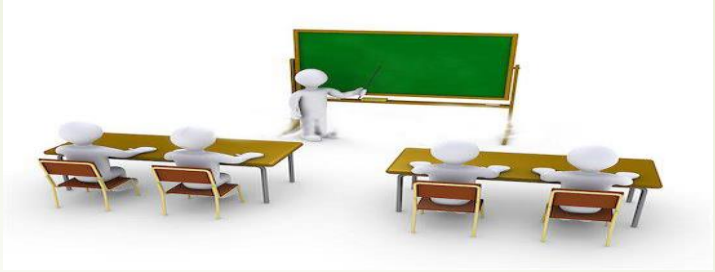


Nguyễn Việt Nam Sơn
nvnamson@gmail.com

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Operator Overloading
Function Overloading

NỘI DUNG BÀI HỌC

1. OPERATOR OVERLOADING
2. FUNCTION OVERLOADING
3. CHIA SẺ KINH NGHIỆM LẬP TRÌNH
4. BÀI TẬP ỨNG DỤNG TẠI LỚP
5. BÀI TẬP VỀ NHÀ

1. OPERATOR OVERLOADING

GIỚI THIỆU VẤN ĐỀ

Đối với những kiểu dữ liệu có sẵn (số nguyên, số thực) thì ta có thể tự do sử dụng các phép toán tử bởi vì ngôn ngữ lập trình đã cung cấp sẵn cho ta

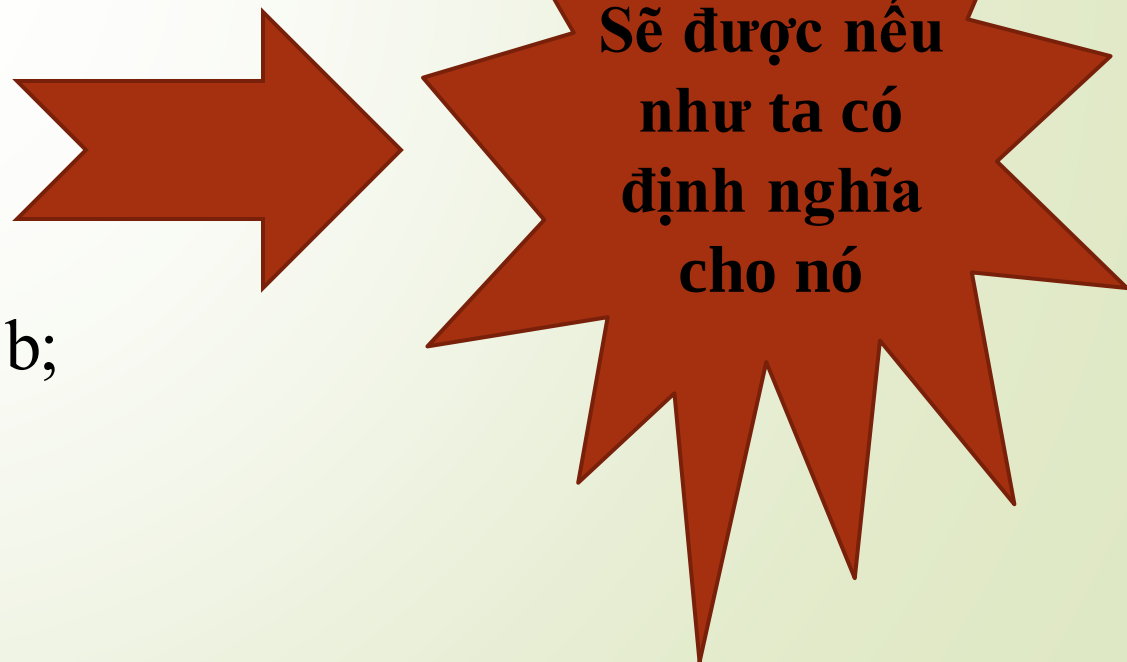
```
int main()
{
    int a = 6, b = 9;
    cout << a + b;
    b = ++a;
    cout << "a = " << a << " & b = " << b;
    return 0;
}
```

