ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA CÔNG NGHỆ PHẦN MỀM

GIÁO TRÌNH OPP C++

THÀNH PHỐ HỒ CHÍ MINH - 2010

CHƯƠNG 1

GIỚI THIỆU VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1.1 LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP) LÀ GÌ ?

Lập trình hướng đối tượng (Object-Oriented Programming, viết tắt là OOP) là một phương pháp mới trên bước đường tiến hóa của việc lập trình máy tính, nhằm làm cho chương trình trở nên linh hoạt, tin cậy và dễ phát triển. Tuy nhiên để hiểu được OOP là gì, chúng ta hãy bắt đầu từ lịch sử của quá trình lập trình – xem xét OOP đã tiến hóa như thế nào.

1.1.1 Lập trình tuyến tính

Máy tính đầu tiên được lập trình bằng mã nhị phân, sử dụng các công tắt cơ khí để nạp chương trình. Cùng với sự xuất hiện của các thiết bị lưu trữ lớn và bộ nhớ máy tính có dung lượng lớn nên các ngôn ngữ lập trình cấp cao đầu tiên được đưa vào sử dụng . Thay vì phải suy nghĩ trên một dãy các bit và byte, lập trình viên có thể viết một loạt lệnh gần với tiếng Anh và sau đó chương trình dịch thành ngôn ngữ máy. Các ngôn ngữ lập trình cấp cao đầu tiên được thiết kế để lập các chương trình làm các công việc tương đối đơn giản như tính toán. Các chương trình ban đầu chủ yếu liên quan đến tính toán và không đòi hỏi gì nhiều ở ngôn ngữ lập trình. Hơn nữa phần lớn các chương trình này tương đối ngắn, thường ít hơn 100 dòng. Khi khả năng của máy tính tăng lên thì khả năng để triển khai các chương trình phức tạp hơn cũng

tăng lên. Các ngôn ngữ lập trình ngày trước không còn thích hợp đối với việc lập trình đòi hỏi cao hơn. Các phương tiện cần thiết để sử dụng lại các phần mã chương trình đã viết hầu như không có trong ngôn ngữ lập trình tuyến tính. Thật ra, một đoạn lệnh thường phải được chép lặp lại mỗi khi chúng ta dùng trong nhiều chương trình do đó chương trình dài dòng, logic của chương trình khó hiểu. Chương trình được điều khiển để nhảy đến nhiều chỗ mà thường không có sự giải thích rõ ràng, làm thế nào để chương trình đến chỗ cần thiết hoặc tại sao như vậy.
Ngôn ngữ lập trình tuyến tính không có khả năng kiểm soát phạm vi nhìn thấy của các dữ liệu. Mọi dữ liệu trong chương trình đều là dữ liệu toàn cục nghĩa là chúng có thể bị sửa đổi ở bất kỳ phần nào của chương trình. Việc dò tìm các thay đổi không mong muốn đó của các phần tử dữ liệu trong một dãy mã lệnh dài và vòng vèo đã từng làm cho các lập trình viên rất mất thời gian.

1.1.2 Lập trình cấu trúc:

Rõ ràng là các ngôn ngữ mới với các tính năng mới cần phải được phát triển để có thể tạo ra các ứng dụng tinh vi hơn. Vào cuối các năm trong 1960 và 1970, ngôn ngữ lập trình có cấu trúc ra đời. Các chương trình có cấu trúc được tổ chức theo các công việc mà chúng thực hiện.
Về bản chất, chương trình chia nhỏ thành các chương trình con riêng rẽ (còn gọi là hàm hay thủ tục) thực hiện các công việc rời rạc trong quá trình lớn hơn, phức tạp hơn. Các hàm này được giữ càng độc lập với nhau càng nhiều càng tốt, mỗi hàm có dữ liệu và logic riêng.Thông tin được chuyển giao giữa các hàm thông qua các tham số, các hàm có thể có các biến cục bộ mà không một ai nằm bên ngoài phạm vi của hàm lại có thể truy xuất được chúng. Như vậy, các hàm có thể được xem là các chương trình con được đặt chung với nhau để xây dựng nên một ứng dụng.
Mục tiêu là làm sao cho việc triển khai các phần mềm dễ dàng hơn đối với các lập trình viên mà vẫn cải thiện được tính tin cậy và dễ bảo quản chương trình. Một chương trình có cấu trúc được hình thành bằng cách bẻ gãy các chức năng cơ bản của chương trình thành các mảnh nhỏ mà sau đó trở thành các hàm. Bằng cách cô lập các công việc vào trong các hàm, chương trình có cấu trúc có thể làm giảm khả năng của một hàm này ảnh hưởng đến một hàm khác. Việc này cũng làm cho việc tách các vấn đề trở nên dễ dàng hơn. Sự gói gọn này cho phép chúng ta có thể viết các chương trình sáng sủa hơn và giữ được điều khiển trên từng hàm. Các biến toàn cục không còn nữa và được thay thế bằng các tham số và biến cục bộ có phạm vi nhỏ hơn và dễ kiểm soát hơn. Cách tổ chức tốt hơn này nói lên rằng chúng ta có khả năng quản lý logic của cấu trúc chương trình, làm cho việc triển khai và bảo dưỡng chương trình nhanh hơn và hữu hiện hơn và hiệu quả hơn.
Một khái niệm lớn đã được đưa ra trong lập trình có cấu trúc là **sự trừu tượng hóa** (**Abstraction**). Sự trừu tượng hóa có thể xem như khả năng quan sát một sự việc mà không cần xem xét đến các chi tiết bên trong của nó. Trong một chương trình có cấu trúc, chúng ta chỉ cần biết một hàm đã cho có thể làm được một công việc cụ thể gì là đủ. Còn làm thế nào mà công việc đó lại thực hiện được là không quan trọng, chừng nào hàm còn tin cậy được thì còn có thể dùng nó mà không cần phải biết nó thực hiện đúng đắn chức năng của mình như thế nào. Điều này gọi là **sự trừu tượng hóa theo chức năng** (**Functional abstraction**) và là nền tảng của lập trình có cấu trúc.
Ngày nay, các kỹ thuật thiết kế và lập trình có cấu trúc được sử rộng rãi. Gần như mọi ngôn ngữ lập trình đều có các phương tiện cần thiết để cho phép lập trình có cấu trúc. Chương trình có cấu trúc dễ viết, dễ bảo dưỡng hơn các chương trình không cấu trúc.
Sự nâng cấp như vậy cho các kiểu dữ liệu trong các ứng dụng mà các lập trình viên đang viết cũng đang tiếp tục diễn ra. Khi độ phức tạp của một chương trình tăng lên, sự phụ thuộc của nó vào các kiểu dữ liệu cơ bản mà nó xử lý cũng tăng theo. Vấn đề trở rõ ràng là cấu trúc dữ liệu trong chương trình quan trọng chẳng kém gì các phép toán thực hiện trên chúng. Điều này càng trở rõ ràng hơn khi kích thước của chương trình càng tăng. Các kiểu dữ liệu được xử lý trong nhiều hàm khác nhau bên trong một chương trình có cấu trúc. Khi có sự thay đổi trong các dữ liệu này thì cũng cần phải thực hiện cả các thay đổi ở mọi nơi có các thao tác tác động trên chúng. Đây có thể là một công việc tốn thời gian và kém hiệu quả đối với các chương trình có hàng ngàn dòng lệnh và hàng trăm hàm trở lên.

