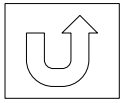
Chương 04. Lớp và đối tượng của lớp

I. Mô tả lớp (khai báo lớp)

II. Tạo và tương tác với các đối tượng

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

I. Mô tả lớp (khai báo lớp)

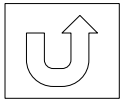


1. Cú pháp mô tả lớp (định nghĩa lớp)

```
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.

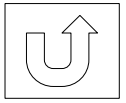


2. Từ khóa public, private, protected

- 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.

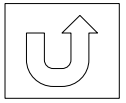


3. Khai báo dữ liệu của lớp

- 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)

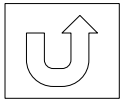
✧ Cú pháp định nghĩa hàm thành viên bên ngoài mô tả lớp như sau:

```
class Ten_lop
{
   private:

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

Ví dụ về lớp

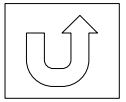
Lớp đối tượng thời gian lưu trữ giờ và phút.

```
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
2. Gửi thông báo tới các đối tượng
3. Mảng đối tượng
4. Con trỏ trỏ tới đối tượng
5. Lệnh gán đối tượng
6. Truy nhập dữ liệu của các đối tượng cùng lớp

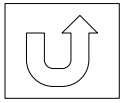


1. Tạo các đối tượng của một lớp

- 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;



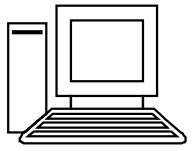
2. Gửi thông điệp tới các đối tượng

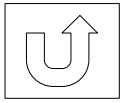
- Khi một đối tượng được tạo thì ta có thể tương tác với nó bằng cách sử dụng các hàm hành viên. Việc gọi hàm thành viên của một đối tượng gọi là gửi thông điệp tới đối tượng đó.
- Cú pháp gửi thông báo tới một đối tượng:

  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)

