Lập Trình Hướng Đối Tượng

CHƯƠNG 4: TÁI ĐỊNH NGHĨA (OVERLOADING)

TRẦN VIỆT KHÁNH

NỘI DUNG

4.1 Tái định nghĩa hàm và toán tử

4.2 Trị trả về của hàm là đối tượng, con trỏ *this

4.3 Cách sử dụng thành phần tĩnh (static)

4.1 Tái định nghĩa hàm và toán tử

1. TOÁN TỬ
2. CHỒNG TOÁN TỬ
3. CÚ PHÁP NẠP CHỒNG TOÁN TỬ
4. TOÁN TỬ MỘT NGÔI
5. TOÁN TỬ HAI NGÔI
6. HỖ TRỢ NGÔN NGỮ .NET KHÁC
7. PHẠM VI SỬ DỤNG CỦA CÁC TOÁN TỬ
8. MỘT SỐ TRƯỜNG HỢP NÊN SỬ DỤNG TOÁN TỬ NẠP CHỒNG
9. YÊU CẦU KHI SỬ DỤNG CHỒNG TOÁN TỬ
10. ƯU VÀ NHƯỢC ĐIỂM CỦA CHỒNG TOÁN TỬ

TOÁN TỬ

- Toán tử được ký hiệu bằng các biểu tượng đặc biệt và được dùng để thực hiện một hành động nào đó. Trong C# có các toán tử +, -, *, /, %, v. v… Thực ra các toán tử này là các hàm.

→ đơn giản là chúng được gọi bằng các cú pháp thay vì gọi hàm.

Ví dụ: x+7

- "+" là toán tử 2 ngôi với toán hạng là x và 7.
- Thay vì gọi hàm cộng với 2 toán hạng là x và 7 thì con người thích ký hiệu "+" này hơn là gọi hàm cộng.

→ Vì vậy ta hãy nghĩ về nó như là một hàm **+(x, 7)**

- **"+"** là tên hàm
- **x, 7** là 2 đối số truyền vào.
- Hàm "+" trả về tổng của các đối số truyền vào.

CHỒNG TOÁN TỬ (Operator Overload)

- Nạp chồng toán tử có nhiều điểm tương tự với nạp chồng hàm, toán tử chính là tên của hàm.

- Việc chồng toán tử thực chất cũng giống như ta xây dựng một hàm thông thường và kết quả thu được như nhau, nhưng chồng toán tử giúp người lập trình có thể sử dụng các ký hiệu toán học thường gặp(+ , -, *, / , v.v…) , dễ nhớ và gần gũi hơn.

CÚ PHÁP NẠP CHỒNG TOÁN TỬ

+ Cú pháp:

Type operator operator_symbol(parameter_list);

Trong đó :

- Type là kiểu giá trị trả về của hàm.
- parameter_list là danh sách các đối số nếu có
- Operator_symbol là các toán tử.

Ví dụ: int operator +(int x,int y);
Phép "+" ở đây đã được nạp chồng.

CÚ PHÁP NẠP CHỒNG TOÁN TỬ

- Trong C# có các hàm toán tử có thể nạp chồng và các phương thức thay thế của nó như sau:

Toán tử	Tên phương thức thay thế	Toán tử	Tên phương thức thay thế
+	Add	>	Compare
-	Subtract	<	Compare
*	Multiply	!=	Compare
/	Divide	>=	Compare
%	Mod	<=	Compare
^	Xor	*=	Multiply
&	BitwiseAnd	- =	Subtract
|	Bitwiseor	^=	Xor
&&	Add	<<=	leftshift
||	Or	%=	Mod
=	Assign	+=	Add
<<	leftshift	&=	BitwiseAnd
>>	Rightshift	|=	Bitwiseor
= =	Equals	/=	Divide

TOÁN TỬ MỘT NGÔI

- Toán tử một ngôi là toán tử chỉ nhận một toán hạng

  Ví dụ: x = -y; // Đặt x bằng âm y

- Toán tử một ngôi khác:

  ++, --, + (+a) , - (-a) , ++ (++a) …

TOÁN TỬ HAI NGÔI

- Các toán tử như toán tử (= =) so sánh giữa hai đối tượng. toán tử (!=) so sánh không bằng giữa hai đối tượng, “<” so sánh nhỏ hơn, “>” so sánh lớn hơn… Đây là các toán tử phải có các cặp toán hạng hay ta còn gọi là toán tử hai ngôi.

Ngoài ra, trong C# còn có toán tử ba ngôi.

Ví dụ:

```
int value1,value2, maxValue;
value1=10; value=20;
maxValue=value1>value2 ? value1:value2;
```

- Trong ví dụ trên toán tử ba ngôi được sử dụng để kiểm tra xem giá trị của value1 có lớn hơn giá trị của value2 không, nếu đúng thì trả về giá trị của value1, tức gán giá trị value1 cho biến maxValue, còn ngược lại thì gán giá trị value2 cho biến maxValue.