Một yếu điểm nữa của việc lập trình có cấu trúc là khi có nhiều lập trình viên làm việc theo nhóm cùng một ứng dụng nào đó. Trong một chương trình có cấu trúc, các lập trình viên được phân công viết một tập hợp các hàm và các kiểu dữ liệu. Vì có nhiều lập trình viên khác nhau quản lý các hàm riêng, có liên quan đến các kiểu dữ liệu dùng chung nên các thay đổi mà lập trình viên tạo ra trên một phần tử dữ liệu sẽ làm ảnh hưởng đến công việc của tất cả các người còn lại trong nhóm. Mặc dù trong bối cảnh làm việc theo nhóm, việc viết các chương trình có cấu trúc thì dễ dàng hơn nhưng sai sót trong việc trao đổi thông tin giữa các thành viên trong nhóm có thể dẫn tới hậu quả là mất rất nhiều thời gian để sửa chữa chương trình.

1.1.3 Sự trừu tượng hóa dữ liệu:

Sự trừu tượng hóa dữ liệu (**Data abstraction**) tác động trên các dữ liệu cũng tương tự như sự trừu tượng hóa theo chức năng. Khi có trừu tượng hóa dữ liệu, các cấu trúc dữ liệu và các phần tử có thể được sử dụng mà không cần bận tâm đến các chi tiết cụ thể. Chẳng hạn như các số dấu chấm động đã được trừu tượng hóa trong tất cả các ngôn ngữ lập trình, Chúng ta không cần quan tâm cách biểu diễn nhị phân chính xác nào cho số dấu chấm động khi gán một giá trị, cũng không cần biết tính bất thường của phép nhân nhị phân khi nhân các giá trị dấu chấm động. Điều quan trọng là các số dấu chấm động hoạt động đúng đắn và hiểu được.
Sự trừu tượng hóa dữ liệu giúp chúng ta không phải bận tâm về các chi tiết không cần thiết. Nếu lập trình viên phải hiểu biết về tất cả các khía cạnh của vấn đề, ở mọi lúc và về tất cả các hàm của chương trình thì chỉ ít hàm mới được viết ra, may mắn thay trừu tượng hóa theo dữ liệu đã tồn tại sẵn trong mọi ngôn ngữ lập trình đối với các dữ liệu phức tạp như số dấu chấm động. Tuy nhiên chỉ mới gần đây, người ta mới phát triển các ngôn ngữ cho phép chúng ta định nghĩa các kiểu dữ liệu trừu tượng riêng.