1. OPERATOR OVERLOADING

GIỚI THIỆU VẤN ĐỀ

Vậy còn đối với những kiểu dữ liệu không có sẵn mà do ta tự tạo ra (PhanSo, SoPhuc) thì có thể làm được như vậy không ?

```
int main()
{
        PhanSo a(1, 2), b(3, 4); // Khởi tạo
        cout << a + b;
        b = ++a;
        cout << "a = " << a << " & b = " << b;
        return 0;
}
```



1. OPERATOR OVERLOADING

GIỚI THIỆU VẤN ĐỀ

Nếu yêu cầu đặt ra là tính tổng của 4 phân số a, b, c, d và kết quả trả ra là 1 phân số tổng, chúng ta có những cách làm thế nào để giải quyết yêu cầu ?

Cách 1: Xây dựng phương thức cộng 2 phân số (Làm demo tại lớp)

```
int main()
{
    PhanSo a(1, 2), b(3, 4), c(5, 6), d(7, 8); // Khởi tạo
    PhanSo Tong = a.Cong(b);
    Tong = Tong.Cong(c);
    Tong = Tong.Cong(d);
    return 0;
    // Hoặc có thể làm trên 1 dòng: PhanSo Tong = a.Cong(b).Cong(c).Cong(d);
}
```

1. OPERATOR OVERLOADING

GIỚI THIỆU VẤN ĐỀ

Cách 2: Chúng ta có thể làm việc linh động hơn bằng cách sử dụng toán tử:

```
int main()
{
    PhanSo a(1, 2), b(3, 4), c(5, 6), d(7, 8); // Khởi tạo
    PhanSo Tong = a + b + c + d; // Sử dụng toán tử cộng
    return 0;
}
```

1. OPERATOR OVERLOADING

GIỚI THIỆU VẤN ĐỀ

Qua ví dụ trên ta thấy việc sử dụng toán tử sẽ giúp cho công việc của chúng ta tiện lợi, dễ dàng hơn rất nhiều, chương trình nhìn gọn gàng, sáng sủa hơn

Đối với các kiểu dữ liệu do người lập trình định nghĩa thêm, ta có thể định nghĩa chồng các toán tử cho các kiểu dữ liệu mới này khiến cho máy tính vẫn hiểu được như bình thường.

1. OPERATOR OVERLOADING

TỔNG QUAN

Được biết đến với nhiều tên gọi khác nhau như:
Chồng toán tử.
Nạp chồng toán tử.
Quá tải toán tử.
Toán tử.

Toán tử là một phương thức đặc biệt của lớp đối tượng nhằm thể hiện trực quan khi được gọi thực hiện

LÀM QUEN

Xây dựng lớp PhanSo và thực hiện đi tính tổng của 2 phân số theo 2 cách:

Cách 1: Làm bằng phương thức bình thường
Cách 2: Làm bằng toán tử

1. OPERATOR OVERLOADING

LÀM QUEN

Cách 1: Ví dụ phương thức tính tổng 2 phân số

File: PhanSo.h

```
#pragma once
#include <iostream>
using namespace std;

class PhanSo
{
private:
	int TuSo, MauSo;
public:
	PhanSo Cong(PhanSo);
	PhanSo(void);
	~PhanSo(void);
};
```

File: PhanSo.cpp

```
#include "PhanSo.h"

PhanSo PhanSo::Cong(PhanSo x)
{
	PhanSo Tong;
	Tong.TuSo = TuSo * x.MauSo + MauSo * x.TuSo;
	Tong.MauSo = MauSo * x.MauSo;

	return Tong;
}
```

1. OPERATOR OVERLOADING

LÀM QUEN

Cách 2: Ví dụ toán tử tính tổng 2 phân số: Cộng 1 phân số với 1 phân số và kết quả trả về là 1 phân số.

