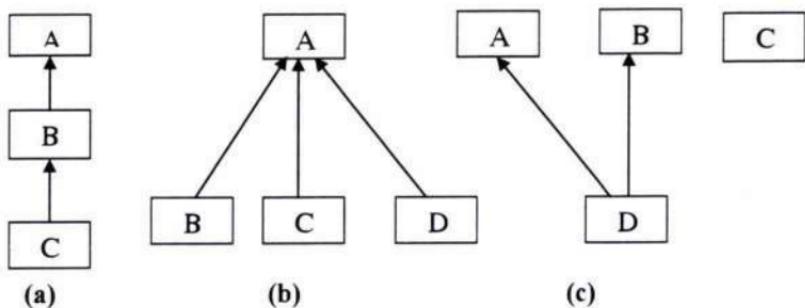


*Hình 5.1. Đơn kế thừa, lớp A là lớp cơ sở của lớp B*

Hình 5.2: Hình (a): Lớp A là lớp cơ sở của lớp B, lớp B là lớp cơ sở của lớp C

Hình (b): Lớp A là lớp cơ sở của các lớp B, C, D

Hình (c): Lớp A, B là lớp cơ sở của lớp D



Hình 5.2 Đa kế thừa

## 5.2. Đơn kế thừa

### 5.2.1. Định nghĩa lớp dẫn xuất từ một lớp cơ sở

Giả sử đã định nghĩa lớp A. Cú pháp để xây dựng lớp B dẫn xuất từ lớp A như sau:

```
class B: mode A
{
    private:
        // Khai báo các thuộc tính của lớp B
    public:
        // Định nghĩa các hàm thành phần của lớp B
};
```

Trong đó mode có thể là **private** hoặc **public** với ý nghĩa như sau:

- Kế thừa theo kiểu public thì tất cả các thành phần public của lớp cơ sở cũng là thành phần public của lớp dẫn xuất.

- Kế thừa theo kiểu private thì tất cả các thành phần public của lớp cơ sở sẽ trở thành các thành phần private của lớp dẫn xuất.

**Chú ý:** Trong cả hai trường hợp ở trên thì thành phần private của lớp cơ sở là không được kế thừa. Như vậy trong lớp dẫn xuất không cho phép truy nhập đến các thành phần private của lớp cơ sở.

### 5.2.2. Truy nhập các thành phần trong lớp dẫn xuất

Thành phần của lớp dẫn xuất bao gồm: các thành phần khai báo trong lớp dẫn xuất và các thành phần mà lớp dẫn xuất thừa kế từ các lớp cơ sở. Quy tắc sử dụng các thành phần trong lớp dẫn xuất được thực hiện theo mẫu như sau:

Tên đối tượng.Tên\_lớp::Tên\_thành\_phần

Khi đó chương trình dịch C++ dễ dàng phân biệt thành phần thuộc lớp nào.

**Ví dụ 5.1.** Giả sử có các lớp A và B như sau:

```
class A
{
public: int n;
void nhap()
{
    cout<<"\n Nhập n = ";
    cin>>n;
}
class B: public A
{
public: int m;
void nhap()
{
    cout<<"\n Nhập m = ";
```

```
    cin>>m;  
}  
};
```

Xét khai báo: B ob;

Lúc đó: ob.B::m là thuộc tính n khai báo trong B

ob.A::n là thuộc tính n thừa kế từ lớp A

ob.D::nhap() là hàm nhap() định nghĩa trong lớp B

ob.A::nhap() là hàm nhap() định nghĩa trong lớp A

**Chú ý:** Để sử dụng các thành phần của lớp dẫn xuất, có thể không dùng tên lớp, chỉ dùng tên thành phần. Khi đó chương trình dịch phải tự phán đoán để biết thành phần đó thuộc lớp nào: trước tiên xem thành phần đang xét có trùng tên với các thành phần nào của lớp dẫn xuất không? Nếu trùng thì đó thành phần của lớp dẫn xuất. Nếu không trùng thì tiếp tục xét các lớp cơ sở theo thứ tự: các lớp có quan hệ gần với lớp dẫn xuất sẽ được xét trước, các lớp quan hệ xa hơn xét sau. Chú ý trường hợp thành phần đang xét có mặt đồng thời trong 2 lớp cơ sở có cùng một đẳng cấp quan hệ với lớp dẫn xuất. Trường hợp này chương trình dịch không thể quyết định được thành phần này thừa kế từ lớp nào và sẽ đưa ra một thông báo lỗi.

### 5.2.3. Định nghĩa lại các hàm thành phần của lớp cơ sở trong lớp dẫn xuất

Trong lớp dẫn xuất có thể định nghĩa lại hàm thành phần của lớp cơ sở. Như vậy có hai phiên bản khác nhau của hàm thành phần trong lớp dẫn xuất. Trong phạm vi lớp dẫn xuất, hàm định nghĩa lại “che khuất” hàm được định nghĩa. Việc sử dụng hàm nào cần tuân theo quy định ở trên.

**Chú ý:** Việc định nghĩa lại hàm thành phần khác với định nghĩa hàm quá tải. Hàm định nghĩa lại và hàm bị định nghĩa lại giống nhau về tên, tham số và giá trị trả về. Chúng chỉ khác nhau về vị trí: một hàm đặt trong lớp dẫn xuất và hàm kia thì ở trong lớp cơ sở. Trong khi đó, các hàm quá tải chỉ có cùng tên, nhưng khác nhau về danh sách tham số và tất cả chúng thuộc cùng một lớp.

Định nghĩa lại hàm thành phần chính là cơ sở cho việc xây dựng tính đa hình của các hàm.

C++ cho phép đặt trùng tên thuộc tính trong các lớp cơ sở và lớp dẫn xuất. Các thành phần cùng tên này có thể cùng kiểu hay khác kiểu. Lúc này bên trong đối tượng của lớp dẫn xuất có tới hai thành phần khác nhau có cùng tên, nhưng trong phạm vi lớp dẫn xuất, tên chung đó nhằm chỉ định thành phần được khai báo lại trong lớp dẫn xuất. Khi muốn chỉ định thành phần trùng tên trong lớp cơ sở phải dùng tên lớp toán tử ‘::’ đặt trước tên hàm thành phần.

**Ví dụ 5.2.** Xét các lớp A và B được xây dựng như sau:

```
class A
{
    private:
        int a, b, c;
    public:
        void nhap()
        {
            cout<<"\na = "; cin>>a;
            cout<<"\nb = "; cin>>b;
            cout<<"\nc = "; cin>>c;
        }
        void hienthi()
        {
            cout<<"\na = "<<a<<" b = "<<b<<" c = "<<c; }
};

class B: private A
{
    private:
        double d;
```

```
public:  
    void nhap_d()  
    { cout<<"\nd = "; cin>>d;  
    }  
};
```

Lớp B kế thừa theo kiểu private từ lớp A, các thành phần public của lớp A là các hàm nhap() và hienthi() trở thành thành phần private của lớp B.

Xét khai báo : B ob; Lúc đó các câu lệnh sau đây là lỗi :

```
ob.nhap();  
ob.d=10;  
ob.a = ob.b = ob.c = 5;  
ob.hienthi();
```

bởi vì đối tượng ob không thể truy nhập vào các thành phần private của lớp A và B.

**Ví dụ 5.3.** Chương trình minh họa đơn kế thừa theo kiểu public:

```
#include <iostream.h>  
#include <conio.h>  
  
class A  
{  
    int a; //Bien private, khong duoc ke thua  
public:  
    int b; //Bien public, se duoc ke thua  
    void get_ab();  
    int get_a(void);  
    void show_a(void);
```

```

};

class B: public A
{
    int c;

public:
    void mul(void);
    void display(void);
};

void A::get_ab(void)
{ a = 5; b = 10; }

int A::get_a()
{ return a; }

void A::show_a()
{ cout << "a = " << a << "\n"; }

void B::mul()
{ c = b*get_a(); }

void B::display()
{
    cout << "a = " << get_a() << "\n";
    cout << "b = " << b << "\n";
    cout << "c = " << c << "\n";
}

void main()
{ clrscr();
  B d;
}

```

```
d.get_ab(); //Ke thua tu A  
d.mul();  
d.show_a(); //Ke thua tu A  
d.display();  
d.b = 20;  
d.mul();  
d.display();  
getch();  
}
```

Chương trình cho kết quả:

a = 5

a = 5

b = 10

c = 50

a = 5

b = 20

c = 100

**Ví dụ 5.4.** Chương trình sau là minh họa khác cho trường hợp kế thừa đơn:

```
#include <conio.h>  
#include <iostream.h>  
  
class Diem  
{  
private:
```

```

    double x, y;

public:
    void nhap()
    {
        cout<<"\n x = "; cin>>x;
        cout<<"\n y = "; cin>>y;
    }

    void hienthi()
    {
        cout<<"\n x = "<<x<<" y = "<<y;
    }
};

class Hinhtron: public Diem
{
private:
    double r;

public:
    void nhap_r()
    {
        cout<<"\n r = "; cin>>r;
    }

    double get_r()
    {
        return r;
    }
}

```

```

};

void main()
{
    Hinhtron h;

    clrscr();

    cout<<"\n Nhap toa do tam va ban kinh hinh tron";

    h.nhap();

    h.nhap_r();

    cout<<"\n Hinh tron co tam:";h.hienthi();

    cout<<"\n Co ban kinh = " << h.get_r();

    getch();
}

```

Chương trình cho kết quả:

Nhap toa do tam va ban kinh hinh tron

x = 2

y = 3

r = 10

Hinh tron co tam:

x = 2 y = 3

Co ban kinh = 10

#### *5.2.4. Hàm tạo đối với tinh kế thừa*

Các hàm tạo của lớp cơ sở là không được kế thừa. Một đối tượng của lớp dẫn xuất về thực chất có thể xem là một đối tượng của lớp cơ sở, vì vậy việc gọi hàm tạo lớp dẫn xuất để tạo đối tượng của lớp dẫn xuất sẽ kéo theo việc gọi đến một hàm tạo của lớp cơ sở. Thứ tự thực hiện của các hàm tạo sẽ là: hàm tạo cho lớp cơ sở, rồi đến hàm tạo cho lớp dẫn xuất.

C++ thực hiện điều này bằng cách: trong định nghĩa của hàm tạo lớp dẫn xuất, ta mô tả một lời gọi tới hàm tạo trong lớp cơ sở. Cú pháp để truyền tham số từ lớp dẫn xuất đến lớp cơ sở như sau:

```
Tên_lớp_dẫn_xuất(danh_sách_đôi);Tên_lớp_cơ_sở(danh_sách_đôi)  
{  
    //thân hàm tạo của lớp dẫn xuất  
};
```

Trong phần lớn các trường hợp, hàm tạo của lớp dẫn xuất và hàm tạo của lớp cơ sở sẽ không dùng tham số giống nhau. Trong trường hợp cần truyền một hay nhiều tham số cho mỗi lớp, ta phải truyền cho hàm tạo của lớp dẫn xuất **tất cả** các tham số mà cả hai lớp dẫn xuất và cơ sở cần đến. Sau đó, lớp dẫn xuất chỉ truyền cho lớp cơ sở những tham số nào mà lớp cơ sở cần.

