**CHƯƠNG 18: NẠP CHỒNG**

1. **Nạp chồng hàm:**

Là các hàm có tên giống nhau, nhưng khác nhau về kiểu dữ liệu của tham số. Khi gọi tên, hàm sẽ được truy cập tới dựa vào tham số truyền vào.

VD:

void tong( int a, int b)

{

return a+b;

}

void tong( float a, float b)

{

return a+b;

}

int main()

{

tong(2,3); // truy cap ham tong( int a, int b)

tong(3.5,6.8); //truy cap ham tong( float a, float b)

1. **Nạp chồng toán tử:**

Toán tử có thể nạp chồng:

|  |  |  |
| --- | --- | --- |
| **Ngôi** | **Nhóm** | **Toán tử** |
| 1 ngôi | Tăng giảm | ++, -- |
|  | Dấu số học | +, - |
|  | Logic | !, ~ |
|  | Con trỏ | \*, & |
|  | Ép kiểu | int, float, double,… |
|  |  |  |
| 2 ngôi | Số học | +, -, \*, /, %, +=, -=, \*=, /=, %=,… |
|  | So sánh | <, >, ==, >=, <=, != |
|  | Logic | &&, ||, &, | |
|  | Nhập xuất | <<, >> |
|  | Gán | = |
|  | Lấy chỉ số mảng | [ ] |

Tóan tử không thể nạp chồng:

|  |  |
| --- | --- |
| **Nhóm** | **Toán tử** |
| Xác định phạm vi | :: |
| Lựa chọn thành viên | . |
| Lựa chọn thành viên thông qua con trỏ | .\* |
| Kích thước | sizeof |
| Căn chỉnh | alignof |
| Trả về thông tin lúc chạy | typeid |
| Toán tử điều kiện | ? : |
| Chỉ thị tiền xử lý | # |
| Chỉ thị tiền xử lý | ## |

Tên của hàm toán tử:

operator@(); Với @ là toán tử

VD:

class phanso

{

private:

int tu;

int mau;

puplic:

void operator\*(phanso a,phanso b);

};

void phanso::operator\*(phanso a,phanso b){

phanso t;

tu=a.tu\*b.tu;

mau=a.mau\*b.mau;

return t;

}

int main(){

phanso x,y;

cout<<”Tong”<<x\*y; //goi ham tinh tong

}

Hàm nạp chồng toán tử có ít nhất 1 đối số, đối số đầu tiên bao giờ cũng thuộc lớp do người dùng định nghĩa.

Đối với các phép toán tăng giảm một đối tượng:

Có 2 trường hợp: ++a hoặc a++

Đối với trường hợp ++a ta viết tên hàm : operator++a();

Đối với trường hợp a++ ta viết tên hàm : operator++a(int);

Có những phép toán mà đối với dữ liệu có sẵn, chương trình có thể tự hiểu và chuyển đổi như –a tương tương với a -= 1 hay là a=a-1. Nhưng với nạp chồng khi ta viết opertator +=() không có nghĩa nó là sự kết hợp giữ operator+() và operator=().

Với toán tử gán( =) giá trị được cung cấp cho đối tượng trước khi gán sẽ được xóa.

Một hàm nạp chồng toán tử phải là một thành viên của class và có ít nhất một đối số do người dùng định nghĩa

Có 2 cách để truyền giá trị cho toán tử nạp chồng là truyền bằng trị và truyền bằng tham trị, với các đối tượng không thay đổi trong suốt quá trình thực hiện chương trình thì ta có thể thêm const.

|  |  |
| --- | --- |
| **Truyền bằng trị** | **Truyền tham trị** |
| operator +=( int a); | operator +=( int &a); |

VD: Hàm truyền bằng tham trị

ostream& operator << (ostream & os, Phanso &ps)

{

os<< ps.tu<<"/"<<ps.mau<<endl;

return os;

}

VD: Hàm truyền bằng trị

operator <(Phanso ps)

