Chương 4 Định nghĩa toán tử trên lớp

4.1. Các phương thức toán tử

4.1.1. Cách đặt tên phương thức toán tử

Các phương thức (hoặc hàm) toán tử được xây dựng như các phương thức thông thường, chỉ có khác cách đặt tên.

Cách khai báo phương thức toán tử như sau:

<kiểu d/liệu trả về> operator <tên toán tử> ([ds tham số]) ;

Định nghĩa phương thức toán tử ở bên ngoài lớp:

<kiểu d/l trả về><tên_lớp>::operator<tên toán tử>([ds tham số])

{ //thân hàm }

Chú ý: Cũng giống như phương thức thông thường, phương thức toán tử có đối số đầu tiên (đối số không tường minh) là con trỏ this.

4.1.2. Phương thức toán tử một ngôi

Toán tử một ngôi (hay một toán hạng) dùng ngay con trỏ this để biểu thị toán hạng duy nhất, do đó phương thức toán tử một ngôi sẽ không có đối tường minh.

Ví dụ: xây dựng phương thức toán tử đổi dấu số phức.

```
class sophuc
{
  private:
     double thuc, ao;
  public:
     sophuc operator – ( );
};
```

```
sophuc sophuc::operator – ( )
{
    sophuc tg;
    tg.thuc = - thuc;          // hoặc tg.thuc = - this -> thuc;
    tg.ao = - ao;              // hoặc tg.ao = - this -> ao;
     return tg;
}
```

Cách dùng:

```
sophuc a, b;
a = -b;          // hoặc a = operator - (b);
```

4.1.3. Phương thức toán tử hai ngôi

Toán tử hai ngôi (hay toán tử hai toán hạng) dùng con trỏ this ứng với toán hạng thứ nhất, nên trong phương thức toán tử hai ngôi chỉ cần dùng một đối tường minh để biểu thị toán tử thứ hai.

Ví dụ: xây dựng phương thức toán tử + hai số phức.

```
class sophuc
{
  private:
     double thuc, ao;
  public:
     sophuc operator + (sophuc u );
};
```

```
sophuc  sophuc::operator + (sophuc  u )
{
    sophuc  tg;
    tg.thuc  =  this -> thuc  +  u.thuc;
        // hoặc tg.thuc  =  thuc  +  u.thuc;
    tg.ao  =  this -> ao  +  u.ao;
        // hoặc tg.ao  =  ao  +  u.ao;
     return  tg;
}

Cách dùng:
sophuc   a, b, c;
c = a + b;        // hoặc   c =  a.operator + (b);
```

- **Danh sách các toán tử có thể nạp chồng:**

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

Chú ý:

- Chỉ có thể định nghĩa lại các toán tử ở trên
- Không làm thay đổi độ ưu tiên của các toán tử
- Với toán tử 2 ngôi: toán tử bên trái là ẩn, toán tử bên phải là đối số

  Do đó số tham số bằng số toán hạng - 1

- **Danh sách các toán tử không thể nạp chồng:**

.	.*	::	?:	sizeof

- **Cách gọi hàm toán tử:**

  * Dùng như cú pháp thông thường của phép toán

  Ví dụ:

```
Phanso  a,b,c;
 c = a + b;
```

  * Dùng như hàm thành phần của đối tượng

  Ví dụ:

```
Phanso   a,b,c;
c = a.operator + (b);
```

Chú ý:

Các hàm toán tử có thể là các thành viên của lớp, hoặc không là các thành viên của lớp.

Một hàm toán tử cài đặt như hàm không thành viên cần là một friend nếu hàm phải truy cập đến các thành viên private hoặc protected.

Ví dụ: Hàm toán tử + không là thành viên của lớp sophuc.

```
#include <iostream.h>
#include <conio.h>
class   sophuc
{
private:
   double   thuc, ao;
```

```
public:
   sophuc(double  t=0,double  a=0)
   {   thuc = t; ao = a;  }
   void In()
   {   cout<<"("<<thuc<<" , "<<ao<<")"; }
   friend   sophuc   operator + (sophuc, sophuc);
};
sophuc  operator + (sophuc  p, sophuc  q)
{
   sophuc Tmp;
   Tmp.thuc =  p.thuc + q.thuc;
   Tmp.ao = p.ao + q.ao;
   return Tmp;
}
```

4.2. Đa năng hóa các toán tử chèn dòng << và trích dòng >>

Hàm toán tử của toán tử << được đa năng hóa có nguyên mẫu hàm như sau:

```
ostream & operator << (ostream & <luồng>, <lớp>  &<đối tượng>);
```

Hàm toán tử của toán tử >> được đa năng hóa có nguyên mẫu hàm như sau:

```
istream  & operator >> (istream & <luồng>, <lớp>  &<đối tượng>);
```

Các hàm trên không là thành viên của lớp, chúng là các hàm **friend**.