**Ví dụ 5.5.** Chương trình sau minh họa cách truyền tham số cho hàm tạo của lớp cơ sở:

```
#include <iostream.h>  
#include <conio.h>  
  
class Diem  
{  
private:  
    double x, y;  
public:  
    Diem()  
    {  
        x=y=0.0;  
    }  
    Diem(double x1, double y1)
```

```

{
    x=x1; y=y1;
}

void in()
{
    cout<<"\nx="<<x<<"y="<<y;
}

};

class Hinhtron: public Diem
{
private:
    double r;
public:
    Hinhtron()
    {
        r = 0.0;
    }

    Hinhtron(double x1,double y1,double r1): Diem (x1,
y1)
    {
        r=r1;
    }

    double get_r()
    {
        return r;
    }
}

```

```

    }

};

void main()
{
    Hinhtron h(2.5, 3.5, 8);

    clrscr();

    cout<<"\n Hinh tron co tam:";

    h.in();

    cout<<"\n Co ban kinh =" << h.get_r();

    getch();
}

```

**Chú ý:** Các tham số mà hàm tạo của lớp dẫn xuất truyền cho hàm tạo của lớp cơ sở không nhất thiết phải lấy hoàn toàn y như từ các tham số nó nhận được. Ví dụ:

Hinhtron(double x1,double y1,double r1):Diem (x1/2, y1/2)

#### 5.2.5. Hàm hủy đối với tinh kế thừa

Hàm hủy của lớp cơ sở cũng không được kế thừa. Khi cả lớp cơ sở và lớp dẫn xuất có các hàm hủy và hàm tạo, các hàm tạo thi hành theo thứ tự dẫn xuất. Các hàm hủy được thi hành theo thứ tự ngược lại. Nghĩa là, hàm tạo của lớp cơ sở thi hành trước hàm tạo của lớp dẫn xuất, hàm hủy của lớp dẫn xuất thi hành trước hàm hủy của lớp cơ sở.

#### Ví dụ 5.6.

```

#include <iostream.h>

#include <conio.h>

class CS
{
public:

```

```

CS()

{cout<<"\nHam tao lop co so";}

~CS()

{cout<<"\nHam huy lop co so";}

};

class DX:public CS

{ public:

DX()

{cout<<"\nHam tao lop dan xuat";}

~DX()

{cout<<"\nHam huy lop dan xuat";}

};

void main()

{ clrscr();

DX ob;

getch();

}

```

Chương trình này cho kết quả như sau :

Ham tao lop co so

Ham tao dan xuat

Ham huy lop dan xuat

Ham huy lop co so

#### *5.2.6. Khai báo protected*

Ta đã biết các thành phần khai báo private không được kế thừa trong lớp dẫn xuất. Có thể giải quyết vấn đề này bằng cách chuyển chúng sang vùng

public. Tuy nhiên cách làm này lại phá vỡ nguyên lý che giấu thông tin của LTHĐT. C++ đưa ra cách giải quyết khác là sử dụng khai báo **protected**. Các thành phần **protected** có phạm vi truy nhập rộng hơn so với các thành phần **private**, nhưng hẹp hơn so với các thành phần **public**.

Các thành phần **protected** của lớp cơ sở hoàn toàn giống các thành phần **private** *ngoại trừ một điểm* là chúng có thể kế thừa từ lớp dẫn xuất trực tiếp từ lớp cơ sở. Cụ thể như sau:

- Nếu kế thừa theo kiểu public thì các thành phần proteted của lớp cơ sở sẽ trở thành các thành phần protected của lớp dẫn xuất.

- Nếu kế thừa theo kiểu private thì các thành phần proteted của lớp cơ sở sẽ trở thành các thành phần private của lớp dẫn xuất.

### 5.2.7. Dẫn xuất **protected**

Ngoài hai kiểu dẫn xuất đã biết là private và public, C++ còn đưa ra kiểu dẫn xuất **protected**. Trong dẫn xuất loại này thì các thành phần public, protected trong lớp cơ sở trở thành các thành phần **protected** trong lớp dẫn xuất.

## 5.3. Đa kế thừa

### 5.3.1. Định nghĩa lớp dẫn xuất từ nhiều lớp cơ sở

Giả sử đã định nghĩa các lớp A, B. Cú pháp để xây dựng lớp C dẫn xuất từ lớp A và B như sau:

```
class C: mode A, mode B  
{  
    private:  
        // Khai báo các thuộc tính  
    public:  
        // Các hàm thành phần  
};
```

trong đó mode có thể là private, public hoặc protected. Ý nghĩa của kiểu dẫn xuất này giống như trường hợp đơn kế thừa.

### 5.3.2. Một số ví dụ về đa kế thừa

**Ví dụ 5.7.** Chương trình sau minh họa một lớp kế thừa từ hai lớp cơ sở:

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

class M
{
protected :
    int m;
public :
    void getm(int x) {m=x;}
};

class N
{
protected :
    int n;
public :
    void getn(int y) {n=y;}
};

class P : public M,public N
{
public :
    void display(void)
    {
        cout<<"m= "<<m<<endl;
    }
}
```

```

        cout<<"n= "<<n<<endl;
        cout<<"m * n = "<<m*n<<endl;
    }
};

void main()
{
    P ob;

    clrscr();
    ob.getm(10);
    ob.getn(20);
    ob.display();
    getch();
}

```

Chương trình cho kết quả như sau:

$m = 10$

$n = 20$

$m * n = 200$

**Ví dụ 5.8.** Chương trình sau minh họa việc quản lý kết quả thi của một lớp không quá 100 sinh viên. Chương trình gồm 3 lớp: lớp cơ sở sinh viên (*sinhvien*) chỉ lưu họ tên và số báo danh, lớp điểm thi (*diemthi*) kế thừa lớp sinh viên và lưu kết quả môn thi 1 và môn thi 2. Lớp kết quả (*ketqua*) lưu tổng số điểm đạt được của sinh viên.

```

#include <iostream.h>
#include <conio.h>
#include <stdio.h>

```

```
class sinhvien
{
    char hoten[25];

protected:
    int sbd;

public:
    void nhap()
    {
        cout<<"\nHo ten :"; gets(hoten);
        cout<<"So bao danh :"; cin>>sbd;
    }

    void hienthi()
    {
        cout<<"So bao danh : "<<sbd<<endl;
        cout<<"Ho va ten sinh vien : "<<hoten<<endl;
    }
};

class diemthi : public sinhvien
{
protected :
    float d1,d2;

public :
    void nhap_diem()
    {
        cout<<"Nhaph diem hai mon thi : \n";
        cin>>d1>>d2;
    }

    void hienthi_diem()
    {
        cout<<"Diem mon 1 :"<<d1<<endl;
    }
};
```

```

        cout<<"Diem mon 2 :"<<d2<<endl;
    }
};

class ketqua : public diemthi
{
    float tong;
public :
void display()
{
    tong = d1+d2;
    hienthi();
    hienthi_diem();
    cout<<"Tong so diem :"<<tong<<endl;
}
};

void main()
{ int i,n; ketqua sv[100];
cout<<"\n Nhap so sinh vien : ";
cin>>n;
clrscr();
for(i=0;i<n;++i)
{ sv[i].nhap();
sv[i].nhap_diem();
}
for(i=0;i<n;++i)
sv[i].display();
}

```

```
    getch();  
}
```

**Ví dụ 5.9.** Chương trình sau là sự mở rộng của chương trình ở trên, trong đó ngoài kết quả hai thi, mỗi sinh viên còn có thể có điểm thường. Chương trình mở rộng thêm một lớp ưu tiên (uutien).

```
#include <iostream.h>  
#include <conio.h>  
#include <stdio.h>  
  
class sinhvien  
{ char hoten[25];  
protected : int sbd;  
public :  
  
void nhap()  
{ cout<<"\nHo ten :"; gets(hoten);  
cout<<"So bao danh :"; cin>>sbd;  
}  
  
void hienthi()  
{ cout<<"So bao danh : "<<sbd<<endl;  
cout<<"Ho va ten sinh vien :  
<<hoten<<endl;  
}  
};  
  
class diemthi : public sinhvien  
{ protected : float d1,d2;  
public :  
  
void nhap_diem()  
{
```

```

        cout<<"Nhap diem hai mon thi : \n";
        cin>>d1>>d2;
    }

void hienthi_diem()
{
    cout<<"Diem mon 1 :"<<d1<<endl;
    cout<<"Diem mon 2 :"<<d2<<endl;
}

};

class uutien
{
protected : float ut;
public :
void nhap_ut()
{
    cout<<"\nNhap diem uu tien :";cin>>ut;
}

void hienthi_ut()
{cout<<"Diem uu tien : "<<ut<<endl; }

};

class ketqua : public diemthi, public uutien
{
float tong;
public :
void display()
{
    tong=d1+d2+ut;
    hienthi();
    hienthi_diem();
    hienthi_ut();
}

```

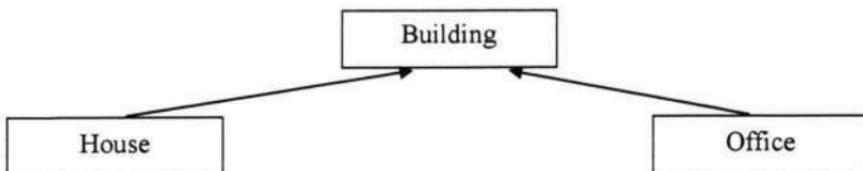
```

        cout<<"Tong so diem :"<<tong<<endl;
    }
};

void main()
{ int i,n; ketqua sv[100];
    cout<<"\n Nhap so sinh vien : ";
    cin>>n;
    clrscr();
    for(i=0;i<n;++i)
    { sv[i].nhap();
        sv[i].nhap_diem();
        sv[i].nhap_ut();
    }
    for(i=0;i<n;++i)
        sv[i].display();
    getch();
}

```

**Ví dụ 5.10.** Xem sơ đồ kế thừa các lớp như sau:



*Hình 5.3 Sơ đồ kế thừa lớp*

Trong đó lớp cơ sở Building lưu trữ số tầng của một tòa nhà, tổng số phòng và tổng diện tích của tòa nhà. Lớp dẫn xuất House kế thừa lớp Building và lưu trữ số phòng ngủ, số phòng tắm. Lớp dẫn xuất Office từ lớp Building

lưu trữ số máy điện thoại và số bình cứu hỏa. Chương trình sau minh họa việc tổ chức lưu trữ theo sơ đồ kể thừa này.

```
#include <iostream.h>
#include <conio.h>
class Building
{ protected :
    int floors; //tong so tang
    int rooms; //tong so phong
    double footage; //tong so dien tich
};
class house : public Building
{    int bedrooms; //tong so phong ngu
    int bathrooms; //tong so phong tam
public :
house(int f, int r, int ft, int br, int bth)
{
floors=f; rooms=r; footage=ft;
bedrooms=br; bathrooms=bth;
}
void show()
{ cout<<'\n';
cout<<" So tang : " <<floors <<'\n';
cout<<" So phong : " <<rooms <<'\n';
cout<<" So tong dien tich : " <<footage<<'\n';
cout<<" So phong ngu : " <<bedrooms <<'\n';
}
```

```

        cout<<" So phong tam : " <<bathrooms<<'\n';
    }

};

class office : public Building
{
    int phones; //tong so may dien thoai
    int extis; //tong so binh cuu hoa
public :
office(int f, int r, int ft, int p, int ext)
{
    floors=f; rooms=r; footage=ft;
    phones=p; extis=ext;
}

void show()
{
    cout<<'\n';
    cout<<" So tang : " <<floors <<'\n';
    cout<<" So phong : " <<rooms <<'\n';
    cout<<" So tong dien tich : " <<footage
        <<"\n";
    cout<<" So may dien thoai : " <<phones
        << "\n";
    cout<<" So binh cuu hoa : " <<extis<<'\n';
}

};

void main()
{
    house h_ob(2,12,5000,6,4);
}

```

```
office o_ob(4,25,12000,30,8);  
cout<<"House : "  
h_ob.show();  
cout<<"Office : "  
o_ob.show();  
getch();  
}
```

Chương trình cho kết quả như sau:

House :

Số tầng : 2

Số phòng : 12

Số tổng diện tích : 5000

Số phòng ngủ : 6

Số phòng tắm : 4

Office :

Số tầng : 4

Số phòng : 25

Số tổng diện tích : 12000

Số máy điện thoại : 30

Số bình cứu hỏa : 8

#### **5.4. Hàm ảo**

##### **5.4.1 *Đặt vấn đề***

Trước khi đưa ra khái niệm về hàm ảo, ta hãy thảo luận ví dụ sau:

Giả sử có 3 lớp A, B và C được xây dựng như sau:

