Lập trình hướng đối tượng và C++

Bài 5: Định nghĩa chồng toán tử

TS. Nguyễn Hiếu Cường
Bộ môn CNPM, Khoa CNTT, Trường Đại học GTVT
Email: cuonggt@gmail.com

Hà Nội, 2022

Nội dung chính

1. Giới thiệu môn học
2. Các khái niệm cơ bản
3. Hàm trong C++
4. Lớp và đối tượng
5. **Định nghĩa chồng toán tử**
6. Hàm tạo và hàm huỷ
7. Dẫn xuất và thừa kế
8. Tương ứng bội
9. Khuôn hình (templates)

Tại sao cần định nghĩa chồng toán tử?

- Trong các lớp thường có các phương thức thực hiện các phép toán: cộng, nhân, ...
- Tuy nhiên, sử dụng không thuận tiện, ví dụ:

```
PS p1, p2, p3, p4, x;
...
// Tính x là tổng bốn phân số:
x = p1.cong(p2.cong(p3.cong(p4)));
```

- Định nghĩa chồng toán tử giúp cho sử dụng tương tự như các phép toán đã biết:

```
x = p1 + p2 + p3 + p4;
```

Lớp Số phức

// Phương thức cong()

```
class SP {
  float a,b;
public:
  SP cong(SP u2);  // hàm cộng
};
SP SP::cong(SP u2) {
    SP u;
    u.a= this->a + u2.a;
    u.b= this->b + u2.b;
    return u;
}
int main()
{
  SP u, u1, u2;
  ...
  u= u1.cong(u2);
}
```

// Phương thức toán tử +

```
class SP {
  float a,b;
public:
  SP operator+(SP u2);  // toán tử +
};
SP SP::operator+(SP u2) {
    SP u;
    u.a= this->a + u2.a;
    u.b= this->b + u2.b;
    return u;
}
int main()
{
  SP u, u1, u2;
  ...
  u= u1 + u2;
}
```

Phương thức toán tử

- Vấn đề: Có thể thực hiện các phép toán theo cách quen thuộc?
- Giải pháp: Mỗi toán tử được xây dựng giống các phương thức, nhưng thêm từ khóa **operator**
- Lưu ý về số đối:
  - Số đối tường minh trong phương thức toán tử ít hơn số ngôi 1

Một số loại toán tử

- Toán tử 1 ngôi: Trong phương thức sẽ **kh□ng c□ □□i**

  - (đảo dấu)

  [ ] (lấy giá trị mảng)

  ++ (tăng 1), **--** (giảm 1)

  Phân biệt các dạng tiền tố, hậu tố dùng đối giả (int)

- Toán tử 2 ngôi: Trong phương thức cần dùng **1 □□i** tường minh

  + - * /

  =

Ví dụ (phương thức toán tử)

```
class SP
{
  double a;     // Phần thực
  double b;     // Phần ảo
public:
  SP operator-();       // 1 ngo^i
  SP operator-(SP s);   // 2 ngo^i
  SP operator+(SP s);
} ;
SP  SP:: operator-(){
  SP  u ;
  u.a = - this->a ;
  u.b = - this->b ;
  return u;
}
SP  SP::operator-(SP s) {
  SP u;
  u.a = a - s.a;
  u.b = b - s.b;
  return u;
}
```

```
SP  SP::operator+(SP s) {
  SP u;
  u.a = a + s.a;
  u.b = b + s.b;
  return u;
}

int main()
{
  SP   u, v, s, p, q;
  ...
  s = u + v;
  p = u - v;    // 2 ngo^i
  u = -v;       // 1 ngo^i
  int a, b=5, c=7;
  a= b+c;
}
```

Ví dụ

Xây dựng lớp Phân số

- Dữ liệu là ts, ms (private)
- Các phương thức (public):

1. Nhập, xuất
2. Toán tử **+** hai phân số
3. Toán tử ***** hai phân số
4. Toán tử **++** để tăng phân số thêm 1 đơn vị (định nghĩa cả hai dạng prefix và postfix)

Sau đó xây dựng hàm main() sử dụng lớp trên

Quy định về định nghĩa toán tử

- Không thể định nghĩa một toán tử mới
  (Chỉ có thể định nghĩa chồng một toán tử đã sẵn có)
- Các toán tử phải được định nghĩa tường minh
  (Định nghĩa toán tử + không có nghĩa là toán tử += được định nghĩa)
- Các toán tử định nghĩa phải bảo toàn **số ngôi** nguyên thủy
- Các toán tử định nghĩa nên bảo toàn **ý nghĩa** nguyên thủy

Một số lưu ý về định nghĩa toán tử

- Hàm toán tử có thể là
  - Hàm thành phần (Phương thức) của lớp
  - Hàm bạn của lớp
- Đa phần các toán tử có thể định nghĩa theo cả hai cách trên
- Một số toán tử phải được định nghĩa:
  - Là hàm bạn: << , >>
  - Là phương thức: ++, = , [ ]

Những toán tử có thể định nghĩa chồng

- Hầu hết toán tử trong C++ đều có thể định nghĩa chồng

Operators that can be overloaded							
+	-	*	/	%	^		
!	=	<	>	+=	-=	*=	
/=	%=	^=			<<	>>	
==	!=	<=	>=	&&	||	++	--
[]	()	new	delete	new[]	delete[]		

- Chỉ có một số rất ít toán tử không thể định nghĩa chồng

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

Hàm toán tử là phương thức của lớp

- Khi đó *this* là đối ẩn thể hiện toán hạng thứ nhất
  - Số đối bằng số ngôi – 1
  - Toán tử 1 ngôi không có đối
  - Toán tử 2 ngôi chỉ có 1 đối

- Hạn chế?

