PHẦN II. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Chương 10. Khái niệm về lập trình hướng đối tượng

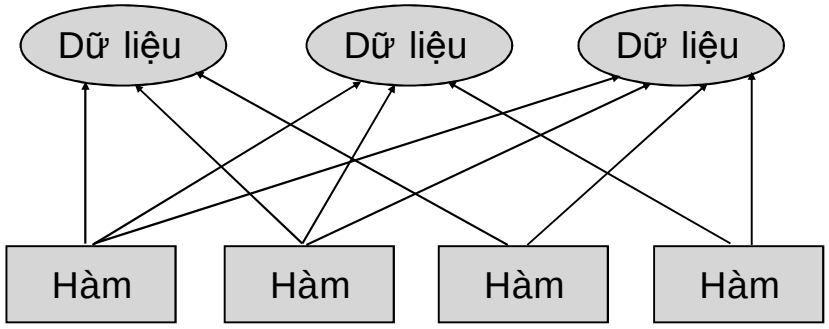
I. Lập trình cấu trúc và lập trình hướng đối tượng

1. Lập trình cấu trúc

- Tư tưởng chính của lập trình cấu trúc (structural programming) là chia chương trình thành các chương trình con (trong C++ gọi là hàm) và các module. Mỗi hàm thực hiện một nhiệm vụ xác định nào đó, còn mỗi module bao gồm một số hàm.
- Khi các chương trình ngày càng lớn và phức tạp thì lập trình cấu trúc bắt đầu bộc lộ những điểm yếu. Và cho dù các chương trình lớn này có được cài đặt tốt đến mấy thì nó vẫn quá phức tạp.

1. Lập trình cấu trúc (tiếp)

- Mô hình lập trình cấu trúc như sau:



1. Lập trình cấu trúc (tiếp)

- Lý do chính làm cho phương pháp lập trình cấu trúc tự bộc lộ những điểm yếu là dữ liệu của chương trình không được coi trọng. Các dữ liệu quan trọng của chương trình được lưu trữ trong các biến toàn cục, nó cho phép mọi hàm có thể truy nhập. Mà các hàm lại được viết bởi nhiều người lập trình khác nhau nên nguy cơ hỏng, mất dữ liệu là rất lớn.
- Hơn nữa, vì nhiều hàm truy nhập cùng một dữ liệu nên khi dữ liệu thay đổi thì các hàm này cũng phải thay đổi theo. Việc tìm các hàm cần thay đổi đã khó nhưng việc thay đổi các hàm này sao cho đúng còn khó hơn.

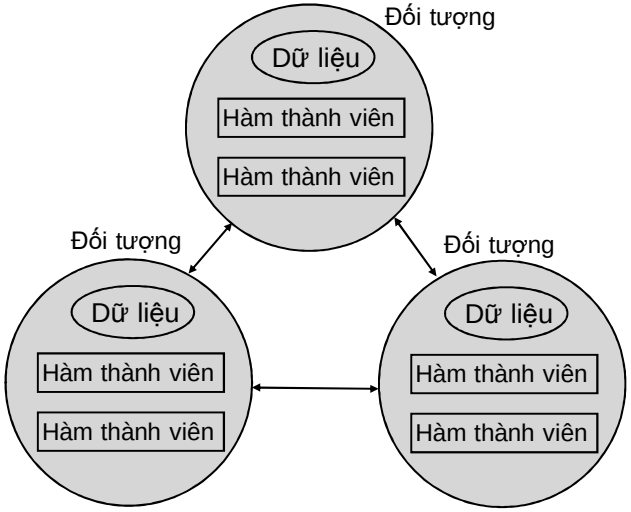
1. Lập trình cấu trúc (tiếp)

- Lập trình cấu trúc thường khó thiết kế chương trình bởi vì các thành phần chính của chương trình cấu trúc (là hàm và cấu trúc dữ liệu) không mô phỏng được thế giới thực. Ví dụ: giả sử ta cần viết mã để tạo giao diện đồ họa với người sử dụng như menu, cửa sổ, nút bấm,… Nếu lập trình cấu trúc thì câu hỏi đặt ra là dùng cấu trúc dữ liệu nào? Các hàm cần làm gì?

2. Lập trình hướng đối tượng

- Ý tưởng chính của lập trình hướng đối tượng (object oriented programming, OOP) là kết hợp cả dữ liệu và các hàm thao tác trên dữ liệu đó vào một thực thể chương trình gọi là đối tượng.
- Cách duy nhất để truy nhập dữ liệu của một đối tượng là thông qua các hàm của đối tượng đó (trong C++, các hàm của đối tượng được gọi là các hàm thành viên). Nếu ta muốn đọc dữ liệu trong một đối tượng thì ta phải gọi một hàm thành viên của đối tượng đó. Hàm thành viên này sẽ đọc dữ liệu và trả về giá trị cho ta. Ta không thể truy nhập trực tiếp dữ liệu của đối tượng.

2. Lập trình hướng đối tượng (tiếp)

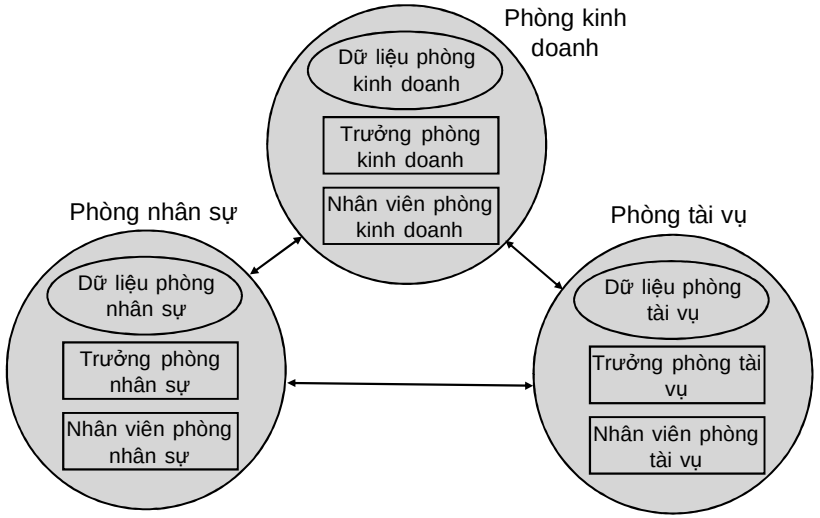


Mô hình lập trình hướng đối tượng

2. Lập trình hướng đối tượng (tiếp)

- Trong lập trình hướng đối tượng dữ liệu được ẩn đi để tránh những thay đổi vô tình làm hỏng dữ liệu. Dữ liệu và hàm tác động lên nó được đóng gói trong một thực thể chương trình.
- Nếu chúng ta muốn thay đổi dữ liệu trong một đối tượng thì chúng ta phải biết chính xác hàm nào tương tác với nó; tức là các hàm thành viên trong đối tượng đó. Không có hàm nào có thể truy nhập dữ liệu. Điều này giúp đơn giản hoá việc viết, gỡ rối, và bảo trì chương trình.

2. Lập trình hướng đối tượng (tiếp)



Mô hình công ty kinh doanh

2. Lập trình hướng đối tượng (tiếp)

- Tóm lại, lập trình hướng đối tượng là một cách tổ chức chương trình (OOP). Hướng đối tượng là phải xem thiết kế chương trình như thế nào chứ không đi vào chi tiết từng lệnh. Cụ thể là các chương trình hướng đối tượng phải được tổ chức xung quanh các đối tượng.

2. Lập trình hướng đối tượng (tiếp)

Người ta đã tổng hợp các đặc tính của LTHDT:

1. Tất cả đều là đối tượng.
2. Chương trình hướng đối tượng có thể coi là một tập hợp các đối tượng tương tác với nhau
3. Mỗi đối tượng trong chương trình có các dữ liệu độc lập của mình và chiếm bộ nhớ riêng của mình.
4. Mỗi đối tượng đều có dạng đặc trưng của lớp các đối tượng đó.
5. Tất cả các đối tượng thuộc về cùng một lớp đều có các hành vi giống nhau.

II. Các khái niệm cơ bản trong lập trình hướng đối tượng

1. Đối tượng (object)
2. Lớp (class)
3. Sự kế thừa (inheritance)
4. Sự sử dụng lại (Reusability)
5. Sự đa hình và chồng hàm (polymorphism and overloading)
6. Che giấu dữ liệu
7. Truyền thông báo

1. Đối tượng (object)

- Như ta đã biết, đối tượng là một thành phần chương trình chứa cả dữ liệu và các hàm thao tác trên dữ liệu đó.
- Trong lập trình hướng đối tượng chúng ta không đi tìm cách chia chương trình thành các hàm mà đi tìm cách chia chương trình thành các đối tượng. Việc chia chương trình thành các đối tượng làm cho việc thiết kế chương trình trở nên dễ dàng hơn vì các đối tượng trong chương trình rất gần gũi với các đối tượng trong thực tế.

1. Đối tượng (tiếp)

Ví dụ về một số đối tượng trong thực tế có thể trở thành đối tượng trong chương trình.

- Các đối tượng vật lý:
  - Các thang máy trong chương trình điều khiển tháng máy
  - Các máy bay trong chương trình điều hành bay
  - Các xe ô tô trong chương trình mô phỏng luồng giao thông.
- Các phần tử trong môi trường người sử dụng máy tính:
  - Các cửa sổ
  - Các menu
  - Các đối tượng đồ họa (như hình chữ nhật, hình tròn, hình tam giác,…)
  - Chuột, bàn phím, các ổ đĩa, máy in

1. Đối tượng (tiếp)

Ví dụ về một số đối tượng trong thực tế có thể trở thành đối tượng trong chương trình.

- Các cấu trúc dữ liệu:
  - Ngăn xếp
  - Hàng đợi
  - Danh sách liên kết
  - Cây nhị phân
- Nhân sự:
  - Nhân viên
  - Sinh viên
  - Khách hàng

1. Đối tượng (tiếp)

Ví dụ về một số đối tượng trong thực tế có thể trở thành đối tượng trong chương trình.

- Các tệp dữ liệu:
  - Một file nhân sự
  - Một từ điển
- Các kiểu dữ liệu của người sử dụng:
  - Thời gian
  - Các số phức
  - Các điểm trong mặt phẳng

1. Đối tượng (tiếp)

Ví dụ về một số đối tượng trong thực tế có thể trở thành đối tượng trong chương trình.

- Các thành phần trong trò chơi:
  - Các viên bi trong trò chơi Line
  - Các quân cờ trong trò chơi cờ tướng, cờ vua
  - …

1. Đối tượng (tiếp)

- *Một câu hỏi đặt ra là khi các đối tượng thực tế trở thành các đối tượng trong chương trình thì cái gì là dữ liệu, cái gì là hàm thành viên của đối tượng?*
  - Các đối tượng trong thực tế thường có trạng thái và khả năng. Trạng thái là các tính chất của đối tượng mà có thể thay đổi. Khả năng là những gì mà đối tượng có thể làm.
  - Khi trở thành đối tượng trong chương trình thì dữ liệu sẽ lưu trạng thái còn các hàm thành viên sẽ đáp ứng với các khả năng của đối tượng.

1. Đối tượng (tiếp)

- Ví dụ một đối tượng thang máy thì dữ liệu có thể là:
  - Tầng hiện tại
  - Số lượng hành khách
  - Các nút ấn

  Các hàm thành viên có thể là:
  - DiXuong()
  - DiLen()
  - MoCua()
  - DongCua()
  - LayTTin()
  - TinhTangSeToi()

2. Lớp (class)

- Một lớp đối tượng (gọi tắt là lớp) là một mô tả về một số đối tượng tương tự nhau. Nó xác định dữ liệu gì và các hàm nào sẽ có trong các đối tượng của lớp đó.
- Khái niệm lớp trong lập trình hướng đối tượng giống khái niệm lớp trong sinh học. Ví dụ: cá chép, cá trôi, cá mè đều thuộc lớp cá.
- Nếu so sánh với các kiểu dữ liệu thì lớp giống như kiểu dữ liệu, còn đối tượng giống như các biến của kiểu đó.

2. Lớp (tiếp)

- Một đối tượng được gọi là một thể hiện (instance) của một lớp bởi vì đối tượng là một trường hợp cụ thể của mô tả lớp. Vì dữ liệu trong các đối tượng của cùng một lớp là khác nhau nên dữ liệu của lớp cũng được gọi là dữ liệu thực thể (instance data).