```
class A
{
public:
void xuat()
{
    cout <<"\n Lop A";
}
};

class B : public A
{
public:
void xuat()
{
    cout <<"\n Lop B";
}
};

class C : public B
{
public:
void xuat()
{
    cout <<"\n Lop C";
}
};
```

Cả 3 lớp này đều có hàm thành phần là xuat(). Lớp C có hai lớp cơ sở là A, B và C kế thừa các hàm thành phần của A và B. Do đó một đối tượng của C sẽ có 3 hàm xuat(). Xem các câu lệnh sau:

C ob; // ob là đối tượng kiểu C

ob.xuat(); // Gọi tới hàm thành phần xuat() của lớp D

ob.B::xuat(); // Gọi tới hàm thành phần xuat() của lớp B

ob.A::xuat(); // Gọi tới hàm thành phần xuat() của lớp A

Các lời gọi hàm thành phần trong ví dụ trên đều xuất phát từ đối tượng ob và mọi lời gọi đều xác định rõ hàm cần gọi.

Ta xét tiếp tình huống các lời gọi không phải từ một biến đối tượng mà từ một con trỏ đối tượng. Xét các câu lệnh:

A \*p, \*q, \*r; // p,q,r là các con trỏ kiểu A

A a; // a là đối tượng kiểu A

B b; // b là đối tượng kiểu B

C c; // c là đối tượng kiểu C

Bởi vì con trỏ của lớp cơ sở có thể dùng để chứa địa chỉ các đối tượng của lớp dẫn xuất, nên cả 3 phép gán sau đều hợp lệ:

p = &a; q = &b; r = &c;

Ta xét các lời gọi hàm thành phần từ các con trỏ p, q, r:

p->xuat(); q->xuat(); r->xuat();

Cả 3 câu lệnh trên đều gọi tới hàm thành phần xuat() của lớp A, bởi vì các con trỏ p, q, r đều có kiểu lớp A. Sở dĩ như vậy là vì một lời gọi (xuất phát từ một đối tượng hay con trỏ) tới hàm thành phần **luôn luôn liên kết với một hàm thành phần cố định** và sự liên kết này xác định trong quá trình biên dịch chương trình. Ta bảo đây là **sự liên kết tĩnh**.

Có thể tóm lược cách thức gọi các hàm thành phần như sau:

1. Nếu lời gọi xuất phát từ một đối tượng của lớp nào đó, thì hàm thành phần của lớp đó sẽ được gọi.
2. Nếu lời gọi xuất phát từ một con trỏ kiểu lớp, thi hàm thành phần của lớp đó sẽ được gọi bắt kè con trỏ chứa địa chỉ của đối tượng nào.

**Vấn đề đặt ra là:** Ta muốn tại thời điểm con trỏ đang trỏ đến đối tượng nào đó thi lời gọi hàm phải liên kết đúng hàm thành phần của lớp mà đối tượng đó thuộc vào chứ không phụ thuộc vào kiểu lớp của con trỏ. C++ giải quyết vấn đề này bằng cách dùng khái niệm *hàm ảo*.

#### 5.4.2. Định nghĩa hàm ảo

Hàm ảo là hàm thành phần của lớp, nó được *khai báo trong lớp cơ sở* và định nghĩa lại trong lớp dẫn xuất. Để định nghĩa hàm ảo thì phần khai báo hàm phải bắt đầu bằng từ khóa *virtual*. Khi một lớp có chứa hàm ảo được kế thừa, lớp dẫn xuất sẽ định nghĩa lại hàm ảo đó cho chính mình. Các hàm ảo triển khai tư tưởng chủ đạo của tính đa hình là “một giao diện cho nhiều hàm thành phần”. Hàm ảo bên trong lớp cơ sở định nghĩa hình thức giao tiếp đối với hàm đó. Việc định nghĩa lại hàm ảo ở lớp dẫn xuất là thi hành các tác vụ của hàm liên quan đến chính lớp dẫn xuất đó. Nói cách khác, định nghĩa lại hàm ảo chính là tạo ra phương thức cụ thể. Trong phần định nghĩa lại hàm ảo ở lớp dẫn xuất, không cần phải sử dụng lại từ khóa *virtual*.

Khi xây dựng hàm ảo, cần tuân theo những quy tắc sau :

1. Hàm ảo phải là hàm thành phần của một lớp ;
2. Những thành phần tĩnh (static) không thể khai báo ảo;
3. Sử dụng con trỏ để truy nhập tới hàm ảo;
4. Hàm ảo được định nghĩa trong lớp cơ sở, ngay khi nó không được sử dụng;

5. Mẫu của các phiên bản (ở lớp cơ sở và lớp dẫn xuất) phải giống nhau. Nếu hai hàm cùng tên nhưng có mẫu khác nhau thì C++ sẽ xem như hàm tái bội;

6. Không được tạo ra hàm tạo ảo, nhưng có thể tạo ra hàm hủy ảo;

7. Con trỏ của lớp cơ sở có thể chứa địa chỉ của đối tượng thuộc lớp dẫn xuất, nhưng ngược lại thì không được;

8. Nếu dùng con trỏ của lớp cơ sở để trỏ đến đối tượng của lớp dẫn xuất thì phép toán tăng giảm con trỏ sẽ không tác dụng đối với lớp dẫn xuất, nghĩa là không phải con trỏ sẽ trỏ tới đối tượng trước hoặc tiếp theo trong lớp dẫn xuất. Phép toán tăng giảm chỉ liên quan đến lớp cơ sở.

**Ví dụ:**

```
class A
{
    ...
    virtual void hienthi()
    {
        cout<<"\nDay la lop A";
    };
};

class B : public A
{
    ...
    void hienthi()
    {
        cout<<"\nDay la lop B";
    }
}
```

```

};

class C : public B

{
    ...

void hienthi()

{
    cout<<"\nDay la lop C";

}

};

class D : public A

{ ...

void hienthi()

{
    cout<<"\nDay la lop D";

}

};


```

**Chú ý:** Từ khoá virtual không được đặt bên ngoài định nghĩa lớp. Xem ví dụ:

```

class A

{
    ...

virtual void hienthi();

};

virtual void hienthi() // sai

{
```

```
cout<<"\nDay la lop A";
```

```
}
```

#### 5.4.3. Quy tắc gọi hàm ảo

Hàm ảo chỉ khác hàm thành phần thông thường khi được gọi từ một con trỏ. Lời gọi tới hàm ảo từ một con trỏ chưa cho biết rõ hàm thành phần nào (trong số các hàm thành phần cùng tên của các lớp có quan hệ thừa kế) sẽ được gọi. Điều này sẽ phụ thuộc vào đối tượng cụ thể mà con trỏ đang trỏ tới: *con trỏ đang trỏ tới đối tượng của lớp nào thì hàm thành phần của lớp đó sẽ được gọi.*

#### 5.4.4. Quy tắc gán địa chỉ đối tượng cho con trỏ lớp cơ sở

C++ cho phép gán địa chỉ đối tượng của một lớp dẫn xuất cho con trỏ của lớp cơ sở bằng cách sử dụng phép gán = và phép toán lấy địa chỉ &.

**Ví dụ :** Giả sử A là lớp cơ sở và B là lớp dẫn xuất từ A. Các phép gán sau là đúng:

A \*p; // p là con trỏ kiểu A

A a; // a là biến đối tượng kiểu A

B b; // b là biến đối tượng kiểu B

p = &a; // p và a cùng lớp A

p = &b; // p là con trỏ lớp cơ sở, b là đối tượng lớp dẫn xuất

**Chú ý:** Không cho phép gán địa chỉ đối tượng của lớp cơ sở cho con trỏ của lớp dẫn xuất, chẳng hạn với khai báo B \*q; A a; thì câu lệnh q = &a; là sai.

**Ví dụ 5.11.** Chương trình sau đây minh họa việc sử dụng hàm ảo:

```
#include <iostream.h>
#include <conio.h>
class A
{
```

```
public:  
    virtual void hienthi()  
    {  
        cout <<"\n Lop A";  
    }  
};  
  
class B : public A  
{  
public:  
    void hienthi()  
    {  
        cout <<"\n Lop B";  
    }  
};  
  
class C : public B  
{  
public:  
    void hienthi()  
    {  
        cout <<"\n Lop C";  
    }  
};  
  
void main()  
{ clrscr();  
    A *p;
```

```

A a; B b; C c;

a.hienthi(); //goi ham cua lop A

p = &b;           //p tro to doi tuong b cua lop B

p->hienthi(); //goi ham cua lop B

p=&c;           //p tro to doi tuong c cua lop C

p->hienthi(); //goi ham cua lop C

getch();

}

```

Chương trình này cho kết quả như sau:

Lop A

Lop B

Lop C

### **Chú ý :**

➤ Cùng một câu lệnh `p->hienthi();` được tương ứng với nhiều hàm khác nhau khi `hienthi()` là hàm ảo. Đây chính là *sự tương ứng bội*. Khả năng này cho phép xử lý nhiều đối tượng khác nhau theo cùng một cách thức.

➤ Cùng với lời gọi: `p->hienthi();` (`hienthi()` là hàm ảo) thì lời gọi này không liên kết với một phương thức cố định, mà tùy thuộc và nội dung con trỏ. Đó là *sự liên kết động* và phương thức được liên kết (được gọi) thay đổi mỗi khi có sự thay đổi nội dung con trỏ trong quá trình chạy chương trình.

**Ví dụ 5.12.** Chương trình sau tạo ra một lớp cơ sở có tên là **num** lưu trữ một số nguyên, và một hàm ảo của lớp có tên là `shownum()`. Lớp **num** có hai lớp dẫn xuất là **outhex** và **outoct**. Trong hai lớp này sẽ định nghĩa lại hàm ảo `shownum()` để chúng in ra số nguyên dưới dạng số hệ 16 và số hệ 8.

```

#include <iostream.h>
#include <conio.h>

```

```
class num
{
public :
    int i;
    num(int x) { i=x; }
    virtual void shownum()
    {
        cout<<"\n So he 10 : ";
        cout <<dec<<i<<'\n';
    }
};

class outhex : public num
{
public :
    outhex(int n) : num(n) {}
    void shownum()
    {
        cout <<"\n So he 10 : "<<dec<<i<<endl;
        cout <<"\n So he 16 : "<<hex << i <<'\n';
    }
};

class outoct : public num
{
public :
    outoct(int n) : num(n) {}
    void shownum()
    {
        cout <<"\n So he 10 : "<<dec<<i<<endl;
        cout <<"\n So he 8 : "<< oct << i <<'\n';
    }
};
```

```
};

void main()
{ clrscr();
    num n (1234);
    outoct o (342);
    outhex h (747);
    num *p;
    p=&n;
    p->shownum(); //goi ham cua lop co so, 100
    p=&o;
    p->shownum(); //goi ham cua lop dan xuat, 12
    p=&h;
    p->shownum(); //goi ham cua lop dan xuat, f
    getch();
}
```

Chương trình trên cho kết quả:

So he 10 : 1234

So he 10 : 342

So he 8 : 526

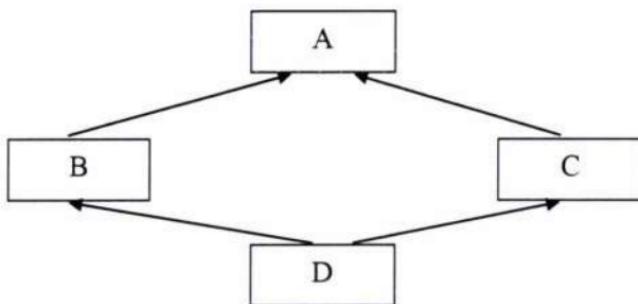
So he 10 : 747

So he 16 : 2eb

## 5.5. Lớp cơ sở ảo

### 5.5.1. Khai báo lớp cơ sở ảo

Một vấn đề tồn tại là khi nhiều lớp cơ sở được kế thừa trực tiếp bởi một lớp dẫn xuất. Để hiểu rõ hơn vấn đề này, xét tình huống các lớp kế thừa theo sơ đồ như sau:



Hình 5.4 Lớp cơ sở ảo

Ở đây, lớp A được kế thừa bởi hai lớp B và C. Lớp D kế thừa trực tiếp cả hai lớp B và C. Như vậy lớp A được kế thừa *hai lần* bởi lớp D: lần thứ nhất nó được kế thừa thông qua lớp B, và lần thứ hai được kế thừa thông qua lớp C. Bởi vì có hai bản sao của lớp A có trong lớp D nên một tham chiếu đến một thành phần của lớp A sẽ tham chiếu về lớp A được kế thừa gián tiếp thông qua lớp B hay tham chiếu về lớp A được kế thừa gián tiếp thông qua lớp C? Để giải quyết tình không rõ ràng này, C++ có một cơ chế mà nhờ đó chỉ có một bản sao của lớp A ở trong lớp D: đó là sử dụng lớp *cơ sở ảo*.

Trong ví dụ trên, C++ sử dụng từ khóa `virtual` để khai báo lớp A là ảo trong các lớp B và C theo mẫu như sau:

```
class B : virtual public A
```

```
{...};
```

```
class C : virtual public A
```

```
{...};
```

```
class D : public B, public C
```

```
{...};
```

Việc chỉ định A là ảo trong các lớp B và C nghĩa là A sẽ chỉ xuất hiện một lần trong lớp D. Khai báo này không ảnh hưởng đến các lớp B và C.

**Chú ý:** Từ khóa virtual có thể đặt trước hoặc sau từ khóa public, private, protected.

### Ví dụ 5.13.

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

class A
{
    float x,y;
public:
    void set(float x1, float y1)
    {
        x = x1; y = y1;
    }
    float getx()
    {
        return x;
    }
    float gety()
    {
        return y;
    }
};

class B : virtual public A
{ };
```

```

class C : virtual public A
{
};

class D : public B, public C
{
};

void main()
{
    clrscr();

    D d;

    cout<<"\nd.B::set(2,3)\n";
    d.B::set(2,3);

    cout<<"\nd.C::getx() = "; cout<<d.C::getx()<<endl;
    cout<<"\nd.B::getx() = "; cout<<d.B::getx()<<endl;
    cout<<"\nd.C::gety() = "; cout<<d.C::gety()<<endl;
    cout<<"\nd.B::gety() = "; cout<<d.B::gety()<<endl;
    cout<<"\nd.C::set(2,3)\n";
    d.C::set(2,3);

    cout<<"\nd.C::getx() = "; cout<<d.C::getx()<<endl;
    cout<<"\nd.B::getx() = "; cout<<d.B::getx()<<endl;
    cout<<"\nd.C::gety() = "; cout<<d.C::gety()<<endl;
    cout<<"\nd.B::gety() = "; cout<<d.B::gety()<<endl;
    getch();
}

```

Chương trình trên sẽ cho kết quả:

```

d.B::set(2,3)

d.C::getx() = 2
d.B::getx() = 2

```

```
d.C::gety() = 3  
d.B::gety() = 3  
d.C::set(2,3)  
d.C::getx() = 2  
d.B::getx() = 2  
d.C::gety() = 3  
d.B::gety() = 3
```

### 5.5.2. Hàm tạo và hàm hủy đối với lớp cơ sở ảo

Ta đã biết, khi khởi tạo đối tượng lớp dẫn xuất thì các hàm tạo được gọi theo thứ tự xuất hiện trong danh sách các lớp cơ sở được khai báo, rồi đến hàm tạo của lớp dẫn xuất. Thông tin được chuyển từ hàm tạo của lớp dẫn xuất sang hàm tạo của lớp cơ sở. Trong tình huống có lớp cơ sở ảo, chẳng hạn như hình vẽ 5.4., cần phải tuân theo quy định sau:

Thứ tự gọi hàm tạo: Hàm tạo của một lớp ảo luôn luôn được gọi trước các hàm tạo khác.

Với sơ đồ kế thừa như hình vẽ 5.4., thứ tự gọi hàm tạo sẽ là A, B, C và cuối cùng là D. Chương trình sau minh họa điều này:

#### Ví dụ 5.14

```
#include <iostream.h>  
#include <conio.h>  
  
class A  
{    float x,y;  
  
public:  
    A() {x = 0; y = 0;}  
    A(float x1, float y1)  
    {
```

```
    cout<<"A::A(float,float)\n";
    x = x1; y = y1;
}

float getx()
{
    return x;
}

float gety()
{
    return y;
}

};

class B : virtual public A
{
public:
    B(float x1, float y1):A(x1,y1)
    {
        cout<<"B::B(float,float)\n";
    }
};

class C : virtual public A
{
public:
    C(float x1, float y1):A(x1,y1)
```

```

    {
        cout<<"C::C(float,float)\n";
    }
};

class D : public B, public C
{
public:
    D(float x1, float y1):A(x1,y1), B(10,4), C(1,1)
    {
        cout<<"D::D(float,float)\n";
    }
};

void main()
{
    clrscr();
    D d(2,3);
    cout<<"\nD d (2,3)\n";
    cout<<"\nd.C::getx() = "; cout<<d.C::getx()<<endl;
    cout<<"\nd.B::getx() = "; cout<<d.B::getx()<<endl;
    cout<<"\nd.C::gety() = "; cout<<d.C::gety()<<endl;
    cout<<"\nd.B::gety() = "; cout<<d.B::gety()<<endl;
    cout<<"\ndl (10,20) \n";
    D dl (10,20);    cout<<"\ndl.C::getx() = "; cout<<d.C::getx()
<<endl;
    cout<<"\ndl.B::getx() = "; cout<<d.B::getx()<<endl;
}

```

```
    cout<<"\nd.C::gety() = "; cout<<d.C::gety()<<endl;
    cout<<"\nd.B::gety() = "; cout<<d.B::gety()<<endl;
    getch();
}
```

Kết quả chương trình trên như sau:

```
A::A(float,float)
B::B(float,float)
C::C(float,float)
D::D(float,float)

D d (2,3)
d.C::getx() = 2
d.B::getx() = 2
d.C::gety() = 3
d.B::gety() = 3

d1 (10,20)
A::A(float,float)
B::B(float,float)
C::C(float,float)
D::D(float,float)

d.C::getx() = 2
d.B::getx() = 2
d.C::gety() = 3
d.B::gety() = 3
```

## Bài tập

1. Xây dựng lớp có tên là Stack với các thao tác cần thiết. Từ đó hãy dẫn xuất từ lớp stack để đổi một số nguyên dương sang hệ đếm bất kỳ.
2. Viết một phân cấp kế thừa cho các lớp hình tứ giác, hình thang, hình bình hành, hình chữ nhật, hình vuông.
3. Tạo một lớp cơ sở có tên là airship để lưu thông tin về số lượng hành khách tối đa và trọng lượng hàng hóa tối đa mà máy bay có thể chở được. Từ đó hãy tạo hai lớp dẫn xuất airplane và balloon, lớp airplane lưu thông tin về kiểu động cơ (gồm động cơ cánh quạt và động cơ phản lực), lớp balloon lưu thông tin về loại nhiên liệu sử dụng cho khí cầu (gồm hai loại là hydrogen và helium). Hãy viết chương trình minh họa.
4. Một nhà xuất bản nhận xuất bản sách. Sách có hai loại: loại có hình ảnh ở trang bìa và loại không có hình ảnh ở trang bìa. Loại có hình ảnh ở trang bìa thì phải thuê họa sĩ vẽ bìa. Viết chương trình thực hiện các yêu cầu:
  - Tạo một lớp cơ sở có tên là SACH để lưu thông tin về tên sách, tác giả, số trang, giá bán và định nghĩa hàm thành phần cho phép nhập dữ liệu cho các đối tượng của lớp SACH.
  - Tạo lớp BIA kế thừa từ lớp SACH để lưu các thông tin: Mã hình ảnh, tiền vẽ và định nghĩa hàm thành phần cho phép nhập dữ liệu cho các đối tượng của lớp BIA.
  - Tạo lớp HOASY để lưu các thông tin họ tên, địa chỉ của họa sỹ và định nghĩa hàm thành phần cho phép nhập dữ liệu cho các đối tượng của lớp HOASY.
  - Tạo lớp SACHVEBIA kế thừa từ lớp BIA và lớp HOASY và định nghĩa hàm thành phần cho phép nhập dữ liệu cho các đối tượng của lớp SACHVEBIA. Viết hàm main() cho phép nhập vào hai danh sách: danh sách các sách có vẽ bìa và danh sách các sách không có vẽ bìa (có thể dùng mảng tĩnh hoặc mảng con trỏ).

5. Xây dựng lớp hình vuông có tên là HVUONG với các dữ liệu thành phần như sau: độ dài cạnh. Các hàm thành phần để nhập dữ liệu, hiển thị dữ liệu, tính diện tích, chu vi hình vuông. Từ lớp HVUONG, xây dựng lớp dẫn xuất có tên là CHUNHAT, là lớp kế thừa của lớp HVUONG và được bổ sung thêm thuộc tính sau: độ dài cạnh thứ hai. Các hàm thành phần để nhập dữ liệu, hiển thị dữ liệu để tính diện tích và chu vi hình chữ nhật. Viết chương trình minh họa.

6. Xây dựng lớp cơ sở CANBO có dữ liệu thành phần là mã cán bộ, mã đơn vị, họ tên, ngày sinh. Các hàm thành phần bao gồm: nhập dữ liệu cán bộ, hiển thị dữ liệu. Lớp dẫn xuất LUONG kế thừa lớp CANBO và có thêm các thuộc tính: phụ cấp, hệ số lương, bảo hiểm. Hàm thành phần để tính lương cán bộ theo công thức:

$$\text{lương} = \text{hệ số lương} * 290000 + \text{phụ cấp} - \text{bao hiểm}$$

Hãy thiết kế chương trình để đáp ứng các yêu cầu:

- Nhập danh sách cán bộ
- In bảng lương các cán bộ theo từng đơn vị.

7. Nhân viên trong một cơ quan được lĩnh lương theo các dạng khác nhau. Dạng người lao động hưởng lương từ ngân sách Nhà nước gọi là cán bộ, công chức (dạng biên chế). Dạng người lao động lĩnh lương từ ngân sách của cơ quan gọi là người làm hợp đồng. Như vậy hệ thống có hai đối tượng: biên chế và hợp đồng.

- Hai loại đối tượng này có đặc tính chung là viên chức làm việc cho cơ quan. Từ đây có thể tạo nên lớp cơ sở để quản lý một viên chức (lớp Nguoi) bao gồm mã số, họ tên, lương.

- Hai lớp kế thừa từ lớp cơ sở trên:

+ Lớp Bienche gồm các thuộc tính: hệ số lương, tiền phụ cấp chức vụ.

+ Lớp Hopdong gồm các thuộc tính: tiền công lao động, số ngày làm việc trong tháng, hệ số vượt giờ.

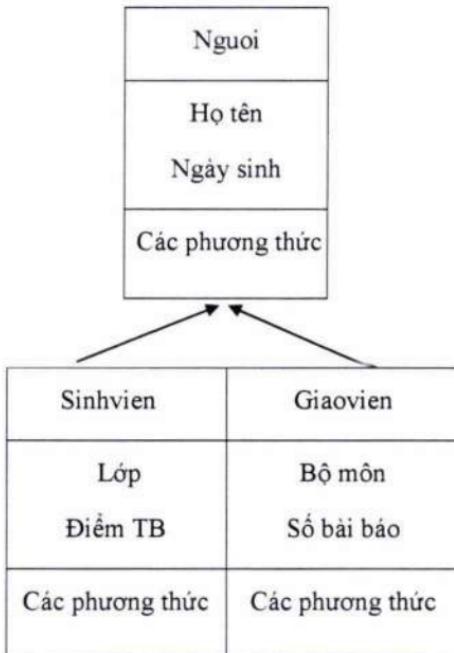
Hãy thiết kế các lớp trên và viết chương trình minh họa.

