Hàm dựng và Hàm hủy

Lập trình hướng đối tượng

Hồ Tuấn Thanh – htthanh@fit.hcmus.edu.vn

Toán tử

Các toán tử thường gặp

- Gán bằng: a=b;
- Cộng, trừ, nhân, chia: c=a+b; d=a%b;
- So sánh: a==b; c<=a;
- +=, -=, *=, /=: a+=b;
- ++, --: a++; ++a; b--; --b;
- Nhập, xuất: cin>>a; cout<<b;
- Ép kiểu float x=10.7; int n=(int)x;
- []: int x=a[i];

Định nghĩa lại toán tử

- Các lớp tự định nghĩa: chưa có sẵn các toán tử
- Muốn sử dụng toán tử: phải tự cài đặt toán tử
- Cài đặt 1 toán tử được gọi là định nghĩa lại toán tử đó (operator overloading)
- Về bản chất, operator cũng là một hàm.
  - VD: **c=a+b** tương đương với **c=a.operator+(b);**
  - VD: **c+=b** tương đương với **c.operator+=(b);**
- Như vậy, đến lúc này, chúng ta đã cài đặt 4 loại hàm: hàm dựng, hàm hủy, hàm toán tử và các hàm khác.

.

Các toán tử có thể overload

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

Qui tắc bất thành văn

- Chỉ cài đặt các toán tử có ý nghĩa với lớp.
- Tránh thay đổi ý nghĩa nguyên thủy của toán tử đó.
- Các cặp toán tử có cùng chức năng, ví dụ x=x+y và x+=y phải được viết cùng nhau và có cùng chức năng.

=

Toán tử gán bằng

```
void main()
{
    PhanSo a(1,2);
    PhanSo b;
    b=a;
}
```

Goi toan tu gan bang

```
PhanSo& PhanSo::operator=(const PhanSo &p)
{
    if(this==&p)
        return *this;
    this->tu=p.tu;
    this->mau=p.mau;
    return *this;
}
```

void, PhanSo: sai

Truong hop: a=a;

