CHƯƠNG 3: NẠP CHỒNG TOÁN TỬ

Giảng viên: Trần Thị Ngân

Email: ngantt@tlu.edu.vn

NỘI DUNG

3.1 Nạp chồng toán tử

3.2 Nạp chồng toán tử bằng hàm thành viên

3.3 Nạp chồng toán tử bằng hàm bạn

Định nghĩa

- Nạp chồng toán tử là khả năng định nghĩa các phương thức của lớp dưới dạng các toán tử **(phép toán)**
- Các toán tử **cùng tên** thực hiện được nhiều chức năng khác nhau được gọi là **nạp chồng (overload) toán tử**
- Ví dụ:

```
PS  operator - ( );
friend PS operator - ( PS x , PS  y );
```

Khai báo nạp chồng toán tử

* Khai báo tổng quát:

<return - type> operator # (d/s đối số);

trong đó:

- ✓ return-type: kiểu trả lại kết quả của hàm toán tử
- ✓ #: tên toán tử cần định nghĩa (+,-,*,/,…)

•Nạp chồng toán tử được định nghĩa trong vị trí public của phần khai báo lớp.

Các toán tử được nạp chồng

+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete

.	.*	::	?:	sizeof

Các toán tử không được nạp chồng

Bài tập

Viết chương trình xây dựng lớp Phân số. Yêu cầu định nghĩa:

- Hàm tạo không tham số, hàm tạo có tham số
- Nhập và xuất phân số
- Tính tổng hai phân số
- Tính hiệu hai phân số (*friend*)
- Áp dụng nhập vào hai phân số và in ra kết quả

Định nghĩa toán tử >> và <<

- Cho phép nạp chồng hai toán tử nhập (>>) và xuất(<<) cho các đối tượng của lớp bất kỳ
- Định nghĩa hai toán tử dưới dạng hàm friend
- Khi đó có thể sử dụng dòng **cin>>** và **cout<<** để nhập và xuất dữ liệu cho đối tượng lớp

Định nghĩa toán tử <<

- Khai báo:

  friend ostream & operetor << (ostream &out, class-name c);

- Định nghĩa:

```
ostream & operetor << (ostream &out, class-name c)
{
    //định nghĩa thao tác in các giá trị của "c" ra màn hình.
    //Lưu ý dùng dòng out<< để xuất thay vì dùng cout<<
        return out;
}
```

Định nghĩa toán tử <<

Trong đó:

- **ostream:** tham chiếu chỉ đến dòng xuất,
- **out**: tên dòng xuất tạm thời sử dụng thay cho cout
- **class-name***:* tên của lớp đối tượng cần định nghĩa toán tử <<

Ví dụ

```
class phanso
{
public:
   int mau, tu;
   ….
   friend ostream& operator<<(ostream &out, phanso a);
   ….
};
ostream& operator<<(ostream &out, phanso a)
{
        out<<“Phan so:”<<a.tu<<“/”<<a.mau;
        return out;
}
```

Định nghĩa toán tử >>

- Khai báo:

friend istream & operator >> (istream &in, *class-name* &c);

- Định nghĩa:

```
istream & operator >> (istream &in, class-name &c)
{
  //định nghĩa thao tác nhập các giá trị cho đối tượng "c".
  //Lưu ý dùng dòng in>> để nhập thay vì dùng cin>>
  return in;
}
```

Định nghĩa toán tử >>

- Trong đó:
  - **istream:** tham chiếu chỉ đến dòng nhập,
  - **in**: tên dòng nhập tạm thời sử dụng thay cho cin
  - **class-name***:* là tên của lớp đối tượng cần định nghĩa toán tử nhập

Ví dụ

```
class phanso
{
public:
   int mau, tu;
   ….
   friend istream& operator>>(istream &in, phanso &a);
   ….
};

istream& operator>>(istream &in, phanso &a)
{
        cout<<“Nhap tu:”;          in>>a.tu;
        cout<<“Nhap mau:”;         in>>a.mau;
        return in;
}
```