File: PhanSo.h

```
1  // Thư viện sử dụng trong chương trình.
2  #pragma once
3  #include <iostream>
4  using namespace std;
5
6  // Khởi tạo cấu trúc lớp CPhanSo.
7  class CPhanSo
8  {
9  private:
10     int TuSo;
11     int MauSo;
12 public:
13     CPhanSo operator +(CPhanSo); // Định nghĩa toán tử: Phân số + phân số. Kết quả trả về là 1 phân số.
14
```

File: PhanSo.cpp

```
// Định nghĩa toán tử: Phân số + phân số. Kết quả trả về là 1 phân số.
CPhanSo CPhanSo ::operator+(CPhanSo x)
{
    CPhanSo Temp;
    Temp.TuSo = TuSo * x.MauSo + MauSo * x.TuSo;
    Temp.MauSo = MauSo * x.MauSo;
    return Temp;
}
```

1. OPERATOR OVERLOADING

LÀM QUEN

Ví dụ khác: Xây dựng operator + với cách thức xử lý như sau: Cộng 1 phân số với 1 số nguyên và kết quả trả về là 1 phân số.

File: PhanSo.h

```
1  // Thư viện sử dụng trong chương trình.
2  #pragma once
3  #include <iostream>
4  using namespace std;
5
6  // Khởi tạo cấu trúc lớp CPhanSo.
7  class CPhanSo
8  {
9  private:
10     int TuSo;
11     int MauSo;
12 public:
13     CPhanSo operator +(int); // Định nghĩa toán tử: Phân số + số nguyên. Kết quả trả về là 1 phân số.
14
```

121 %

File: PhanSo.cpp

```
// Định nghĩa toán tử: Phân số + số nguyên. Kết quả trả về là 1 phân số.
CPhanSo CPhanSo ::operator+(int x)
{
    CPhanSo Temp;
    Temp.TuSo = TuSo + x * MauSo;
    Temp.MauSo = MauSo;
    return Temp;
}
```

NHẬN XÉT TỔNG QUAN VÀ TÌNH HUỐNG ĐẶT RA

Qua 2 ví dụ trên, ta thấy được cách xử lý theo thứ tự giữa 1 phân số (lớp đối tượng hiện tại) với 1 tham số đằng sau tùy ý (có thể là 1 phân số khác hoặc 1 số nguyên).

Vậy nếu yêu cầu của đề bây giờ thứ tự sẽ là số nguyên + phân số (số nguyên đứng trước rồi sau đó đến toán tử rồi mới đến phân số) thì chương trình sẽ không thể chạy được, vậy phải giải quyết làm sao đây ?

1. OPERATOR OVERLOADING

TRẢ LỜI

Muốn xử lý được trường hợp này thì ta phải khai báo toán tử không thuộc về lớp đối tượng nữa, rồi truyền vào theo thứ tự tham số đúng như đề yêu cầu.

Nếu khai báo đối tượng không thuộc về lớp, mà muốn sử dụng được các thuộc tính của lớp (được cài đặt bởi từ khóa private) thì ta phải khai báo hàm bạn (**friend**) và để phương thức ở bên trong lớp. Còn nếu không dùng hàm bạn (**friend**) thì ta phải để phương thức ở bên ngoài lớp, truy xuất và xử lý thuộc tính nhờ vào Getter & Setter.

1. OPERATOR OVERLOADING

HÀM BẠN (FRIEND)

Từ khóa **friend**:

- Giúp cho hàm có thể truy xuất tới các thuộc tính của lớp và sử dụng chúng như bình thường mặc dù bản thân của hàm không thuộc về lớp đó. Lưu ý hàm friend sẽ khai báo ở bên trong lớp.

1. OPERATOR OVERLOADING

PHÂN TÍCH TỪNG TRƯỜNG HỢP

TH1: Khai báo toán tử nằm bên trong lớp => dùng hàm bạn (**friend**) .
Định nghĩa toán tử: Số nguyên + phân số. Kết quả trả về là 1 phân số.