1.1.4 Lập trình hướng đối tượng:

Khái niệm hướng đối tượng được xây dựng trên nền tảng của khái niệm lập trình có cấu trúc và sự trừu tượng hóa dữ liệu. Sự thay đổi căn bản ở chỗ, một chương trình hướng đối tượng được thiết kế xoay quanh dữ liệu mà chúng ta có thể làm việc trên đó, hơn là theo bản thân chức năng của chương trình. Điều này hoàn toàn tự nhiên một khi chúng ta hiểu rằng mục tiêu của chương trình là xử lý dữ liệu. Suy cho cùng, công việc mà máy tính thực hiện vẫn thường được gọi là xử lý dữ liệu. Dữ liệu và thao tác liên kết với nhau ở một mức cơ bản (còn có thể gọi là mức thấp), mỗi thứ đều đòi hỏi ở thứ kia có mục tiêu cụ thể, các chương trình hướng đối tượng làm tường minh mối quan hệ này.
Lập trình hướng đối tượng liên kết cấu trúc dữ liệu với các thao tác, theo cách mà tất cả thường nghĩ về thế giới quanh mình. Chúng ta thường gắn một số các hoạt động cụ thể với một loại hoạt động nào đó và đặt các giả thiết của mình trên các quan hệ đó.
Ví dụ1.1: Chúng ta biết rằng một chiếc xe có các bánh xe, di chuyển được và có thể đổi hướng của nó bằng cách quẹo tay lái. Tương tự như thế, một cái cây là một loại thực vật có thân gỗ và lá. Một chiếc xe không phải là một cái cây, mà cái cây không phải là một chiếc xe, chúng ta có thể giả thiết rằng cái mà chúng ta có thể làm được với một chiếc xe thì không thể làm được với một cái cây. Chẳng hạn, thật là vô nghĩa khi muốn lái một cái cây, còn chiếc xe thì lại chẳng lớn thêm được khi chúng ta tưới nước cho nó.
Lập trình hướng đối tượng cho phép chúng ta sử dụng các quá trình suy nghĩ như vậy với các khái niệm trừu tượng được sử dụng trong các chương trình máy tính. Một mẫu tin (record) nhân sự có thể được đọc ra, thay đổi và lưu trữ lại; còn số phức thì có thể được dùng trong các tính toán. Tuy vậy không thể nào lại viết một số phức vào tập tin làm mẫu tin nhân sự và ngược lại hai mẫu tin nhân sự lại không thể cộng với nhau được. Một chương trình hướng đối tượng sẽ xác định đặc điểm và hành vi cụ thể của các kiểu dữ liệu, điều đó cho phép chúng ta biết một cách chính xác rằng chúng ta có thể có được những gì ở các kiểu dữ liệu khác nhau.
Chúng ta còn có thể tạo ra các quan hệ giữa các kiểu dữ liệu tương tự nhưng khác nhau trong một

chương trình hướng đối tượng. Người ta thường tự nhiên phân loại ra mọi thứ, thường đặt mối liên hệ giữa các khái niệm mới với các khái niệm đã có, và thường có thể thực hiện suy diễn giữa chúng trên các quan hệ đó. Hãy quan niệm thế giới theo kiểu cấu trúc cây, với các mức xây dựng chi tiết hơn kế tiếp nhau cho các thế hệ sau so với các thế hệ trước. Đây là phương pháp hiệu quả để tổ chức thế giới quanh chúng ta. Các chương trình hướng đối tượng cũng làm việc theo một phương thức tương tự, trong đó chúng cho phép xây dựng các các cơ cấu dữ liệu và thao tác mới dựa trên các cơ cấu có sẵn, mang theo các tính năng của các cơ cấu nền mà chúng dựa trên đó, trong khi vẫn thêm vào các tính năng mới.

Lập trình hướng đối tượng cho phép chúng ta tổ chức dữ liệu trong chương trình theo một cách tương tự như các nhà sinh học tổ chức các loại thực vật khác nhau. Theo cách nói lập trình đối tượng, xe hơi, cây cối, các số phức, các quyển sách đều được gọi là các **lớp** (**Class**).

Một lớp là một bản mẫu mô tả các thông tin cấu trúc dữ liệu, lẫn các thao tác hợp lệ của các phần tử dữ liệu. Khi một phần tử dữ liệu được khai báo là phần tử của một lớp thì nó được gọi là một **đối tượng** (**Object**). 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. Một **thực thể** (**Instance**) là một vật thể có thực bên trong bộ nhớ, thực chất đó là một đối tượng (nghĩa là một đối tượng được cấp phát vùng nhớ).

Mỗi một đối tượng có riêng cho mình một bản sao các phần tử dữ liệu của lớp còn gọi là các **biến thực thể** (**Instance variable**). Các phương thức định nghĩa trong một lớp có thể được gọi bởi các đối tượng của lớp đó. Điều này được gọi là gửi một **thông điệp** (**Message**) cho đối tượng. Các thông điệp này phụ thuộc vào đối tượng, chỉ đối tượng nào nhận thông điệp mới phải làm việc theo thông điệp đó. Các đối tượng đều độc lập với nhau vì vậy các thay đổi trên các biến thể hiện của đối tượng này không ảnh hưởng gì trên các biến thể hiện của các đối tượng khác và việc gửi thông điệp cho một đối tượng này không ảnh hưởng gì đến các đối tượng khác.

1.2 MỘT SỐ KHÁI NIỆM MỚI TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Trong phần này, chúng ta tìm hiểu các khái niệm như sự đóng gói, tính kế thừa và tính đa hình. Đây là các khái niệm căn bản, là nền tảng tư tưởng của lập trình hướng đối tượng. Hiểu được khái niệm này, chúng ta bước đầu tiếp cận với phong cách lập trình mới, phong cách lập trình dựa vào đối tượng làm nền tảng mà trong đó quan điểm che dấu thông tin thông qua sư đóng gói là quan điểm trung tâm của vấn đề.