Giong ham dung sao chep

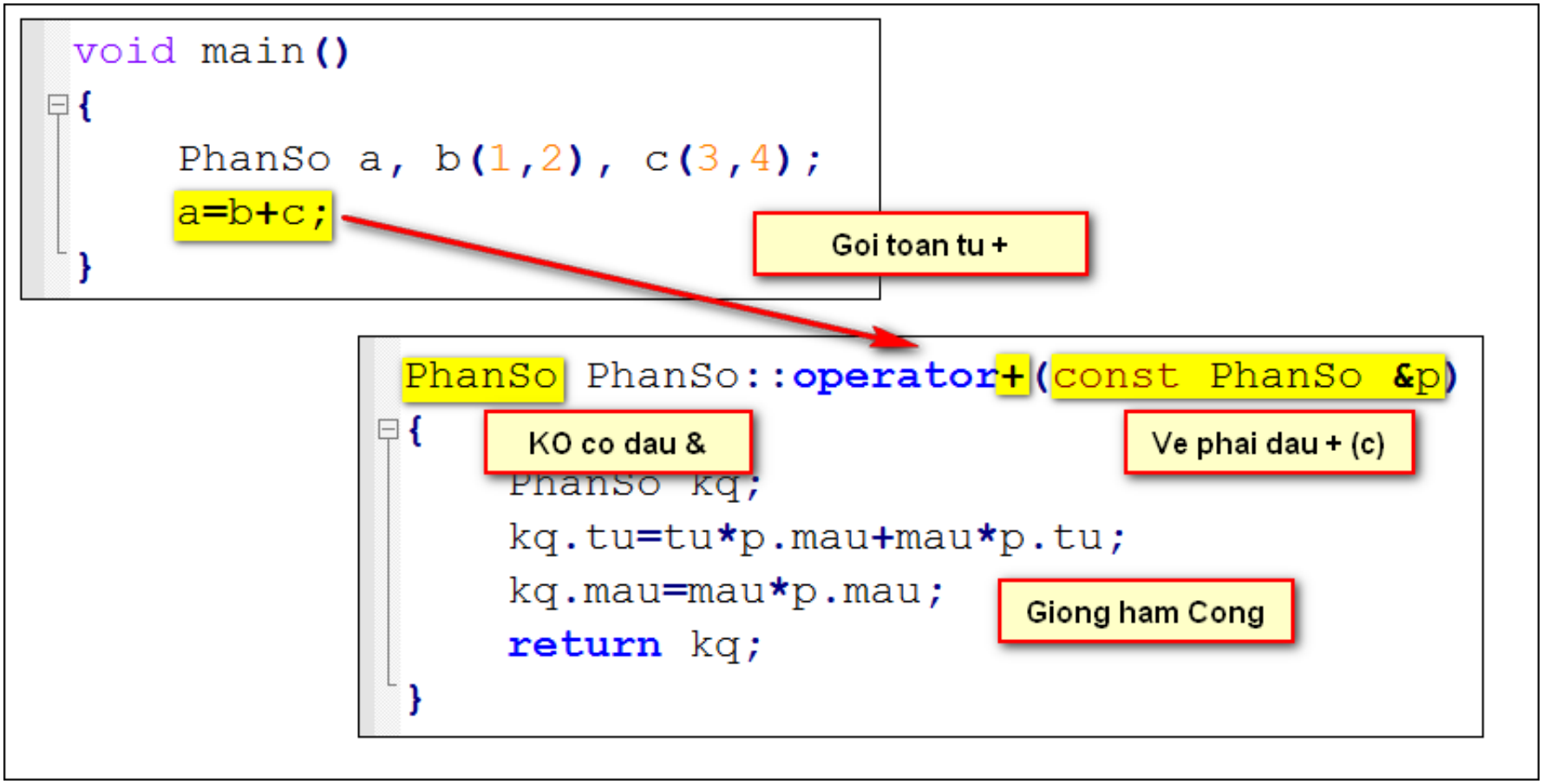
Nho cau lenh nay

Lưu ý

- Toán tử gán bằng được gọi khi thực hiện gán bằng giữa 2 biến (b=a)
- Cài đặt gần giống hàm dựng sao chép
- Thêm kiểm tra **this==&p** để tránh trường hợp gán a=a
- Kiểu trả về **TenLop&** (VD: PhanSo&) và nhớ return *this

+, -, *, /

Toán tử +


void main()
{
PhanSo a, b(1,2), c(3,4);
a=b+c;
}
Goi toan tu +
PhanSo PhanSo::operator+(const PhanSo &p)
{
KO co dau &
Ve phai dau + (c)
PhanSo kq;
kq.tu=tu*p.mau+mau*p.tu;
kq.mau=mau*p.mau;
Giong ham Cong
return kq;
}

Lưu ý

- Không có dấu & ở kiểu trả về
- Tham số: vế phải toán tử
- Không cộng dồn vào đối tượng gọi toán tử (this)

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

Toán tử ==

```
void main()
{
    PhanSo a(1,2), b(3,4);
    int kq=(a==b);
    if(kq==1)
        cout<<"a = b";
    else
        cout<<"a != b";
}
```

Goi toan tu so sanh bang

```
int PhanSo::operator==(const PhanSo &p)
{
    int d=tu*p.mau-mau*p.tu;
    if(d==0)
        return 1;
    else
        return 0;
}
```