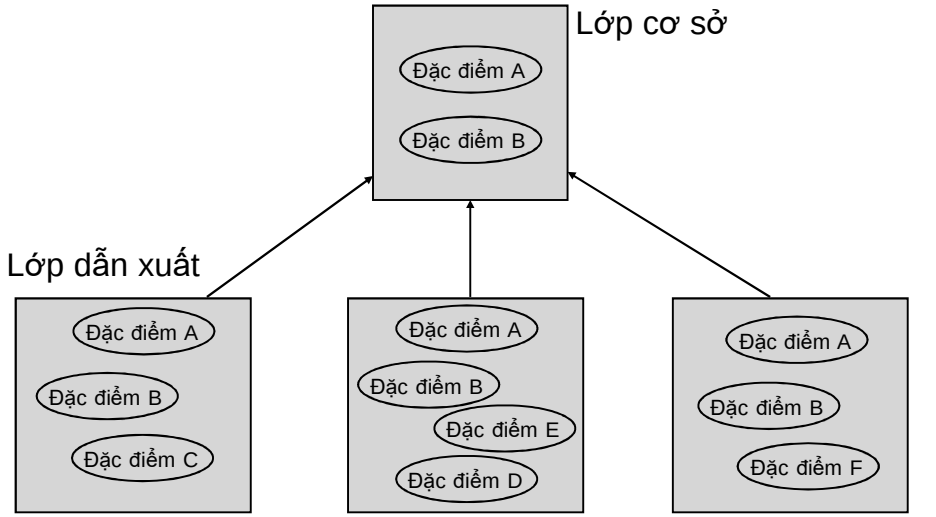
3. Sự kế thừa (inheritance)

- Trong cuộc sống ta thấy các lớp lại được chia thành các lớp con. Chẳng hạn như lớp động vật được chia thành cá, chim, động vật có vú...; lớp xe cộ được chia thành xe con, xe buýt, xe tải, xe máy,...
- Nguyên tắc phân chia thành các lớp con là các lớp con đều có các đặc điểm giống với lớp mà nó tách ra. Ví dụ: xe con, xe tải, xe buýt và xe máy, tất cả đều có tay lái, động cơ, được dùng để vận chuyển người và hàng hoá. Đây là các đặc điểm của xe cộ. Ngoài các đặc điểm này, mỗi lớp con còn có các đặc điểm của riêng nó: Các xe buýt có chỗ ngồi cho nhiều người, xe tải có thùng xe để chở hàng hoá.

3. Sự kế thừa (tiếp)

- Trong lập trình hướng đối tượng một lớp có thể làm cơ sở cho một hoặc nhiều lớp con khác nhau. Một lớp như vậy gọi là lớp cơ sở. Các lớp mà được định nghĩa là có các đặc điểm của lớp cơ sở nhưng thêm vào các đặc điểm mới của riêng nó gọi là các lớp dẫn xuất. Như vậy, các lớp dẫn xuất kế thừa những đặc điểm của lớp cơ sở.

3. Kế thừa (tiếp)



4. Sự sử dụng lại (Reusability)

- Khi một lớp đã được viết hoàn hảo thì có thể bán cho những người lập trình khác để sử dụng trong các chương trình của riêng họ. Việc này gọi là sự sử dụng lại.
- Việc sử dụng lại tương tự như việc sử dụng một thư viện hàm trong lập trình cấu trúc. Tuy nhiên, trong lập trình hướng đối tượng, nhờ có sự kế thừa mà ý tưởng sử dụng lại được mở rộng rất nhiều. Người lập trình có thể lấy một lớp đã có, thêm các đặc điểm và khả năng cho nó mà không cần thay đổi gì. Để làm được điều này người lập trình chỉ đơn giản tạo ra một lớp dẫn xuất kế thừa toàn bộ các đặc điểm của lớp đã có và còn có thể thêm vào các đặc điểm mới.

4. Sự sử dụng lại (Tiếp)

- Ví dụ: giả sử ta đã viết (hoặc mua) một lớp tạo hệ thống menu. Lớp menu này rất tốt và ta không muốn thay đổi nó, nhưng ta lại muốn làm cho một số mục menu nhấp nháy. Để làm được điều này ta chỉ đơn giản tạo ra một lớp dẫn xuất kế thừa tất cả các khả năng của lớp đã có nhưng có thêm các mục menu nhấp nháy.
- Việc có thể sử dụng lại các phần mềm đã có là một lợi ích chính của lập trình hướng đối tượng.

5. Sự đa hình và chồng hàm (polymorphism and overloading)

- Trong lập trình hướng đối tượng ta có thể sử dụng các hàm và các toán tử theo nhiều cách khác nhau tùy thuộc vào những gì mà chúng tác động. Đây gọi là sự đa hình.
- Sự đa hình có thể thực hiện theo hai cách:
  - Đa hình tại thời điểm biên dịch thông qua việc chồng hàm và chồng toán tử.
  - Đa hình tại thời điểm chạy chương trình thông qua việc sử dụng hàm ảo.

5. Sự đa hình và chồng hàm (tiếp)

- Chồng hàm cho phép có nhiều hàm trùng tên nhưng có đối số khác nhau. Chồng toán tử cho phép sử dụng các toán tử đã có (chẳng hạn như +, -) tác động trên các kiểu dữ liệu mới do người lập trình định nghĩa. Khi các hàm hay các toán tử này được gọi thì trình biên dịch biết cách chọn hàm, toán tử nào để thực hiện.
- Trường hợp lớp dẫn xuất và lớp cơ sở có hàm thành viên giống hệt nhau thì khi gọi hàm thành viên này trình biên dịch sẽ không xác định được là gọi hàm nào, hàm trong lớp cơ sở hay lớp dẫn xuất. Chỉ đến khi chạy chương trình mới biết được hàm nào được gọi dựa vào kiểu đối tượng gọi hàm đó.

6. Che giấu thông tin

- Trong LTHDT người ta phân biệt hai công việc: thứ nhất là công việc tạo ra các lớp đối tượng (class creators), thứ hai là công việc sử dụng các lớp đối tượng này.
- Khi tạo lớp, người tạo lớp sẽ xác định những gì cho phép người sử dụng lớp truy nhập, phần còn lại được che giấu và không cho người sử dụng lớp quyền truy nhập.

6. Che giấu dữ liệu (tiếp)

- Khả năng che giấu dữ liệu cho phép những người tạo lớp có thể thay đổi hay định nghĩa lại lớp mà vẫn chắc chắn rằng không ảnh hưởng tới chương trình của những người sử dụng lớp này.
- C++ sử dụng các từ khóa sau để xác định khả năng truy nhập các thông tin dữ liệu từ bên ngoài lớp:

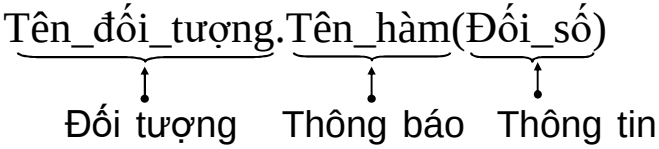
  public, private, và protected.

7. Truyền thông điệp

- Chương trình hướng đối tượng bao gồm một tập các đối tượng và mối quan hệ giữa các đối tượng với nhau.
- 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 điệp giúp chúng ta dễ dàng xây dựng hệ thống mô tả được đầy đủ, trung thực hệ thống trong thực tế. Truyền thông điệp cho một đối tượng tức là báo cho nó phải thực hiện một việc gì đó. Cách đáp ứng của đối tượng được mô tả qua các hàm thành viên của đối tượng.

7. Truyền thông điệp (tiếp)

- Thông điệp truyền đi phải chỉ ra được tên đối tượng nhận thông điệp, tên hàm cần thực hiện và thông tin truyền đi. Cấu trúc một thông điệp như sau:



III. Các ngôn ngữ lập trình hướng đối tượng

- Tư tưởng lập trình hướng đối tượng có thể cài đặt trong nhiều ngôn ngữ lập trình khác nhau như C, Pascal. Tuy nhiên, nếu sử dụng những ngôn ngữ không phải là ngôn ngữ hướng đối tượng thì sẽ gặp rất nhiều khó khăn, nhất là với những chương trình lớn và phức tạp. Những ngôn ngữ được thiết kế để hỗ trợ cho việc mô tả, cài đặt các khái niệm của phương pháp lập trình hướng đối tượng gọi chung là ngôn ngữ hướng đối tượng.

III. Các ngôn ngữ lập trình hướng đối tượng (tiếp)

- Các ngôn ngữ lập trình được gọi là ngôn ngữ hướng đối tượng phải có các đặc điểm sau:
  - Bao gói thông tin: có thể đưa dữ liệu và các hàm thao tác trên dữ liệu đó vào một cấu trúc (gọi là lớp)
  - Cơ chế che giấu dữ liệu
  - Tự động tạo lập và xóa bỏ các đối tượng
  - Sự kế thừa
  - Sự đa hình (chồng hàm, chồng toán tử và liên kết động)
- C++, Smalltalk, Object Pascal,… là các ngôn ngữ lập trình hướng đối tượng.

IV. Phân tích và thiết kế theo hướng đối tượng

Các bước phân tích hướng đối tượng:

Bước 1: Tìm hiểu bài toán

Bước 2: Xác định rõ các đặc tả yêu cầu của người sử dụng, của hệ thống phần mềm.

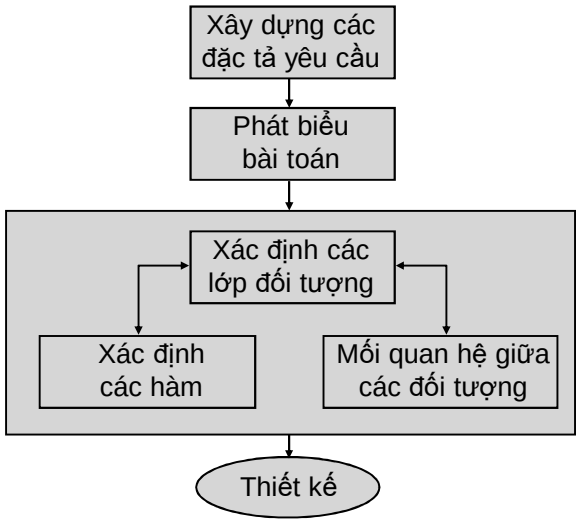
Bước 3: Xác định các đối tượng và thuộc tính của chúng. Từ thuộc tính suy ra các dữ liệu của đối tượng

Bước 4: Xác định các hàm mà đối tượng sẽ phải thực hiện (hành vi của đối tượng)

Bước 5: Xác định mối quan hệ tương tác giữa các đối tượng.

Bước 6: Thiết kế lớp theo ký pháp UML

Quá trình phân tích hướng đối tượng thể hiện qua sơ đồ sau:



V- Các công việc của LTHĐT

1) Tạo ra lớp: Khai báo lớp => Khai báo biến và khai báo hàm thành viên
2) Sử dụng lớp: Tạo đối tượng của lớp => tương tác với đối tượng và cho các đối tượng tương tác với nhau.

Chương 11. Lớp và đối tượng của lớp

I. Mô tả lớp (khai báo lớp)

1. Cú pháp mô tả lớp (định nghĩa lớp)

```
class Tên_lớp
{
      private:

      public:
};
```
← Dấu chấm phẩy

- Tên_lớp đặt theo quy tắc đặt tên
- Mô tả lớp đặt trước hàm main() hoặc để trong một file header.

2. Từ khóa public, private, protected

- Khi định nghĩa lớp ta quy định quyền truy nhập các thành phần của lớp bằng các từ khóa public, private và protected. Theo sau các từ khóa này là dấu 2 chấm.
- Phần của lớp nằm sau từ khóa private: chỉ có thể truy nhập từ bên trong lớp, tức là chỉ có các thành viên của lớp mới có quyền truy nhập. Trong C++, nếu không sử dụng từ khóa private thì mặc định là private.
- Phần của lớp nằm sau từ khóa public: có thể truy nhập từ bất kỳ đâu trong chương trình.

2. Từ khóa public, private, protected (tiếp)

- Phần của lớp nằm sau từ khóa protected: có thể truy nhập từ bên trong lớp và từ các lớp dẫn xuất.
- Thông thường người ta thường để tất cả dữ liệu là private để che giấu dữ liệu, tránh những thay đổi vô tình làm hỏng dữ liệu. Tuy nhiên, các hàm thành viên nên để là public sao cho các phần khác của chương trình có thể gọi chúng để bảo đối tượng làm cái gì đấy.

3. Khai báo dữ liệu của lớp

- Khai báo dữ liệu của lớp là khai báo các biến để lưu trữ các thuộc tính của đối tượng.
- Việc khai báo các biến của lớp không tạo ra các ô nhớ. Nó chỉ đơn giản báo cho trình biên dịch biết về tên biến và kích thước bộ nhớ sẽ cần khi đối tượng được tạo. Khi khai báo các biến của lớp ta không khởi tạo được giá trị cho biến vì chưa có ô nhớ.

  Ví dụ:

```
private:
    int x,y;
```

4. Khai báo và định nghĩa các hàm thành viên của lớp

- Các hàm thành viên lớp được khai báo và định nghĩa giống như các hàm thông thường.
- Ta có thể định nghĩa các hàm thành viên ngay trong mô tả lớp và không cần khai báo các hàm này nữa. Thông thường thì chỉ với các hàm thành viên nhỏ (chỉ có một vài dòng lệnh) người ta mới định nghĩa ngay trong mô tả lớp. Bởi vì nếu ta định nghĩa hàm thành viên ngay trong mô tả lớp thì mặc định nó là hàm inline. Hàm inline khác hàm các hàm bình thường ở chỗ: khi dịch chương trình, trình biên dịch không để mã của hàm ở một vùng nhớ riêng mà chèn ngay vào vị trí lời gọi hàm. Bởi vậy, nếu để hàm inline lớn sẽ làm tăng kích thước chương trình.

4. Khai báo và định nghĩa các hàm thành viên của lớp (tiếp)

- Nếu định nghĩa các hàm thành viên bên ngoài mô tả lớp thì bên trong mô tả lớp phải có khai báo về các hàm thành viên này. Các hàm thành viên định nghĩa bên ngoài lớp thì trước tên hàm phải có tên lớp, giữa tên hàm và tên lớp cách nhau bởi hai dấu hai chấm liền nhau (::). Hai dấu hai chấm này là toán tử quy định phạm vi (scope resolution operator).