{

if((float)(tu/mau)<(float)(ps.tu/ps.mau))

return 1;

}

Hàm nạp chồng toán tử là thành viên của lớp, nhưng có những hàm nạp chồng toán tử cùng tên, giống nhau cả về tham số nhưng khác nhau về kết quả trả về. Vì vậy khi gọi tên hàm chương trình sẽ không biết phải sử dụng hàm nào. Do đó namespace được triển khai để giúp chương trình tìm được hàm cần sử dụng. Toán tử nạp chồng khi được gọi ra sẽ được chương trình tìm kiếm trong không gian tên của nó thông qua các tham số.

class complex {

private:

double re, im;

public:

complex& operator+=(complex a){

re += a.re;

im += a.im;

return ∗this;

}

complex& operator+=(double a){

re += a;

return ∗this;

}

};

complex operator+(complex a, complex b)

{

return a += b;

}

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;

auto r2 = x+2;

auto r3 = 2+x;

auto r4 = 2+3;

}

Ta cũng có thể xây dựng một hàm chuyển kiểu giống như khi người ta cho complex{3} có nghĩa là cho một số phức với phần thực là 3 và phần ảo là 0.

Ta xây dựng lớp như sau để định nghĩa:

class complex

{

private:

double re, im;

public:

complex(): re{0}, im{0} { }

complex(double r): re{r}, im{0} { }

complex(double r, double i): re{r}, im{i} { }

};

Tuy nhiên cách nào sẽ trỏ nên phức tạp nếu trong trường hợp hàm nạp chồng có 3 đối số, sẽ có rất nhiều trường hợp xảy ra. Nên chúng ta có thể viết theo cách khác:

bool operator==(complex,complex);

void f(complex x, complex y)

{

x==y;

x==3;

3==y;

}

1. **Literal**

Literal là giá trị cử chính nó đang mang. Ví dụ trong câu lệnh cout<<”Hello world”;

Thì literal chính là “Hello world”. Một số phức có thể được tạo trong quá trình biên dịch khi ta sử dụng constexpt:

class complex {

public:

constexpr complex(double r =0, double i =0) : re{r}, im{i} { }

};

void main()

{

complex z1 {1.2+12e3i};

}

Trong 1 vài trường hợp ta sẽ cần sử dụng dữ liệu ở private. Nhưng nếu dùng hàm thông thường thì chương trình sẽ không cho phép thực hiện. Do đó, ta phải dùng hàm thành viên lấy ra giá trị thuộc tính ở private.

class complex

{

double re,im;

public:

constexp double real() const {return re;}

constexp double image() const {return im;}

void real (double r) {re=r;}

void hinh (double i) {im=i:}

}

1. **Ép kiểu trong c++**

Chúng ta có thể sử dụng toán tử operator để ép kiểu dữ liệu trong c++

**CHƯƠNG 19: TOÁN TỦ ĐẶC BIỆT**

1. **Toán tử đặc biệt**

Đó là các loại toán tử sau: [], (), ++, --,new delete

Nạp chồng toán tử truy cập đến phần tử mảng

#include <iostream>

using namespace std;

const int MAX = 15;

class Mang

{

private:

int A[MAX];

public:

Mang()

{

register int i;

for (i = 0; i < MAX; i++)

{ A[i] = i; }

}

int& operator[](int i)

{

if (i > MAX)

{

cout << "\n======================\n" << endl;

  cout << "Chi muc vuot gioi han!" << endl;

return mang[0];

}

return mang[i];

}

};

int main()

{

Mang V;

cout << "Gia tri cua V[3] la: " << V[3] << endl;

cout << "Gia tri cua V[6] la: " << V[6] << endl;

cout << "Gia tri cua V[16]la:"<<V[16]<<endl;

return 0;

}

Toán tử () được dùng với hàm toán tử truyền tham số bất kì.