1.2.1 Sự đóng gói (Encapsulation)

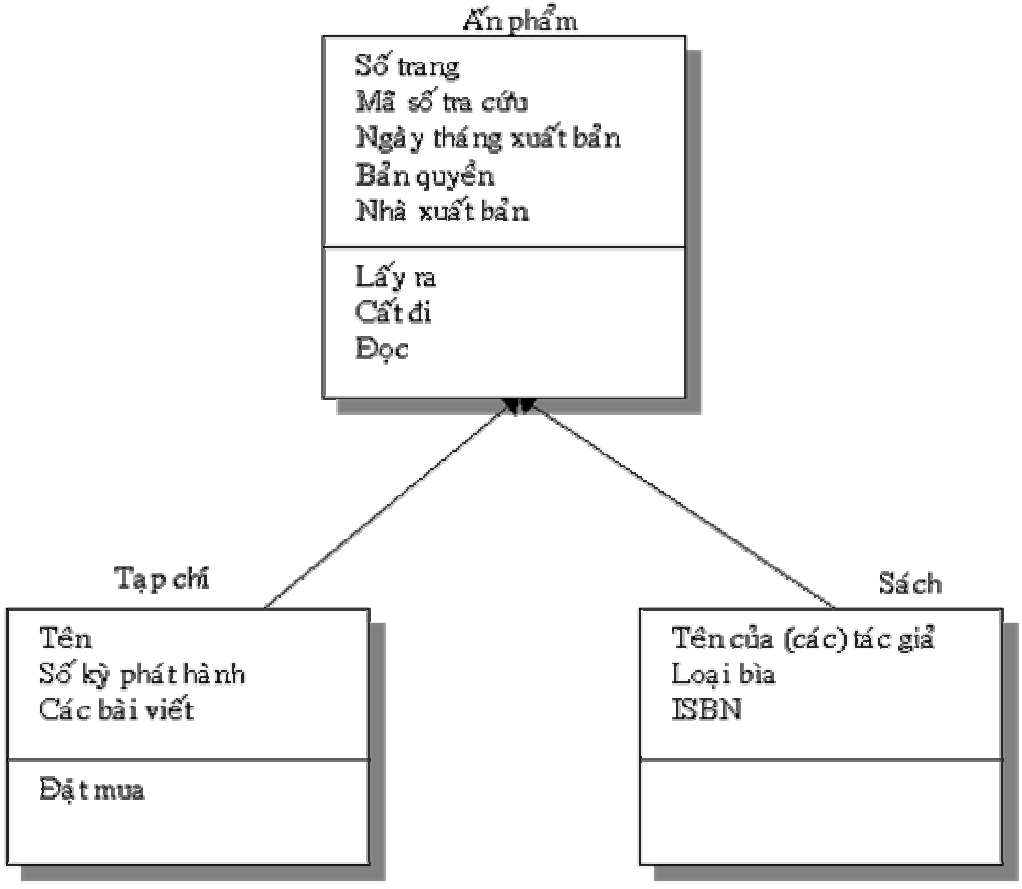
Sự đóng gói là cơ chế ràng buộc dữ liệu và thao tác trên dữ liệu đó thành một thể thống nhất, tránh được các tác động bất ngờ từ bên ngoài. Thể thống nhất này gọi là đối tượng.

Trong một đối tượng, dữ liệu hay thao tác hay cả hai có thể là **riêng** (**private**) hoặc **chung** (**public**) của đối tượng đó. Thao tác hay dữ liệu riêng là thuộc về đối tượng đó chỉ được truy cập bởi các thành phần của đối tượng, điều này nghĩa là thao tác hay dữ liệu riêng không thể truy cập bởi các phần khác của chương trình tồn tại ngoài đối tượng. Khi thao tác hay dữ liệu là chung, các phần khác của chương trình có thể truy cập nó mặc dù nó được định nghĩa trong một đối tượng. Các thành phần chung của một đối tượng dùng để cung cấp một giao diện có điều khiển cho các thành thành riêng của đối tượng.Cơ chế đóng gói là phương thức tốt để thực hiện cơ chế che dấu thông tin so với các ngôn ngữ lập trình cấu trúc.

1.2.2 Tính kế thừa (Inheritance)

Chúng ta có thể xây dựng các lớp mới từ các lớp cũ thông qua sự kế thừa. Một lớp mới còn gọi là **lớp dẫn xuất** (**derived class**), có thể thừa hưởng dữ liệu và các phương thức của **lớp cơ sở** (**base class**) ban đầu. Trong lớp này, có thể bổ sung các thành phần dữ liệu và các phương thức mới vào

những thành phần dữ liệu và các phương thức mà nó thừa hưởng từ lớp cơ sở. Mỗi lớp (kể cả lớp dẫn xuất) có thể có một số lượng bất kỳ các lớp dẫn xuất. Qua cơ cấu kế thừa này, dạng hình cây của các lớp được hình thành. Dạng cây của các lớp trông giống như các cây gia phả vì thế các lớp cơ sở còn được gọi là **lớp cha** (**parent class**) và các lớp dẫn xuất được gọi là **lớp con** (**child class**).
Ví dụ 1.2: Chúng ta sẽ 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 có thể tạo một ấn phẩm tổng quát bằng cách định nghĩa các thành phần dữ liệu tương ứng với số trang, mã số tra cứu, ngày tháng xuất bản, bản quyền và nhà xuất bản. Các ấn phẩm có thể được lấy ra, cất đi và đọc. Đó là các phương thức thực hiện trên một ấn phẩm. Tiếp đó chúng ta định nghĩa hai lớp dẫn xuất tên là tạp chí và sách. Tạp chí có tên, số ký 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 (cứng hay mềm) và số hiệu ISBN của nó. Như vậy chúng ta có thể thấy, sách và tạp chí có chung các đặc trưng ấn phẩm, trong khi vẫn có các thuộc tính riêng của chúng.