CODE MINH HỌA NẠP CHỒNG TOÁN TỬ

```
using System;
 namespace VietKhanh
{
    class Box
    {
        private double chieu_dai;
        private double chieu_rong;
        private double chieu_cao;

        public double tinhTheTich()
        {
            return chieu_dai * chieu_rong * chieu_cao;
        }

        public void setChieuDai(double len)
        {
            chieu_dai = len;
        }
```

CODE MINH HỌA NẠP CHỒNG TOÁN TỬ

```
        public void setChieuRong(double wid)
        {
            chieu_rong = wid;
        }

        public void setChieuCao(double hei)
        {
            chieu_cao = hei;
        }

        // nap chong toan tu "+" de cong hai doi tuong Box.
        public static Box operator +(Box b, Box c)
        {
            Box box = new Box();
            box.chieu_dai = b.chieu_dai + c.chieu_dai;
            box.chieu_rong = b.chieu_rong + c.chieu_rong;
            box.chieu_cao = b.chieu_cao + c.chieu_cao;
            return box;
        }
    }
}
```

CODE MINH HỌA NẠP CHỒNG TOÁN TỬ

- **Code xử lý trên Form.**

```
//tao cac doi tuong Box1, Box2 va Box3
Box Box1 = new Box();
Box Box2 = new Box();
Box Box3 = new Box();
double the_tich = 0.0;

// nhap thong tin Box1
Box1.setChieuDai(6.0);
Box1.setChieuRong(7.0);
Box1.setChieuCao(5.0);

// nhap thong tin Box2
Box2.setChieuDai(12.0);
Box2.setChieuRong(13.0);
Box2.setChieuCao(10.0);
```

CODE MINH HỌA NẠP CHỒNG TOÁN TỬ

```
// tinh va hien thi the tich Box1
the_tich = Box1.tinhTheTich();
lblTheTichHop1.Text = the_tich.ToString();

// tinh va hien thi the tich Box2
the_tich = Box2.tinhTheTich();
 lblTheTichHop2.Text = the_tich.ToString();

// cong hai doi tuong Box1 và Box2 thông qua nạp
      chồng toán tử operator +(Box b, Box c)
Box3 = Box1 + Box2;

// tinh va hien thi the tich Box3
the_tich = Box3.tinhTheTich();
```

Nạp chồng phương thức (Function Overload)

- C# cho phép nhiều hàm (phương thức) trong cùng một phạm vi (namespace, class) có thể trùng tên, nhưng phải khác nhau về các tham số khi gọi. Điều này được gọi là nạp chồng phương thức hay còn gọi là định nghĩa chồng.
- Ví dụ:

```
class C {
    public:
        int compare(int x, int y);
        int compare(int x, int y) const;
        int compare(float x, float y);
};
```

Nạp chồng phương thức (Function Overload)

+ Định nghĩa chồng ở lớp con sẽ che mất các phương thức cùng tên của lớp cha.

```
class D: C { // lớp con D kế thừa lớp cha C (học thêm ở bài sau)
        public:
                // Định nghĩa trùng tên với lớp cha
                int compare(string s1, string s2);
};
        // Sử dụng lớp con D
        D d = new D();
        d.compare("1234", "abcd"); // OK
        d.compare(10, 20); // lỗi
        // Sử dụng lớp cha C
        d.C::compare(10, 20); // OK
```

Bài tập áp dụng

- **Xây dựng class Phân số và thực hiện các phép tính cộng, trừ, nhân, chia, hai phân số, thực hiện rút gọn phân số.**

- **Các bước thực hiện như sau:**

Bước 1: Thiết kế class phân số

```
class PHANSO
{
    int tuso, mauso;
    public PHANSO()
    {
    }
    public PHANSO(int t, int m)
    {
       tuso = t;
       mauso = m;
    }
    int UCLN(int a, int b)
    {
        while(a!=b)
        {
            if (a >b)
                a = a - b;
            else
                b = b - a;
         }
        return a;
    }
```

Thiết kế class phân số (tt)

```
private void RutGon(int tso,int mso)
{
      int x = UCLN(tso, mso);
      tuso = tso / x;
      mauso = mso / x;
}
public void Cong(PHANSO ps1,PHANSO ps2)
{
      tuso = ps1.tuso*ps2.mauso + ps2.tuso*ps1.mauso;
      mauso = ps1.mauso*ps2.mauso;
      RutGon(tuso,mauso);
}
public void Tru(PHANSO ps1,PHANSO ps2)
{
       tuso = ps1.tuso * ps2.mauso - ps2.tuso * ps1.mauso;
       mauso = ps1.mauso * ps2.mauso;
       RutGon(tuso, mauso);
}
```

Thiết kế class phân số (tt)

```
    public void Nhan(PHANSO ps1, PHANSO ps2)
    {
        tuso = ps1.tuso * ps2.tuso;
        mauso = ps1.mauso * ps2.mauso;
        RutGon(tuso, mauso);
    }
    public void Chia(PHANSO ps1, PHANSO ps2)
    {
        tuso = ps1.tuso * ps2.mauso ;
        mauso = ps1.mauso * ps2.tuso;
        RutGon(tuso, mauso);
    }
    public void   NhapPhanSo(string strtuso,string strmauso)
    {
        tuso = int.Parse(strtuso);
        mauso = int.Parse(strmauso);
    }
    public string Xuat()
    {
        return tuso + "/" + mauso;
    }
}
```