4. Khai báo và định nghĩa các hàm thành viên của lớp (tiếp)

- Cú pháp định nghĩa hàm thành viên bên ngoài mô tả lớp như sau:

```
class Ten_lop
{
   private:

   public:
     Kieu Ten_ham();
};
Kieu Ten_lop::Ten_ham()
  {
    //Cac lenh cua ham
  }
```


Ví dụ về lớp

```
Lớp đối tượng thời gian lưu trữ giờ và phút.
class airtime
  {
    private:
      int hours;          //Tu 0 den 23
      int minutes;        //Tu 0 den 59
    public:
      void set();         //Khai bao ham thanh vien
      void display()      //Ham inline
       {
         cout<<hours<<':'<<minutes;
       }
  };
void airtime::set()
  {
    char kt;  //Dung de chua dau hai cham

    cout<<"Nhap vao thoi gian (dang 20:45): ";
    cin>>hours>>kt>>minutes;
  }
```


II. Tạo và tương tác với các đối tượng

1. Tạo các đối tượng của một lớp

- Việc tạo ra lớp chỉ là tạo ra bản thiết kế để sau này tạo các đối tượng.
- Cú pháp tạo các đối tượng giống cú pháp tạo các biến (khai báo biến).

  Tên_lớp Tên_đối_tượng;
- Trong C++, các đối tượng được đối xử như các biến, còn các lớp được đối xử như các kiểu dữ liệu.

  Ví dụ: airtime t1,t2;

2. Gửi thông điệp tới các đối tượng

- Khi một đối tượng được tạo thì ta có thể tương tác với nó bằng cách sử dụng các hàm hành viên. Việc gọi hàm thành viên của một đối tượng gọi là gửi thông điệp tới đối tượng đó.
- Cú pháp gửi thông báo tới một đối tượng:

  Tên_đối_tượng.Tên_hàm();

  Ví dụ: t1.display();

  Sau đây là một chương trình hoàn chỉnh về việc tạo lớp và các đối tượng của lớp.

3. Mảng đối tượng

- Bởi vì C++ đối xử với các đối tượng như các biến nên ta cũng có thể khai báo một mảng các đối tượng. Mảng các đối tượng rất hữu ích khi chúng ta muốn tạo một số lượng lớn các đối tượng của cùng một lớp. Ví dụ: ta có một lớp nhân viên và ta muốn tạo 500 đối tượng cho 500 nhân viên thì cách tốt nhất là tạo một mảng 500 đối tượng nhân viên.
- Cú pháp tạo mảng đối tượng giống cú pháp khai báo biến mảng:

  Tên_lớp Tên_mảng_đối_tượng[Số_đối_tượng];

  Dữ liệu của các đối tượng trong mảng được lưu trữ liên tiếp nhau trong bộ nhớ.

3. Mảng đối tượng (tiếp)

- Để gửi thống báo tới một đối tượng cụ thể trong mảng đối tượng ta phải dùng thêm ký hiệu của mảng để xác định đối tượng muốn gửi thống báo tới. Ví dụ:

  airtime at[20];

  at[2].display();

  Lệnh này gửi thống báo tới đối tượng thứ 3 trong mảng đối tượng at.

  Chương trình về mảng đối tượng thời gian airtime.

4. Con trỏ trỏ tới đối tượng

- Các đối tượng được lưu trữ trong bộ nhớ nên chúng cũng có địa chỉ. Bởi vậy, con trỏ có thể trỏ tới các đối tượng giống như trỏ tới các biến kiểu cơ bản.
- Cú pháp khai báo biến con trỏ trỏ tới đối tượng như sau:

  Tên_lớp *Tên_con_trỏ;

  Ví dụ: airtime *p;

  //p có thể trỏ tới các đối tượng lớp airtime.
- Để đưa địa chỉ của đối tượng vào biến con trỏ ta dùng toán tử lấy địa chỉ &

  Ví dụ: airtime t1; //tạo đối tượng t1

  airtime* p= &t1; //tạo con trỏ p trỏ tới t1

  airtime *q = new airtime;

4. Con trỏ trỏ tới đối tượng (tiếp)

- Để truy nhập tới các thành viên của đối tượng do con trỏ p trỏ tới ta có 2 cách:
  - Sử dụng toán tử truy nhập gián tiếp và toán tử dấu chấm: (*p).Thành_viên

    Ví dụ: (*p).display();
  - Sử dụng toán tử truy nhập thành viên -> (gồm dấu trừ và dấu lớn hơn liền nhau): p->Thành_viên

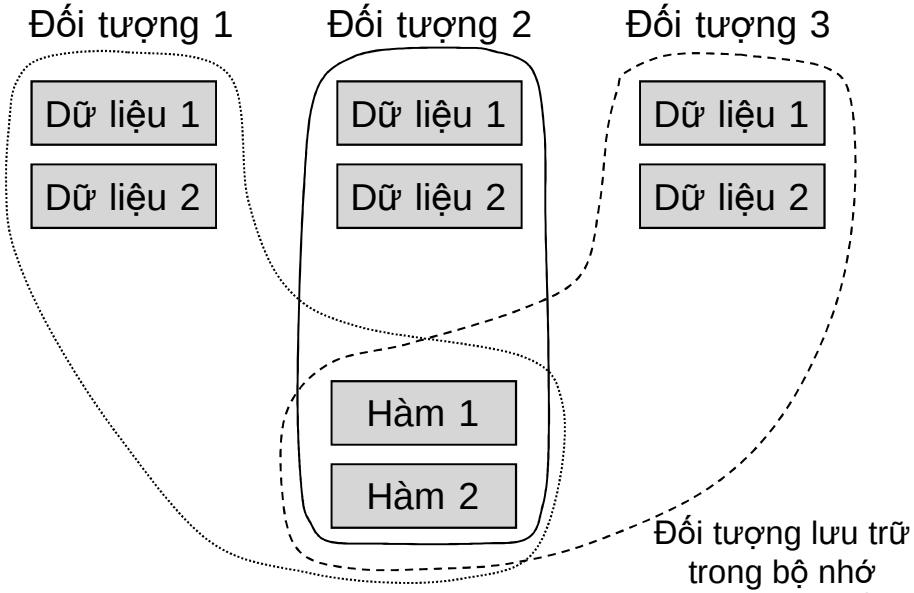
    Ví dụ: p->display();

  Cách thứ hai gọn hơn cách nhất. Với con trỏ trỏ tới đối tượng người ta hay dùng cách thứ hai.

5. Lệnh gán đối tượng

- Với các biến kiểu cơ bản ta có thể gán giá trị của một biến cho một biến cùng kiểu. Vậy có thể gán giá trị của một đối tượng cho một đối tượng được không? Câu trả lời là có, bởi vì C++ coi các đối tượng như các biến.
- Nhưng đối tượng bao gồm cả dữ liệu và các hàm thành viên, khi gán một đối tượng cho một đối tượng khác thì trình biên dịch sẽ làm như thế nào? Trình biên dịch chỉ copy các mục dữ liệu, không copy các hàm thành viên. Bởi vì tất cả các đối tượng của cùng một lớp có các hàm thành viên giống nhau. Trong bộ nhớ chỉ có một bản các hàm thành viên, các đối tượng sử dụng chung các hàm thành viên này. Các hàm thành viên sẽ tác động trên dữ liệu của đối tượng nào gọi nó.

5. Lệnh gán đối tượng (tiếp)



5. Lệnh gán đối tượng (tiếp)

- Ví dụ: giả sử t1, t2 là hai đối tượng thời gian airtime, sau khi lấy giá trị giờ và phút cho t1 ta gán t1 cho t2.

```
airtime t1, t2;
t1.set();
t1.display();
t2 = t1;
t2.display();
```

vdp2c23.cpp

6. Truy nhập dữ liệu của các đối tượng cùng lớp

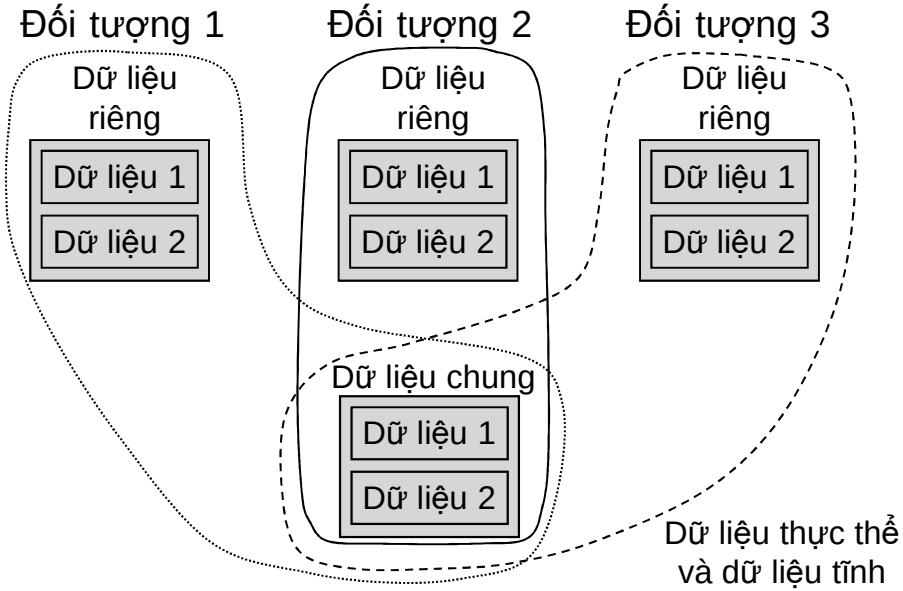
- Các hàm thành viên có thể truy nhập trực tiếp dữ liệu private của các đối tượng cùng lớp.
- *Bài toán:* Tính tổng hai số phức.
- BTVN: Tính tổng hai phân số.

III. Các thành viên tĩnh của lớp (static member)

1. Dữ liệu thành viên tĩnh

- Dữ liệu riêng gắn với một đối tượng cụ thể. Chúng tồn tại khi đối tượng được tạo và mất đi khi đối tượng bị hủy. Nhưng nếu chúng ta cần một biến mà có thể dùng cho cả lớp đối tượng chứ không phải cho một đối tượng cụ thể thì làm thế nào? Có thể chúng ta sẽ nghĩ tới các biến ngoài, nhưng các biến ngoài lại không gắn với một lớp cụ thể và có nhiều vấn đề không tốt. Dữ liệu thành viên tĩnh sẽ giải quyết được vấn đề này.

1. Dữ liệu thành viên tĩnh (tiếp)



1. Dữ liệu thành viên tĩnh (tiếp)

- Để có dữ liệu thành viên tĩnh ta phải dùng hai lệnh: một lệnh khai báo biến nằm trong mô tả lớp, một lệnh định nghĩa biến đó nằm ngoài mô tả lớp. Ví dụ:

```
class aclass
 {
   private:
     static int a;       //Khai bao thanh vien tinh
     .....
 };
int aclass::a=100;   //Dinh nghia, khoi tao = 100
```


1. Dữ liệu thành viên tĩnh (tiếp)

- Dữ liệu thành viên tĩnh có thể được khởi tạo khi định nghĩa. Nếu ta không khởi tạo thì chúng được tự động khởi tạo bằng 0.
- Ta có thể truy nhập dữ liệu thành viên tĩnh từ bất kỳ hàm thành viên thông thường nào. Tuy nhiên, người ta thường dùng một loại hàm đặc biệt dành cho cả lớp để truy nhập dữ liệu thành viên tĩnh. Hàm này gọi là hàm tĩnh (static function).

2. Hàm thành viên tĩnh

- Việc khai báo và định nghĩa hàm thành viên tĩnh giống như các hàm thành viên thông thường chỉ khác là dùng thêm từ khóa static.
- Lời gọi hàm thành viên tĩnh không giống lời gọi hàm thành viên thông thường. Lời gọi hàm thành viên tĩnh không gắn với đối tượng mà gắn với tên lớp bằng toán tử quy định phạm vi: tên_lớp::tên_hàm_tĩnh.
- Hàm thành viên tĩnh chỉ truy nhập được các dữ liệu tĩnh, bởi vì chúng không biết gì về các đối tượng của lớp. Thậm chí ta có thể gọi hàm thành viên tĩnh trước khi tạo bất kỳ đối tượng nào của lớp.

2. Hàm thành viên tĩnh (tiếp)

```
class aclass
 {
   private:
     .....
   public:
     static void stafunc();  //Khai bao
};

void main()
{
   ......
   aclass::stafunc();  //Goi ham thanh vien tinh
}
void aclass::stafunc()  //Dinh nghia
{
      //Ham tinh chi co the truy nhap du lieu thanh vien tinh
}
```

Bài tập chương 11

Bài 1. Viết chương trình nhập vào một thời gian có giờ và phút. Tính và đưa ra màn hình thời gian sau n phút nhập vào từ bàn phím.

Bài 2. Viết chương trình nhập vào n số phức. Đưa các số phức đã nhập ra màn hình. Yêu cầu trong chương trình phải tạo đối tượng động.

Bài 3. Nhập thông tin của một số cán bộ. Mỗi cán bộ có thông tin về mã cán bộ, tên. Mã cán bộ là số thứ tự của cán bộ, được lấy tự động. Đưa ra màn hình thông tin về các cán bộ và tổng số cán bộ đã nhập.

Bài tập chương 11

Bài 4. Viết chương trình nhập vào danh sách sinh viên cho tới khi không muốn nhập thì thôi, mỗi sinh viên có thông tin về mã sinh viên, tên và điểm tbc. Mã SV là các số nguyên được lấy tự động có giá trị từ 11 trở đi. Đưa ra màn hình số lượng và danh sách sinh viên đã nhập. Yêu cầu trong chương trình có sử dụng biến tĩnh và hàm tĩnh, sử dụng đối tượng động.