Ví dụ: Đa năng hóa toán tử chèn dòng << và trích dòng >> trên lớp diem.

```
#include <iostream.h>
#include <conio.h>
class  diem
{
  private:
    float  x, y;
  public:
    diem();
    friend  ostream & operator << (ostream & out, diem & p);
    friend  istream & operator >> (istream & in, diem & p);
};
```

```
diem:: diem()
{ x = y = 0; }
ostream  & operator << (ostream & out, diem &p)
{  out << "(" << p.x << " , " << p.y << ")";
   return   out;
}
istream  & operator >> (istream & in, diem & p)
{  cout << "Toa do x = ";
    in >> p.x;
    cout << "Toa do y = ";
    in >> p.y;
    return   in;
}
```

```
void main()
{
  clrscr();
  diem  a;
  cin >> a;
  cout << "Diem " << a;
  getch();
}
```

Bài tập chương 4

1/ Nạp chồng toán tử toán tử nhập >>, xuất << , +, -, *, /, của lớp phân số.

2/ Nạp chồng toán tử + hai vector, tích vô hướng của 2 vector.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
class Phanso
{
   int ts, ms;
public:
   void Toigian();
   Phanso operator - ();
   Phanso operator + (Phanso p);
   Phanso operator - (Phanso p);
   Phanso operator * (Phanso p);
   Phanso operator / (Phanso p);
   friend ostream & operator <<(ostream & out, const Phanso &p);
   friend istream & operator >>(istream & in, Phanso & p);
};
//-----------------
ostream & operator << (ostream & out, const Phanso &p)
{
   out << p.ts << "/" << p.ms;
   return out;
}
```

```
//……
istream & operator >> (istream & in, Phanso & p)
{
   cout<<"Tu so: ";
   in >> p.ts;
   cout << "Mau so: ";
   in >> p.ms;
   return in;
}
//……..
void Phanso::Toigian()
{
   int a, b, d;
  a = abs(ts);
  b = abs(ms);
  while (a != b)
    if (a > b)
      a -= b;
    else
      b -= a;
   ts = ts/a; ms = ms/a;
};
```

```
Phanso Phanso::operator - ()
{ Phanso p;
            p.ts = -ts; p.ms = ms;
            p.Toigian();
            return p;
};
Phanso Phanso::operator + (Phanso p)
{
   Phanso q;
   q.ts = p.ts * ms + p.ms * ts;
   q.ms = p.ms * ms;
   q.Toigian();
   return q;
}
Phanso Phanso::operator - (Phanso p)
{
   Phanso q;
   q.ts = ts * p.ms - ms * p.ts;
   q.ms = ms * p.ms;
   q.Toigian();
   return q;
}
```

```
//.....
Phanso Phanso::operator * (Phanso p)
{
   Phanso q;
   q.ts = ts * p.ts;
   q.ms = ms * p.ms;
   q.Toigian();
   return q;
}
//
Phanso Phanso::operator / (Phanso p)
{
   Phanso q;
   q.ts = ts * p.ms;
   q.ms = ms * p.ts;
   q.Toigian();
   return q;
}
```

```
// Ham main
void main()
{
   Phanso p, q;
   clrscr();
   cout<<"Nhap phan so thu nhat: "; cin >> p;
   cout << p;
   cout<<"\nNhap phan so thu hai: "; cin >> q;
   cout << q << "\n";
   cout << p <<" + " << q << " = " << p+q << "\n";
   cout<<"Doi dau: "; cout << -(p+q) << "\n";
   cout << p <<" * "<< q << " = " << p*q << "\n";
   cout << p <<" / " << q <<" = " << p/q;// t;
   getch();
}
```

Back

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
class vecto
{
   private:
      int n;
      float *v;
   public:
            vecto operator + (vecto &);
            float operator * (vecto);
            friend ostream & operator << (ostream & out, const vecto &);
            friend istream & operator >> (istream & in, vecto & );
}
istream & operator >> (istream & in, vecto &a)
{
   do
   {   cout << "Nhap so chieu cua vector: ";
       in >> a.n;
   } while(a.n <= 0);
   for(int i=0; i<a.n; i++)
      {
            cout << "Nhap thanh phan thu "<< i+1 << ": ";
            in >> a.v[i];
      }
   return in;
}
```

```
ostream & operator << (ostream & out, const vecto &a)
{
   out << "(";
   for(int i=0; i < a.n - 1; i++)
            out << a.v[i] << " , ";
   out << a.v[a.n-1] << ")";
   return out;
}
vecto vecto:: operator + (vecto &a)
{
   vecto c;
   c.n = n;
   for(int i=0; i<n; i++)
      c.v[i] = v[i] + a.v[i];
   return c;
}
float vecto::operator * (vecto a)
{
   float tg = 0;
   for(int i=0; i<n; i++)
      tg += v[i] * a.v[i];
   return tg;
}
```

```
//-------------
void main()
{
   vecto a,b,c;
   clrscr();
   cin >> a;
   cin >> b;
   cout << "Vector thu nhat: " << a;
   cout <<"\nVector thu hai: " << b;
   c = a + b;
   cout << "\nVector tong: " << c;
   cout<<"\nTich vo huong: "<<a*b;
   getch();
}
```

Back