  Không xử lý được khi hai toán hạng thuộc hai lớp khác nhau

```
PS  p ;
cout <<  p  ;
```

Hàm toán tử là hàm bạn của lớp

- Định nghĩa các toán tử vào/ra

```
friend void operator>> (istream& is, A &a);
friend void operator<< (ostream& os, A &a);
```

Hoặc:

```
friend istream& operator>> (istream& is, A &a);
friend ostream& operator<< (ostream& os, A &a);
```

Ví dụ

1. Viết lại các lớp Phân số, trong đó:

- Định nghĩa các phép toán +, * để cộng, nhân hai phân số.
- Định nghĩa phép toán ++ để tăng phân số thêm 1 (theo hai cách: tăng trước và tăng sau).
- Thay các hàm nhap(), xuat() bằng các toán tử >> và <<.

2. Viết lớp SP (Số phức):

- Dữ liệu: a, b (thực, ảo)
- Định nghĩa các phương thức (toán tử):
  - << để in số phức dưới dạng a+b*i*
  - >> để nhập số phức
  - + để cộng hai số phức

Toán tử gán

- Khi nào cần định nghĩa chồng toán tử gán?
  - Trong mỗi lớp đều có một toán tử gán mặc định
  - Nếu trong thành phần dữ liệu của lớp
    - không có con trỏ hoặc biến tham chiếu → toán tử gán mặc định
    - có con trỏ hoặc biến tham chiếu → phải định nghĩa toán tử gán
- Toán tử gán có mấy ngôi?
  - Là toán tử hai ngôi

    a = b;
  - Khi định nghĩa chồng toán tử gán cần phải:
    - Định nghĩa là hàm thành phần của lớp
    - Có một tham số trong hàm toán tử

Xây dựng toán tử gán

- Định nghĩa toán tử gán cho lớp A

```
void operator= (const A& d);    //nếu chỉ cần a= b
A& operator= (const A& d);      //nếu gán liên tiếp a= b =c
```

```
class PS {
  int ts, ms;
public:
  void operator=(PS& p) {
    ts= p.ts;
    ms= p.ms;
  }
};
int main(){
  PS p1, p2;
  ...
  p2= p1;
}
```

```
class PS {
  int ts, ms;
public:
  PS& operator=(PS& p) {
    this->ts= p.ts;
    this->ms= p.ms;
    return (*this);
  }
};
int main() {
  PS p1, p2, p3;
  ...
  p3= p2= p1;
}
```

Ví dụ (toán tử gán)

```
class DT  // Lớp ĐA THỨC
{
    int n;
    float *a;   // co con tro
public:
    DT()  { }
    DT(int n1);
    ~DT()  { delete a; }
    void hien();
    DT& operator=(DT& d);
};
DT::DT(int n1)
{
    n= n1;
    a= new float[n+1];
    for (int i=0; i<=n; ++i)
    {
        cout<<"a["<<i<<"]= ";
        cin>>a[i];
    }
}
```

```
void DT::hien() {
    for (int i=0; i<n; ++i)
        cout<<a[i]<<"  ";
}
DT& DT::operator=(DT& d)
{
    n= d.n;
    a= new float[n+1];
    for (int i=0; i<=n; ++i)
       a[i]= d.a[i];
    return (*this);
}
void main()
{
    DT x(3);
    cout<<endl<<"x:"<<endl; x.hien();
    DT y;
    y = x;  // Su dung toan tu gan
    cout<< endl <<"y:"<< endl;
    y.hien();
}
```

Ví dụ (toán tử vào/ra)

```
#include <iostream>
using namespace std;
class DT
{
    int n;
    float *a;
public:
    friend ostream& operator <<
         (ostream& os, DT& d);
    friend istream& operator >>
         (istream& is, DT& d);
    DT& opeartor=(DT& d);
};

ostream& operator <<(ostream& os,
                          DT& d)
{
    for (int i=0; i<=d.n; ++i)
      os<<d.a[i]<<"  ";
    return os;
}
```

```
istream& operator >>(istream& is,
                               DT& d) {
    cout<<"Bac da thuc: ";
    cin>>d.n;
    d.a= new float[d.n +1];
    for (int i=0; i<=d.n; ++i)
      is>>d.a[i];
    return is;
}
DT& DT::opeartor=(DT& d) { ... }
int main() {
    DT d1, d2;
    cout<<endl<<"Nhap da thuc:"<<endl;
    cin>>d1;
    cout<<endl<<"Da thuc:"<<endl;
    cout<<d1;
    d2 = d1;
    cout<<d2;
}
```

