Chương 17

17.1 Hàm khởi tạo và hàm huỷ

17.1.1 Hàm khởi tạo

Một thành viên có cùng tên với lớp của nó được gọi là một phương thức khởi tạo. Ví dụ:

```
class Vector {
public:
    Vector(int s);
};
```

Hàm khởi tạo là một hàm thành viên đặc biệt của một lớp. Nó sẽ được tự động gọi đến khi một đối tượng của lớp đó được khởi tạo.

17.1.2 Hàm huỷ

Hàm huỷ cũng là một hàm thành viên đặc biệt giống như hàm tạo, nó được dùng để phá huỷ hoặc xoá một đối tượng trong lớp.Ví dụ:

```
Class Vector {
public:
    Vector(int s) :elem{new double[s]}, sz{s} { }; // Hàm tạo
~   Vector() { delete[] elem; } // Hàm huỷ: giải phóng bộ nhớ
private:
double∗ elem;
int sz;
};
```

17.2 Khởi tạo đối tượng của lớp

17.2.1 Hàm khởi tạo không tham số (mặc định)

Hàm tạo loại này sẽ không truyền vào bất kì một đối số nào.Ví dụ:

```
class Vector {
public:
    Vector(); // Hàm khởi tạo mặc định};
```

17.2.2 Hàm khởi tạo sao chép

Hàm khởi tạo sao chép là một hàm tạo mà tạo một đối tượng bằng việc khởi tạo nó với một đối tượng của cùng lớp đó, mà đã được tạo trước đó.

Một hàm khởi tạo sao chép sẽ có nguyên mẫu chung như sau:

```
ClassName(const ClassName &old_obj){ //Code }
```

Trong đó Classname là tên của lớp, old_obj là đối tượng cũ sẽ lấy làm gốc để sao chép sang đối tượng mới.

17.3 Sao chép và di chuyển

17.3.1 Sao chép

Sao chép một lớp X bằng 2 cách:

+Sao chép hàm tạo: X(const X&)

+Sao chép phép gán: X& operator=(const X&)

17.3.2 Sao chép cơ sở

Đối với mục đích sao chép, một cơ sở là một thành viên: để sao chép một đối tượng của một lớp bạn có thể sao chép cơ sở của nó. Ví dụ:

```
struct B1 {
        B1();
        B1(const B1&);      };
struct B2 {
        B2(int);
        B2(const B2&);      };
struct D : B1, B2 {
        D(int i) :B1{}, B2{i}, m1{}, m2{2*i} {}
        D(const D& a) :B1{a}, B2{a}, m1{a.m1}, m2{a.m2} {}
        B1 m1;
        B2 m2;};
D d {1}; // construct with int argument
D dd {d}; // copy construct
```

17.3.3 Cắt đối tượng

Cắt đối tượng xảy ra khi một đối tượng lớp có nguồn gốc được gán cho một đối tượng lớp cơ sở, các thuộc tính bổ sung của một đối tượng lớp có nguồn gốc được cắt ra để tạo thành đối tượng lớp cơ sở.

17.3.4 Hàm tạo di chuyển

Hàm tạo di chuyển lấy giá trị tham chiếu tới một đối tượng của lớp, và được dùng để hiện thực chuyển quyền sở hữu tài nguyên của đối tượng tham số.

17.4 Tạo toán tử mặc định

17.4.1 Mặc định rõ ràng

Vì việc tạo ra các hoạt động mặc định khác có thể bị chặn, nên phải có một cách để lấy lại mặc định. Ngoài ra, một số người muốn xem danh sách đầy đủ các thao tác trong chương trình văn bản ngay cả khi danh sách đầy đủ đó không cần thiết.

Ví dụ:

```
class gslice {
        valarray<siz e_t> siz e;
        valarray<siz e_t> stride;
        valarray<siz e_t> d1;
public:
        gslice() = default;
        ~gslice() = default;
        gslice(const gslice&) = default;
        gslice(gslice&&) = default;
        gslice& operator=(const gslice&) = default;
        gslice& operator=(gslice&&) = default;   };
```

17.4.2 Toán tử mặc định

```
struct S {
```

```
        string a;
        int b;
        };
S f(S arg){
        S s0 {};
        S s1 {s0};
        s1 = arg;
        return s1;
        }
```

17.4.3 Sử dụng thao tác mặc định

Phần này trình bày một số ví dụ chứng minh cách sao chép, di chuyển và hủy được liên kết một cách hợp lý. Nếu chúng không được liên kết, các lỗi hiển nhiên khi bạn nghĩ về chúng sẽ không được trình biên dịch bắt gặp.