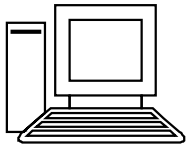
- Để gửi thống báo tới một đối tượng cụ thể trong mảng đối tượng ta phải dùng thêm ký hiệu của mảng để xác định đối tượng muốn gửi thống báo tới. Ví dụ:

  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;

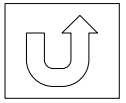
4. Con trỏ trỏ tới đối tượng (tiếp)

- 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();

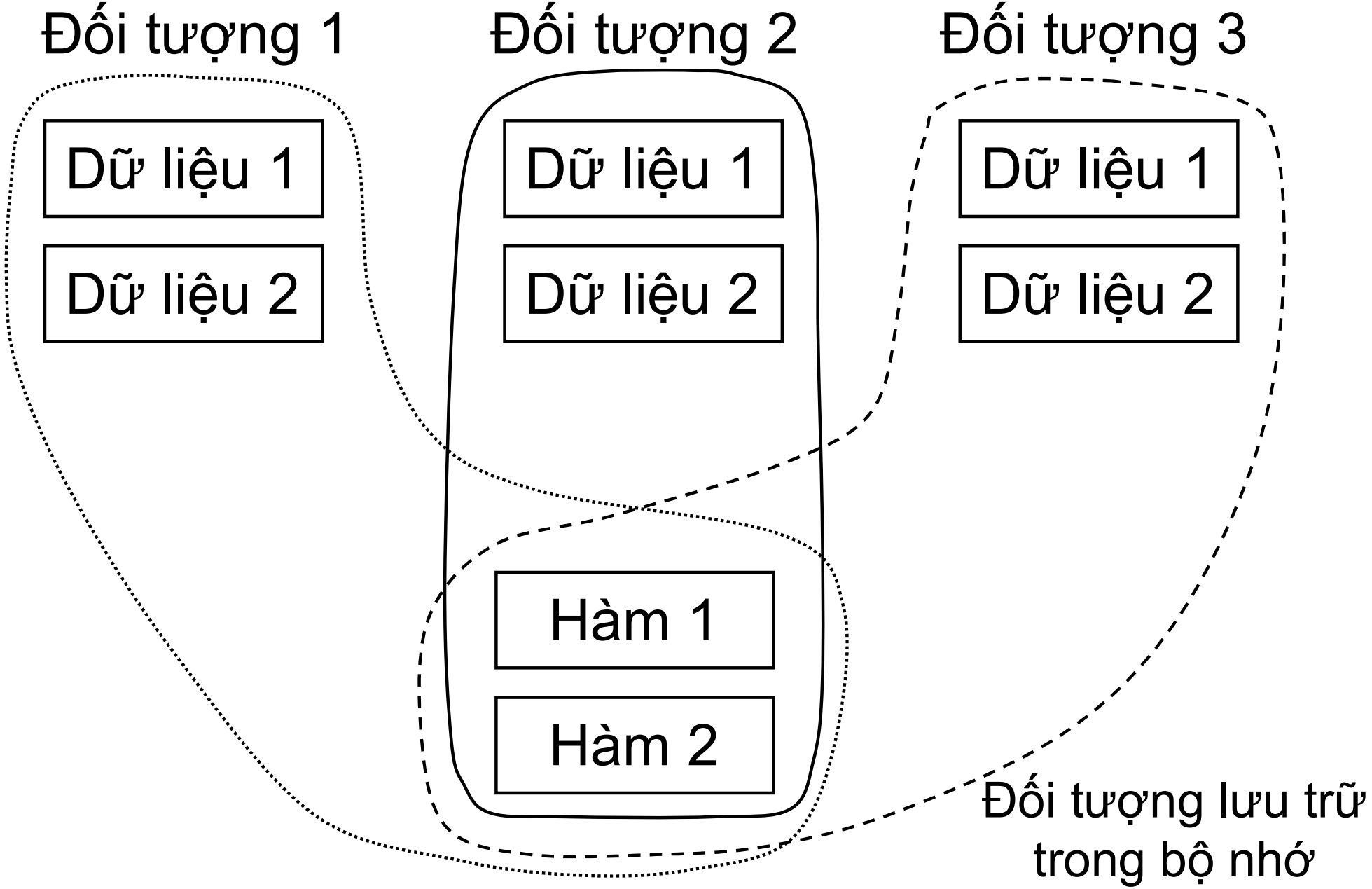
  Cách thứ hai gọn hơn cách nhất. Với con trỏ trỏ tới đối tượng người ta hay dùng cách thứ hai.



5. Lệnh gán đối tượng

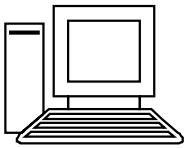
- Với các biến kiểu cơ bản ta có thể gán giá trị của một biến cho một biến cùng kiểu. Vậy có thể gán giá trị của một đối tượng cho một đối tượng được không? Câu trả lời là có, bởi vì C++ coi các đối tượng như các biến.
- Nhưng đối tượng bao gồm cả dữ liệu và các hàm thành viên, khi gán một đối tượng 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.



vdp2c23.cpp

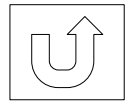
```
airtime t1, t2;
t1.set();
t1.display();
t2 = t1;
t2.display();
```

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)

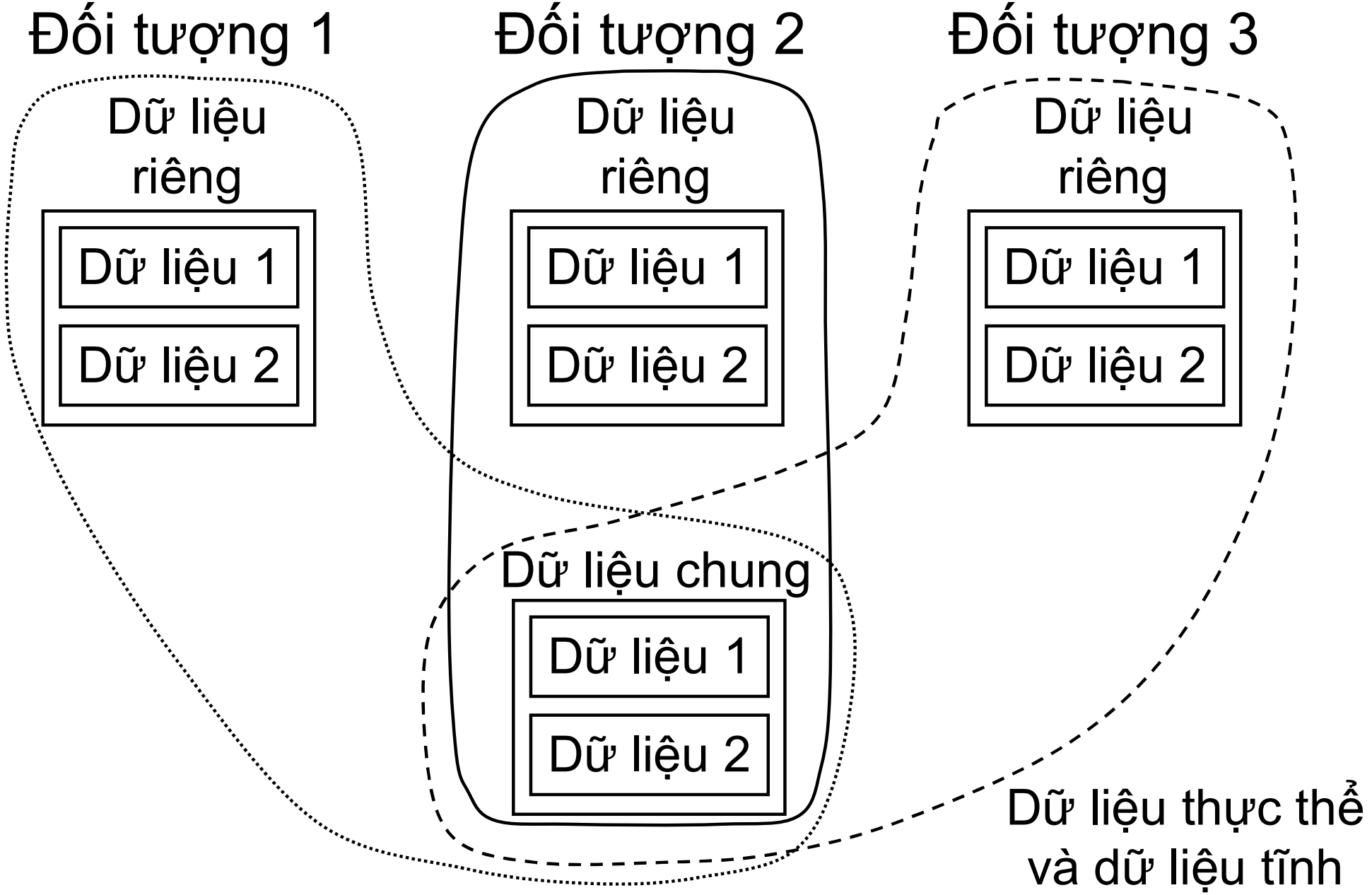
1. Dữ liệu thành viên tĩnh
2. Hàm thành viên tĩnh



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)



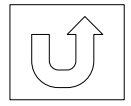
1. Dữ liệu thành viên tĩnh (tiếp)

✧ 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).

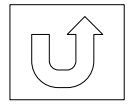


2. Hàm thành viên tĩnh

- 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 4

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 4

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 chung và hàm chung, sử dụng đối tượng động.