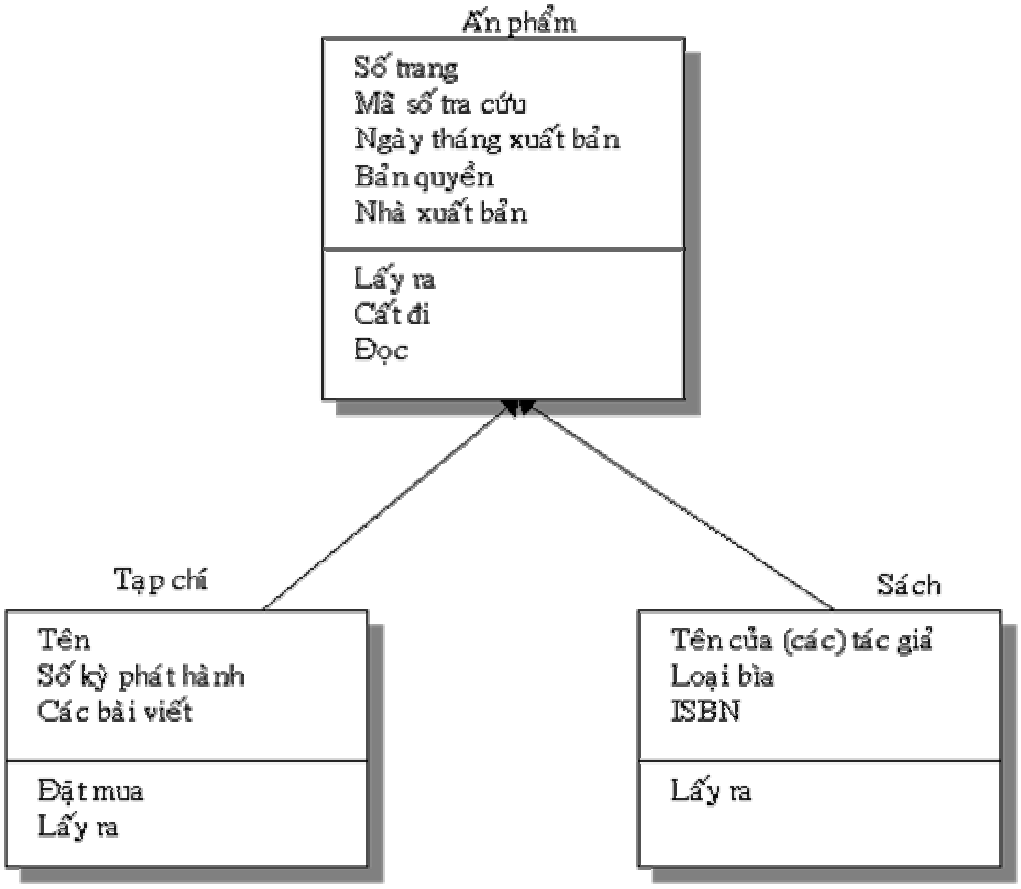
Hình 1.1: Lớp ấn phẩm và các lớp dẫn xuất của nó.

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.

1.2.3 Tính đa hình (Polymorphism)

Đó là khả năng để cho một thông điệp có thể thay đổi cách thực hiện của nó theo lớp cụ thể của đối tượng nhận thông điệp. Khi một lớp dẫn xuất được tạo ra, nó có thể thay đổi cách thực hiện các phương thức nào đó mà nó thừa hưởng từ lớp cơ sở của nó. Một thông điệp khi được gởi đến một đối tượng của lớp cơ sở, sẽ dùng phương thức đã định nghĩa cho nó trong lớp cơ sở. Nếu một lớp dẫn xuất định nghĩa lại một phương thức thừa hưởng từ lớp cơ sở của nó thì một thông điệp có cùng tên với phương thức này, khi được gởi tới một đối tượng của lớp dẫn xuất sẽ gọi phương thức đã định nghĩa cho lớp dẫn xuất.

Ví dụ 1.3: Xét lại ví dụ 1.2, chúng ta thấy rằng cả tạp chí và và sách đều phải có khả năng lấy ra. Tuy nhiên phương pháp lấy ra cho tạp chí có khác so với phương pháp lấy ra cho sách, mặc dù kết quả cuối cùng giống nhau. Khi phải lấy ra tạp chí, thì phải sử dụng phương pháp lấy ra riêng cho tạp chí (dựa trên một bản tra cứu) nhưng khi lấy ra sách thì lại phải sử dụng phương pháp lấy ra riêng cho sách (dựa trên hệ thống phiếu lưu trữ). Tính đa hình cho phép chúng ta xác định một phương thức để lấy ra một tạp chí hay một cuốn sách. Khi lấy ra một tạp chí nó sẽ dùng phương thức lấy ra dành riêng cho tạp chí, còn khi lấy ra một cuốn sách thì nó sử dụng phương thức lấy ra tương ứng với sách. Kết quả là chỉ cần một tên phương thức duy nhất được dùng cho cả hai công việc tiến hành trên hai lớp dẫn xuất có liên quan, mặc dù việc thực hiện của phương thức đó thay đổi tùy theo từng lớp. Tính đa hình dựa trên sự nối kết (Binding), đó là quá trình gắn một phương thức với một hàm thực sự. Khi các phương thức kiểu đa hình được sử dụng thì trình biên dịch chưa thể xác định hàm nào tương ứng với phương thức nào sẽ được gọi. Hàm cụ thể được gọi sẽ tuỳ thuộc vào việc phần tử nhận thông điệp lúc đó là thuộc lớp nào, do đó hàm được gọi chỉ xác định được vào lúc chương trình chạy. Điều này gọi là sự kết nối muộn (Late binding) hay kết nối lúc chạy (Runtime binding) vì nó xảy ra khi chương trình đang thực hiện.