bool hoac int

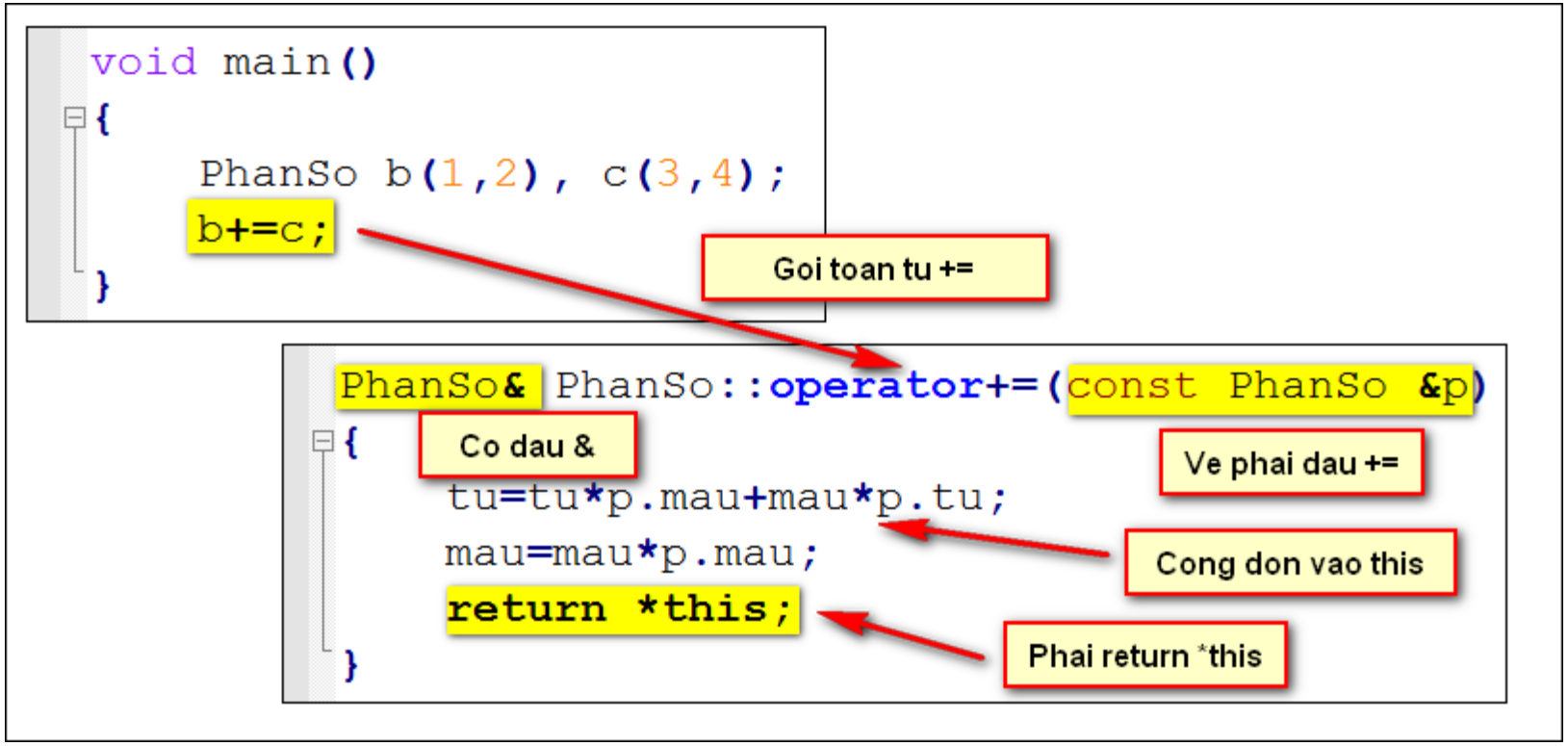
Ve phai dau ==

Lưu ý

- Trả về kiểu int (1, 0) hoặc bool (true, false)
- Tham số: vế phải toán tử

+=, -=, *=, /=

Toán tử +=


void main()
{
PhanSo b(1,2), c(3,4);
b+=c;
}
Goi toan tu +=
PhanSo& PhanSo::operator+=(const PhanSo &p)
{
Co dau &
Ve phai dau +=
tu=tu*p.mau+mau*p.tu;
mau=mau*p.mau;
Cong don vao this
return *this;
Phai return *this
}

Lưu ý

- Trả về có dấu &
  - PhanSo hoặc void: SAI
- Tham số: vế phải toán tử
- Cộng/trừ/nhân/chia dồn vào this
- Nhớ có return *this

>>, <<

To

```
void main()
{
    PhanSo a;
    cin>>a;
}
```

```
// Khai bao lop PhanSo
class PhanSo
{
private:
    int tu, mau;
};

// Khai bao operator>>, o ngoai lop
istream& operator>>(istream& is, PhanSo& p);

// Cai dat operator>>
istream& operator>>(istream& is, PhanSo& p)
{
    cout<<"Nhap tu: ";
    // cin>>p.tu;
    is>>p.tu;
    cout<<"Nhap mau: ";
    // cin>>p.mau;
    is>>p.mau;
    return is;
}
```