Ví dụ (các phép toán khác)

```
// Lop Da thuc
#include <iostream>
using namespace std;
class  DT
{
 int n;        // Bac da thuc
 double *a;
public:
 friend ostream& operator<<(ostream& os,
                            const DT &d);
 friend istream&  operator>>(istream&is,
                                DT &d);

 // Cong hai da thuc
 DT operator+(const DT &d2);

 // Tinh gia tri da thuc
 double operator^(const double &x);
 double F(DT d,double x);
};
```

```
DT DT::operator+(const DT &d2){
  DT d;
  int k,i;

  k = n > d2.n ? n : d2.n ;
  d.a = new double[k+1];

  for (i=0; i<=k ; ++i)
    if (i<=n && i<=d2.n)
      d.a[i] = a[i] + d2.a[i];
    else if (i<=n)
      d.a[i] = a[i];
    else
      d.a[i] = d2.a[i];

  i=k;
  while(i>0 && d.a[i]==0.0)
    --i;
  d.n = i;
  return d ;
}
```

Ví dụ (tiếp)

```
double DT::operator^(const double &x)
{
  double s=0.0 , t=1.0;
  for (int i=0 ; i<= n ; ++i)
  {
    s += a[i]*t;
    t *= x;
  }
  return s;
}

double F(DT d,double x) {
  double s=0.0 , t=1.0;
  int n;
  n = int(d[-1]);
  for (int i=0; i<=n; ++i)
  {
    s += d[i]*t;
    t *= x;
  }
  return s;
}
```

```
ostream& operator<< (ostream& os, const
                                    DT&d)
{
  os << "Cac he so da thuc: " ;
  for (int i=0 ; i<= d.n ; ++i)
    os <<  d.a[i] <<"  " ;
  return os;
}

istream&  operator>> (istream& is, DT &d)
{
  cout << "Bac da thuc: " ;
  cin >> d.n;
  d.a = new double[d.n+1];
  cout << "Nhap cac he so da thuc:\n" ;
  for (int i=0 ; i<= d.n ; ++i)
  {
    cout << "He so bac " <<i<< " = ";
    is >> d.a[i] ;
  }
  return is;
}
```

Ví dụ (tiếp)

```
void main()
{
  DT p, q, r, f;
  double x1, x2, g1, g2;

  cout <<"\nNhap da thuc p\n " ;
  cin >> p;
  cout << "\nDa thuc p " << p ;
  cout <<"\nNhap da thuc q\n " ;
  cin >> q;
  cout << "\nDa thuc q " << q ;
  r = p+q;
  cout << "\nDa thuc r " << r ;
  f = p - q;
  cout << "\nNhap so thuc x1: " ; cin >> x1;
  cout << "\nNhap so thuc x2: " ; cin >> x2;
  g1 = f^x1;
  g2 = F(f,x2);
  cout  << "\n f("<<x1<<") = " << g1;
  cout  << "\n f("<<x2<<") = " << g2;
}
```

Tóm tắt

- Các quy định trong định nghĩa chồng toán tử
- Những toán tử có thể định nghĩa chồng
- Những toán tử phải là phương thức của lớp: (), [], =
- Những toán tử phải là hàm bạn : <<, >>
- Định nghĩa một số loại toán tử quan trọng

Bài tập

1. Xây dựng lớp SP (số phức)
   - Dữ liệu (private): phần thực, phần ảo
   - Các phương thức toán tử (public): + để cộng 2 số phức
   - Toán tử >> để nhập điểm
   - Toán tử << để xuất điểm ra màn hình
   - Toán tử gán

2. Xây dựng lớp DIEM biểu diễn điểm trong không gian 2 chiều, trong đó có dữ liệu (private) gồm hoành độ, tung độ và các phương thức (public) sau:
   - Toán tử >> để nhập điểm
   - Toán tử << để xuất điểm ra màn hình
   - Toán tử * để "nhân" hai điểm theo công thức: $(x_1, y_1) * (x_2, y_2)$ là điểm có tọa độ (x_1*x_2, y_1*y_2)
   - Phương thức tính khoảng cách giữa hai điểm

Bài tập

3. Xây dựng lớp DT (Đa thức), trong đó:

- Các thuộc tính

```
int n;          // là bậc của đa thức
float *a;       // là con trỏ xác định vùng bộ nhớ chứa các hệ số
```

- Phương thức nhap() để nhập các hệ số của đa thức
- Phương thức xuat() để in các hệ số của đa thức ra màn hình
- Phương thức gia_tri(t) để tính giá trị của đa thức tại x = t.

4. Xây dựng lớp DT (Đa thức)

- Dữ liệu: n, *a
- Các phương thức toán tử: >>, <<, =
- Phương thức gia_tri(t) để tính giá trị đa thức tại x = t;