Hình 1.2: Minh họa tính đa hình đối với lớp ấn phẩm và các lớp dẫn xuất của nó.

1.3 CÁC NGÔN NGỮ VÀ VÀI ỨNG DỤNG CỦA OOP

Xuất phát từ tư tưởng của ngôn ngữ SIMULA67, trung tâm nghiên cứu Palo Alto (PARC) của hãng XEROR đã tập trung 10 năm nghiên cứu để hoàn thiện ngôn ngữ OOP đầu tiên với tên gọi là Smalltalk. Sau đó các ngôn ngữ OOP lần lượt ra đời như Eiffel, Clos, Loops, Flavors, Object Pascal, Object C, C++, Delphi, Java…
Chính XEROR trên cơ sở ngôn ngữ OOP đã đề ra tư tưởng giao diện biểu tượng trên màn hình (icon base screen interface), kể từ đó Apple Macintosh cũng như Microsoft Windows phát triển giao diện đồ họa như ngày nay. Trong Microsoft Windows, tư tưởng OOP được thể hiện một cách rõ nét nhất đó là "chúng ta click vào đối tượng", mỗi đối tượng có thể là control menu, control menu box, menu bar, scroll bar, button, minimize box, maximize box, … sẽ đáp ứng công việc tùy theo đặc tính của đối tượng. Turbo Vision của hãng Borland là một ứng dụng OOP tuyệt vời, giúp lập trình viên không quan tâm đến chi tiết của chương trình gia diện mà chỉ cần thực hiện các nội dung chính của vấn đề.

2.1 LỊCH SỬ CỦA C++

Vào những năm đầu thập niên 1980, người dùng biết C++ với tên gọi "C with Classes" được mô tả trong hai bài báo của Bjarne Stroustrup (thuộc AT&T Bell Laboratories) với nhan đề "Classes: An Abstract Data Type Facility for the C Language" và "Adding Classes to C : AnExercise in Language Evolution". Trong công trình này, tác giả đã đề xuất khái niệm lớp, bổ sung việc kiểm tra kiểu tham số của hàm, các chuyển đổi kiểu và một số mở rộng khác vào ngôn ngữ C. Bjarne Stroustrup nghiên cứu mở rộng ngôn ngữ C nhằm đạt đến một ngôn ngữ mô phỏng (simulation language) với những tính năng hướng đối tượng.
Trong năm 1983, 1984, ngôn ngữ "C with Classes" được thiết kế lại, mở rộng hơn rồi một trình biên dịch ra đời. Và chính từ đó, xuất hiện tên gọi "C++". Bjarne Stroustrup mô tả ngôn ngữ C++ lần đầu tiên trong bài báo có nhan đề "Data Abstraction in C". Sau một vài hiệu chỉnh C++ được công bố rộng rãi trong quyển "The C++ Programming Language" của Bjarne Stroustrup xuất hiện đánh dấu sự hiện diện thực sự của C++, người lập tình chuyên nghiệp từ đây đã có một ngôn ngữ đủ mạnh cho các dữ án thực tiễn của mình.
Về thực chất C++ giống như C nhưng bổ sung thêm một số mở rộng quan trọng, đặc biệt là ý tưởng về đối tượng, lập trình định hướng đối tượng.Thật ra các ý tưởng về cấu trúc trong C++ đã xuất phát vào các năm 1970 từ Simula 70 và Algol 68. Các ngôn ngữ này đã đưa ra các khái niệm về lớp và đơn thể. Ada là một ngôn ngữ phát triển từ đó, nhưng C++ đã khẳng định vai trò thực sự của mình.