#incluce<iostream>

using namespace std;

class Sosanh{

private:

int a;

int b;

public:

int operator() (int a, int b)

{

if( a>b)

{

return a;

}

}

int oprator() (int a, int b, int c)

{

return a+b+c;

}

};

int main()

{

Sosanh f;

cout<< f( 3,5)<<endl;

cout<< f( 7,8,3);

}

Tăng giảm tiền tố hậu tố:

class Phanso{

private :

int tu;

int mau;

public:

Phanso operator ++(int);

Phanso operator ++();

}

Phanso Phanso::operator ++(int)

{

tu=tu+mau;

return \*this;

}

Phanso Phanso::operator ++()

{

tu=tu+mau;

return \*this;

}

Xác định new và delete cho 1 lớp cụ thể:

void \*operator new(size\_t);

void operator delete(void\*, size\_t);

Kiểu dữ liệu do người dùng định nghĩa:

VD: nhị phân, 5giây, 10km, 20phút,…

Những kiểu dữ liệu này luôn có hậu tố đi theo như giây, phút, giờ, m, km,…

Khi đó ta sẽ dùng operator để đọc 1 thông tin nhập vào với hậu tố:

operator “” <hậu tố> (tham số);

VD:Ta nhập hai quãng đường 10km và 5km. khi tính toán thì máy tính không thể lấy 10km+5km mà lấy như sau 10+5 rồi thêm hậu tố km vào kết quả.

* KM operator “” km(float d)

{

return KM(d);

}

Trả về giá trị trước hậu tố để tính toán.

1. **Lớp chuỗi:**

Các phương thức với lớp chuỗi

* Tạo chuỗi mặc định.
* Sao chép chuỗi.
* Di chuyển chuỗi.
* Hủy chuỗi.

Truy cập vào các kí tự trong chuỗi bằng cách sử dụng [].

Trình bày chuỗi đáp ứng 3 yêu cầu: dễ dàng truy cập vào từng kí tự, dễ thêm kí tự vào cuối và tiết kiệm bộ nhớ.

Hàm thành viên của class chuỗi gồm có:

Hàm tạo chuỗi rỗng

String::String()  
 :sz{0}, ptr{ch}

{

ch[0]=0;

}

Hàm tạo copy chuỗi

String::String(const char∗ p)

:sz{strlen(p)},

ptr{(sz<=short\_max) ? ch : new char[sz+1]},

space{0}

{

strcpy(ptr,p);

}

Hàm tạo di chuyển chuỗi

String::String(String&& x)

{

move\_from(x);

}

Toán tử copy:

String& String::operator=(const String& x)

{

if (this==&x) return ∗this;

char∗ p = (shor t\_max<sz) ? ptr : 0;

copy\_from(x);

delete[] p;

return ∗this;

}

Toán tử di chuyển:

String& String::operator=(String&& x)

{

if (this==&x) return ∗this;

if (short\_max<sz) delete[] ptr;

move\_from(x); // does not throw

return ∗this;

}

Thêm kí tự vào cuối:

String& String::operator+=(char c)

{

if (sz==short\_max) {

int n = sz+sz+2;

ptr = expand(ptr,n);

space = n−sz−2;

}

else if (short\_max<sz)

{

if (space==0)

{

int n = sz+sz+2;

char∗ p = expand(ptr,n);

delete[] ptr;

ptr = p;

space = n−sz−2;

}

else

−−space;

}

ptr[sz] = c;

ptr[++sz] = 0;

return ∗this;

}

1. Hàm bạn:

Là cho phép hàm ngoài class có thể truy cập vào thuộc tính private.

class Phanso{

private :

int tu;

int mau;

};

friend istream& operator >> (istream & is, Phanso &ps);

Hàm bạn sẽ được gọi thông qua đối số của nó.

Cách sử dụng hàm bạn và hàm thành viên:

Hàm thành viên chỉ sử dụng cho một đối tượng thuộc lớp của nó, còn hàm bạn được sử dụng chỉ với mục đích truy cập vào một phần của thuộc tính lớp nhằm mục đích khác chứ không thể hiện rõ nó là một phương thức của đối tượng.