VTrai: cin => istream
VPhai: a => PhanSo
Phai cai dat o ngoai lop PhanSo

VTrai

VPhai

Cai dat giong ham Nhap

Thay cin bang is

LOI: truy xuat private cua PhanSo

To

```
void main()
{
    PhanSo a;
    cin>>a;
}
```

```
class PhanSo
{
private:
    int tu, mau;
public:
    // Mang khai bao operator>> bo trong lop
    // Them tu khoa friend o dang truoc
    // operator>> chi la ban, ko thuoc lop PhanSo
    friend istream& operator>>(istream& is, PhanSo& p);
};

// Cai dat operator>>
istream& operator>>(istream& is, PhanSo& p)
{
    cout<<"Nhap tu: ";
    // cin>>p.tu;
    is>>p.tu;
    cout<<"Nhap mau: ";
    // cin>>p.mau;
    is>>p.mau;
    return is;
}
```

VTrai: cin => istream
VPhai: a => PhanSo
Phai cai dat o ngoai lop PhanSo

Khai bao trong lop
Them tu khoa friend

VTrai

VPhai

Cai dat giong ham Nhap

Thay cin thanh is

friend => truy xuat duoc private

Lưu ý

- Kiểu trả về istream & (nhớ có dấu &)
- Tham số 1: istream &
- Tham số 2: PhanSo (có dấu &, vì trong operator thay đổi giá trị tu, mau cua tham số)
- Nhớ dùng từ khóa friend khi khai báo

To

```
void main()
{
    PhanSo a;
    cout<<a;
}
```

```
// Khai bao lop PhanSo
class PhanSo
{
private:
    int tu, mau;
};

// Khai bao operator<<, o ngoai lop
ostream& operator<<(ostream& os, const PhanSo& p);

// Cai dat operator<<
ostream& operator<<(ostream& os, const PhanSo& p)
{
    // cout<<p.tu<"/"<<p.mau;
    os<<p.tu<"/"<<p.mau;
    return os;
}
```

VTrai: cout => ostream
VPhai: a => PhanSo
Phai cai dat ngoai lop PhanSo

VTrai

VPhai

Cai dat giong ham Xuat
Thay cout bang os

LOI: Truy xuat private lop PhanSo

```
void main()
{
    PhanSo a;
    cout<<a;
}
```

```
// Khai bao lop PhanSo
class PhanSo
{
private:
    int tu, mau;
public:
    // Mang khai bao operator<< bo trong lop
    // Them tu khoa friend o dang truoc
    // operator<< chi la ban, ko thuoc lop PhanSo
    friend ostream& operator<<(ostream& os, const PhanSo& p);
};

// Cai dat operator<<
ostream& operator<<(ostream& os, const PhanSo& p)
{
    // cout<<p.tu<"/"<<p.mau;
    os<<p.tu<"/"<<p.mau;
    return os;
}
```

VTrai: cout => ostream
VPhai: a => PhanSo
Phai cai dat ngoai lop PhanSo

Khai bao trong lop
Them tu khoa friend

VTrai

VPhai

Cai dat giong ham Xuat
Thay cout bang os
friend: truy xuat private lop PhanSo

Lưu ý

- Kiểu trả về ostream & (nhớ có dấu &)
- Tham số 1: ostream &
- Tham số 2: PhanSo
- Nhớ dùng từ khóa friend khi khai báo

Từ khóa friend

- Dùng để khai báo một hàm/một lớp là bạn của một lớp khác
- BẠN: không phải bà con thân thích, không phải là thành viên của lớp
- BẠN: có thể truy cập private của lớp mà không cần sử dụng các hàm Lay, Gan
- BẠN: 1 chiều. A friend B:
  - A truy xuất private của B
  - B KHÔNG THỂ truy xuất private của A

