CHƯƠNG 16

16.1 Cơ bản về lớp(class)

Một lớp(class) trong C++ sẽ có những đặc điểm sau:
• Là kiểu dữ liệu do người lập trình viên định nghĩa.
• Một lớp sẽ bao gồm các thành viên dữ liệu và các hàm thành viên.
• Các hàm thành viên có thể xác định ý nghĩa của việc khởi tạo, sao chép, di chuyển và huỷ.
• Để các thành viên truy cập vào đối tượng ta sử dụng dấu chấm và mũi tên −> đối với con trỏ.
• Các toán tử, chẳng hạn như + , ! , và [] , có thể được định nghĩa cho một lớp.
• Dữ liệu và hàm bên trong một lớp được gọi là các thành viên của lớp đó.
• Một cấu trúc là một lớp mà các thành viên mặc định là công khai .

16.2.1 Chức năng thành viên

Một tính năng quan trọng của lớp là các hàm thành viên. Mỗi kiểu dữ liệu có thể có các hàm tích hợp riêng của nó (được gọi là các phương thức) có quyền truy cập vào tất cả các thành viên (public và private) của kiểu dữ liệu.

16.2.2 Sao chép mặc định

Các đối tượng một lớp có thể khởi tạo bằng cách sao chép đối tượng của lớp đó.
Ví dụ:

```
Date d1 = my_birthday;   hoặc Date d2 {my_bir thday};
```

Các đối tượng một lớp có thể được sao chép bằng phép gán.
Ví dụ:

```
void f (Date & d){
        d = my_birthday;        }
```

16.2.3 Kiểm soát truy cập

Ví dụ:

```
class Date {
private:
        int d, m, y;
public:
        void init(int  dd, int mm, int yy);
        void add_year(int n);
        void add_month(int n);
        void add_day(int n);                };
```

Chức năng của các thành viên được định nghĩa và sử dụng như sau:

```
void Date::add_year(int n){
        y += n;          }
```

Các thành viên bên ngoài, sẽ bị ngăn chặn truy cập vào các thành viên riêng tư. Ví dụ:

```
void timewarp (Date& d){
        d.y −= 200;     // error: Date::y is private       }
```

Việc đặt các dữ liệu trong biến riêng tư, hàm init () sẽ được khởi tạo bằng cách như sau:

```
Date dx;
dx.m = 3; // error : m is private
dx.init(25,3,2011); // OK
```

16.2.4 Lớp và cấu trúc

Khai báo lớp: class X{/*…*/};
Theo mặc định, các thành viên của một lớp là riêng tư và các thành viên của struct được công khai.

```
struct S {…};
class S {public: / * ... * /};
```

16.2.5 Hàm khởi tạo

Ví dụ:
```
class Date {
        int d, m, y;
public:
        Date(int dd, int mm, int yy); // constructor     };
```

Constructor là một loại hàm thành viên đặc biệt của class, được gọi tự động khi một đối tượng của class đó được khởi tạo. Các constructors thường được sử dụng để khởi tạo các biến thành viên của class theo các giá trị mặc định phù hợp hoặc do người dùng cung cấp, hoặc để thực hiện bất kỳ các bước thiết lập cần thiết nào cho class.

Ví dụ:
```
Date today = Date(23,6,1983);
Date today = Date {23,6,1983};
Date my_birthday; // Lỗi: chưa được khởi tạo
Date release1_0(10,12);  // Lỗi : đối số thứ ba bị thiếu
```

16.2.6 Explicit Constructors

Từ khóa explicit cho hàm khởi tạo sẽ ngăn trình biên dịch sử dụng hàm tạo đó để chuyển đổi ngầm định.

Ví dụ giả định: bạn có một lớp MyString(int size) với một hàm tạo xây dựng một chuỗi có kích thước đã cho. Bạn có một hàm print(const MyString&) và bạn gọi print(3) (khi bạn thực sự dự định gọi print("3")). Bạn mong đợi nó sẽ in "3", nhưng thay vào đó, nó sẽ in một chuỗi trống có độ dài 3.

Nếu một hàm tạo được khai báo rõ ràng và được định nghĩa bên ngoài lớp, thì không thể lặp lại hàm tạo đó:

```
class Date {
        int d, m, y;
public:
        explicit  Date(int dd);     };
Date::Date(int  dd) { /* ... */ } // OK
explicit  Date::Date(int  dd) { /* ... */ } // error
```

16.2.7 Bộ khởi tạo trong lớp

Khi chúng ta sử dụng một số hàm tạo, việc khởi tạo thành viên có thể bị lặp lại.