Bài tập áp dụng

Viết chương trình xây dựng lớp số phức. Yêu cầu:

- Định nghĩa toán tử nhập >> và xuất <<
- Định nghĩa toán tử +, - để cộng, trừ hai số phức
- Định nghĩa toán tử *, / để nhân, chia hai số phức(friend)
- Định nghĩa toán tử == để so sánh hai số phức(friend)
- Áp dụng nhập vào hai số phức và in ra màn hình các kết quả

```
#include …
class sophuc
{
private:
     float t;          //phan thuc
     float a;          //phan ao
public:
     friend ostream& operator<<( ostream &os , sophuc x );
     friend istream& operator>>( istream &is , sophuc &x );
     sophuc operator+( sophuc y );
     sophuc operator-( sophuc y );
     friend sophuc operator*( sophuc x, sophuc y );
     friend sophuc operator/( sophuc x, sophuc y );
     friend int operator==( sophuc x, sophuc y );
};
```

```
ostream& operator<<( ostream &os , sophuc x )
{
    os<<x.t;
    if(x.a<0) os<<x.a<<"i";
    else os<<"+"<<x.a<<"i";
    return os;
}
```

```
istream& operator>>( istream &is, sophuc &x )
{
    cout<<"Nhap phan thuc:";
    is>>x.t;
    cout<<"Nhap phan ao:";
    is>>x.a;
    return is;
}
```

```
sophuc sophuc::operator+(sophuc y )
{
    sophuc kq;
    kq.t = t + y.t;
    kq.a = a + y.a;
    return kq;
}
```

```
sophuc sophuc::operator-(sophuc y )
{
    sophuc kq;
    kq.t = t - y.t;
    kq.a = a - y.a;
    return kq;
}
```

```
sophuc operator*(sophuc x,sophuc y )
{
    sophuc kq;
    kq.t = x.t*y.t - x.a*y.a;
    kq.a = x.t*y.a + x.a*y.t;
    return kq;
}
```

```
sophuc operator/(sophuc x, sophuc y )
{
    sophuc kq;
    kq.t = x.t*y.t + x.a*y.a;
    kq.a = x.a*y.t - x.t*y.a;
    return kq;
}
```

```
int operator==( sophuc x, sophuc y )
{
    float dd1,dd2;
    dd1 = x.t*x.t + x.a*x.a;
    dd2 = y.t*y.t + y.a*y.a;
    if (dd1==dd2) return 1;
    else return 0;
}
```

```
void main()
 {
      sophuc a,b;
      sophuc kq;
      cout<<"Nhap so phuc thu nhat \n";    cin>>a;
      cout<<"\n Nhap so phuc thu hai:\n"; cin>>b;
      //kq=a+b;
      kq=a.operator+(b);
      cout<<"\n\nTong:"<<kq;
      kq=a-b;
      cout<<"\n\nHieu:"<<kq;
      getch();
 }
```

Toán tử chuyển đổi kiểu

- Toán tử chuyển đổi kiểu được dùng để chuyển đổi một đối tượng của lớp thành đối tượng lớp khác hoặc thành đối tượng của một kiểu dữ liệu đã có sẵn
- Hàm chuyển đổi kiểu phải là hàm thành viên không tĩnh (*không static)* và không hàm bạn (friend)

Toán tử chuyển đổi kiểu

- cú pháp:

  operator **<***kiểu-cần-chuyển***>**();

- ví dụ:

```
class date
{ ….
   public:
      operator int();    //chuyển kiểu date về kiểu int
};
```

Ví dụ

Xây dựng lớp Date. Yêu cầu:

- Định nghĩa hàm toán tử nhập và xuất
- Định nghĩa toán tử chuyển đổi để chuyển một giá trị Date thành kiểu số nguyên
- Áp dụng nhập vào một ngày bất kỳ và in ra số nguyên tương ứng của ngày đó