Chương 12. Chồng hàm (function overloading)

I. Chồng hàm

II. Các loại biến

I. Chồng hàm (function overloading)

1. Sự cần thiết phải chồng hàm

2. Trình biên dịch và các hàm chồng

1. Sự cần thiết phải chồng hàm

- *Bài tập 1:* Viết hàm tính trung bình cộng của một mảng int, long, float và double.
  - Với bài tập này, bình thường ta phải viết 4 hàm để tính trung bình cho 4 mảng khác nhau và khi gọi hàm ta phải nhớ 4 tên hàm này. Tuy nhiên, C++ cho phép nhiều hàm có tên giống nhau chỉ cần khác nhau về đối số. Việc sử dụng cùng một tên cho nhiều hàm gọi là chồng hàm. Chồng hàm giúp người sử dụng không phải nhớ nhiều tên hàm khác nhau.

2. Trình biên dịch và các hàm chồng

- Làm thế nào mà trình biên dịch có thể phân biệt được các hàm có cùng tên? Trình biên dịch sẽ tạo ra một tên mới cho mỗi hàm bằng cách kết hợp tên hàm với tên kiểu của các đối số.

  Ví dụ: tbc_int_int(), tbc_long_int()
- *Bài tập về nhà:*
  - Viết chương trình tính bình phương của một số int, long, float, double.
  - Làm thế nào để lấy địa chỉ của các hàm được chồng?

II. Các loại biến

1. Sự khác nhau giữa khai báo và định nghĩa
2. Thời gian tồn tại và phạm vi hoạt động của các loại biến

1. Sự khác nhau giữa khai báo và định nghĩa

- Một khai báo (declaration) chỉ xác định tên và kiểu dữ liệu. Nhiệm vụ của khai báo là cung cấp thông tin cho trình biên dịch, nó không yêu cầu trình biên dịch làm bất cứ việc gì.
- Trái lại, một định nghĩa (definition) yêu cầu trình biên dịch phải cấp phát bộ nhớ cho biến.
- Trong một số trường hợp khai báo cũng yêu cầu trình biên dịch cấp phát bộ nhớ, chẳng hạn như khai báo biến. Tuy nhiên, với định nghĩa thì trong bất kỳ trường hợp nào cũng yêu cầu cấp phát bộ nhớ.

2. Thời gian tồn tại và phạm vi hoạt động của các loại biến

- Các loại biến có hai đặc tính chính là phạm vi hoạt động và thời gian tồn tại. Phạm vi hoạt động liên quan đến phần chương trình nào có thể truy nhập (sử dụng) biến. Thời gian tồn tại là khoảng thời gian trong đó biến tồn tại. Phạm vi hoạt động của biến có thể là trong một lớp, một hàm, một file hay một số file. Thời gian tồn tại của một biến có thể trùng với một đối tượng, một hàm hay toàn bộ chương trình.
- Có các loại biến sau: biến tự động, biến thanh ghi, biến trong khối lệnh, biến ngoài, biến tĩnh và đối tượng.

a) Các biến tự động (automatic variable)

- Các biến tự động là các biến được khai báo trong một hàm. Sở dĩ gọi chúng là các biến tự động bởi vì chúng được tự động tạo khi hàm được gọi và bị hủy khi hàm kết thúc.
  - Biến tự động có phạm vi hoạt động trong một hàm. Do đó, một biến i được khai báo trong một hàm hoàn toàn khác với một biến i được khai báo trong một hàm khác.
  - Mặc định các biến tự động không được khởi tạo, bởi vậy ngay sau khi chúng được hình thành chúng sẽ có một giá trị vô nghĩa.

b) Các biến thanh ghi (register variable)

- Biến thanh ghi là một loại biến tự động đặc biệt. Nó được đặt trong các thanh ghi của CPU chứ không phải trong bộ nhớ. Việc truy nhập các biến thanh ghi nhanh hơn các biến thông thường. Biến thanh ghi có lợi nhất khi được dùng làm biến điều khiển cho lệnh lặp bên trong nhất trong các lệnh lặp lồng nhau. Ta chỉ nên dùng một đến hai biến thanh ghi trong một hàm.
- Để khai báo biến thanh ghi ta dùng từ khóa register trước khai báo biến thông thường.

  Ví dụ: register int a;

c) Các biến trong khối lệnh

- Các biến tự động có thể được khai báo ở bất kỳ đâu trong một hàm hoặc trong một khối lệnh. Khối lệnh là phần chương trình nằm giữa hai dấu ngoặc { và }, chẳng hạn như thân lệnh if hay thân lệnh lặp. Các biến được khai báo trong một khối lệnh có phạm vi hoạt động chỉ trong khối lệnh đó.

d) Các biến ngoài (external variable)

- Các biến ngoài là các biến được khai báo ở bên ngoài tất cả các hàm và các lớp. Các biến ngoài có phạm vi hoạt động từ vị trí khai báo đến cuối file khai báo chúng. Thời gian tồn tại của các biến ngoài là thời gian tồn tại của chương trình, tức là khi chương trình kết thúc thì các biến ngoài mới bị hủy. Khác với các biến tự động, các biến ngoài được tự động khởi tạo bằng 0 nếu ta không khởi tạo.

d) Các biến ngoài (*tiếp*)

```
//Bat dau file
int a;  //a la bien ngoai
.....
class aclass
  {
    .......
  };
void afunc();
.......
//Cuoi file
```

d) Các biến ngoài (*tiếp*)

- Nếu chương trình được chia thành nhiều file thì các biến ngoài chỉ có thể dùng được trong file khai báo chúng, không dùng được trong các file khác. Để các file khác có thể sử dụng một biến ngoài đã được định nghĩa ở một file ta phải khai báo biến đó dùng từ khóa extern.
- Để các biến ngoài chỉ truy nhập được trong file khai báo chúng, không truy nhập được từ file khác ta dùng từ khóa static. Trong ngữ cảnh này, từ khóa static có nghĩa là hạn chế phạm vi hoạt động của biến.

  Ví dụ: (*trang sau*)

d) Các biến ngoài (*tiếp*)

Ví dụ 1: Truy nhập biến ngoài trên nhiều file

```
//Bat dau file 1
int a;    //a la bien ngoai
//Cuoi file 1

//Bat dau file 2
extern int a;     //khai bao su dung bien ngoai a o file 1
          //Trong file 2 co the truy nhap bien a
//Cuoi file 2

//Bat dau file 3
//Khong khai bao su dung bien ngoai a nen trong file 3
// khong the truy nhap bien a
//Cuoi file 3
```


d) Các biến ngoài (*tiếp*)

Ví dụ 2: Hạn chế việc truy nhập biến ngoài

```
//Bat dau file 1
static int a;   //dinh nghia bien ngoai a
                //bien a chi truy nhap duoc trong file nay
//Cuoi file 1


//Bat dau file 2
extern int a;  //Khong dung duoc khai bao nay
//Cuoi file 2
```


d) Các biến ngoài (*tiếp*)

- Có hai vấn đề khi sử dụng biến ngoài:
  - Vì biến ngoài có thể truy nhập được từ bất kỳ hàm nào trong chương trình nên rất dễ bị thay đổi làm mất dữ liệu.
  - Vì các biến ngoài có phạm vi hoạt động ở mọi nơi trong chương trình nên ta phải quan tâm đến vấn đề kiểm soát tên biến để sao cho không có hai biến nào trùng tên.

e) Các biến tĩnh cục bộ (local static)

- Các biến tĩnh cục bộ được sử dụng khi ta muốn duy trì giá trị của một biến khai báo trong hàm giữa các lời gọi hàm. Tức là khi hàm kết thúc biến tĩnh vẫn còn và vẫn chứa giá trị, khi hàm được gọi lần 2 lại có thể sử dụng giá trị này. Phạm vi hoạt động của biến tĩnh cục bộ là trong hàm nhưng thời gian tồn tại của nó là suốt thời gian chương trình chạy.
- *Trong lập trình hướng đối tượng người ta không dùng biến tĩnh một mình mà dùng trong lớp đối tượng.*

f) Đối tượng

- Đối tượng được C++ đối xử như các biến. Các đối tượng có thể được tạo ở dạng biến tự động, biến ngoài,… những không tạo được ở dạng biến thanh ghi.
- Phạm vi hoạt động của thành viên dữ liệu riêng (được khai báo private) của lớp đối tượng là chỉ trong các hàm thành viên của lớp. Còn phạm vi hoạt động của các hàm thành viên (được khai báo public) là tất cả các hàm và các lớp trong chương trình.
- Thời gian tồn tại của dữ liệu riêng là (dù là private hay public) là thời gian tồn tại của đối tượng.

Chương 13. Hàm tạo và hàm hủy (constructor & destructor)

I. Giới thiệu về hàm tạo và hàm hủy
  I.1. Giới thiệu về hàm tạo và hàm hủy
  I.2. Hàm tạo và hàm hủy do người lập trình viết
II. Hàm tạo có đối số
  II.1. Hàm tạo hai đối số
  II.2. Hàm tạo mặc định
  II.3. Hàm tạo một đối số
III. Hàm tạo sao chép

I.1. Giới thiệu về hàm tạo và hàm hủy

- Hàm tạo và hàm hủy là hai hàm thành viên đặc biệt của đối tượng. Hàm tạo được thực hiện tự động khi đối tượng được tạo, còn hàm hủy được tự động thực hiện khi đối tượng bị hủy.
- Chúng ta thường viết hàm tạo để khởi tạo đối tượng, viết hàm hủy để giải phóng bộ nhớ cấp phát bởi hàm tạo.
- Dù người lập trình có viết hay không viết hàm tạo và hàm hủy thì trình biên dịch vẫn tạo ra những mã lệnh để tạo đối tượng, cấp phát bộ nhớ cho nó và thực hiện một số khởi tạo nào đó; tạo ra mã lệnh để giải phóng bộ nhớ chiếm bởi đối tượng và thực hiện nhiều hoạt động dọn dẹp khác khi đối tượng bị hủy.

Chương 4. Hàm tạo và hàm hủy

I. Giới thiệu về hàm tạo và hàm hủy
  I.1. Giới thiệu về hàm tạo và hàm hủy
  I.2. Hàm tạo và hàm hủy do người lập trình viết
II. Hàm tạo có đối số
  II.1. Hàm tạo hai đối số
  II.2. Hàm tạo mặc định
  II.3. Hàm tạo một đối số
III. Hàm tạo sao chép

I.2. Hàm tạo và hàm hủy do người lập trình viết

- Người lập trình có thể tự định nghĩa hàm tạo và hàm hủy của riêng mình.
- Hàm tạo và hàm hủy có thể định nghĩa ngay trong mô tả lớp. Cả hai hàm này đều không có kiểu trả về, kể cả kiểu void. Hàm tạo có tên trùng với tên lớp, hàm hủy cũng có tên trùng với tên lớp nhưng có dấu ~ đứng trước.
- Ví dụ: Định nghĩa một lớp chỉ có hàm tạo và hàm hủy, sau đó tạo 2 đối tượng của lớp này và xem các hàm tạo và hàm hủy thực hiện thế nào.

I.2. Hàm tạo và hàm hủy do người lập trình viết (tiếp)

- Ta có thể dùng hàm tạo để khởi tạo giá trị cho các biến của đối tượng. Có 2 cách khởi tạo:
  - Dùng danh sách khởi tạo: danh sách khởi tạo nằm cùng dòng với tên hàm tạo, bắt đầu bằng dấu hai chấm, sau đó là các biến cần khởi tạo cách nhau bởi dấu chấm, giá trị khởi tạo đặt trong ngoặc đơn ngay sau tên biến. Ví dụ: giả sử lớp Alpha có 2 biến nguyên là a và b, hàm tạo khởi tạo giá trị cho 2 biến này như sau:

    Alpha() : a(5), b(6)
    { }
  - Dùng lệnh gán giá trị trong thân của hàm tạo. Cách này chỉ áp dụng với một số biến lớn như biến mảng, đối tượng.
- Ví dụ:

Chương 4. Hàm tạo và hàm hủy

II.1. Hàm tạo hai đối số

- Ví dụ: Viết lớp số phức có 1 hàm tạo không đối số để khởi tạo phần thực và phần ảo bằng 0, có 1 hàm tạo hai đối số để khởi tạo phần thực và phần ảo bằng đối số, có một hàm hủy.
- Bài tập về nhà: Viết một lớp Stack có thể chứa các số nguyên. Nhập vào 1 số nguyên dương, đưa ra số nhị phân.

II.1. Hàm tạo hai đối số (tiếp)

- Cú pháp "Gọi" hàm tạo có hai đối số: Bởi vì hàm tạo được thực hiện tự động nên ta không thể sử dụng lời gọi hàm thông thường để truyền đối số cho nó. Việc truyền đối số cho hàm tạo được thực hiện khi tạo đối tượng. Giá trị của các đối số được đặt trong ngoặc đơn sau tên đối tượng.