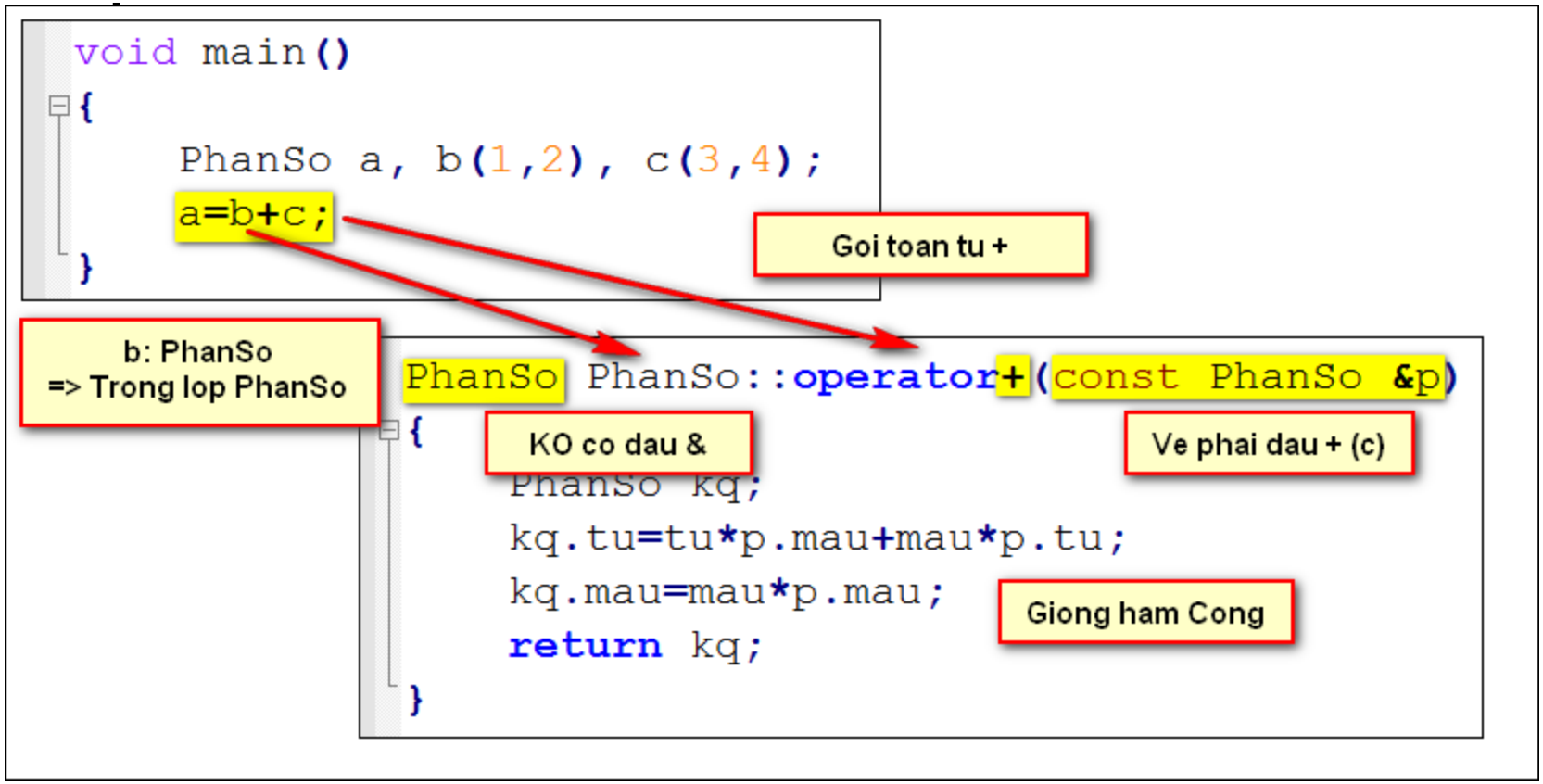
Từ khóa friend

- Tại sao phải khai báo hàm là friend
  - Để truy xuất private của lớp
  - VD: truy xuất trực tiếp tu, mau (private) mà không cần sử dụng các hàm LayTu, LayMau, GanTu, GanMau (public)
- Tại sao không khai báo operator<<, >> ở trong lớp
  - Không thể: vì vế trái của << và >> là cout và cin (không phải PhanSo)

Hai cách cài đặt toán tử

Cài đặt trong lớp

- Chọn lớp nào???
- Dựa vào về trái của toán tử



Cài đặt ngoài lớp

- Nên khai báo friend trong lớp

```
class PhanSo
{
private:
public:
    friend PhanSo operator+(const PhanSo &p1, const PhanSo &p2);
};

PhanSo operator+(const PhanSo &p1, const PhanSo &p2)
{
    PhanSo kq;
    kq.tu=p1.tu*p2.mau+p1.mau*p2.tu;
    kq.mau=p1.mau*p2.mau;
    return kq;
}
```