File: PhanSo.h

```
// Thư viện sử dụng trong chương trình.
#pragma once
#include <iostream>
using namespace std;

// Khởi tạo cấu trúc lớp CPhanSo.
class CPhanSo
{
private:
    int TuSo;
    int MauSo;
public:
    friend CPhanSo operator +(int, CPhanSo); // Định nghĩa toán tử: Số nguyên + phân số. Kết quả trả về là 1 phân số.

```

File: PhanSo.cpp

```
// Định nghĩa toán tử: Số nguyên + phân số. Kết quả trả về là 1 phân số.
CPhanSo operator+(int x, CPhanSo y)
{
    CPhanSo Temp;
    Temp.TuSo = y.TuSo + x * y.MauSo;
    Temp.MauSo = y.MauSo;
    return Temp;
}
```

1. OPERATOR OVERLOADING

PHÂN TÍCH TỪNG TRƯỜNG HỢP

TH2: Khai báo toán tử nằm bên ngoài lớp => truy xuất nhờ Getter & Setter . Định nghĩa toán tử: Số nguyên - phân số. Kết quả trả về là 1 phân số.

File: PhanSo.h

```
class CPhanSo
{
private:
    int TuSo;
    int MauSo;
public:
    void SetterTuSo(int);
    void SetterMauSo(int);
    int GetterTuSo();
    int GetterMauSo();
};
CPhanSo operator -(int, CPhanSo); // Định nghĩa toán tử: số nguyên - phân số. Kết quả trả về kiểu phân số.
```

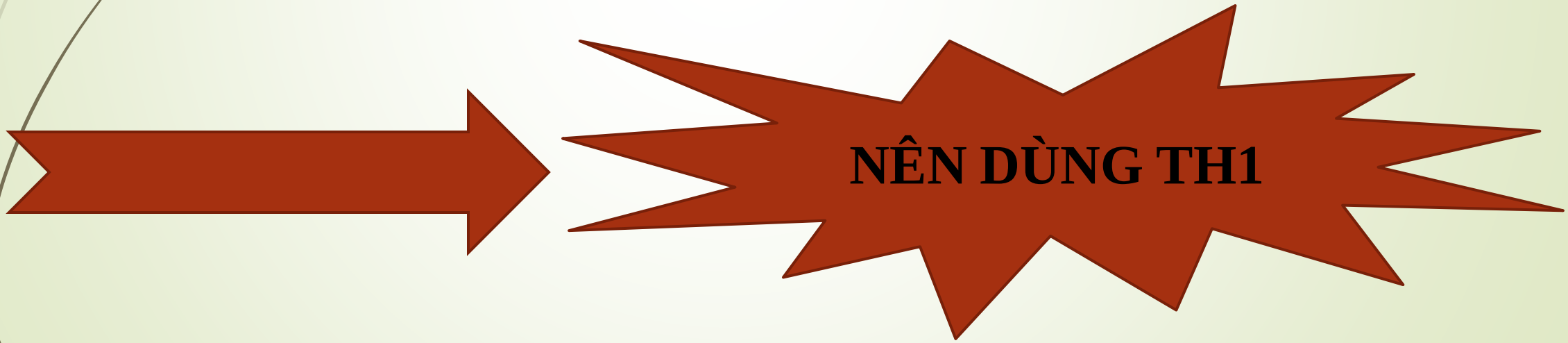
File: PhanSo.cpp

```
// Định nghĩa toán tử: số nguyên - phân số. Kết quả trả về kiểu phân số.
CPhanSo operator -(int x, CPhanSo y)
{
    CPhanSo Temp;
    Temp.SetterTuSo(x * y.GetterMauSo() - y.GetterTuSo());
    Temp.SetterMauSo(y.GetterMauSo());
    return Temp;
}
```