Tên_lớp Tên_đối_tượng(Danh sách đối số);

Chương 4. Hàm tạo và hàm hủy

I. Giới thiệu về hàm tạo và hàm hủy
   I.1. Giới thiệu về hàm tạo và hàm hủy
   I.2. Hàm tạo và hàm hủy do người lập trình viết
II. Hàm tạo có đối số
   II.1. Hàm tạo hai đối số
   II.2. Hàm tạo mặc định
   II.3. Hàm tạo một đối số
III. Hàm tạo sao chép

II.2. Hàm tạo mặc định

- Nếu ta không định nghĩa hàm tạo thì trình biên dịch sẽ tạo ra một hàm tạo mặc định, hàm tạo này không có đối số.
- Tuy nhiên, nếu ta tự định nghĩa hàm tạo, dù là có đối số hay không có đối số thì trình biên dịch sẽ không tạo ra hàm tạo mặc định nữa.
- Bài tập 2: Nếu trong lớp airtime ta định nghĩa hàm tạo 2 đối số thì khai báo sau sẽ sinh ra lỗi.

  airtime t1, t2;

  Hãy thay đổi lớp airtime để khai báo trên vẫn đúng trong khi lớp vẫn có hàm tạo 2 đối số.

Chương 4. Hàm tạo và hàm hủy

I. Giới thiệu về hàm tạo và hàm hủy
   I.1. Giới thiệu về hàm tạo và hàm hủy
   I.2. Hàm tạo và hàm hủy do người lập trình viết
II. Hàm tạo có đối số
   II.1. Hàm tạo hai đối số
   II.2. Hàm tạo mặc định
   II.3. Hàm tạo một đối số
III. Hàm tạo sao chép

II.3. Hàm tạo một đối số

- Hàm tạo một đối số có vai trò quan trọng trong lập trình hướng đối tượng. Nó được dùng để chuyển đổi một đối tượng lớp này sang lớp khác. Việc chuyển đổi này thường dùng với các lớp biểu diễn kiểu dữ liệu. Chính vì lý do này mà hàm tạo con được gọi là hàm chuyển đổi.
- Để thấy được cách khai báo và sử dụng hàm tạo một đối số ta xét ví dụ sau: Giả sử lớp TypeA có hàm tạo một đối số kiểu int được khai báo như sau: (Trang sau)

II.3. Hàm tạo một đối số (tiếp)

```
class TypeA
{
   public:
     TypeA(int i)  //Ham tao mot doi so
    {
      //Thuc hien chuyen doi gia tri int toi gia tri TypeA
    }
};
void main()
  {
   int b=50;       //Khai bao mot bien int co gia tri 50

   TypeA ta1(b);//Khoi tao doi tuong ta1 bang gia tri int
   //TypeA ta1=b;          //Tuong duong voi ta1(b)
  }
```


II.3. Hàm tạo một đối số (tiếp)

- Dấu = trong cách thứ hai không phải là toán tử gán, nó chỉ có tác dụng gọi hàm tạo một đối số.
- Bài tập về nhà: Hãy xây dựng một lớp về xâu ký tự trong đó có sử dụng hàm tạo một đối số để chuyển đổi một xâu ký tự thông thường thành đối tượng xâu ký tự.

Chương 4. Hàm tạo và hàm hủy

III. Hàm tạo sao chép

- Hàm tạo sao chép (copy constructor) cho phép ta tạo ra một đối tượng là bản sao của một đối tượng đã có.
- Hàm tạo sao chép là một hàm tạo chỉ có một đối số, đối số này là đối tượng của lớp chứa hàm tạo.

III. Hàm tạo sao chép (tiếp)

- Khi khai báo một biến kiểu cơ bản ta có thể khởi tạo giá trị cho nó. Ví dụ:

```
int a=45;       //tao bien a va khoi tao bang 45
int b=a;        //tao bien b la ban sao cua a
//int b(a);     //tuong duong voi lenh int b=a;
```

Dấu = trong khai báo trên không phải là lệnh gán, nó chỉ có nghĩa là khởi tạo.

- Với đối tượng ta cũng có thể tạo và khởi tạo cho nó giống như với một biến kiểu cơ bản. Ví dụ: giả sử có lớp Alpha

```
Alpha a1;       //tao mot doi tuong Alpha
a1.getdata();//dua du lieu vao doi tuong a1
Alpha a2=a1;//tao doi tuong a2 la ban sao cua a1, o day ham
 //tao sao chep duoc goi de sao chep du lieu tu a1 sang a2
```


III. Hàm tạo sao chép (tiếp)

- Nếu không định nghĩa hàm tạo sao chép thì trình biên dịch sẽ tạo ra một hàm tạo sao chép mặc định. Hàm tạo sao chép mặc định sao chép y nguyên tất cả dữ liệu từ một đối tượng này sang một đối tượng khác. Bởi vậy, nếu ta chỉ cần sao chép dữ liệu từ đối tượng này sang đối tượng khác thì không cần định nghĩa hàm tạo sao chép. Còn nếu ta muốn làm những việc khác thì ta phải định nghĩa riêng một hàm tạo sao chép.
- Hàm tạo sao chép phải được khai báo để truyền đối số theo tham chiếu và đối số nên để là const.

III. Hàm tạo sao chép (tiếp)

- Hàm tạo sao chép được gọi trong các trường hợp sau:
  - Khi khởi tạo đối tượng
  - Khi truyền đối tượng cho hàm theo giá trị
  - Khi hàm trả về đối tượng
- Ví dụ về sử dụng hàm tạo, hàm hủy, biến ngoài và biến tự động: Viết chương trình tạo ra một lớp sao cho khi tạo đối tượng có thể khởi tạo đối tượng bằng một xâu ký tự; hàm tạo đối tượng cần hiển thị dữ liệu của đối tượng, đây là đối tượng thứ mấy và tổng số đối tượng đang tồn tại là bao nhiêu; còn hàm huỷ cần hiển thị là huỷ đối tượng thứ mấy và sau khi huỷ thì còn bao nhiêu đối tượng đang tồn tại; trước khi hàm main kết thúc cần hiển thị tổng số đối tượng đã tạo ra trong thời gian chương trình chạy.

Chương 14. Chồng toán tử, hàm bạn

I. Tại sao phải chồng toán tử?

II. Chồng các toán tử hai ngôi

III. Chồng các toán tử một ngôi

IV. Chuyển đổi giữa các đối tượng và kiểu dữ liệu cơ bản

V. Chuyển đổi giữa các lớp

VI. Chồng toán tử gán = và toán tử [ ]

VII. Chồng toán tử nhập/xuất - Hàm bạn (friend function)

I. Tại sao phải chồng toán tử?

- Chồng toán tử là sử dụng các toán tử có sẵn để tác động trên các toán hạng khác nhau, tức là ta có thể định nghĩa tác động của các toán tử trên các đối tượng lớp.
- Chồng toán tử giúp chương trình dễ viết, dễ đọc và dễ hiểu. Ví dụ: giả sử ta muốn cộng hai đối tượng của lớp airtime rồi gán kết quả nhận được vào một đối tượng airtime khác. Khi đó, ta viết

  at3=at1+at2

  sẽ dễ hiểu hơn là viết

  at3=at1.add(at2) hay at3.add(at1,at2)

Chương 14. Chồng toán tử

I. Tại sao phải chồng toán tử?

II. Chồng các toán tử hai ngôi

III. Chồng các toán tử một ngôi

IV. Chuyển đổi giữa các đối tượng và kiểu dữ liệu cơ bản

V. Chuyển đổi giữa các lớp

VI. Chồng toán tử gán = và toán tử [ ]

Chương 14. Chồng toán tử

II. Chồng các toán tử hai ngôi

II.1. Chồng các toán tử số học

II.2. Chồng các toán tử quan hệ

II.3. Chồng các toán tử gán phức hợp

II.1. Chồng các toán tử số học

- Ví dụ 1: Viết chương trình cộng hai số phức nhập vào từ bàn phím bằng toán tử cộng +.

- Bài về nhà 1: Xây dựng lớp đối tượng xâu ký tự để có thể dùng phép cộng ghép nhiều xâu ký tự thông thường thành một xâu.

II.1. Chồng các toán tử số học

- Các toán tử có thể chồng là +, -, *, /
- Để chồng một toán tử ta phải định nghĩa một hàm xác định phép toán mà toán tử đó sẽ thực hiện. Hàm chồng toán tử giống như các hàm bình thường, chỉ khác tên hàm là từ khóa operator kết hợp với toán tử: operatorX, trong đó X là toán tử. Ví dụ để chồng toán tử + ta có tên hàm là operator+
- Lời gọi hàm chồng toán tử có thể dùng cú pháp giống như hàm bình thường. Ví dụ:

  t3 = t1.operator+(t2);

  Nhưng từ khóa operator, dấu chấm và cặp dấu ngoặc () là không cần thiết. Bởi vậy ta viết:

  t3 = t1 + t2;

Chương 14. Chồng toán tử

II. Chồng các toán tử hai ngôi

- II.1. Chồng các toán tử số học
- II.2. Chồng các toán tử quan hệ
- II.3. Chồng các toán tử gán phức hợp

II.2. Chồng các toán tử quan hệ

- Ta có thể chồng tất cả các phép toán so sánh (quan hệ).
- Bài tập 2: Viết hàm thành viên chồng toán tử so sánh nhỏ hơn (<) để so sánh hai đối tượng lớp airtime.

Chương 14. Chồng toán tử

II. Chồng các toán tử hai ngôi

II.1. Chồng các toán tử số học

II.2. Chồng các toán tử quan hệ

II.3. Chồng các toán tử gán phức hợp

II.3. Chồng các toán tử gán phức hợp

- Có thể chồng các toán tử phức hợp sau:

  +=, -=, *=, /=
- Toán tử gán khác với các toán tử hai ngôi thông thường ở chỗ là chúng thay đổi đối tượng gọi chúng.
- Mục đích chính của toán tử gán là thay đổi đối tượng nhưng chúng cũng thường được dùng để trả về giá trị.
- Bài tập 3: Chồng toán tử gán += cho lớp airtime sao cho có thể dùng nó để gán các đối tượng airtime cho nhau.

II.3. Chồng các toán tử gán phức hợp (tiếp)

- Khi trả về đối tượng ta nên dùng lệnh trả về đặc biệt sau:

  Ví dụ: return airtime(hours,minutes);

  Lệnh đặc biệt này tạo đối tượng trả về, hàm tạo sao chép không thực hiện.

Chương 14. Chồng toán tử

I. Tại sao phải chồng toán tử?

II. Chồng các toán tử hai ngôi

III. Chồng các toán tử một ngôi

IV. Chuyển đổi giữa các đối tượng và kiểu dữ liệu cơ bản

V. Chuyển đổi giữa các lớp

VI. Chồng toán tử gán = và toán tử [ ]

III. Chồng các toán tử một ngôi

- Toán tử một ngôi là các toán tử chỉ có một toán hạng. Ví dụ: toán tử tăng ++, toán tử giảm --, toán tử dấu âm – và toán tử phủ định logic !. Hay dùng nhất là toán tử tăng giảm. Toán tử tăng và giảm có thể dùng ở trước hoặc sau toán hạng.
- Bài tập 4: Chồng toán tử ++ cho lớp airtime để tăng đối tượng airtime lên 1 phút, toán tử ++ có thể đứng trước và sau đối tượng.

III. Chồng các toán tử một ngôi

- Toán tử một ngôi là các toán tử chỉ có một toán hạng. Ví dụ: toán tử tăng ++, toán tử giảm --, toán tử dấu âm – và toán tử phủ định logic !. Hay dùng nhất là toán tử tăng giảm. Toán tử tăng và giảm có thể dùng ở trước hoặc sau toán hạng.
- Bài tập 4: Chồng toán tử ++ cho lớp airtime để tăng đối tượng airtime lên 1 phút, toán tử ++ có thể đứng trước và sau đối tượng.

Chương 14. Chồng toán tử

IV. Chuyển đổi giữa các đối tượng và kiểu dữ liệu cơ bản

- Việc chuyển đổi giữa các kiểu cơ bản được thực hiện tự động bởi vì các hàm chuyển đổi giữa các kiểu cơ bản đã có sẵn.
- Khi ta tạo ra một lớp và muốn chuyển đổi giữa các đối tượng lớp và các kiểu dữ liệu cơ bản thì chúng ta phải viết hàm chuyển đổi.
- Việc chuyển đổi từ các kiểu dữ liệu cơ bản sang các đối tượng được thực hiện bằng hàm tạo một đối số.
- Việc chuyển đổi từ các đối tượng lớp sang các kiểu cơ bản được thực hiện bằng hàm chồng toán tử ép kiểu.

IV. Chuyển đổi giữa các đối tượng và kiểu dữ liệu cơ bản

- Hàm chồng toán tử ép kiểu không có kiểu trả về, tên hàm bắt đầu bằng từ khoá operator sau đó là dấu cách rồi đến tên kiểu. Ví dụ: hàm chuyển đổi các đối tượng lớp sang kiểu long có dạng như sau:

```
operator long()
 {
   //Thuc hien chuyen doi o day
   return longvar;
 }
```

Mặc dù trong hàm có lệnh trả về nhưng hàm lại không có kiểu trả về, kiểu trả về ẩn trong tên hàm.

IV. Chuyển đổi giữa các đối tượng và kiểu dữ liệu cơ bản

- Cách gọi hàm chồng toán tử ép kiểu:

  Tên_kiểu(Đối tượng)

  Ví dụ: long(doituong); **ó** (long) doituong;
- Hàm chồng toán tử ép kiểu được gọi tự động khi ta gán một đối tượng cho một biến kiểu cơ bản hoặc khi khởi tạo một biến kiểu cơ bản.
- Bài tập 5: Xây dựng một lớp đối tượng chiều dài đo bằng đơn vị Anh: feet và inches. 1 foot = 12 inches, 1 meter = 3.280833 feet. Một chiều dài 6 feet 2 inches được viết là 6’-2”. Đặt tên lớp là English.