8. Viết chương trình quản lý sách báo, tạp chí của thư viện trong trường đại học, hằng tháng gửi về khoa họ tên của các giáo viên và sinh viên đã quá thời hạn mượn sách.

9. Viết chương trình tính và in bảng lương hàng tháng của giáo viên và người làm hợp đồng trong một trường đại học. Giả sử việc tính toán tiền lương được căn cứ vào các yếu tố sau:

- Đối với giáo viên: số tiết dạy trong tháng, tiền dạy một tiết.
- Đối với người làm hợp đồng: tiền công ngày, số ngày làm việc

10. Giả sử cuối năm học cần trao phần thưởng cho các sinh viên giỏi, các giáo viên có tham gia nghiên cứu khoa học. Điều kiện khen thưởng của sinh viên là có điểm trung bình lớn hơn 8. Điều kiện khen thưởng của giáo viên là có ít nhất một bài báo nghiên cứu khoa học. Sơ đồ cấu trúc phân cấp lớp như sau:

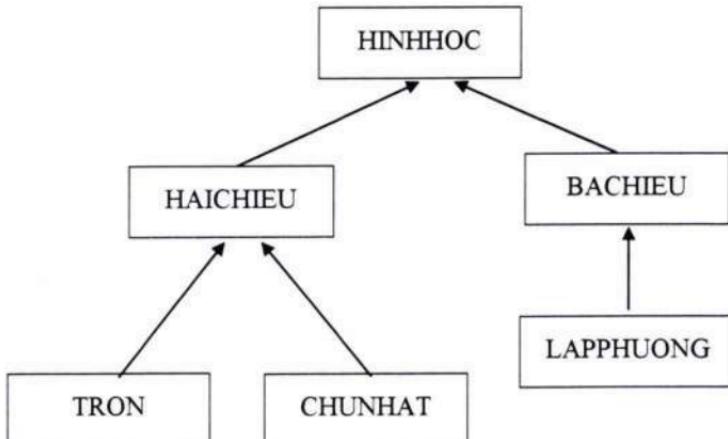


### Yêu cầu:

- Xây dựng các lớp theo sơ đồ kế thừa ở trên, mỗi lớp có các hàm thành phần để nhập, xuất dữ liệu, hàm kiểm tra khen thưởng.

- Hãy viết hàm main() cho phép nhập vào dữ liệu của không quá 100 sinh viên và không quá 30 giáo viên. In ra danh sách sinh viên và giáo viên được khen thưởng.

11. Giả sử ta có sơ đồ kế thừa của các lớp như sau:



### Yêu cầu:

- Thiết kế các lớp để có thể in ra các thông tin của các hình (tròn, chữ nhật, lập phương) bao gồm: diện tích, chu vi, thể tích.

- Viết chương trình minh họa.

12. Viết chương trình quản lý việc mượn và trả sách ở một thư viện theo phương pháp lập trình hướng đối tượng. Chương trình cho phép:

- Đăng ký bạn đọc mới với thông tin là mã và tên bạn đọc, số điện thoại
- Nhập sách mới với thông tin là mã sách, tên sách, số lượng, nhà xuất bản.
- Mượn và trả sách.
- Thống kê bạn đọc.
- Thống kê sách.

## Chương 6

### KHUÔN HÌNH

#### 6.1. Khuôn hình hàm

##### 6.1.1. Khái niệm

Ta đã biết hàm quá tải cho phép dùng một tên duy nhất cho nhiều hàm để thực hiện các công việc khác nhau. Khái niệm khuôn hình hàm cũng cho phép sử dụng cùng một tên duy nhất để thực hiện các công việc khác nhau, tuy nhiên so với định nghĩa hàm quá tải, nó có phần mạnh hơn và chặt chẽ hơn. Mạnh hơn vì chỉ cần viết định nghĩa khuôn hình hàm một lần, rồi sau đó chương trình biên dịch làm cho nó thích ứng với các kiểu dữ liệu khác nhau. Chặt chẽ hơn bởi vì dựa theo khuôn hình hàm, tất cả các hàm thể hiện được sinh ra bởi chương trình dịch sẽ tương ứng với cùng một định nghĩa và như vậy sẽ có cùng một giải thuật.

##### 6.1.2. Tạo một khuôn hình hàm

Giả thiết rằng chúng ta cần viết một hàm **min** đưa ra giá trị nhỏ nhất trong hai giá trị có cùng kiểu. Ta có thể viết một định nghĩa như thế với kiểu **int** như sau:

```
int min (int a, int b)
{
    if (a<b)
        return a;
    else
        return b;
}
```

Nếu ta muốn sử dụng hàm min cho kiểu double, float, char,... ta lại phải viết lại định nghĩa hàm **min**, ví dụ:

```
float min (float a, float b)
{
    if (a < b)
        return a;
    else
        return b;
}
```

Như vậy ta phải viết rất nhiều định nghĩa hàm hoàn toàn tương tự nhau, chỉ có kiểu dữ liệu là thay đổi. Chương trình dịch C++ cho phép giải quyết đơn giản vấn đề trên bằng cách định nghĩa một khuôn hình hàm duy nhất theo cú pháp:

```
template <danh sách tham số kiểu> <kiểu trả về>
tên hàm(khai báo tham số)

{
    // định nghĩa hàm
}
```

trong đó <danh sách tham số kiểu> là các kiểu dữ liệu được khai báo với từ khoá *class*, cách nhau bởi dấu phẩy. Kiểu dữ liệu là một kiểu bất kỳ, kể cả kiểu *class*.

**Ví dụ 6.1.** Xây dựng khuôn hình cho hàm tìm giá trị nhỏ nhất của hai số:

```
template <class Kieuso> Kieuso min(Kieuso a, Kieuso b)
{
    if (a<b)
        return a;
```

```
    else  
        return b;  
}
```

### 6.1.3. Sử dụng khuôn hình hàm

Để sử dụng khuôn hình hàm **min** vừa tạo ra, chỉ cần sử dụng hàm **min** ng những điều kiện phù hợp, trong trường hợp này là hai tham số của hàm **ii** cùng kiểu dữ liệu. Như vậy, nếu trong một chương trình có hai tham số **iyen n và m** (kiểu int) với lời gọi **min(n,m)** thì chương trình dịch tự động sản h ra hàm **min()**, gọi là một hàm thể hiện, tương ứng với hai tham số kiểu **float**. Nếu chúng ta gọi **min()** với hai tham số kiểu **float**, chương trình biên dịch **ig** tự động sản sinh một hàm thể hiện **min** khác tương ứng với các tham số **u float** và cứ thế với các kiểu dữ liệu khác.

#### Chú ý:

- Các biến truyền cho danh sách tham số của hàm phải chính xác với kiểu i báo.
- Muốn áp dụng được với kiểu lớp thì trong lớp phải định nghĩa các toán ài bởi tương ứng.

### 6.1.4. Các tham số kiểu của khuôn hình hàm

Khuôn hình hàm có thể có một hay nhiều tham số kiểu, mỗi tham số đi sau từ khoá class. Các tham số này có thể ở bất kỳ đâu trong định nghĩa khuôn hình hàm, nghĩa là :

- Trong dòng tiêu đề (ở dòng đầu khai báo template).
- Trong các khai báo biến cục bộ.
- Trong các chỉ thị thực hiện.

Trong mọi trường hợp, mỗi tham số kiểu phải xuất hiện ít nhất một lần g khai báo danh sách tham số hình thức của khuôn hình hàm. Điều đó hoàn

tôan logic, bởi vì nhờ các tham số này, chương trình dịch mới có thể sà ra hàm thể hiện cần thiết.

Ở khuôn hình hàm min trên, mới chỉ cho phép tìm min của hai số kiểu, nếu muốn tìm min hai số khác kiểu thì khuôn hình hàm trên chưa đáp ứng được. Ví dụ sau sẽ khắc phục được điều này.

### Ví dụ 6.2.

```
#include <iostream.h>

template <class kieusol, class kieuso2>
kieusol min(kieusol a, kieuso2 b)

{
    return a<b ? a : b;
}

void main(){
    float a = 2.5;
    int b = 8;
    cout << "so nho nhat la :" << min(a,b);
}
```

**Ví dụ 6.3.** Giả sử trong lớp SO các số int đã xây dựng, ta có xây các toán tử tài bội `<`, `<<` cho các đối tượng của class SO (xem chương 4) dung file ttclsso.h như sau:

```
class SO

{
private:
    int giatri;
public:
    SO(int x=0)
```

```

{
    giatri = x;
}
SO (SO &tso)
{
    giatri = tso.giatri;
}
SO (){}; //Giong nhu ham thiet lap ngam dinh
~SO()
{
}
int operator<(SO & s)
{
    return (giatri <s.giatri);
}
}

friend istream& operator>>(istream&, SO&);
friend ostream& operator<<(ostream&, SO&);
};

Chương trình sau đây cho phép thử hàm min trên hai đối tượng kiểu class:
```

**Ví dụ 6.4** Chương trình sau đây cho phép thử hàm min trên hai đối tượng của class:

```
#include <iostream.h>
#include <ttclss.h>
template <class kieusol, class kieuso2> kieusol
min(kieusol a, kieuso2 b)
```

```

{
    if (a<b)
        return a;
    else
        return b;
}

void main(){
    float a =2.5;
    int b = 8;
    cout << "so nho nhat la :" << min(a,b)<<endl;
    SO so1(15),so2(20);
    cout << "so nho nhat la :" << min(so2,so1);
}

```

#### *6.1.5. Định nghĩa chồng các khuôn hình hàm*

Tương tự việc định nghĩa các hàm quá tải, C++ cho phép định nghĩa chồng các khuôn hình hàm, tức là có thể định nghĩa một hay nhiều khuôn hình hàm có cùng tên nhưng với các tham số khác nhau. Điều đó sẽ tạo ra nhiều họ các hàm (mỗi khuôn hình hàm tương ứng với họ các hàm).

Ví dụ có ba họ hàm min :

- Một họ gồm các hàm tìm giá trị nhỏ nhất trong hai giá trị
- Một họ gồm các hàm tìm giá trị nhỏ nhất trong ba giá trị
- Một họ gồm các hàm tìm giá trị nhỏ nhất trong một mảng giá trị.

Một cách tổng quát, ta có thể định nghĩa một hay nhiều khuôn hình cùng tên, mỗi khuôn hình có các tham số kiểu cũng như là các tham số biểu thức riêng. Hơn nữa, có thể cung cấp các hàm thông thường với cùng tên với cùng

một khuôn hình hàm, trong trường hợp này ta nói đó là sự cụ thể hoá một hàm thể hiện.

Trong trường hợp tổng quát khi có đồng thời cả hàm quá tải và khuôn hình hàm, chương trình dịch lựa chọn hàm tương ứng với một lời gọi hàm dựa trên các nguyên tắc sau:

Đầu tiên, kiểm tra tất cả các hàm thông thường cùng tên và chú ý đến sự tương ứng chính xác; nếu chỉ có một hàm phù hợp, hàm đó được chọn; Còn nếu có nhiều hàm cùng thỏa mãn sẽ tạo ra một lỗi biên dịch và quá trình tìm kiếm bị gián đoạn.

Nếu không có hàm thông thường nào tương ứng chính xác với lời gọi, khi đó ta kiểm tra tất cả các khuôn hình hàm có trùng tên với lời gọi, khi đó ta kiểm tra tất cả các khuôn hình hàm có trùng tên với lời gọi; nếu chỉ có một tương ứng chính xác được tìm thấy, hàm thể hiện tương ứng được sản sinh và vẫn đề được giải quyết; còn nếu có nhiều hơn một khuôn hình hàm điều đó sẽ gây ra lỗi biên dịch và quá trình dừng.

