**DỊCH SÁCH C++**

**CHƯƠNG 18: NẠP CHỒNG.**

Là hình thức sử dụng tên hàm định nghĩa giống nhau, nhưng khác nhau về kiểu dữ liệu, thức tự các biến. Thường được sử dụng với các hàm thường xuyên xuất hiện trong chương trình, các hàm đều có cùng toán tử hay mục đích tính toán nhưng đối tượng khác nhau( có thể về kiểu dữ liệu hay thức tự). Để tránh việc đặt nhiều tên khác nhau, nhưng lại không nhớ, dẫn đến sai sót trong quá trình lập trình.

1. NẠP CHỒNG TOÁN TỬ

Đây là các phép toán được sử dụng trong nạp chồng do các phép này trả về kết quả dựa vào dữ liệu của đối tượng:

+ − ∗ / % ˆ &

| ̃ ! = < > +=

-= ∗= /= %= ˆ= &= |=

<< >> >>= <<= == != <=

>= && || ++ −− −>∗ ,

−> [] () new new[] delete delete[]

Các phép toán sau đây không được sử dụng trong nạp chồng:

:: (phạm vi dữ liệu)

. (lựa chọn thành viên)

.\* (lựa chọn thành viên của đối tượng con trỏ)

Sizeof kiểu dữ liệu của đối tượng

alignof quan hệ giữa các đối tượng

typeid dạng của đối tượng

* Các phép toán này hoạt động dựa vào tên chứ không phải dựa vào giá trị của biến mảng.
* Cùng là toán tử biểu thức có điều kiện không được thực hiện: ?:
* Tên của một hàm toán tử được quy định: **operator**@

(với @ là các toán tử muốn nạp chồng” + ,- ,\* ,/,…”)

a. Toán tử nhị phân và toán tử một ngôi.

Toán tử hai ngôi: a+b, a-b, a\*b,….

Nạp chồng toán tử hai ngôi được xác định với hàm thành viên hoặc không thành viên 1 đối số hoặc hai đối số.

VD: 1 đối số: operator+(*aa*)

2 đối số: *aa*.operator+(*bb*); operator(*aa,bb*)

Toán tử 1 ngôi: phép tăng hay giảm( i++, i--)

Nạp chồng toán tử một ngôi có thể sử dụng hàm thành viên tĩnh 1 đối số hoặc hàm không thành viên lấy 1 đối số.

VD: aa.operator+(), operator+(aa)

b. Ý nghĩa cũa nạp chồng

Một số toán tử được cài sẵn có ý nghĩa tương đương với một số sự kết hợp trên cùng 1 đối số.

VD: a++ tương đương a+=1 tương đương a=a+1

Nhưng quan hệ này không được áp dụng cho kiểu toán tử do người dùng định nghĩa.

Toán tử =( gán), &( lấy địa chỉ), và “, “ có ý nghĩ xác định trước khi áp dụng vào đối tượng lớp, ý nghĩa này có thể được xóa bỏ.

c. Toán tử và kiểu do người dùng định nghĩa.(class hoặc struct)

Một hàm nạp chồng toán tử phải là một thành viên hay có một đối số thuộc kiểu do người dùng định nghĩa. Đối số đầu tiên phải là một thành viên của kiểu dữ liệu định sẵn.

d. Truyền đối tượng.

* Truyền bằng trị

VD: void Point::operator+=(Point delta);

* Truyền bằng tham chiếu

VD: Matrix operator+(const Matrix&, const Matrix&);

e. Phạm vi chứa hàm toán tử.

Namespace được dùng để phân biệt phạm vi chứa các hàm giống nhau. Để khi gọi hàm, chương trình sẽ biết được sẽ phải sử dụng hàm nào để thực hiện.

⮘Cú pháp: Định nghĩa: namespace *ten\_cua\_namespace* { *phan\_code*}

Gọi tên: *ten\_cua\_namespace* :: *ten\_ham*;

VD:

namespace std { *// simplified std*

class string {

*// ...*

};

class ostream {

*// ...*

ostream& operator<<(const char∗);*// output C-style string*

};

extern ostream cout;

ostream& operator<<(ostream&, const string&);*// output* std::string

} *// namespace std*

int main()

{

const char∗ p = "Hello";

std::string s = "world";

std::cout << p << ", " << s << "!\n";

}

std::cout << p std::cout.operator<<(p) (nạp chồng toán tử nhập).

Vì std::ostream không là thành viên của lớp std::string nên std::cout << s

operator<<(std::cout,s).

Cũng giống như việc chương trình có thể tự tìm hàm dựa vào đối số. Thì không gian tên cũng vậy, nó có thể được tìm dựa vào các toán tử chứa trong nó, cout nằm trong std nên khi nhắc đến << có nghĩa là nhắc đến cout.

VD: xét toán tử nhị phân @, x thuộc dạng X, y thuộc dạng Y. x@y được thực hiện như sau:

* Nếu X là 1 class khi nhắc @ thì tìm đến @ trong class X và tìm trong các ngữ cảnh liên quan.
* Nếu X là namespace của N thì tìm operator@ trong X.
* Nếu Y là namespace của M thì tìm operator@ trong M.

Chú ý: thiếu ẩn toán tử sẽ giúp cho toán tử lúc nào cũng truy cập được và người dùng có thể dễ dàng thay đổi ý nghĩa khác mà không cần sửa đổi khai báo.

Thư viên iostream cho phép người dùng sử dụng << để xuất dữ liệu mà không cần sửa đổi class ostream.

1. DẠNG SỐ PHỨC.
2. Hàm thành viên và hàm không thành viên.

Hàm thành viên là hàm được viết bên trong class, hàm không thành viên là hàm viết bên ngoài class.

Nếu hàm nạp chồng là hàm thành viên

1. Tính toán tổng hợp.

VD: 3+ z với z là số phức, 3 là số thực. Hai kiểu dữ liệu khác nhau sẽ không gọi hàm ra để thực hiện được. Khi đó ta có thể định nghĩa các hàm như sau:

complex operator+(complex a, complex b)

{

return a += b; // goi complex::operator+=(complex)

}

complex operator+(complex a, double b)

{

return {a.real()+b,a.imag()};

}

complex operator+(double a, complex b)

{

return {a+b.real(),b.imag()};

}

void f(complex x, complex y)

{

auto r1 = x+y; // calls operator+(complex,complex)

auto r2 = x+2; // calls operator+(complex,double)

auto r3 = 2+x; // calls operator+(double,complex)

auto r4 = 2+3; // built-in integer addition

}