Chúng ta có thể giải quyết vấn đề đó bằng cách giới thiệu các đối số mặc định để giảm số lượng hàm tạo. Ngoài ra, chúng ta có thể thêm trình khởi tạo vào các thành viên dữ liệu:

```
class Date {
        int d {today.d}; int m {today.m};          int y {today.y};
public:
        Date(int, int, int);
        Date(int, int);
        Date(int);
        Date();
        Date(const char*);
```

16.2.8 Định nghĩa hàm trong lớp

Một hàm thành viên của một lớp là một hàm được định nghĩa bên trong định nghĩa lớp giống như bất kỳ biến nào khác. Nó hoạt động trên bất kỳ đối tượng nào của lớp mà nó là một thành viên, và có sự truy cập tới tất cả thành viên của một lớp cho đối tượng đó. Ví dụ:

```
class Date {
public:
void add_month(int  n) { m+=n; }
```

```
    private:
    int d, m, y;    };
```

16.2.9 Mutability

Để trở nên hữu ích ngoài định nghĩa về các hằng số đơn giản của các kiểu tích hợp, chúng ta phải có khả năng định nghĩa các hàm hoạt động trên các đối tượng const của các kiểu do người dùng định nghĩa. Đối với các chức năng tự do nghĩa là các hàm nhận const T & đối số. Đối với các lớp, điều đó có nghĩa là chúng ta phải có khả năng xác định các hàm thành viên hoạt động trên các đối tượng const.

16.2.9.1 Chức năng thành viên const

Một đối tượng được tuyên bố là const không thể được sửa đổi và do đó, chỉ có thể gọi các hàm thành viên const vì các chức năng này đảm bảo không sửa đổi đối tượng.

Khi một hàm được tuyên bố là const, nó có thể được gọi trên bất kỳ loại đối tượng, đối tượng const cũng như các đối tượng không phải là const, trong khi một hàm thành viên không phải const chỉ có thể được gọi cho các đối tượng không phải const.

Ví dụ:

```
class Date {    int d, m, y;
public:
        int year() const;
        void add_year(int n); // add n years     };
void f(Date& d, const Date& cd){
int i = d.year(); // OK
d.add_year(1); // OK
int j = cd.year(); // OK
cd.add_year(1); // error : cannot change value of a const Date    }
```

16.2.9.3 Trạng thái – thay đổi được đối tượng

Một thành viên là mutable có thể được sửa đổi bởi một hàm thành viên const.

Ví dụ:

```
class Date {
public:
string string_rep() const;
private:
mutable bool cache_valid;
mutable string cache;
void compute_cache_value() const; };
```

Định nghĩa string_rep ():

```
string Date::string_rep() const{
if (!cache_valid) {
        compute_cache_value();
        cache_valid = true;     }
return cache;   }
```

Bây giờ chúng ta có thể sử dụng string_rep () cho cả đối tượng const và không phải const:

```
void f(Date d, const Date cd)   {
        string s1 = d.string_rep();
        string s2 = cd.string_rep(); // OK!     }
```

16.2.9.4 Mutability through Indirection

Const không áp dụng (tạm thời) cho các đối tượng được truy cập thông qua con trỏ hoặc tham chiếu.

Một con trỏ thành viên không có bất kỳ ngữ nghĩa đặc biệt nào để phân biệt nó với các con trỏ khác.

16.2.10 Self-Reference

Một lớp tự tham chiếu chứa một thành viên con trỏ trỏ đến một đối tượng lớp cùng loại lớp. Các đối tượng lớp tự tham chiếu có thể được liên kết với nhau để tạo thành các cấu trúc dữ liệu hữu ích như danh sách, hàng đợi, ngăn xếp và cây.

16.2.11 Quyền truy cập thành viên

Một thành viên của lớp X có thể được truy cập bằng cách sử dụng dấu "." hoặc bởi " ->" đối với con trỏ.Ví dụ:

```
struct X {
void f ();
int m;   };
void người dùng (X x, X ∗ px)   {
x.m = 1;// OK
px–> m = 1;// OK          }
```

16.2.12 Thành viên tĩnh static

Static dùng để khai báo thành viên dữ liệu dùng chung cho mọi thể hiện của lớp.