1. OPERATOR OVERLOADING

LỜI KHUYÊN

Nếu toán tử chồng không là hàm thành viên của lớp thì nên sử dụng từ khóa friend thay vì truy xuất đến các thành phần dữ liệu 1 cách phức tạp



1. OPERATOR OVERLOADING

ÁP DỤNG HÀM BẠN ĐỂ TẠO RA TOÁN TỬ NHẬP XUẤT

Thay vì ta phải viết phương thức nhập xuất cho các kiểu dữ liệu ta tự định nghĩa thì ta có thể tự tạo ra toán tử hỗ trợ nhập xuất cho chúng. Toán tử nhập xuất chúng ta tạo ra vừa có thể sử dụng nhập vào từ bàn phím (cin >>) và xuất ra màn hình (cout <<) hoặc nhập dữ liệu vào từ File (FileIn >>) và xuất dữ liệu ra File (FileOut <<) **=> Tính phổ dụng rất cao**

1. OPERATOR OVERLOADING

TOÁN TỬ NHẬP XUẤT TRÊN LỚP PHÂN SỐ

File: PhanSo.h

```
#pragma once
#include <iostream>
using namespace std;

class PhanSo
{
private:
    int TuSo, MauSo;
public:
    friend istream& operator >>(istream &, PhanSo &); // Toán tử nhập
    friend ostream& operator <<(ostream &, PhanSo); // Toán tử xuất
};
```

File: PhanSo.cpp

```
#include "PhanSo.h"

// Toán tử nhập.
istream& operator >>(istream &is, PhanSo &ps)
{
    cout << "\nNhap vao tu so: ";
    is >> ps.TuSo;

    do{
        cout << "\nNhap vao mau so: ";
        is >> ps.MauSo;

        if(ps.MauSo == 0)
        {
            cout << "\nMau so phai khac 0. Xin kiem tra lai !";
        }
    }while(ps.MauSo == 0);
    return is;
}

// Toán tử xuất
ostream& operator <<(ostream &os, PhanSo ps)
{
    os << ps.TuSo << "/" << ps.MauSo;
    return os;
}
```

1. OPERATOR OVERLOADING

CON TRỎ THIS

Định nghĩa: là con trỏ đặc biệt trong C++.

Có 2 cách làm việc với con trỏ this.

1. OPERATOR OVERLOADING

DÙNG ĐỂ TRUY CẬP THUỘC TÍNH – PHƯƠNG THỨC CỦA ĐỐI TƯỢNG

```
void CPhanSo::Nhap()
{
        cout<<"Nhap tu so: ";
        cin>>this->TuSo;
        cout<<"Nhap mau so: ";
        cin>>this->MauSo;
}
```

1. OPERATOR OVERLOADING

GIỮ ĐỊA CHỈ CỦA ĐỐI TƯỢNG ĐANG GỌI PHƯƠNG THỨC

```
CPhanSo& CPhanSo::operator+=(const CPhanSo &b)
{
        *this = *this + b;
        return *this;
}
```

1. OPERATOR OVERLOADING

TOÁN TỬ GÁN BẰNG

Cú pháp:

<Tên lớp đối tượng>& operator = (const <Tên lớp đối tượng>&)

File: PhanSo.h

```
#pragma once
#include <iostream>
using namespace std;

class PhanSo
{
private:
    int TuSo, MauSo;
public:
    PhanSo& operator =(const PhanSo &);
    PhanSo(void);
    ~PhanSo(void);
};
```

File: PhanSo.cpp

```
#include "PhanSo.h"

PhanSo& PhanSo::operator=(const PhanSo &ps)
{
    TuSo = ps.TuSo;
    MauSo = ps.MauSo;
}
```