2.2 CÁC MỞ RỘNG CỦA C++

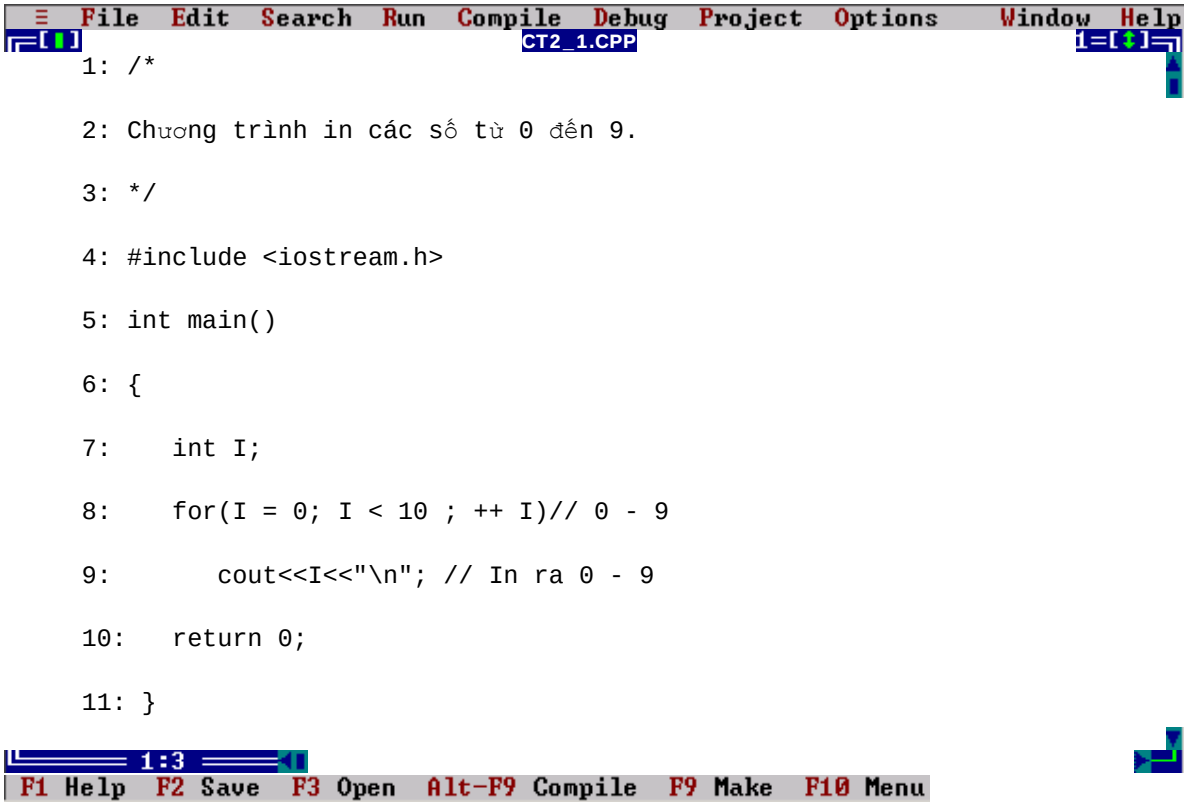
2.2.1 Các từ khóa mới của C++

Để bổ sung các tính năng mới vào C, một số từ khóa (keyword) mới đã được đưa vào C++ ngoài các từ khóa có trong C. Các chương trình bằng C nào sử dụng các tên trùng với các từ khóa cần phải thay đổi trước khi chương trình được dịch lại bằng C++. Các từ khóa mới này là :

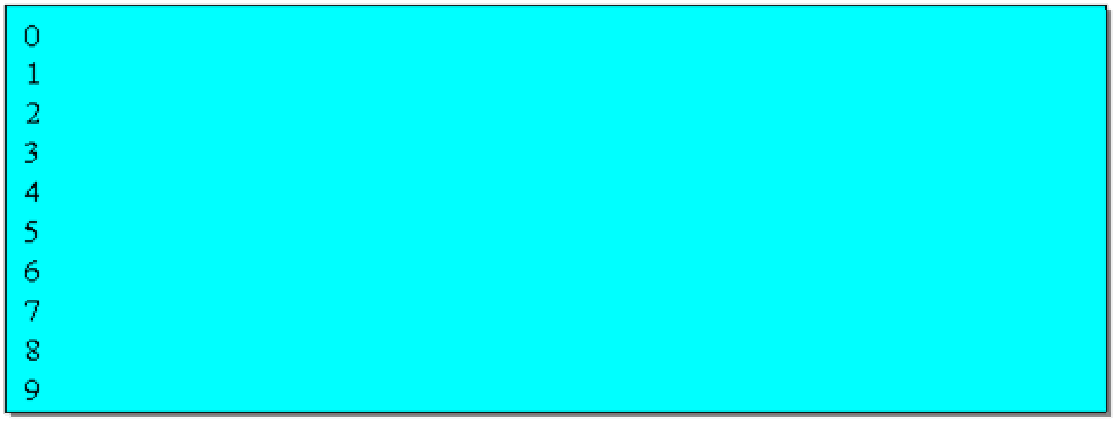
asm	**catch**	**class**	**delete**	**friend**	**inline**
new	**operator**	**private**	**protected**	**public**	**template**
this	**throw**	**try**	**virtual**		

2.2.2 Cách ghi chú thích

C++ chấp nhận hai kiểu chú thích. Các lập trình viên bằng C đã quen với cách chú thích bằng /*…*/. Trình biên dịch sẽ bỏ qua mọi thứ nằm giữa /*…*/. Ví dụ 2.1: Trong chương trình sau :



Mọi thứ nằm giữa /*…*/ từ dòng 1 đến dòng 3 đều được chương trình bỏ qua. Chương trình này còn minh họa cách chú thích thứ hai. Đó là cách chú thích bắt đầu bằng // ở dòng 8 và dòng 9. Chúng ta chạy ví dụ 2.1, kết quả ở hình 2.1.