Cuối cùng, nếu không có khuôn hình hàm phù hợp, ta kiểm tra một lần nữa tất cả các hàm thông thường cùng tên với lời gọi. Trong trường hợp này chúng ta phải tìm kiếm sự tương ứng dựa vào cả các chuyển kiều cho phép trong C/C++.

## 6.2. Khuôn hình lớp

### 6.2.1. Khái niệm

Bên cạnh khái niệm khuôn hình hàm, C++ còn cho phép định nghĩa khuôn hình lớp. Cũng giống như khuôn hình hàm, ở đây ta chỉ cần viết định nghĩa các khuôn hình lớp một lần rồi sau đó có thể áp dụng chúng với các kiểu dữ liệu khác nhau để được các lớp thể hiện khác nhau.

### 6.2.2. Tạo một khuôn hình lớp

Trong chương trước ta đã định nghĩa cho lớp SO, giá trị các số là kiểu **int**. Nếu ta muốn làm việc với các số kiểu **float**, **double**,... thì ta phải định nghĩa lại một lớp khác tương tự, trong đó kiểu dữ liệu **int** cho dữ liệu **giatri** sẽ được thay bằng **float**, **double**,...

Để tránh sự trùng lặp trong các tinh huống như trên, chương trình dịch C++ cho phép định nghĩa một khuôn hình lớp và sau đó, áp dụng khuôn hình lớp này với các kiểu dữ liệu khác nhau để thu được các lớp thể hiện như mong muốn.

#### Ví dụ:

```
template <class kieuso> class SO
```

```
{
```

```
    kieuso giatri;
```

```
    public :
```

```
        SO (kieuso x =0);
```

```
        void Hienthi();
```

```
...
```

```
};
```

Cũng giống như các khuôn hình hàm, template <class kieuso> xác định rằng đó là một khuôn hình trong đó có một tham số kiểu kieuso . C++ sử dụng từ khóa **class** chỉ để nói rằng kieuso đại diện cho một kiểu dữ liệu nào đó.

Việc định nghĩa các hàm thành phần của khuôn hình lớp, người ta phân biệt hai trường hợp:

Khi hàm thành phần được định nghĩa bên trong định nghĩa lớp thì không có gì thay đổi.

Khi hàm thành phần được định nghĩa bên ngoài lớp, khi đó cần phải nhắc lại cho chương trình biết các tham số kiểu của khuôn hình lớp, có nghĩa là phải nhắc lại template <class kieuso> chẳng hạn, trước định nghĩa hàm. Ví dụ hàm Hienthi() được định nghĩa ngoài lớp:

```
template <class kieuso> void SO<kieuso>::Hienthi()
{
    cout <<giatri;
}
```

#### 6.2.3. Sử dụng khuôn hình lớp

Sau khi một khuôn hình lớp đã được định nghĩa, nó sẽ được dùng để khai báo các đối tượng theo dạng sau :

Tên\_lớp <Kiểu> Tên\_đối\_tượng;

Ví dụ câu lệnh khai báo SO <int> so1; sẽ khai báo một đối tượng so1 có thành phần dữ liệu giatri có kiểu nguyên int.

SO <int> có vai trò như một kiểu dữ liệu lớp; người ta gọi nó là một lớp thể hiện của khuôn hình lớp SO. Một cách tổng quát, khi áp dụng một kiểu dữ liệu nào đó với khuôn hình lớp SO ta sẽ có được một lớp thể hiện tương ứng với kiểu dữ liệu.

Tương tự với các khai báo SO <float> so2; cho phép khai báo một đối tượng so2 mà thành phần dữ liệu giatri có kiểu float.

#### Ví dụ 6.5.

```
#include <iostream.h>
#include <conio.h>
template <class kieuso> class SO
{
    kieuso giatri;
```

```
public :  
    SO (kieuso x =0);  
  
    void Hienthi(){  
        cout<<"Gia tri cua so :"<<giatri<<endl;  
    }  
};  
  
void main(){  
    clrscr();  
    SO <int> soint(10); soint.Hienthi();  
    SO <float> sofl(25.4); sofl.Hienthi();  
    getch();  
}
```

Kết quả trên màn hình là:

Gia tri cua so: 10

Gia tri cua so: 25.4

#### 6.2.4. Các tham số trong khuôn hình lớp

Hoàn toàn giống như khuôn hình hàm, các khuôn hình lớp có thể có các tham số kiểu và tham số biểu thức.

Ví dụ một lớp mà các thành phần có các kiểu dữ liệu khác nhau được khai báo theo dạng:

```
template <class T, class U,... class Z>  
class <ten lop>{  
    T x;  
    U y;  
    ....
```

```
    z fct1 (int);  
    ....  
};
```

Một lớp thể hiện được khai báo bằng cách liệt kê đăng sau tên khuôn hình lớp các tham số thực, là tên kiểu dữ liệu, với số lượng bằng các tham số trong danh sách của khuôn hình lớp (template<...>)

#### 6.2.5. *Tóm tắt*

Khuôn hình lớp/hàm là phương tiện mô tả ý nghĩa của một lớp/hàm tổng quát, còn lớp/hàm thể hiện là một bản sao của khuôn hình tổng quát với các kiểu dữ liệu cụ thể.

Các khuôn hình lớp/hàm thường được tham số hoá. Tuy nhiên vẫn có thể sử dụng các kiểu dữ liệu cụ thể trong các khuôn hình lớp/hàm nếu cần.

## Bài tập

1. Viết khuôn hình hàm để tìm số lớn nhất của hai số bất kỳ
2. Viết khuôn hình hàm để trả về giá trị trung bình của một mảng, các tham số hình thức của hàm này là tên mảng, kích thước mảng.
3. Cài đặt hàng đợi template.
4. Viết khuôn hình hàm để sắp xếp kiểu dữ liệu bất kỳ.
5. Xây dựng khuôn hình lớp cho các tọa độ điểm trong mặt phẳng, các thành phần dữ liệu của lớp là toadox, toadoy.
6. Xây dựng khuôn hình lớp cho vector để quản lý các vector có thành phần có kiểu tùy ý.

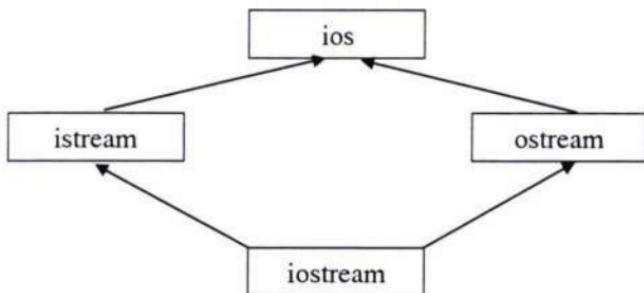
## PHỤ LỤC 1

### Các dòng xuất nhập

C++ sử dụng khái niệm dòng (stream) và đưa ra các lớp dòng để tổ chức việc nhập xuất. Dòng có thể xem như một dãy tuần tự các byte. Thao tác nhập là đọc các byte từ dòng (gọi là dòng nhập – input) vào bộ nhớ. Thao tác xuất là đưa các byte từ bộ nhớ ra dòng (gọi là dòng xuất- output). Các thao tác này là độc lập thiết bị. Để thực hiện việc nhập, xuất lên một thiết bị cụ thể, chúng ta chỉ cần gắn dòng tin với thiết bị này.

#### 1.1. Các lớp stream

Có 4 lớp stream quan trọng là: lớp cơ sở ios, từ lớp ios dẫn xuất đến hai lớp istream và ostream. Hai lớp istream và ostream lại dẫn xuất với lớp ostream. Sơ đồ kế thừa giữa các lớp như sau;



➤ Lớp **ios**: định nghĩa các thuộc tính được sử dụng làm các cờ định dạng cho việc xuất nhập và các cờ kiểm tra lỗi, các phương thức của lớp ios phục vụ việc định dạng dữ liệu nhập xuất, kiểm tra lỗi.

➤ Lớp **iostream**: cung cấp toán tử nhập >> và nhiều phương thức nhập khác, chẳng hạn các phương thức: get, getline, read, ignore, peek, seekg, tellg, ...

➤ Lớp **ostream**: cung cấp toán tử xuất << và nhiều phương thức xuất khác, chẳng hạn các phương thức: put, write, flush, seekp, tellp, ...

➤ Lớp **iostream**: thừa kế các phương thức nhập của các lớp istream và ostream.

## 1.2. Dòng cin và toán tử nhập >>

### 1.2.1 Dòng cin

**cin** là đối tượng của lớp **istream**. Đó là dòng nhập và được nói là “bị ràng buộc tới” hoặc kết nối tới thiết bị nhập chuẩn, thông thường là bàn phím. Các thao tác nhập trên dòng **cin** đồng nghĩa với nhập dữ liệu từ bàn phím.

### 1.2.2. Toán tử trích >>

Toán tử trích >> được sử dụng như sau để đọc dữ liệu từ dòng **cin**:

**cin>> biến;**

Để nhập giá trị của nhiều biến trên một dòng lệnh ta dùng cú pháp sau:

**cin>>biến 1>>biến 2>>...>>biến n;**

## 1.3. Nhập ký tự và chuỗi ký tự

Có thể dùng các phương thức sau (định nghĩa trong lớp **istream**) để nhập ký tự và chuỗi: **cin.get**    **cin.getline**    **cin.ignore**

### 1.3.1. Phương thức **get()** có 3 dạng:

**Dạng 1:** `int cin.get();`

Dùng để đọc một ký tự (kè cả khoảng trắng).

**Dạng 2:** `istream& cin.get(char &ch);`

Dùng để đọc một ký tự (kè cả khoảng trắng) và đặt vào một biến kiểu `char` được tham chiếu bởi `ch`.

**Dạng 3:** `istream& cin.get(char *str, int n, char d = '\n');`

Dùng để đọc một dãy ký tự (kè cả khoảng trắng) và đưa vào vùng nhớ do `str` trả tới. Quá trình đọc kết thúc khi xảy ra một trong hai tình huống sau:

+ Gặp ký tự giới hạn (cho trong `d`). Ký tự giới hạn mặc định là ‘\n’.

+ Đã nhận đủ (`n-1`) ký tự.

### **Chú ý:**

- + Ký tự kết thúc chuỗi ‘\0’ được bổ sung vào cuối chuỗi nhận được.
- + Ký tự giới hạn vẫn còn lại trên dòng nhập để dành cho các lệnh nhận tiếp theo.
  - + Ký tự <Enter> còn lại trên dòng nhập có thể làm trôi phương thức get() sang 3.

**Ví dụ:** Xét đoạn chương trình:

```
char hoten[25], diachi[50], quequan[30] ;  
  
cout<<"\nHọ tên:";  
  
cin.get(ht,25);  
  
cout<<"\nĐịa chỉ : ";  
  
cin.get(diachi,50);  
  
cout<<"\nQuê quán : ";  
  
cin.get(quequan,30);  
  
cout <<"\n" <<hoten<<" "<<diachi<<" "<<quequan;
```

Đoạn chương trình dùng để nhập họ tên, địa chỉ và quê quán. Nếu gõ ‘ào Nguyen van X <Enter> thì câu lệnh get đầu tiên sẽ nhận được chuỗi ‘Nguyen van X’ cắt vào mảng hoten. Ký tự <Enter> còn lại sẽ làm trôi 2 câu lệnh get tiếp theo. Do đó câu lệnh cuối cùng sẽ chỉ in ra Nguyen van X.