De truy xuat duoc private tu, mau

KO co PhanSo::

Ve trai dau +

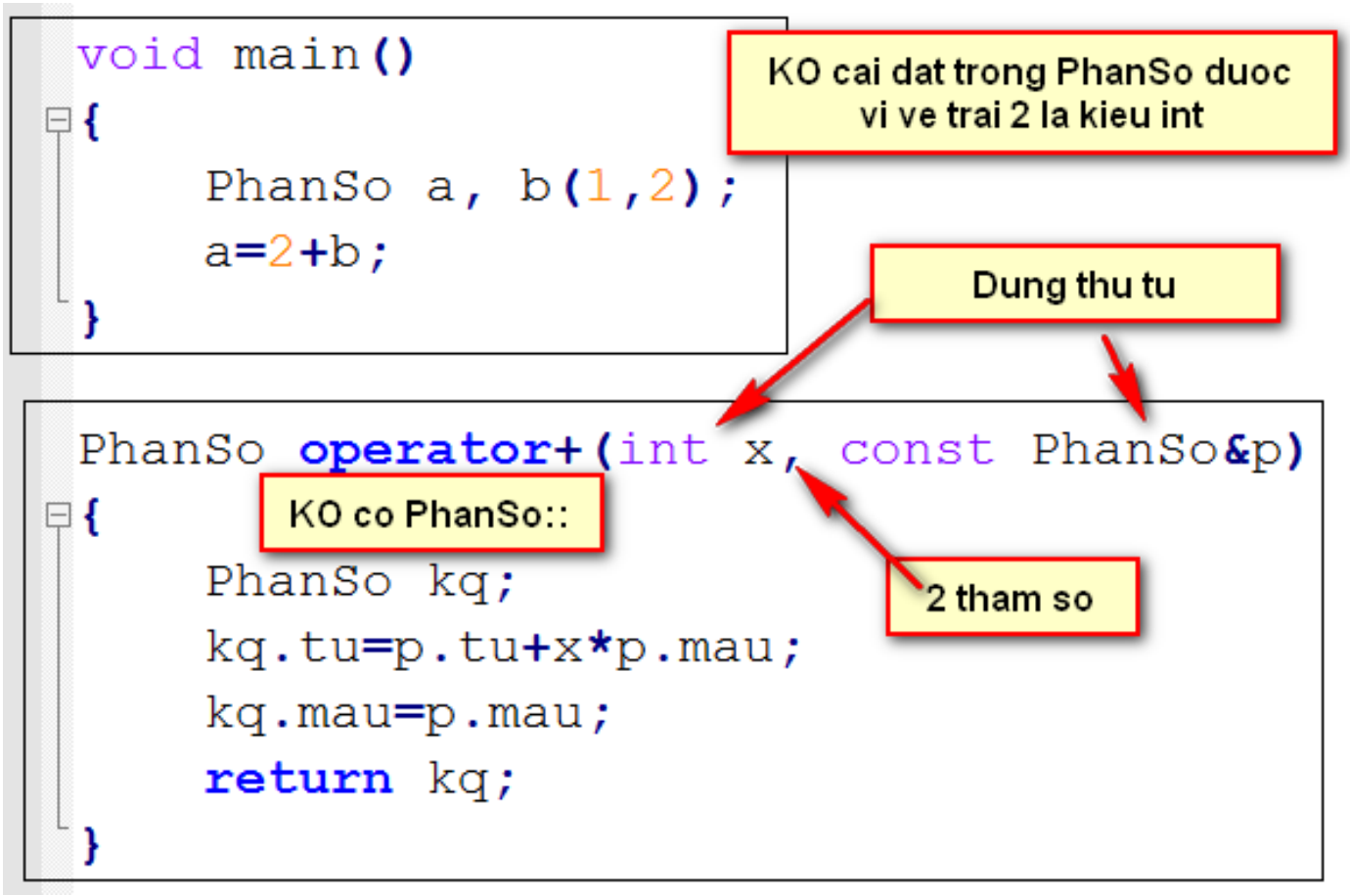
Ve phai dau +

Co 2 tham so

Lưu ý

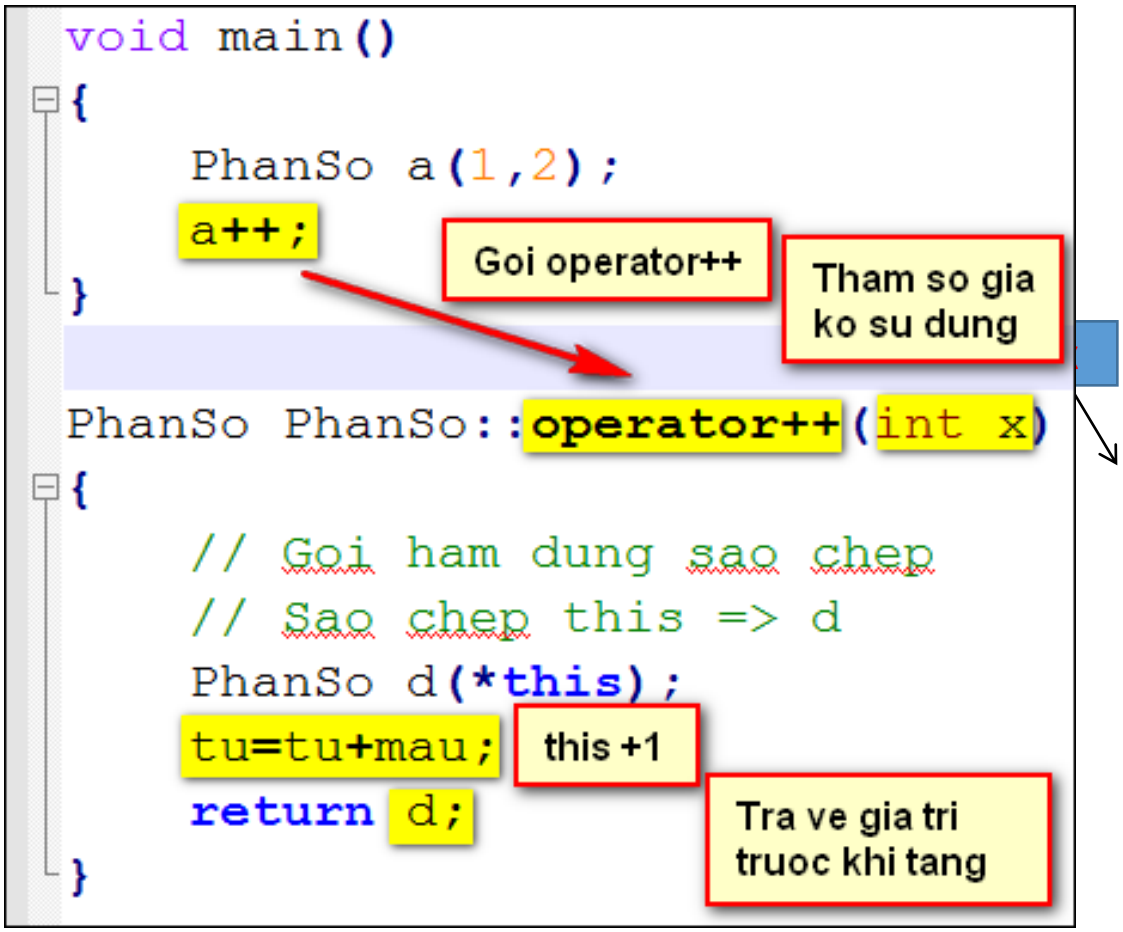
- Đa số toán tử đều có thể cài đặt bằng cả 2 cách
  - Chọn 1 cách để cài đặt thôi
- Một số toán tử bắt buộc phải CÀI ĐẶT TRONG LỚP. VD: Toán tử gán bằng (=)
- Một số trường hợp bắt buộc phải CÀI ĐẶT NGOÀI LỚP. VD: <<, >>
  - Lí do: vế trái không thuộc lớp mà ta có thể viết code