1. OPERATOR OVERLOADING

VẤN ĐỀ VỀ TOÁN TỬ GÁN BẰNG

Cũng như phương thức tạo lập sao chép. C++ đã hỗ trợ sẵn cho ta toán tử gán mặc định. Hàm này cũng có chức năng tương tự như hàm dựng sao chép mặc định: sao chép từng bit của đối tượng nguồn cho đối tượng đích. Nhưng cần phải cẩn thận nếu như trong thuộc tính của đối tượng có biến con trỏ thì khi sử dụng toán tử gán nó chỉ sao chép phần địa chỉ chứ không thực sự sao chép nội dung vùng nhớ. Do đó cả 2 đối tượng vẫn đang sử dụng chung 1 vùng nhớ. Điều này không theo mong muốn của việc sao chép và có thể dẫn đến các lỗi nghiêm trọng khi chạy chương trình

VẤN ĐỀ VỀ TOÁN TỬ GÁN BẰNG

Do vậy, nếu lớp đối tượng có biến con trỏ và có nhu cầu gán bằng 1 đối tượng khác.

⇒ Cần xây dựng toán tử gán bằng cho lớp

Lưu ý: toán tử gán bằng khác hàm dựng sao chép 1 số điểm sau:

– Xóa phần bộ nhớ nó đang kiểm soát trước khi gán bằng đối tượng mới.

– Kiểm tra kỹ việc đối tượng tự gán bằng chính nó.

1. OPERATOR OVERLOADING

GIẢI PHÁP TỰ XÂY DỰNG TOÁN TỬ GÁN BẰNG ĐỐI VỚI KIỂU DỮ LIỆU CON TRỎ

File: PhanSo.h

```
#pragma once
#include <iostream>
using namespace std;

// Khởi tạo cấu trúc lớp CMangSoNguyen.
class CMangSoNguyen
{
private:
    int *Array;
    int Size;
public:
    CMangSoNguyen& operator =(const CMangSoNguyen &); // Định nghĩa toán tử gán bằng.
};
```

File: PhanSo.cpp

```
// Định nghĩa toán tử gán bằng.
CMangSoNguyen& CMangSoNguyen ::operator=(const CMangSoNguyen &src)
{
    if(this != &src) // Tránh trường hợp gán bằng chính nó.
    {
        delete[] Array; // Xóa vùng nhớ đang giữ.
        Size = src.Size; // Cập nhật Size
        Array = new int[Size]; // Cấp lại vùng nhớ mới.

        // Gán từng phần tử sang.
        for(int i = 0; i < Size; ++i)
        {
            Array[i] = src.Array[i];
        }
    }
    return *this; // Trả về đối tượng hiện tại.
}
```

1. OPERATOR OVERLOADING

DANH SÁCH CÁC TOÁN TỬ CÓ THỂ VIẾT CHỒNG

Các toán tử có thể viết chồng

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

- Các toán tử :: hay . hay .* không được phép định nghĩa bởi người dùng
- Toán tử: sizeof, typeid, ?: không được định nghĩa chồng
- Toán tử =, ->, [], () chỉ được viết chồng bằng các hàm non-static

1. OPERATOR OVERLOADING

CẦN GHI NHỚ

Kết luận: Bộ 3 hàm này luôn đi chung với nhau nếu đối tượng có thuộc tính kiểu dữ liệu con trỏ:

1/ Hàm dựng sao chép.

2/ Toán tử gán bằng.

3/ Hàm hủy.

1. OPERATOR OVERLOADING

NHỮNG LƯU Ý KHI SỬ DỤNG

Tránh thay đổi ý nghĩa nguyên thủy của toán tử đó
Vd: Khai báo operator + nhưng bên trong lại cài đặt nhân.

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.

1. OPERATOR OVERLOADING

NHỮNG LƯU Ý KHI SỬ DỤNG

Chỉ viết toán tử nếu nó là hợp lý.