Để khắc phục tình trạng trên, có thể dùng một trong các cách sau:

- + Dùng phương thức get() dạng 1 hoặc dạng 2 để lấy ra ký tự <Enter> trên dòng nhập trước khi dùng get (dạng 3).
- + Dùng phương thức ignore để lấy ra một số ký tự không cần thiết trên dòng nhập trước khi dùng get dạng 3. Phương thức này viết như sau:

```
cin.ignore(n); // Lấy ra (loại ra hay loại bỏ) n ký tự trên dòng nhập.
```

Như vậy để có thể nhập được cả quê quán và cơ quan, cần sửa lại đoạn chương trình trên như sau:

```
char hoten[25], diachi[50], quequan[30] ;  
  
cout<<"\nHọ tên : ";  
  
cin.get(hoten,25);  
  
cin.get(); // Nhấn < Enter>  
  
cout<<"\nĐịa chỉ : ";  
  
cin.get(diachi,50);  
  
ignore(1); // Bỏ qua <Enter>  
  
cout<<"\nQuê quán : ";  
  
cin.get(quequan,30);  
  
cout <<"\n" <<hoten<<" "<<diachi<<" "<<quequan;
```

### 1.3.2. Phương thức getline()

Phương thức getline để nhập một dãy ký tự từ bàn phím, được mô tả như sau:

```
istream& cin.getline(char *str, int n, char d = '\n');
```

#### Ví dụ:

```
char hoten[25], diachi[50];  
  
cout<<"\nHọ tên:";  
  
cin.getline(hoten,25);  
  
cout<<"\nĐịa chỉ";  
  
cin.getline(diachi,50);  
  
cout <<"\n" <<hoten<<" "<<diachi;
```

### *1.3.3. Phương thức ignore*

Phương thức ignore dùng để bỏ qua (loại bỏ) một số ký tự trên dòng nhập. Trong nhiều trường hợp, đây là việc làm cần thiết để không làm ảnh hưởng đến các phép nhập tiếp theo. Phương thức ignore được mô tả như sau:

```
istream& cin.ignore(int n=1);
```

Phương thức sẽ bỏ qua (loại bỏ) n ký tự trên dòng nhập.

## **1.4. Dòng cout và toán tử xuất <<**

### *1.4.1. Dòng cout*

**cout** là một đối tượng kiểu ostream và được nói là “bị ràng buộc tới” thiết bị chuẩn, thông thường là màn hình. Các thao tác xuất trên dòng cout đồng nghĩa với xuất dữ liệu ra màn hình.

### *1.4.2. Toán tử xuất <<*

C++ định nghĩa chồng toán tử dịch trái << để gửi các ký tự sang dòng xuất.

Cách dùng toán tử xuất để xuất dữ liệu từ bộ nhớ ra dòng cout như sau:

```
cout<<biểu thức;
```

Trong đó biểu thức biểu thị một giá trị cần xuất ra màn hình. Giá trị sẽ được biến đổi thành một dãy ký tự trước khi đưa ra dòng xuất.

**Chú ý:** Để xuất nhiều giá trị trên một dòng lệnh, có thể viết như sau:

```
cout<<biểu thức 1<<biểu thức 2 <<...<<biểu thức n;
```

### *1.4.3. Các phương thức định dạng*

#### **1. Phương thức int cout.width();**

cho biết độ rộng quy định hiện tại.

#### **2. Phương thức int cout.width(int n);**

thiết lập độ rộng quy định mới là n và trả về độ rộng quy định trước đó.

**Chú ý:** Độ rộng quy định n chỉ có tác dụng cho một giá trị xuất. Sau đó C++ lại áp dụng độ rộng quy định bằng 0. Ví dụ với các câu lệnh:

```
int m=1234, n=56;  
cout<<"\nAB";  
cout.width(6);  
cout<<m;  
cout<<n;
```

thì kết quả là:

AB 123456

(giữa B và số 1 có 2 dấu cách).

### **3. Phương thức int cout.precision();**

cho biết độ chính xác hiện tại (đang áp dụng để xuất các giá trị thực).

### **4. Phương thức int cout.precision(int n);**

thiết lập độ chính xác sẽ áp dụng là n và cho biết độ chính xác trước đó. Độ chính xác được thiết lập sẽ có hiệu lực cho tới khi gặp một câu lệnh thiết lập độ chính xác mới.

### **5. Phương thức char cout.fill();**

cho biết ký tự động hiện tại đang được áp dụng.

### **6. Phương thức char cout.fill( char ch);**

quy định ký tự động mới sẽ được dùng là ch và cho biết ký tự độn đang dùng trước đó. Ký tự độn được thiết lập sẽ có hiệu lực cho tới khi gặp một câu lệnh chọn ký tự độn mới.

#### **Ví dụ:**

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

```

void main()
{
    clrscr();
    float x=-3.1551, y=-23.45421;
    cout.precision(2);
    cout.fill('*');
    cout<<"\n";
    cout.width(8);
    cout<<x;
    cout<<"\n";
    cout.width(8);
    cout<<y;
    getch();
}

```

Sau khi thực hiện, chương trình in ra màn hình 2 dòng sau:

\*\*\*-3.16

\*\*-23.45

#### *1.4.4. Cờ định dạng*

Mỗi cờ định dạng chứa trong một bit. Cờ có 2 trạng thái: Bật (on) – có giá trị 1, Tắt (off) – có giá trị 0. Các cờ có thể chứa trong một biến kiểu long. Trong tập tin iostream.h đã định nghĩa các cờ sau:

ios::left	ios::right	ios::internal
ios::dec	ios::oct	ios::hex
ios::fixed	ios::scientific	ios::showpos
ios::uppercase	ios::showpoint	ios::showbase

Có thể chia cờ định dạng thành các nhóm:

## Nhóm 1 gồm các cờ căn lề:

ios::left              ios::right              ios::internal

**Cờ ios::left:** khi bật cờ ios::left thì giá trị in ra nằm bên trái vùng quy định, các ký tự độn nằm sau.

**Cờ ios::right:** khi bật cờ ios::right thì giá trị in ra nằm bên phải vùng quy định, các ký tự độn nằm trước.

Chú ý mặc định cờ ios::right bật.

**Cờ ios::internal:** cờ ios::internal có tác dụng giống như cờ ios::right chỉ khác là dấu (nếu có) in đầu tiên.

Chương trình sau minh họa cách dùng các cờ căn lề:

### Ví dụ:

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

void main()
{
    clrscr();
    float x=-87.1551, y=23.45421;
    cout.precision(2);
    cout.fill('*');
    cout.setf(ios::left); //bat co ios::left
    cout<<"\n";
    cout.width(8);
    cout<<x;
    cout<<"\n";
    cout.width(8);
```

```
cout<<y;  
cout.setf(ios::right); //bat co ios::right  
cout<<"\n";  
cout.width(8);  
cout<<x;  
cout<<"\n";  
cout.width(8);  
cout<<y;  
cout.setf(ios::internal); //bat co ios::internal  
cout<<"\n";  
cout.width(8);  
cout<<x;  
cout<<"\n";  
cout.width(8);  
cout<<y;  
getch();  
}
```

Sau khi thực hiện chương trình in ra 6 dòng như sau:

-87.16\*\*

23.45\*\*

\*\*-87.16

\*\*\*23.45

-\*\*87.16

\*\*\*23.45

## Nhóm 2 gồm các cờ định dạng số nguyên:

ios::dec

ios::oct

ios::hex

+ Khi ios::dec bật (mặc định): số nguyên được in dưới dạng cơ số 10

+ Khi ios::oct bật: số nguyên được in dưới dạng cơ số 8

+ khi ios::hex bật: số nguyên được in dưới dạng cơ số 16

## Nhóm 3 gồm các cờ định dạng số thực:

ios::fixed

ios::scientific

ios::showpoint

Mặc định : cờ ios::fixed bật (on) và cờ ios::showpoint tắt (off).

+ Khi ios::fixed bật và cờ ios::showpoint tắt thì số thực in ra dưới dạng thập phân, số chữ số phần phân (sau dấu chấm) được tính bằng độ chính xác n nhưng khi in thì bỏ đi các chữ số 0 ở cuối.

Ví dụ nếu độ chính xác n = 4 thì

Số thực -87.1500 được in: -87.15

Số thực 23.45425 được in: 23.4543

Số thực 678.0 được in: 678

+ Khi ios::fixed bật và cờ ios::showpoint bật thì số thực in ra dưới dạng thập phân, số chữ số phần phân (sau dấu chấm) được in ra đúng bằng độ chính xác n.

Ví dụ nếu độ chính xác n = 4 thì

Số thực -87.1500 được in: -87.1500

Số thực 23.45425 được in: 23.4543

Số thực 678.0 được in: 6780000

+ Khi ios::scientific bật và cờ ios::showpoint tắt thì số thực in ra dưới dạng khoa học. Số chữ số phần phân (sau dấu chấm) được tính bằng độ chính xác n nhưng khi in thì bỏ đi các chữ số 0 ở cuối.

Ví dụ nếu độ chính xác n=4 thì

Số thực -87.1500 được in: -87.15e+01

Số thực 23.45425 được in: 23.4543e+01

Số thực 678.0 được in: 678e+02

+ Khi ios::scientific bật và cờ ios::showpoint bật thì số thực in ra dưới dạng mũ. Số chữ số phần phân (sau dấu chấm) của phần định trị được in đúng bằng độ chính xác n.

Ví dụ nếu độ chính xác n=4 thì

Số thực -87.1500 được in: -87.150e+01

Số thực 23.45425 được in: 23.4543e+01

Số thực 678.0 được in: 67800e+01

#### Nhóm 4 gồm các hiển thị:

ios::uppercase      ios::showpos      ios::showbase

Cờ ios::showpos

+ Nếu cờ ios::showpos tắt (mặc định) thì dấu cộng không được in trước số dương.

+ Nếu cờ ios::showpos tắt thì dấu cộng được in trước số dương.

Cờ ios::showbase bật thì số nguyên hệ 8 được in bắt đầu bằng ký tự 0 và số nguyên hệ 16 được bắt đầu bằng các ký tự 0x. Ví dụ nếu a = 40 thì:

Dạng in hệ 8 là: 050

Dạng in hệ 16 là: 0x28

Cờ ios::showbase tắt (mặc định) thì không in 0 trước số nguyên hệ 8 và không 0x trước số nguyên hệ 16. Ví dụ nếu a = 40 thì:

Dạng in hệ 8 là: 50

Dạng in hệ 16 là: 28

### Cờ ios::uppercase

+ Nếu cờ ios::uppercase bật thì các chữ số hệ 16 (như A, B, C,...) được in dưới dạng chữ hoa.

+ Nếu cờ ios::uppercase tắt (mặc định) thì các chữ số hệ 16 (như A, B, C,...) được in dưới dạng chữ thường.

### 1.4.5. Các phương thức bật tắt cờ

Các phương thức này định nghĩa trong lớp ios.

#### a. Phương thức long cout.setf(long f) ;

sẽ bật các cờ liệt kê trong f và trả về một giá trị long biểu thị các cờ đang bật.

#### b. Phương thức long cout.unsetf(long f) ;

sẽ tắt các cờ liệt kê trong f và trả về một giá trị long biểu thị các cờ đang bật.

#### c. Phương thức long cout.flags(long f) ;

có tác dụng giống như cout.setf(long).

#### d. Phương thức long cout.flags() ;

sẽ trả về một giá trị long biểu thị các cờ đang bật.

### 1.4.6. Các bộ phận định dạng

Các bộ phận định dạng (định nghĩa trong tập tin iostream.h) bao gồm:

dec // như cờ ios::dec

oct // như cờ ios::oct

hex // như cờ ios::hex

endl // xuất ký tự '\n' (chuyển dòng)

flush // đẩy dữ liệu ra thiết bị xuất

**Ví dụ:** Xét chương trình sau:

```

#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
void main()
{
    clrscr();
    cout.setf(ios::showbase);
    cout<<"ABC"<<endl<<hex<<40<<"    "<<41;
    getch();
}

```

#### *1.4.7. Các hàm định dạng*

Các hàm định dạng (định nghĩa trong <iomanip.h>) bao gồm:

set(int n)	// như cout.width(int n)
setprecision(int n)	// như cout.setprecision(int n)
setfill( char ch)	// như cout.setfill( char ch)
setiosflags( long l)	// như cout.setiosflags( long f)
resetiosflags( long l)	// như cout.resetiosflags( long f)

Các hàm định dạng có tác dụng như các phương thức định dạng nhưng được viết nối đuôi trong toán tử xuất nên tiện sử dụng hơn.

#### **Chú ý**

- Các hàm định dạng (cũng như các bộ phận định dạng) cần viết trong toán tử xuất. Một hàm định dạng đứng một mình như một câu lệnh sẽ không có tác dụng định dạng.

➤ Muốn sử dụng các hàm định dạng cần bổ sung vào đầu chương trình câu lệnh: #include <iomanip.h>

Chương trình trong ví dụ 1.2. có thể viết lại theo các phương án sau:

### **Phương án 1:**

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    cout << setiosflags(ios::showbase);
    cout<<"ABC"<<endl<<hex<<40<<" "<<41;
    getch();
}
```