BẮT BUỘC cài đặt ngoài lớp

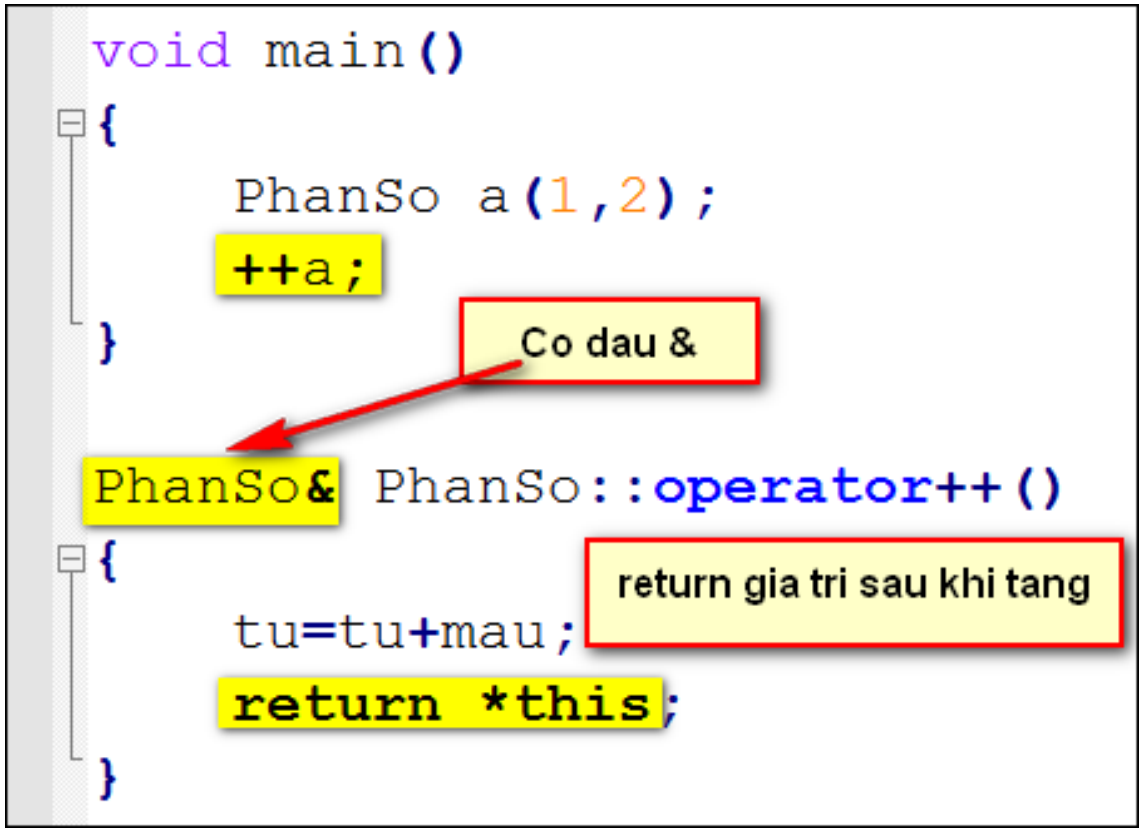


++, --

Toán tử a++


void main()
{
PhanSo a(1,2);
a++;
}
Goi operator++
Tham so gia ko su dung
PhanSo PhanSo::operator++(int x)
{
// Goi ham dung sao chep
// Sao chep this => d
PhanSo d(*this);
tu=tu+mau;
this +1
return d;
Tra ve gia tri truoc khi tang
}

Toán tử ++a


void main()
{
PhanSo a(1,2);
++a;
}
Co dau &
PhanSo& PhanSo::operator++()
{
tu=tu+mau;
return gia tri sau khi tang
return *this;
}

Lưu ý

- Có 2 toán tử ++:
  - Pre-increment: ++a, trả về giá trị sau khi tăng
  - Post-increment: a++, trả về giá trị trước khi tăng, có tham số giả
- Có 2 toán tử --:
  - Pre-increment: --a, trả về giá trị sau khi giảm
  - Post-increment: a--, trả về giá trị trước khi giảm , có tham số giả

Tóm tắt

- Trong một lớp, muốn sử dụng toán tử, phải cài đặt (overload) toán tử đó
- Có 2 cách cài đặt toán tử: trong lớp và ngoài lớp
- Từ khóa friend thường dùng khi muốn một hàm có thể truy xuất private của một lớp