Vd: Có thể có toán tử trừ giữa 2 ngày nhưng không thể có toán tử nhân giữa 2 ngày mặc dù ta có thể viết được

1. OPERATOR OVERLOADING

NHỮNG LƯU Ý KHI SỬ DỤNG

Ưu tiên viết toán tử ở bên trong lớp hơn các toán tử viết ở bên ngoài lớp nếu chúng phải truy xuất đến các thành phần thuộc tính của lớp. Ngược lại đối với các toán tử không cần truy cập đến thành phần thuộc tính của lớp thì ưu tiên viết bên ngoài lớp.

2. FUNCTION OVERLOADING

GIỚI THIỆU

- **Nhu cầu**
  - Thực hiện một công việc với nhiều cách khác nhau. Nếu các hàm khác tên sẽ khó quản lý.
- **Khái niệm nạp chồng/quá tải (overload) hàm**
  - Hàm cùng tên nhưng có tham số đầu vào hoặc đầu ra khác nhau.
  - Nguyên mẫu hàm (prototype) khi bỏ tên tham số phải khác nhau.
  - Cho phép người dùng chọn phương pháp thuận lợi nhất để thực hiện công việc.

2. FUNCTION OVERLOADING

Ví dụ các phương thức sau đây được nạp chồng:

1. PhanSo Cong(PhanSo); // Cộng giữa phân số với phân số, kết quả trả về là phân số.

2. PhanSo Cong(int); // Cộng giữa phân số với số nguyên, kết quả trả về là phân số

3. friend PhanSo Cong(int, PhanSo); // Số nguyên + phân số, kết quả trả về là 1 phân số.

2. FUNCTION OVERLOADING

Bản chất các hàm dựng (Constructor) cũng là những dạng nạp chồng hàm.

1. PhanSo(); // Hàm dựng mặc định
2. PhanSo(int, int); // Hàm dựng 2 tham số
3. PhanSo(int); // Hàm dựng 1 tham số
4. PhanSo(const PhanSo &); // Hàm dựng sao chép

2. FUNCTION OVERLOADING

NHỮNG ĐIỀU CẦN LƯU Ý TRÁNH MẮC PHẢI

- Sự nhập nhằng, mơ hồ (ambiguity)
  - Do sự tự chuyển đổi kiểu

```
float f(float x) { return x / 2; }
double f(double x) { return x / 2; }

void main()
{
	float x = 29.12;
	double y = 17.06;
	printf("%.2f\n", f(x)); // float
	printf("%.2lf\n", f(y));// double
	printf("%.2f", f(10));  // ???
}
```

2. FUNCTION OVERLOADING

NHỮNG ĐIỀU CẦN LƯU Ý TRÁNH MẮC PHẢI

- Sự nhập nhằng, mơ hồ (ambiguity)
  - Do sự tự chuyển đổi kiểu

```
void f(unsigned char c)
{
      printf("%d", c);
}
void f(char c)
{
      printf("%c", c);
}
void main()
{
      f('A');      // char
      f(65);       // ???
}
```

2. FUNCTION OVERLOADING

NHỮNG ĐIỀU CẦN LƯU Ý TRÁNH MẮC PHẢI

- Sự nhập nhằng, mơ hồ (ambiguity)
  - Do việc sử dụng tham chiếu

```
int f(int a, int b)
{
      return a + b;
}
int f(int a, int &b)
{
      return a + b;
}
void main()
{
      int x = 1, y = 2;
      printf("%d", f(x, y));  // ???
}
```

2. FUNCTION OVERLOADING

NHỮNG ĐIỀU CẦN LƯU Ý TRÁNH MẮC PHẢI

- Sự nhập nhằng, mơ hồ (ambiguity)
  - Do việc sử dụng tham số mặc định

```
int f(int a)
{
      return a*a;
}
int f(int a, int b = 0)
{
      return a*b;
}
void main()
{
      printf("%d\n", f(2912, 1706));
      printf("%d\n", f(2912));      //???
}
```

3. CHIA SẺ KINH NGHIỆM KHI LẬP TRÌNH

CÓ THỂ BẠN KHÔNG BIẾT

Hàm **friend** ngoài chức năng là hàm bạn thì còn được các lập trình viên sử dụng để che giấu 1 phương thức nào đó do họ viết thêm bổ sung vào chương trình để tránh bị những lập trình viên làm chung nhìn thấy hoặc họ sợ sẽ gây rối cho những lập trình viên cùng làm việc chung với họ.