IV. Chuyển đổi giữa các đối tượng và kiểu dữ liệu cơ bản

- Bài tập về nhà: Chuyển đổi đối tượng xâu ký tự sang xâu ký tự thông thường.

Chương 14. Chồng toán tử

V. Chuyển đổi giữa các lớp

- Trong nhiều trường hợp, việc chuyển đổi giữa các lớp là không có ý nghĩa.
- Có 2 cách để chuyển đổi từ một lớp này sang một lớp khác:
  - Dùng hàm tạo một đối số
  - Dùng hàm chồng toán tử ép kiểu
- Ví dụ: Viết 2 lớp alpha và beta cùng các hàm cần thiết để chuyển từ alpha sang beta và từ beta sang alpha.

Chương 14. Chồng toán tử

I. Tại sao phải chồng toán tử?

II. Chồng các toán tử hai ngôi

III. Chồng các toán tử một ngôi

IV. Chuyển đổi giữa các đối tượng và kiểu dữ liệu cơ bản

V. Chuyển đổi giữa các lớp

VI. Chồng toán tử gán = và toán tử [ ]

Chương 14. Chồng toán tử

VI. Chồng toán tử gán = và toán tử [ ]

VI.1. Chồng toán tử gán đơn giản =

VI.2. Chồng toán tử chỉ số []

VI.1. Chồng toán tử gán đơn giản =

- Chúng ta có thể sử dụng toán tử gán để gán các đối tượng cho nhau mà không phải làm gì cả. Tuy nhiên, khi đối tượng sử dụng con trỏ hay làm những việc như đếm, đánh số thứ tự cho chính nó,… thì ta phải viết hàm chồng toán tử gán.
- Hàm chồng toán tử gán và hàm tạo sao chép đều thực hiện sao chép dữ liệu từ đối tượng này sang đối tượng khác. Chỉ khác là hàm tạo sao chép tạo ra một đối tượng mới rồi sao chép dữ liệu của một đối tượng khác vào đối tượng mới này, còn hàm chồng toán tử gán chỉ sao chép dữ liệu tới một đối tượng đã có.

VI.1. Chồng toán tử gán đơn giản

- Ví dụ: Tạo lớp omega gồm 2 mục dữ liệu: biến xâu name chứa xâu ký tự phản ánh tên đối tượng, biến nguyên snumber chứa số seri (thứ tự) của đối tượng. Chồng toán tử gán sao cho khi gán hai đối tượng thì nội dung biến xâu của đối tượng thay đổi còn số seri thì không thay đổi.

Chương 14. Chồng toán tử

VI. Chồng toán tử gán = và toán tử [ ]

VI.1. Chồng toán tử gán đơn giản =

VI.2. Chồng toán tử chỉ số []

VI.2. Chồng toán tử chỉ số []

- Toán tử chỉ số thường được dùng để truy nhập các phần tử của mảng. Chồng toán tử chỉ số để có thể sử dụng ký hiệu [ ] truy nhập các phần tử của một đối tượng mảng.
- Bài tập 6: Tạo một lớp mảng có sử dụng toán tử [ ] để nhập vào và đưa ra các phần tử của mảng.

Lưu ý

- Khi các hàm chồng toán tử cần trả về chính đối tượng gọi hàm thì ta nên dùng:
  - return *this
  - Khai báo kiểu trả về là tham chiếu
- this là con trỏ có sẵn, chứa địa chỉ của đối tượng gọi hàm thành viên, do đó *this là đối tượng gọi hàm.
- Nếu khai báo kiểu trả về là tham chiếu thì khi trả về đối tượng sẽ không tạo ra đối tượng trung gian.

VII. Hàm bạn

1. Giới thiệu về hàm bạn
2. Những thuận lợi khi dùng hàm bạn
3. Hàm bạn phá vỡ nguyên tắc bao gói thông tin

VII.1. Giới thiệu về hàm bạn

- Hàm bạn (**friend function**) là một hàm thông thường, không phải là thành viên của một lớp nhưng có thể truy nhập được tới các thành viên **private** và **protected** của lớp đó.
- Để cho một hàm thông thường là hàm bạn của một lớp, trong mô tả lớp ta viết khai báo hàm với từ khóa friend đứng trước.

  Ví dụ: friend void show();
- *Chú ý: Trong mô tả lớp chỉ chứa khai báo hàm bạn, không chứa định nghĩa hàm bạn.*

VII.1. Giới thiệu về hàm bạn

- Khai báo hàm bạn có thể bất kỳ phần nào trong mô tả lớp. Tuy nhiên nên để ở phần public vì nó là phần giao diện của lớp, nghĩa là bất kỳ người sử dụng lớp nào cũng có thể gọi hàm bạn.
- Theo nguyên tắc bao bọc và cất giấu dữ liệu trong LTHĐT, các hàm không phải là thành viên của lớp không thể truy nhập tới dữ liệu **private** và **protected** của một đối tượng. Tuy nhiên, trong một số trường hợp nguyên tắc này rất bất tiện. Các hàm bạn là một cách giải toả sự bất tiện này.
- Một hàm có thể khai báo là bạn của nhiều lớp.

VII.2. Những thuận lợi khi dùng hàm bạn

- Cho phép khi gọi hàm chồng toán tử thì bên trái toán tử không phải là đối tượng cũng được.

  Ví dụ:

VII.2. Những thuận lợi khi dùng hàm bạn

- Hàm bạn cho phép dùng ký hiệu hàm: Đôi khi một hàm bạn cho một cú pháp gọi hàm rõ ràng hơn hàm thành viên. Ví dụ, giả sử chúng ta muốn một hàm tính bình phương một đối tượng obj, khi đó cách viết sqr(obj) rõ ràng hơn cách viết obj.sqr()

VII.2. Những thuận lợi khi dùng hàm bạn

- Hàm bạn như chiếc cầu nối giữa các lớp: Giả sử ta có một hàm tính toán trên các đối tượng của hai lớp khác nhau. Có thể hàm này có đối số là các đối tượng của hai lớp đó và tính toán trên dữ liệu private của chúng. Làm thế nào để có thể dùng trực tiếp dữ liệu private của hai lớp nếu chúng không có liên quan gì với nhau? Hàm bạn của hai lớp sẽ làm được điều này.
- Ví dụ:

VII.3. Hàm bạn phá vỡ nguyên tắc bao gói thông tin

- Khi đưa vào hàm bạn, một mặt nó thêm vào sự linh hoạt cho ngôn ngữ, mặt khác nó không còn giữ nguyên tắc là chỉ có các thành viên mới có thể truy nhập dữ liệu **private** của lớp.
- Một hàm thông thường muốn là bạn của lớp phải được khai báo ở bên trong mô tả lớp đó. Thường thì người lập trình không truy nhập được mã nguồn của các lớp nên không thể chuyển một hàm thành một hàm bạn của lớp. Ở khía cạnh này tính toàn vẹn của lớp vẫn còn được giữ.

VII.3. Hàm bạn phá vỡ nguyên tắc bao gói thông tin

- Mặc dù vậy, hạm bạn vẫn gây ra sự lộn xộn trong tư tưởng LTHĐT.
- Tóm lại, luôn sử dụng hàm thành viên trừ khi có một lý do bắt buộc phải sử dụng hàm bạn.

Bài tập chương 14

Bài 1. Viết chương trình nhập vào 2 số phức. Tính tổng, hiệu và tích của 2 số phức đã nhập. Yêu cầu sử dụng các toán tử +, -, * cho số phức.

Bài 2. Viết chương trình sử dụng đối tượng ngăn xếp để đưa ra màn hình số nhị phân của một số nguyên dương n.

Bài tập chương 14

Bài 3. Viết chương trình sử dụng đối tượng ngăn xếp để tìm và đưa ra màn hình tất cả các số nguyên tố nhỏ hơn một số nguyên dương n nhập vào từ bàn phím theo thứ tự giảm dần.

Bài 4. Viết chương trình nhập vào danh sách n tên. Sắp xếp danh sách tên theo vần ABC. Sử dụng đối tượng xâu tự tạo để chứa tên.

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa

II. Hàm tạo, hàm huỷ và sự kế thừa

III. Điều khiển việc truy nhập lớp cơ sở

IV. Kế thừa nhiều mức

V. Hợp thành và kế thừa

VI. Kế thừa bội

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa

- I.1. Tầm quan trọng của kế thừa trong OOP
- I.2. Sự sử dụng lại
- I.3. Sự kế thừa và thiết kế hướng đối tượng
- I.4. Cú pháp kế thừa
- I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất
- I.6. Các hàm không được kế thừa
- I.7. Sự kế thừa và mối quan hệ loại

I.1. Tầm quan trọng của kế thừa trong OOP

- Sự kế thừa là khái niệm trung tâm thứ hai trong OOP.
- Sự kế thừa cho phép sử dụng lại, có nghĩa là đưa một lớp đã có vào sử dụng trong một tình huống lập trình mới. Nhờ việc sử dụng lại mà ta có thể giảm được thời gian và công sức khi viết một chương trình.
- Sự kế thừa còn đóng vai trò quan trọng trong việc thiết kế hướng đối tượng. Nó giúp ta giải quyết được những chương trình phức tạp.

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa

- I.1. Tầm quan trọng của kế thừa trong OOP
- I.2. Sự sử dụng lại
- I.3. Sự kế thừa và thiết kế hướng đối tượng
- I.4. Cú pháp kế thừa
- I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất
- I.6. Các hàm không được kế thừa
- I.7. Sự kế thừa và mối quan hệ loại

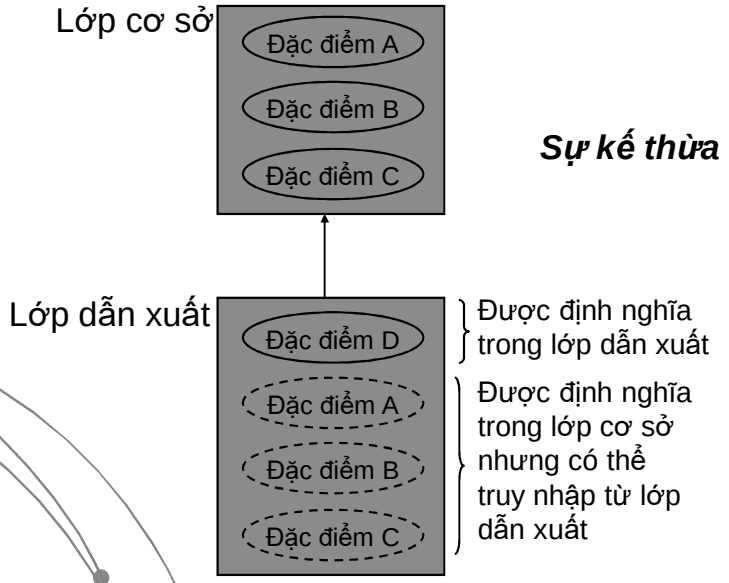
I.2. Sự sử dụng lại

- Những người lập trình đã tìm nhiều cách để tránh viết lại mã đã có:
  - Copy mã từ một chương trình đã có sang một chương trình mới rồi sữa để nó có thể chạy được trong chương trình mới này. Công việc này thường gây ra rất nhiều lỗi và mất nhiều thời gian để sửa lỗi.
  - Tạo các hàm để trong các thư viện hàm để khi sử dụng không cần thay đổi. Đây là giải pháp tốt nhưng khi chuyển sang môi trường lập trình mới các hàm này vẫn phải thay đổi thì mới dùng được.

I.2. Sự sử dụng lại (tiếp)

- Giải pháp tốt nhất đã xuất hiện trong OOP, đó là sử dụng thư viện lớp. Bởi vì một lớp mô phỏng được các thực thể thế giới thực và để sử dụng trong môi trường mới nó cần ít thay đổi hơn các hàm. Quan trọng hơn cả là OOP cho phép thay đổi một lớp mà không cần thay đổi mã của nó: sử dụng kế thừa để rút ra một lớp từ một lớp đã có. Lớp đã có (lớp cơ sở) không bị thay đổi, còn lớp mới (lớp rút ra từ lớp đã có, lớp dẫn xuất) có thể sử dụng tất cả đặc điểm của lớp đã có và thêm vào những đặc điểm của riêng nó.

I.2. Sự sử dụng lại (tiếp)



Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa

I.1. Tầm quan trọng của kế thừa trong OOP

I.2. Sự sử dụng lại

I.3. Sự kế thừa và thiết kế hướng đối tượng

I.4. Cú pháp kế thừa

I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất

I.6. Các hàm “không được kế thừa”

I.7. Sự kế thừa và mối quan hệ loại

I.3. Sự kế thừa và thiết kế hướng đối tượng

- Sự kế thừa giúp cho việc thiết kế chương trình được linh động hơn, phản ánh mối quan hệ trong thế giới thực chính xác hơn.
- Trong lập trình hướng đối tượng có 2 mối quan hệ giữa các thành phần của chương trình:
  - Mối quan hệ "có": Một công nhân có tên, mã số, lương,… Một chiếc xe đạp có khung, 2 bánh, tay lái,… Mối quan hệ "có" trong thế giới thực có thể mô phỏng trong chương trình hướng đối tượng bằng một lớp, trong lớp có các thành viên của lớp. Lớp công nhân có chứa một biến lưu trữ tên, một biến lưu trữ mã số, một biến lưu trữ lương; lớp xe đạp có một đối tượng khung, hai đối tượng bánh, một đối tượng tay lái. Mối quan hệ có được các ngôn ngữ thủ tục (C, Pascal) mô phỏng bằng cấu trúc (struct), bản ghi (record). Mối quan hệ "có" được gọi là sự hợp thành.