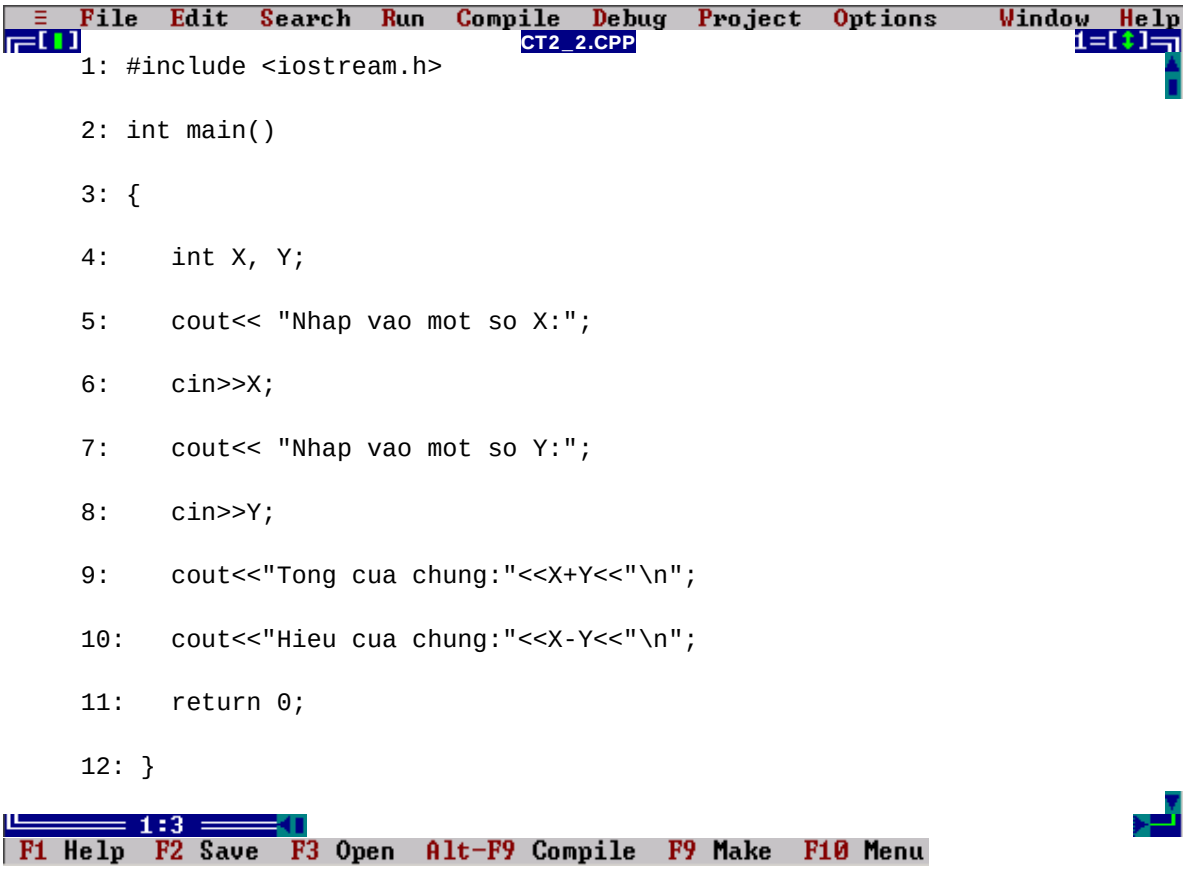


Hình 2.1: Kết quả của ví dụ 2.1

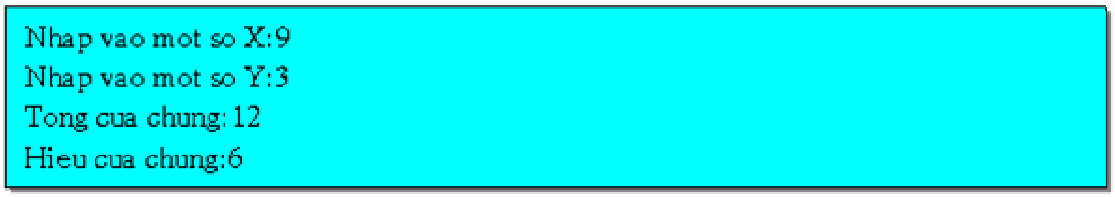
Nói chung, kiểu chú thích /*…*/ được dùng cho các khối chú thích lớn gồm nhiều dòng, còn kiểu // được dùng cho các chú thích một dòng.

2.2.3 Dòng nhập/xuất chuẩn

Trong chương trình C, chúng ta thường sử dụng các hàm nhập/xuất dữ liệu là printf() và scanf(). Trong C++ chúng ta có thể dùng dòng nhập/xuất chuẩn (standard input/output stream) để nhập/xuất dữ liệu thông qua hai biến đối tượng của dòng (stream object) là **cout** và **cin**.
Ví dụ 2.2: Chương trình nhập vào hai số. Tính tổng và hiệu của hai số vừa nhập.



Để thực hiện dòng xuất chúng ta sử dụng biến **cout** (console output) kết hợp với toán tử chèn (insertion operator) << như ở các dòng 5, 7, 9 và 10. Còn dòng nhập chúng ta sử dụng biến **cin** (console input) kết hợp với toán tử trích (extraction operator) >> như ở các dòng 6 và 8. Khi sử dụng **cout** hay **cin**, chúng ta phải kéo file iostream.h như dòng 1. Chúng ta sẽ tìm hiểu kỹ về dòng nhập/xuất ở chương 8. Chúng ta chạy ví dụ 2.2 , kết quả ở hình 2.2.



Hình 2.2: Kết quả của ví dụ 2.2