=> Xem demo ngay tại lớp

3. CHIA SẺ KINH NGHIỆM KHI LẬP TRÌNH

CÓ THỂ BẠN KHÔNG BIẾT

Operator Overloading có ở bên C# nhưng Java thì không có vì nó các lập trình viên Java cho rằng Operator Overloading sẽ làm cho đoạn mã bị rối lên và rất khó để bảo trì hay muốn mở rộng.

3. CHIA SẺ KINH NGHIỆM KHI LẬP TRÌNH

CÓ THỂ BẠN KHÔNG BIẾT

Operator bên C# bắt buộc phải là phương thức tĩnh (static), tức là phương thức lớp (class – level method)

```
class PhanSo
{
    private int TuSo, MauSo;

    public static PhanSo operator +(PhanSo x, PhanSo y)
    {
        PhanSo Tong = new PhanSo();
        Tong.TuSo = x.TuSo * y.MauSo + x.MauSo * y.TuSo;
        Tong.MauSo = x.MauSo * y.MauSo;

        return Tong;
    }
}
```

3. CHIA SẺ KINH NGHIỆM KHI LẬP TRÌNH

CÓ THỂ BẠN KHÔNG BIẾT

Toán tử gán không được phép viết chồng trong C#.
Toán tử gán sẵn có của C# thực hiện sao chép vật lý theo bit hay là theo cơ chế ép kiểu tùy theo tình huống

Trong C#, khi thực hiện chồng toán tử 2 ngôi, chẳng hạn như phép toán +, khi đó toán tử += sẽ tự động được chồng mà ta không cần phải cài đặt như bên C++

3. CHIA SẺ KINH NGHIỆM KHI LẬP TRÌNH

CÓ THỂ BẠN KHÔNG BIẾT

Số lượng các toán tử có thể viết chồng trong C++ nhiều hơn trong C#. Cụ thể trong C# không thể viết chồng các toán tử =, **()**, **[]**, **&&**, || và toán tử **new**

4. BÀI TẬP ỨNG DỤNG TẠI LỚP

Cài đặt các toán tử sau cho lớp PhanSo:

1. =
2. +, -, *, /
3. <, <=, ==, !=, >=, >
4. +=, -=, *=, /=
5. <<, >>
6. ++, --

5. BÀI TẬP VỀ NHÀ

- **Bài 01:** Xây dựng lớp CThoiGian thực hiện các toán tử sau:
  - Toán tử so sánh: >, <, ==, >=, <=, !=
  - Toán tử cộng trừ: +, -
  - Toán tử tăng giảm: ++, -- (tăng và giảm một giây)
  - Toán tử nhập xuất: <<, >>

5. BÀI TẬP VỀ NHÀ

➢ **Bài 02:** Xây dựng lớp CNgay thực hiện các toán tử sau:

- Toán tử so sánh: >, <, ==, >=, <=, !=
- Toán tử cộng trừ: +, -
- Toán tử tăng giảm: ++, -- (tăng và giảm một ngày)
- Toán tử nhập xuất: <<, >>
- Toán tử tăng lên một số ngày: +=
- Toán tử giảm xuống một số ngày: -=

5. BÀI TẬP VỀ NHÀ

- **Bài 03:** Xây dựng lớp CDonThuc thực hiện các toán tử sau:
  - Toán tử so sánh: >, <, ==, >=, <=, !=
  - Toán tử toán học: +, -, *, /
  - Toán tử một ngôi: ! (đạo hàm), ~ (nguyên hàm)
  - Toán tử nhập xuất: <<, >>