I.3. Sự kế thừa và thiết kế hướng đối tượng (tiếp)

- Mối quan hệ "loại": Xe đạp đua, xe đạp địa hình, xe đạp thiếu nhi đều là các loại xe đạp. Tất cả các loại xe đạp đều có đặc điểm chung: hai bánh, một khung. Một xe đạp đua, ngoài các đặc điểm chung này còn có đặc điểm là lốp nhỏ và nhẹ. Một xe đạp địa hình, ngoài các đặc điểm chung của một xe đạp còn có lốp to, dày và phanh tốt. Mối quan hệ "loại" này được mô phỏng trong chương trình hướng đối tượng bằng sự kế thừa. Ở đây, những gì là chung, khái quát được mô tả bằng một lớp cơ sở, những gì cụ thể, rõ ràng được mô tả bằng một lớp dẫn xuất. Sự kế thừa là một công cụ rất hữu ích trong thiết kế chương trình hướng đối tượng.

Chương 15. Sự kế thừa

I.4. Cú pháp kế thừa

```
class TenLopCoSo
 {
   //Cac thanh vien cua lop co so
 };

class TenLopDanXuat : public TenLopCoSo
 {
   //Cac thanh vien cua lop dan xuat
 };
```

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa

I.1. Tầm quan trọng của kế thừa trong OOP

I.2. Sự sử dụng lại

I.3. Sự kế thừa và thiết kế hướng đối tượng

I.4. Cú pháp kế thừa

I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất

I.6. Các hàm không được kế thừa

I.7. Sự kế thừa và mối quan hệ loại

I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất

- Những gì được kế thừa? Tất cả dữ liệu và hàm thành viên của lớp cơ sở, tức là một đối tượng lớp dẫn xuất kế thừa tất cả dữ liệu và hàm thành viên của lớp cơ sở.
- Truy nhập dữ liệu lớp cơ sở từ lớp dẫn xuất: Mặc dù các đối tượng lớp dẫn xuất chứa các thành viên dữ liệu được định nghĩa trong lớp cơ sở nhưng trong lớp dẫn xuất ta không thể truy nhập các thành viên dữ liệu **private** của lớp cơ sở (trừ các thành viên **public** và **protected**).

I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất

- Gọi hàm thành viên lớp cơ sở từ lớp dẫn xuất:
  - Nếu các hàm thành viên lớp cơ sở và dẫn xuất có tên khác nhau thì khi gọi hàm ta chỉ dùng tên hàm.
  - Nếu các hàm thành viên lớp cơ sở và dẫn xuất có tên giống nhau thì khi gọi hàm ta gắn thêm tên lớp cơ sở trước tên hàm bằng toán tử ::

    Tên_lớp_cơ_sở::Tên_thành_viên

I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất

- Trong OOP, các hàm thành viên lớp cơ sở và lớp dẫn xuất làm các công việc tương tự nhau luôn được chồng hàm (trùng) để cho rõ ràng và dễ nhớ.
- Nếu sử dụng chồng hàm thì khi gọi hàm thành viên lớp cơ sở phải gắn với tên lớp, nếu không trình biên dịch sẽ hiểu là gọi hàm thành viên lớp dẫn xuất.

Ví dụ

- Viết chương trình tính thể tích và diện tích bề mặt của hình trụ có bán kính r và chiều cao h. Biết rằng hình trụ là một loại hình tròn có bán kính r được kéo dài với chiều cao h.

Bài tập về nhà

- Viết chương trình quản lý nhân sự của một trường đại học. Nhân sự chia làm 3 loại: Giáo viên, Cán bộ quản lý và nhân viên phục vụ. Thông tin lưu trữ về giáo viên gồm có: tên, mã số, học hàm, học vị. Thông tin lưu trữ về Cán bộ quản lý gồm có tên, mã số và chức vụ. Thông tin lưu trữ về Nhân viên phục vụ gồm có tên và mã số.

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa

- I.1. Tầm quan trọng của kế thừa trong OOP
- I.2. Sự sử dụng lại
- I.3. Sự kế thừa và thiết kế hướng đối tượng
- I.4. Cú pháp kế thừa
- I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất
- I.6. Các hàm “không được kế thừa”
- I.7. Sự kế thừa và mối quan hệ loại

I.7. Các hàm không được kế thừa

- Có một vài hàm đặc biệt không được tự động kế thừa. Đó là các hàm làm những công việc cho riêng lớp cơ sở và lớp dẫn xuất. Có 3 hàm như vậy: hàm chồng toán tử gán =, hàm tạo (constructor) và hàm hủy (destructor).
- Hàm tạo lớp cơ sở phải tạo dữ liệu lớp cơ sở, hàm tạo lớp dẫn xuất phải tạo dữ liệu lớp dẫn xuất. Bởi vì các hàm tạo lớp cơ sở và lớp dẫn xuất tạo dữ liệu khác nhau nên chúng không thể thay thế nhau được. Do đó các hàm tạo không được tự động kế thừa.

I.7. Các hàm không được kế thừa

- Toán tử gán = trong lớp dẫn xuất phải gán các giá trị cho dữ liệu của lớp dẫn xuất, toán tử = trong lớp cơ sở phải gán các giá trị cho dữ liệu của lớp cơ sở. Đây là các công việc khác nhau, bởi vậy toán tử này cũng không được tự động kế thừa.
- Hàm hủy lớp dẫn xuất hủy dữ liệu của lớp dẫn xuất. Nó không huỷ các đối tượng lớp cơ sở; nó phải gọi hàm hủy lớp cơ sở để làm việc này. Hơn nữa, các hàm hủy này làm các công việc khác nhau nên chúng không được tự động kế thừa.

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa

I.1. Tầm quan trọng của kế thừa trong OOP

I.2. Sự sử dụng lại

I.3. Sự kế thừa và thiết kế hướng đối tượng

I.4. Cú pháp kế thừa

I.5. Truy nhập thành viên lớp cơ sở từ lớp dẫn xuất

I.6. Các hàm không được kế thừa

I.7. Sự kế thừa và mối quan hệ loại

I.8. Sự kế thừa và mối quan hệ loại

- Sự kế thừa trong LTHĐT thể hiện được mối quan hệ "loại" trong thế giới thực. Các đối tượng lớp dẫn xuất là một loại đối tượng lớp cơ sở.
- Trong C++, ta có thể gán một đối tượng lớp dẫn xuất cho một đối tượng lớp cơ sở và có thể truyền đối tượng lớp dẫn xuất cho một hàm có đối số lớp cơ sở. Tuy nhiên ta không nên làm điều này vì đối tượng lớp dẫn xuất thường có kích thước lớn hơn đối tượng lớp cơ sở.

I.8. Sự kế thừa và mối quan hệ loại

```
class alpha                  //lớp cơ sở
 {
   public:
     void memfunc()          //hàm thành viên public
       {}
 };
class beta:public alpha      //lớp dẫn xuất
 { };
void main()
 {
   void anyfunc(alpha);      //khai báo, hàm có một đối số
   alpha aa;                 //đối tượng kiểu alpha
   beta  bb;                 //đối tượng kiểu beta
   aa=bb;                    //đối tượng beta được gán cho biến alpha
   anyfunc(bb);  //đối tượng beta được truyền như đối số alpha
 }
```

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa

II. Hàm tạo, hàm huỷ và sự kế thừa

III. Điều khiển việc truy nhập lớp cơ sở

IV. Kế thừa nhiều mức

V. Hợp thành và kế thừa

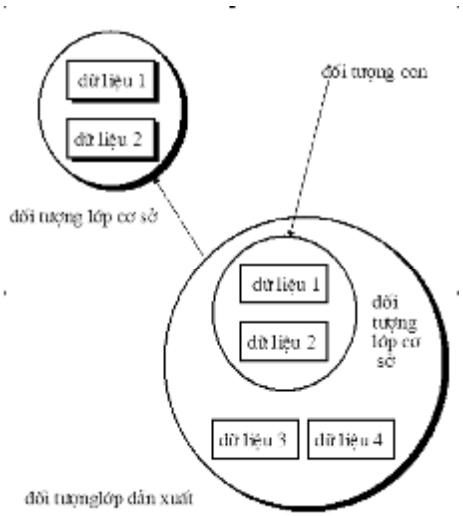
VI. Kế thừa bội

II. Hàm tạo, hàm huỷ và sự kế thừa

1. Hàm tạo, hàm hủy với đối tượng lớp dẫn xuất
2. Khi nào phải viết hàm tạo cho lớp dẫn xuất

II.1. Hàm tạo, hàm hủy với đối tượng lớp dẫn xuất

- Khi chúng ta định nghĩa một đối tượng lớp dẫn xuất, không chỉ hàm tạo của nó được thực hiện mà hàm tạo trong lớp cơ sở cũng được thực hiện. Trên thực tế, hàm tạo lớp cơ sở được thực hiện trước. Bởi vì đối tượng lớp cơ sở là đối tượng con - một phần - của đối tượng lớp dẫn xuất, và chúng ta cần tạo các bộ phận trước khi tạo toàn thể.



II.1. Hàm tạo, hàm hủy với đối tượng lớp dẫn xuất

- Như vậy, khi tạo đối tượng lớp dẫn xuất, các hàm tạo lớp cơ sở được gọi trước sau đó hàm tạo lớp dẫn xuất mới được gọi.
- Khi đối tượng bị hủy, hàm hủy được gọi theo thứ tự ngược lại: đối tượng lớp dẫn xuất được hủy trước sau đó đến đối tượng lớp cơ sở.
- Trong danh sách khởi tạo của hàm tạo lớp dẫn xuất có thể gọi hàm tạo lớp cơ sở. Ví dụ:

```
danxuat(int n):coso(n)
{   }
```

II. Hàm tạo, hàm huỷ và sự kế thừa

1. Hàm tạo, hàm hủy với đối tượng lớp dẫn xuất
2. Khi nào phải viết hàm tạo cho lớp dẫn xuất

II.2. Khi nào phải viết hàm tạo cho lớp dẫn xuất

- Chúng ta không thể tạo đối tượng bằng một hàm tạo mà không có trong lớp của đối tượng đó. Bởi vậy, ngay cả khi chúng ta có một hàm tạo *n* đối số trong lớp cơ sở thì chúng ta vẫn phải định nghĩa một hàm tạo >=*n* đối số trong lớp dẫn xuất.
- Bất kỳ khi nào chúng ta cần hàm tạo có đối số để tạo đối tượng lớp dẫn xuất thì ta phải viết hàm tạo này trong lớp dẫn xuất.

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa
II. Hàm tạo, hàm huỷ và sự kế thừa
III. Điều khiển việc truy nhập lớp cơ sở
IV. Kế thừa nhiều mức
V. Hợp thành và kế thừa
VI. Kế thừa bội

III. Điều khiển việc truy nhập lớp cơ sở

1. Các định danh truy nhập
2. Che giấu dữ liệu lớp cơ sở

III.1. Các định danh truy nhập

- Khi chưa có sự kế thừa, các hàm thành viên lớp có thể truy nhập tới tất cả những gì có trong lớp dù nó là public hay private, nhưng bên ngoài lớp đó chỉ có thể truy nhập tới các thành viên public.
- Khi sự kế thừa xuất hiện, khả năng truy nhập mở rộng hơn cho lớp dẫn xuất. Các hàm thành viên của lớp dẫn xuất có thể truy nhập các thành viên public và protected của lớp cơ sở nhưng vẫn không thể truy nhập các thành viên private. Bên ngoài lớp dẫn xuất vẫn chỉ có thể truy nhập các thành viên public của lớp dẫn xuất.

III.1. Các định danh truy nhập

Sự kế thừa và khả năng truy nhập

Định danh truy nhập	có thể truy nhập từ trong lớp	có thể truy nhập từ lớp dẫn xuất	có thể truy nhập từ bên ngoài lớp
public	có	có	có
protected	có	có	không
private	có	không	không

III.2. Che giấu dữ liệu lớp cơ sở

- Dữ liệu trong lớp cơ sở nên để ở private hay protected? Để dữ liệu ở protected có thuận lợi là không cần viết thêm các hàm lớp cơ sở để truy nhập dữ liệu từ lớp dẫn xuất. Tuy nhiên, đây không phải là cách tốt nhất.
- Nói chung, dữ liệu lớp nên để private (trừ một số trường hợp đặc biệt). Dữ liệu public có thể bị thay đổi bởi bất kỳ hàm nào ở bất kỳ đâu trong chương trình, điều này nên tránh. Dữ liệu protected có thể bị thay đổi bởi các hàm trong bất kỳ lớp dẫn xuất nào.
- Bất kỳ người nào rút ra một lớp từ một lớp khác đều có quyền truy nhập tới dữ liệu **protected** của lớp đó. Sẽ an toàn và tin cậy hơn nếu lớp dẫn xuất không thể truy nhập trực tiếp dữ liệu lớp cơ sở.

III.2. Che giấu dữ liệu lớp cơ sở

- Một giao diện lớp bao gồm các hàm dùng để truy nhập cái gì đó. Thiết kế các lớp cho chúng ta hai giao diện: một giao diện **public** để bên ngoài lớp sử dụng và một giao diện **protected** để để các lớp dẫn xuất sử dụng. Không nên để giao diện nào truy nhập trực tiếp dữ liệu. Một thuận lợi của việc để dữ liệu lớp cơ sở ở **private** là chúng ta có thể thay đổi nó mà không làm ảnh hưởng tới các lớp dẫn xuất.
- Để có truy nhập dữ liệu lớp cơ sở, chúng ta viết thêm các hàm thành viên cho lớp cơ sở. Các hàm này nên để ở **protected**, bởi vì chúng là phần giao diện **protected** được dùng bởi lớp dẫn xuất và các lớp không phải là lớp dẫn xuất không thể truy nhập được.