Khi chúng ta khai báo một thành viên của một lớp là static, nghĩa là, dù cho có bao nhiêu đối tượng của lớp được tạo, thì sẽ chỉ có một bản sao của thành viên static.

Một thành viên static được chia sẻ bởi tất cả đối tượng của lớp. Tất cả dữ liệu static được khởi tạo về 0 khi đối tượng đầu tiên được tạo, nếu không có mặt sự khởi tạo khác.

16.2.13 Các loại thành viên

Một lớp thành viên (thường được gọi là lớp lồng nhau) có thể tham chiếu đến các kiểu và các thành viên tĩnh bao bọc trong một lớp. Nó chỉ có thể tham chiếu đến các thành viên không tĩnh khi nó được cung cấp cho một đối tượng của lớp bao quanh tham khảo.

Một lớp lồng nhau có quyền truy cập vào các thành viên của lớp bao quanh nó, ngay cả với các thành viên riêng (giống như một hàm thành viên có), nhưng không có khái niệm về một đối tượng hiện tại của lớp bao quanh.

16.3 Lớp cụ thể

Một lớp cụ thể là một lớp có một triển khai cho tất cả các phương thức của nó được kế thừa từ trừu tượng hoặc được thực hiện thông qua các giao diện. Nó cũng không định nghĩa bất kỳ phương thức trừu tượng nào của riêng nó. Điều này có nghĩa là một thể hiện của lớp có thể được tạo/cấp phát với từ khóa new mà không phải thực hiện bất kỳ phương thức nào trước.

Một lớp trừu tượng có nghĩa là được sử dụng như một lớp cơ sở nơi một số hoặc tất cả các hàm được khai báo hoàn toàn là ảo và do đó không thể được khởi tạo. Một lớp cụ thể là một lớp bình thường không có chức năng hoàn toàn ảo và do đó có thể được khởi tạo

16.3.1 Chức năng thành viên lớp

Một hàm thành viên của một lớp là một hàm có định nghĩa hoặc nguyên mẫu của nó trong định nghĩa lớp như bất kỳ biến nào khác. Nó hoạt động trên bất kỳ đối tượng nào của lớp mà nó là thành viên và có quyền truy cập vào tất cả các thành viên của một lớp cho đối tượng đó.

Bạn có thể xác định cùng một chức năng bên ngoài lớp bằng cách sử dụng toán tử độ phân giải phạm vi (::)

16.3.2 Chức năng hàm trợ giúp

Thông thường, một lớp có một số hàm được liên kết với nó mà không cần được định nghĩa trong lớp bởi vì họ không cần quyền truy cập trực tiếp vào đại diện.

Các chức năng của người trợ giúp (điều mà tôi tin rằng mọi người có ý nghĩa nhất khi họ nói nó) thường là các chức năng bao gồm một số chức năng hữu ích mà bạn sẽ sử dụng lại, rất có thể là lặp đi lặp lại. Bạn có thể tạo các hàm trợ giúp được sử dụng cho nhiều loại mục đích khác nhau.

16.3.3 Nạp chồng toán tử

Nạp chồng toán tử (Operator Overloading) được dùng để định nghĩa toán tử cho có sẵn trong c++ phục vụ cho dữ liệu riêng do bạn tạo ra. Nếu gặp một biểu thức phức tạp, số lượng phép tính nhiều thì việc sử dụng các phương thức trên khá khó khăn và có thể gây rối cho người lập trình. Vì thế ta sẽ nạp chồng lại các toán tử để có thể tạo một cái nhìn trực quan vào code, giảm thiểu các lỗi sai không đáng có.

Ví dụ: bool operator>(Date, Date;

16.3.4 Tầm quan trọng của concrete classes

Mục đích của concrete là làm tốt và hiệu quả một việc duy nhất, tương đối đơn giản. Nó thường không nhằm mục đích cung cấp cho người dùng các phương tiện để sửa đổi hành vi của một loại cụ thể. Trong đặc biệt, các loại concrete không nhằm mục đích hiển thị hành vi đa hình thời gian chạy.

Một lớp học cụ thể là một lớp có một thực hiện cho tất cả các phương pháp của nó. Họ không thể có bất kỳ phương pháp không bị cản trở nào. Nó cũng có thể mở rộng một lớp trừu tượng hoặc thực hiện một giao diện miễn là nó thực hiện tất cả các phương pháp của họ. Đó là một lớp học hoàn chỉnh và có thể được lập tức.