17.4.3.1 Trình tạo mặc định

Trình tạo mặc định là người xây dựng không lấy bất kỳ đối số nào. Nó không có thông số. Ví dụ:

```
struct X {      X(int);};
X a {1};
```

Nếu chúng ta muốn có hàm tạo mặc định, chúng ta có thể định nghĩa một hoặc khai báo rằng chúng ta muốn hàm tạo mặc định do trình biên dịch tạo ra như sau:

```
struct Y {
        string s; int n;
        Y(const string& s);
        Y() = default; }; }
```

17.4.3.2 Bất biến

[1] Thiết lập một bất biến trong một phương thức khởi tạo (bao gồm cả khả năng thu nhận tài nguyên).

[2] Duy trì tính bất biến với các thao tác sao chép và di chuyển (với các tên và kiểu thông thường).

[3] Thực hiện mọi thao tác dọn dẹp cần thiết trong trình hủy (bao gồm cả giải phóng tài nguyên).

17.4.3.3 Tài nguyên bất biến

Nhiều ứng dụng quan trọng và rõ ràng nhất của bất biến liên quan đến quản lý tài nguyên.Ví dụ:

```
template<class T> class Handle {
T* p;
public:
Handle(T* pp) :p{pp} { }
T& operator*() { return *p; }
```

~Handle() { delete p; } };

17.4.3.4 Bất biến được chỉ định một phần

Các ví dụ rắc rối dựa trên các bất biến nhưng chỉ thể hiện một phần chúng thông qua các hàm tạo hoặc hàm hủy là hiếm hơn nhưng không phải là chưa từng thấy.Ví dụ:

```
    class Tic_tac_toe {
public:
Tic_tac_toe(): pos(9) {} // always 9 positions
Tic_tac_toe& operator=(const Tic_tac_toe& arg)
{
for(int i = 0; i<9; ++i)
pos.at(i) = arg.pos.at(i);
return ∗this;
}
// ... other operations ...
enum State { empty, nought, cross };
private:
vector<State> pos;
};
```

17.4.3.5 Deleted Functions

Chúng ta có thể ‘‘xóa’’ một hàm; nghĩa là, chúng ta có thể nói rằng một hàm không tồn tại để cố gắng sử dụng nó (một cách ngầm hiểu hoặc rõ ràng) là một lỗi. Cách sử dụng rõ ràng nhất là loại bỏ chức năng. Ví dụ:

```
    class Base {
// ...
Base& operator= (const Base&) = delete;// không cho phép sao chép
Base (const Base&) = delete;
Base& operator=(Base&&) = delete; // không cho phép sao chép
Base(Base&&) = delete;
};
Base x1;
Base x2 {x1}; // lỗi: không có hàm tạo bản sao
```

Lưu ý sự khác biệt giữa một hàm đã xóa và một hàm chưa được khai báo. Trong trường hợp trước đây, trình biên dịch lưu ý rằng lập trình viên đã cố gắng sử dụng hàm đã xóa và đưa ra lỗi. Trong trường hợp sau, trình biên dịch tìm kiếm các lựa chọn thay thế, chẳng hạn như không gọi hàm hủy hoặc sử dụng operator new().Ví dụ:

```
    class Not_on_stack {
        ~Not_on_stack() = delete;    };
    class Not_on_free_store {
```

```
        void* operator new(siz e_t) = delete;       };

    void f(){
        Not_on_free_store v2;
        Not_on_stack* p1 = new Not_on_stack;  }
```

17.5 Lời khuyên

Thiết kế hàm tạo, phép gán và hàm hủy dưới dạng một tập hợp các thao tác.

Nếu một hàm tạo có được một tài nguyên, thì lớp của nó cần một hàm hủy để giải phóng tài nguyên đó.

Nếu một lớp có một hàm ảo, nó cần một hàm hủy ảo.

Nếu một lớp không có hàm tạo, nó có thể được khởi tạo bằng cách khởi tạo thành viên.

Ưu tiên khởi tạo {} hơn khởi tạo = và ()

Nếu một lớp là một vùng chứa, hãy cung cấp cho nó một phương thức khởi tạo danh sách khởi tạo.

Khởi tạo các thành viên và căn cứ theo thứ tự khai báo của chúng.

Ưu tiên khởi tạo thành viên hơn là gán trong một phương thức khởi tạo.

Sử dụng bộ khởi tạo trong lớp để cung cấp các giá trị mặc định.

Nếu một lớp có một thành viên con trỏ, nó có thể cần một hàm hủy và các hoạt động sao chép không mặc định.

Khi thêm một thành viên mới vào một lớp, hãy kiểm tra xem có các hàm tạo do người dùng xác định cần được cập nhật để khởi tạo thành viên đó hay không.