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa
II. Hàm tạo, hàm huỷ và sự kế thừa
III. Điều khiển việc truy nhập lớp cơ sở
IV. Kế thừa nhiều mức
V. Hợp thành và kế thừa
VI. Kế thừa bội

IV. Kế thừa nhiều mức

- Sự kế thừa nhiều mức có nghĩa là không chỉ lớp **beta** có thể rút ra từ lớp **alpha**, lớp **gama** cũng có thể rút ra từ **beta**, lớp **delta** có thể rút ra từ **gama** v.v...

```
class alpha
  { };
class beta:public alpha
  { };
class gama:public beta
  { };
class delta:public gama
  { };
```


IV. Kế thừa nhiều mức

- Một lớp có thể truy nhập tới tất cả các lớp tổ tiên của nó, cụ thể là các hàm thành viên **delta** có thể truy nhập tới dữ liệu **public** hoặc **protected** trong **gama**, **beta**, và **alpha**. Tất nhiên là chúng không thể truy nhập tới các thành viên **private** của bất kỳ lớp nào trừ của chính nó.

Chương 15. Sự kế thừa

I. Giới thiệu về kế thừa
II. Hàm tạo, hàm huỷ và sự kế thừa
III. Điều khiển việc truy nhập lớp cơ sở
IV. Kế thừa nhiều mức
V. Hợp thành và kế thừa
VI. Kế thừa bội

V. Hợp thành và kế thừa

- Sự hợp thành là đặt một đối tượng bên trong một đối tượng khác hay, đứng về phía lập trình, là định nghĩa một đối tượng của một lớp ở bên trong mô tả của một lớp khác.

```
class alpha
  { };
class beta
{
   private:
     alpha  obj;
};
```


V. Hợp thành và kế thừa

- Khi nào sử dụng sự kế thừa tốt hơn sự hợp thành?
  - Khi cần mối quan hệ “loại” giữa các lớp.
  - Khi cần một mảng đối tượng lớp cơ sở để chứa các đối tượng của các lớp dẫn xuất.
  - Khi cần một mảng con trỏ lớp cơ sở để chứa địa chỉ các đối tượng của các lớp dẫn xuất.

Chương 15. Sự kế thừa

VI. Kế thừa bội

- Kế thừa bội có nghĩa là một lớp dẫn xuất kế thừa từ hai hay nhiều lớp cơ sở khác nhau.

```
class Base1
  { };
class Base2
  { };
class Derv:public Base1,public Base2
  { };
```

Chương 16. Đa hình động, Hàm ảo

I. Hàm ảo và sự đa hình

II. Ứng dụng của sự đa hình

III. Lớp trừu tượng, hàm tạo và hàm hủy ảo

I. Hàm ảo và sự đa hình

1. Giới thiệu về hàm ảo và sự đa hình
2. Gọi hàm thành viên qua con trỏ lớp cơ sở
3. Sự liên kết động

I.1. Giới thiệu về hàm ảo và sự đa hình

- Dạng đa hình thứ hai trong LTHĐT liên quan tới sự kế thừa, hàm ảo và con trỏ. Ở đây sự đa hình thái thể hiện ở chỗ: Lời gọi tới một hàm thành viên sẽ làm cho các hàm thành viên khác nhau được thực hiện tuỳ thuộc vào kiểu đối tượng gọi hàm đó. Sự đa hình này còn được gọi là sự liên kết động.
- Hàm ảo là hàm thành viên của lớp, giống như các hàm thành viên thông thường, chỉ khác là được khai báo với từ khóa virtual đặt trước.

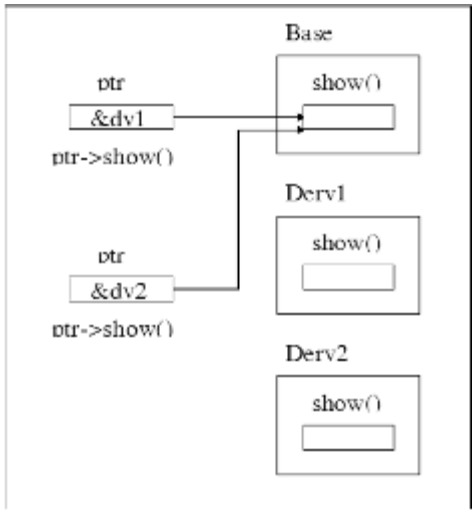
```
virtual void nhap();
```


I.2. Gọi hàm thành viên qua con trỏ lớp cơ sở

- Con trỏ lớp cơ sở có thể chứa địa chỉ của đối tượng các lớp dẫn xuất. Bởi vì đối tượng lớp dẫn xuất là một loại đối tượng lớp cơ sở nên các con trỏ trỏ tới đối tượng của một lớp dẫn xuất có kiểu phù hợp với các con trỏ trỏ tới đối tượng của lớp cơ sở.
- Khi một lớp cơ sở và các lớp dẫn xuất của nó có các hàm thành viên trùng nhau, nếu các hàm này được gọi qua con trỏ lớp cơ sở thì hàm được thực hiện luôn là hàm thành viên lớp cơ sở.

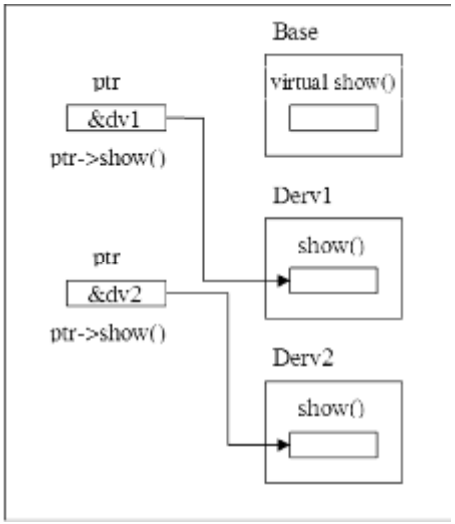
I.2. Gọi hàm thành viên qua con trỏ lớp cơ sở

- Sở dĩ các hàm thành viên lớp cơ sở luôn được thực hiện vì trình biên dịch bỏ qua nội dung của con trỏ và chọn hàm thành viên phù hợp với kiểu con trỏ là lớp cơ sở.



I.2. Gọi hàm thành viên qua con trỏ lớp cơ sở

- Để gọi được hàm thành viên của lớp dẫn xuất qua con trỏ lớp cơ sở ta cho các hàm thành viên của lớp cơ sở là hàm ảo.
- Khi dùng hàm ảo, trình biên dịch lựa chọn hàm để thực hiện dựa trên nội dung của con trỏ, chứ không phải tên kiểu của con trỏ. Đây là sự đa hình thái, vì một lời gọi hàm mà có thể thực hiện các hàm khác nhau, tuỳ thuộc vào nội dung của con trỏ.



I.3. Sự liên kết động

- Nếu trong lớp cơ sở có hàm ảo trùng tên với một hàm thành viên lớp dẫn xuất thì khi gọi hàm thành viên lớp dẫn xuất này qua con trỏ lớp cơ sở trình biên dịch sẽ không biết gọi hàm nào. Bởi vậy trình biên dịch phải sắp xếp để lựa chọn hàm thực hiện tại thời điểm chạy chương trình.
- Việc lựa chọn một hàm tại thời điểm chạy chương trình được gọi là sự liên kết động (**dynamic binding**). Còn việc lựa chọn hàm thực hiện theo cách thông thường, tại thời điểm biên dịch, được gọi là sự liên kết tĩnh (**static binding**).

I.3. Sự liên kết động

- Sự liên kết động cần nhiều thời gian và bộ nhớ hơn sự liên kết tĩnh: Lời gọi hàm lâu hơn, đối tượng lớp dẫn xuất lớn hơn.
- Tóm lại, để cài đặt sự liên kết động cần có kế thừa, hàm ảo, con trỏ và sự trùng hàm thành viên.

II. Ứng dụng của sự đa hình

1. Mảng con trỏ trỏ tới các đối tượng của các lớp khác nhau
2. Phân lập các phần chương trình

Ví dụ: Tính diện tích các hình: Hình tam giác biết 3 cạnh a,b,c; hình chữ nhật biết 2 cạnh a,b; hình tròn biết bán kính r. Nhập vào một số hình. Đưa ra diện tích các hình đã nhập. Yêu cầu cài đặt đa hình động.

II.1. Mảng con trỏ trỏ tới các đối tượng của các lớp khác nhau

- Một ứng dụng của sự đa hình là sử dụng mảng con trỏ lớp cơ sở để chứa địa chỉ của các đối tượng lớp dẫn xuất khác nhau.
- Ví dụ: Viết chương trình quản lý giảng viên và sinh viên. Thông tin về giảng viên có tên và số bài báo đã đăng, thông tin về sinh viên có tên và điểm TBC. Nhập vào một số giảng viên và sinh viên. Đưa ra màn hình thông tin về các giảng viên và sinh viên đã nhập, có kèm theo đánh giá: giáo viên giỏi nếu có số bài báo >=20, sinh viên giỏi nếu có điểm TBC>=9.0.

II.2. Phân lập các phần chương trình

- Sự đa hình thái cũng có thể được sử dụng để giúp cho việc phân lập, hay gỡ bỏ sự phụ thuộc của một phần chương trình vào phần chương trình khác.
- Các chương trình OOP được chia thành hai phần được viết bởi những người lập trình khác nhau tại các thời điểm khác nhau. Đó là phần tạo lớp và phần sử dụng các lớp.

II.2. Phân lập các phần chương trình

- Việc lập trình sẽ đơn giản hơn nếu chương trình của người sử dụng lớp chỉ phải làm việc với một lớp thay vi nhiều lớp khác nhau. Điều này thực hiện được bằng sự đa hình, đó là dùng các tham chiếu hoặc các con trỏ trỏ tới các đối tượng để truyền và trả về từ một hàm.
- Ví dụ 1: Sử dụng đối số là tham chiếu

II.2. Phân lập các phần chương trình

- Ví dụ 2: Sử dụng đối số là con trỏ
- Ví dụ 3: Viết lại chương trình quản lý giảng viên và sinh viên trong đó có sử dụng hàm để truyền và trả về các đối tượng khác nhau.

III. Lớp trừu tượng, hàm tạo và hàm hủy ảo

1. Lớp trừu tượng (abstract class)
2. Hàm tạo ảo và hàm hủy ảo

III.1. Lớp trừu tượng

- Lớp trừu tượng là lớp mà không có đối tượng nào được tạo ra từ nó, nó chỉ đóng vai trò là lớp cơ sở cho các lớp dẫn xuất. Các lớp trừu tượng được cài đặt trong C++ bằng các hàm ảo tinh khiết (**pure virtual function**).
- Hàm ảo tinh khiết là một hàm ảo mà khi khai báo hàm có thêm ký hiệu =0 vào sau khai báo hàm. Thân của hàm ảo có thể có hoặc không có.

  Ví dụ: virtual void show()=0;

  hoặc virtual void show()=0

  { //Các lệnh của hàm }

III.1. Lớp trừu tượng

- Dấu bằng ở đây không phải là toán tử gán, giá trị 0 không được gán cho cái gì. Cú pháp =0 chỉ đơn giản là cách chỉ cho trình biên dịch biết rằng một hàm là tinh khiết.
- Để trình liên kết ngăn chặn việc tạo một đối tượng từ lớp trừu tượng, chúng ta phải định nghĩa ít nhất một hàm ảo tinh khiết trong lớp trừu tượng đó.

III.2. Hàm tạo và hàm hủy ảo

- Câu hỏi đặt ra là “các hàm tạo có bao giờ là ảo không?”. Không, không bao giờ. Các hàm ảo không thể tồn tại cho đến khi hàm tạo đã hoàn thành nhiệm vụ của nó, bởi vậy các hàm tạo không thể là ảo.
- Ngoài ra, khi tạo một đối tượng trình biên dịch cần biết loại đối tượng tạo ra. Do đó không có các hàm tạo ảo.
- Trái lại, các hàm huỷ có thể và thường nên là ảo.

III.2. Hàm tạo và hàm hủy ảo

- Khi một lớp cơ sở có hàm ảo thì hàm hủy cũng nên để ảo. Nếu không để là ảo thì hàm hủy lớp dẫn xuất sẽ không được thực hiện khi hủy đối tượng lớp dẫn xuất thông qua con trỏ lớp cơ sở.
- Ví dụ:

III.2. Hàm tạo và hàm hủy ảo

- Khi nào một lớp cơ sở cần hàm hủy ảo? Khi thỏa mãn 3 điều kiện sau:
  - Cần tạo các lớp dẫn xuất từ lớp cơ sở
  - Các đối tượng lớp dẫn xuất được hủy qua con trỏ lớp cơ sở
  - Các hàm hủy trong lớp cơ sở và dẫn xuất thực hiện các công việc quan trọng chẳng hạn như giải phóng bộ nhớ.
- Tóm lại, cứ khi nào trong lớp cơ sở có hàm ảo thì nên để hàm hủy của nó là ảo.

Bài tập

Bài 1. Viết chương trình tính diện tích của các hình: hình chữ nhật có 2 cạnh, hình tròn có bán kính. Yêu cầu trong chương trình có cài đặt đa hình động.