Ví dụ

```
//Dinh nghia kieu du lieu moi co ten la String
typedef char *string;
//Dinh nghia mang chua so ngay
int ngay[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
//Mo ta lop
class N_Date
{
  int d,m,y;//Ngay - Thang - Nam
public:
  //Toan tu chuyen doi kieu,
  operator int();
  //Dinh nghia phuogn thuc nhap va xuat
  friend ostream& operator<<(ostream &out, N_Date x);
  friend istream& operator>>(istream &in, N_Date &x);
};
```

Ví dụ

```
istream& operator>>(istream &in, N_Date &x)
{
   cout<<"Ngay:";  in>>x.d;
   cout<<"Thang:"; in>>x.m;
   cout<<"Nam:";   in>>x.y;
   return in;
}
```

Ví dụ

```
ostream& operator<<(ostream &out, N_Date x)
{
   out<<x.d<<"/"<<x.m<<"/"<<x.y;
   return out;
}
```

Ví dụ

```
N_Date::operator int()
{
	int s=0,i;
	if( y== 1900)
	{
		s=s+d;
		for(i=0;i<m;i++)    s = s + ngay[i];
	}
	else
	{
		s=s+d;
		s=s+(y-1)*365;
		for(i=1900;i<y;i++)
			if((i%4==0)&&(i%100!=0)) s=s+1;
		for(i=0;i<m;i++)
		{
			s=s+ngay[i];
			if((i%4==0)&&(i%100!=0)&&(i==2)) s=s+1;
		}
	}
	return s;
}
```

Ví dụ

```
void main()
{
   N_Date x;
   long n;
   cout<<"Nhap ngay x: \n";
   cin>>x;
   n = long(x);
   cout<<"\n Ngay x = "<<x;
   cout<<"\n So nguyen tuong ung:"<<n;
   getch();
}
```

Toán tử []

- Là toán tử cho phép truy cập đến từng thành phần của đối tượng **(chuỗi ký tự, mảng)**
- là **Toán tử hai ngôi**, có dạng: **a[b]**
  - **a:** đối tượng cần truy cập
  - **b**: chỉ số vị trí phần tử cần truy cập
- Toán tử này phải là thành viên của lớp

NỘI DUNG

3.1 Nạp chồng toán tử

3.2 Nạp chồng toán tử bằng hàm thành viên

3.3 Nạp chồng toán tử bằng hàm bạn

Phân loại

- **Hàm thành viên**: hàm này không có đối số cho toán tử một ngôi hay có một đối số cho toán tử hai ngôi ⇔ tương tự hàm thành phần thông thường
- **Hàm không thành viên** - **Hàm bạn**: hàm có một đối số cho toán tử một ngôi và hai đối số cho toán tử hai ngôi. Hàm được khai báo thêm từ khóa *friend* trước tên hàm ⇔ tương tự hàm bạn

Quy tắc sử dụng

Loại	Khai báo	Sử dụng
Thành phần	type operator #();	a.operator #(); hoặc #a;
	type operator #(type b);	a.operator #(b); hoặc a#b;
Thân thiện	friend type operator #(type a);	#a
	friend type operator #(type a, type b);	a#b

Trong đó: # là toán tử cần định nghĩa

Ví dụ

Loại	Khai báo	Sử dụng
Thành phần	PS operator -();	kq = a.operator-(); hoặc kq = -a;
	PS operator -(PS b);	kq = a.operator -(b); hoặc kq = a-b;
friend	friend PS operator -(PS a);	kq = -a
	friend PS operator - (PS a, PS b);	kq = a - b

Trong đó: # là toán tử cần định nghĩa

Các nguyên tắc cơ bản

- Có thể định nghĩa nạp chồng trên các kiểu có sẵn hoặc trên các kiểu mới *(kiểu do người dùng định nghĩa)*
- Toán tử gán được sử dụng mặc định cho mọi lớp mà không cần định nghĩa
- Toán tử lấy địa chỉ (&) cũng được sử dụng mặc định cho các đối tượng
- Khi đa năng hóa ( ), [], -> hoặc = thì method nạp chồng phải là hàm thành viên