### **Phương án 2:**

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    cout <<"ABC"<<endl<< setiosflags(ios::showbase)
        << hex<<40<<" "<<41;
    getch();
}
```

## 1.5. Các dòng chuẩn

Có 4 dòng (đối tượng của các lớp stream) đã định nghĩa trước, được cài đặt khi chương trình khởi động là:

- cin dòng input chuẩn gắn với bàn phím, giống như stdin của C.
- cout dòng output chuẩn gắn với màn hình, giống như stdout của C.
- cerr dòng output lỗi chuẩn gắn với màn hình, giống như stderr của C.
- clog giống cerr nhưng có thêm bộ đệm.

### Chú ý

- Có thể dùng các dòng cerr và clog để xuất ra màn hình như đã dùng đối với cout.
- Vì clog có thêm bộ đệm, nên dữ liệu được đưa vào bộ đệm. Khi đầy bộ đệm thì đưa dữ liệu bộ đệm ra dòng clog. Vì vậy trước khi kết thúc xuất cần dùng phương thức: clog.flush(); để đẩy dữ liệu từ bộ đệm ra clog.

Chương trình sau minh họa cách dùng dòng clog. Chúng ta nhận thấy, nếu bỏ câu lệnh clog.flush() thì sẽ không nhìn thấy kết quả xuất ra màn hình khi chương trình tạm dừng bởi câu lệnh getch().

### Ví dụ:

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

void main()
{
    clrscr();
    float x=-87.1500, y=23.45425, z=678.0;
    clog.setf(ios::scientific);
    clog.precision(4);
```

```
clog.fill('*');

clog<<"\n";

clog.width(10);

clog<<x;

clog<<"\n";

clog.width(10);

clog<<y;

clog<<"\n";

clog.width(10);

clog<<z;

clog.flush();

getch();

}

}
```

## 1.6. Xuất ra máy in

Bốn dòng chuẩn không gắn với máy in. Như vậy không thể dùng các dòng này để xuất dữ liệu ra máy in. Để xuất dữ liệu ra máy in (cũng như nhập, xuất trên tệp) cần tạo ra các dòng tin mới và cho nó gắn với thiết bị cụ thể. C++ cung cấp 3 lớp stream để làm điều này, đó là các lớp:

ofstream   dùng để tạo các dòng xuất (ghi tệp)

ifstream   dùng để tạo các dòng nhập (đọc tệp)

fstream   dùng để tạo các dòng nhập, dòng xuất hoặc dòng nhập-xuất

Mỗi lớp có 4 hàm tạo dùng để khai báo các dòng (đối tượng dòng tin). Để tạo một dòng xuất và gắn nó với máy in ta có thể dùng một trong những hàm tạo sau đây:

ofstream Tên\_dòng(int fd),

```
ofstream Tên_dòng(int fd, char *buf, int n);
```

Trong đó:

- Tên\_dòng là tên biến đối tượng kiểu ofstream chúng ta tự đặt.

- fd(file descriptor) là chỉ số tập tin. Chỉ số tập tin định sẵn đối với stdprn (máy in chuẩn) là 4.

- Các tham số buf và n xác định một vùng nhớ n byte do buff trả tới. Vùng nhớ sẽ được làm bộ đệm cho dòng xuất.

**Ví dụ:** Câu lệnh ofstream prn(4); sẽ tạo dòng tin xuất prn và gắn nó với máy in chuẩn. Dòng prn sẽ có bộ đệm mặc định. Dữ liệu trước hết chuyển vào bộ đệm, khi đầy bộ đệm thi dữ liệu sẽ được đẩy từ bộ đệm ra dòng prn và có thể sử dụng phương thức flush hoặc bộ phận định dạng flush. Cách viết như sau:

```
prn.flush // Phương thức
```

```
prn<<flush; //Bộ phận định dạng
```

Các câu lệnh sau sẽ xuất dữ liệu ra prn (máy in) và ý nghĩa của chúng như sau:

```
prn<<"\n Tong=" <<(4+9); //Đưa một dòng vào bộ đệm
```

```
prn<<"\n Tich=" <<(4*9); // Đưa dòng tiếp theo vào bộ đệm
```

```
prn.flush(); //Đẩy dữ liệu từ bộ đệm ra máy in (in 2 dòng)
```

Các câu lệnh dưới đây sẽ xuất dữ liệu ra máy in nhưng xuất từng lòng một:

```
prn<<"\n Tong=" <<(4+9)<<flush; // In một dòng
```

```
prn<<"\n Tich=" <<(4*9)<<flush; //In dòng tiếp theo
```

**Ví dụ:** Các câu lệnh

```
char buf[512];
```

```
ofstream prn(4,buf,512);
```

sẽ tạo dòng tin xuất prn và gắn nó với máy in chuẩn. Dòng xuất prn sử dụng 512 byte của mảng buf làm bộ đệm. Các câu lệnh dưới đây cũng xuất ra máy in:

```
prn<<"\n Tong="<<(4+9); //Đưa dữ liệu vào bộ đệm  
prn<<"\n Tich="<<(4*9) ; //Đưa dữ liệu vào bộ đệm  
prn.flush(); // Xuất 2 dòng (ở bộ đệm) ra máy in
```

**Chú ý:** Trước khi kết thúc chương trình, dữ liệu từ bộ đệm sẽ được tự động đẩy ra máy in.

## PHỤ LỤC 2

### Thứ tự ưu tiên của các phép toán

Các phép toán được chia thành 16 nhóm. Các phép toán trong cùng nhóm có mức độ ưu tiên như nhau.

Về trình tự kết hợp:

- + Các phép tính của nhóm 2, nhóm 14 và toán tử gán (nhóm 15) kết hợp từ phải sang trái.
- + Các phép toán còn lại kết hợp từ trái qua phải.

#### 1. Nhóm một

- () Gọi hàm (Function call)
- [] Chỉ số mảng (Array subscript)
- > Chọn gián tiếp một thành phần (indirect component selector)
- :: Xác định phạm vi truy nhập (scope access/resolution)
- . Chọn trực tiếp một thành phần (direct component selector)

#### 2. Nhóm hai

- () Gọi hàm (Function call)
- ! Phù định logic (Logical negation -NOT)
- Lấy phân bù theo bit (Bitwise (1's) complement)
- + Dấu cộng (Unary plus)
- Dấu trừ (Unary minus)
- ++ Phép tăng một (Preincrement or postincrement)
- Phép giảm một (Predecrement or postdecrement)
- & Phép lấy địa chỉ (Address)
- \* Truy nhập gián tiếp (Indirection)

`sizeof` Cho kích thước của toán hạng (returns size of operand, in bytes)

`new` Cấp phát bộ nhớ động (dynamically allocates C++ storage)

`delete` Giải phóng bộ nhớ (dynamically deallocates C++ storage)

### 3. Nhóm ba

\* Nhân (Multiply)

/ Chia (Divide)

% Lấy phần dư (Remainder - modulus)

### 4. Nhóm bốn

\* Gọi gián tiếp tới thành phần từ một biến đối tượng

->\* Gọi gián tiếp tới thành phần từ một con trỏ đối tượng

### 5. Nhóm năm

+ Cộng (Binary plus)

- Trừ (Binary minus)

### 6. Nhóm sáu

<< Dịch trái (Shift left)

>> Dịch phải (Shift right)

### 7. Nhóm bảy

< Nhỏ hơn (Less than)

<= Nhỏ hơn hoặc bằng (Less than or equal to)

> Lớn hơn (Greater than)

>= Lớn hơn hoặc bằng (Greater than or equal to)

### 8. Nhóm tám

== Bằng (Equal to)

**!=** Không bằng (Not equal to)

## 9. Nhóm chín

**&** Phép và theo bit (Bitwise AND)

## 10. Nhóm mười

**^** Phép hoặc loại trừ theo bit (Bitwise XOR)

## 11. Nhóm mười một

**|** Phép hoặc theo bit (Bitwise OR)

## 12. Nhóm mười hai

**&&** Phép và logic (Logical AND)

## 13. Nhóm mười ba

**&&** Phép hoặc logic (Logical OR)

## 14. Nhóm mười bốn

**?:** Toán tử điều kiện (a ? x : y means "if a then x, else y")

## 15. Nhóm mười lăm

**=** Phép gán đơn giản (Simple assignment)

**\*=** Phép gán sau khi nhân (Assign product)

**/=** Phép gán sau khi chia (Assign quotient)

**%=** Phép gán sau khi lấy phần dư (Assign remainder)

**+=** Phép gán sau khi cộng (Assign sum)

**-=** Phép gán sau khi trừ (Assign difference)

**&=** Phép gán sau khi AND theo bit (Assign bitwise AND)

**^=** Phép gán sau khi XOR theo bit (Assign bitwise XOR)

**|=** Phép gán sau khi OR theo bit (Assign bitwise OR)

<<= Phép gán sau khi dịch trái (Assign left shift)

>>= Phép gán sau khi dịch phải (Assign right shift)

## 16. Nhóm mười sáu

, Toán tử phẩy dùng để phân cách các phần tử

Tất cả các toán tử nói trên đều có thể định nghĩa chòng trừ các toán tử sau:

. Chọn trực tiếp một thành phần

\* Gọi gián tiếp tới thành phần từ một biến đối tượng

:: Toán tử xác định phạm vi truy nhập

## TÀI LIỆU THAM KHẢO

1. Ivar Jacobson, *Object - Oriented Software Engineering*, Addison-Wesley Publishing Company, 1992.
2. Michael Blaha, William Premerlani, *Object - Oriented Modeling and Design for Database Applications*, Prentice Hall, 1998.
3. Phạm Văn Át, *C++ và Lập trình hướng đối tượng*, NXB Khoa học và Kỹ thuật, 1999.
4. Đoàn Văn Ban, *Phân tích và thiết kế hướng đối tượng*, NXB Khoa học và Kỹ thuật, 1997.
5. Nguyễn Thanh Thùy, *Lập trình hướng đối tượng với C++*, NXB Khoa học và Kỹ thuật, 1999.

**NHÀ XUẤT BẢN ĐẠI HỌC THÁI NGUYÊN**

**Địa chỉ: Phường Tân Thịnh - TP. Thái Nguyên - tỉnh Thái Nguyên**

Điện thoại: 0280.3840.018 - Fax: 0280.3840.017

Website: nxb.tnu.edu.vn \* E-mail: nxb.dhtn@gmail.com

---

## **GIÁO TRÌNH**

# **KỸ THUẬT LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

*Chịu trách nhiệm xuất bản*

**PGS.TS NGUYỄN ĐỨC HẠNH**

Giám đốc - Tổng biên tập

*Biên tập:*

HOÀNG ĐỨC NGUYÊN

*Thiết kế bìa:*

NGUYỄN NGỌC DUNG

*Trình bày:*

ĐÀO THÁI SƠN

*Sửa bản in:*

LÊ THÀNH NGUYÊN

**ISBN: 978-604-915-496-6**

In 200 cuốn, khổ 17 x 24 cm, tại Xưởng in – Nhà xuất bản Đại học Thái Nguyên (Địa chỉ: phường Tân Thịnh, thành phố Thái Nguyên). Giấy phép xuất bản số: 1193--2017/CXBIPH/02-49/ĐHTN. Quyết định xuất bản số: 125/QĐ-NXBĐHTN. In xong và nộp lưu chiểu quý II năm 2017.