Bước 2: Viết code cho các chức năng cộng, trừ nhân, chia.

```
Void main()
{
        PHANSO a = new PHANSO(tuso1, mauso1);
       PHANSO b = new  PHANSO(tuso2, mauso2);
       PHANSO c = new PHANSO();
       c.Cong(a,b);
       lblKetQua.Text = c.Xuat();
}
```

Bộ test mẫu:

Test	Phân số 1	Phân số 2	Cộng	Trừ	Nhân	Chia
test 1	3/4	2/3	17/12	1/12	1/2	9/8
test 2	7/2	5/4	19/4	9/4	35/8	14/5

DEMO

HỖ TRỢ NGÔN NGỮ .NET KHÁC

C# cung cấp khả năng cho phép nạp chồng toán tử cho các lớp mà chúng ta xây dựng. Trong khi đó các ngôn ngữ .NET khác không cho phép điều này.

+ Vì vậy ta phải xây dựng các hàm (phương thức) thay thế cho các phép toán có sử dụng nạp chồng toán tử để các ngôn ngữ khác có thể gọi và sử dụng được . Để làm được điều này khi sử dụng nạp chồng toán tử nào đó thì phải xây dựng cho nó một phương thức có chức năng tương tự như toán tử đã chồng.

Ví dụ: Khi chúng ta nạp chồng toán tử (+) thì phải cung cấp một phương thức Add( ) cũng làm chức năng cộng hai đối tượng. Khi sử dụng chồng toán tử (= =) thì nên cung cấp thêm phương thức Equals(), v.v...

PHẠM VI SỬ DỤNG CỦA CÁC TOÁN TỬ

Phạm trù	Toán tử	Hạn chế
Nhị phân toán học	+, *, /, -, %	Không
Thập phân toán học	+, -, ++, --	Không
Nhị phân bit	&, |, ^, <<, >>	Không
Thập phân bit	!, ~, true, false	Không
So sánh	==, !=, >=, <, <=, >	Phải nạp chồng theo từng cặp.

MỘT SỐ TRƯỜNG HỢP SỬ DỤNG TOÁN TỬ NẠP CHỒNG

1. Trong toán học, mọi đối tượng toán học như: tọa độ, vector, ma trận, hàm số v…v… Nếu viết chương trình làm những mô hình toán học hay vật lý, bạn nhất định sẽ mô tả những đối tượng này.

2. Những chương trình đồ họa sẽ sử dụng các đối tượng toán học và toạ độ khi tính toán vị trí của nó trên màn hình.

3. Một lớp mô tả số lượng tiền.

4. Việc sử lý từ hay chương trình phân tích văn bản có lớp để mô tả các câu văn, mệnh đề và bạn phải sử dụng các toán hạng để liên kết các câu lại với nhau.

YÊU CẦU KHI SỬ DỤNG CHỒNG TOÁN TỬ

+ Khi định nghĩa những toán tử trong kiểu dữ liệu giá trị, kiểu dữ liệu xây dựng sẵn.

+ Phải cung cấp các phương thức thay thế cho toán tử được nạp chồng. Bởi vì hầu hết các ngôn ngữ khác không cho phép chồng toán tử nên khi ta sử dụng thêm phương thức có chức năng tương tự như toán tử giúp cho chương trình có thể phù hợp với nhiều ngôn ngữ khác nhau.

+ Sử dụng chồng toán tử trong trường hợp kết quả trả về rõ ràng.

Ví dụ: Khi ta sử dụng toán tử “or” hoặc “and” giữa một giá trị thời gian này với một thời gian khác thì kết quả trả về sẽ không rõ ràng vì vậy trong trường hợp này ta không nên sử dụng chồng toán tử.

ƯU VÀ NHƯỢC ĐIỂM CỦA CHỒNG TOÁN TỬ

Ưu điểm :

+ Nạp chồng toán tử là một phương thức ngắn gọn giúp mã nguồn dễ nhìn hơn, dễ quản lý,sáng sủa hơn.

+ Nạp chồng toán tử là đường dẫn cho những đối tượng.

Nhược điểm:

+ Ta phải sử dụng nạp chồng toán tử một cách hạn chế và phù hợp với các toán tử của lớp được xây dựng sẵn, không sử dụng toán tử quá mới hay quá riêng rẽ. Nếu sử dụng toán tử một cách lạm dụng thì sẽ làm chương trình dễ nhầm lẫn.

4.2 Trị trả về của hàm là đối tượng, con trỏ *this

Cú pháp con trỏ:

+ Kiểu dữ liệu: * Tên con trỏ;

Cách dùng con trỏ:

+ * Tên con trỏ = giá trị (nội dung).

Ví dụ:

```
int x=10;
int *px;
px = &x;
```

4.2 Trị trả về của hàm là đối tượng, con trỏ *this

Con trỏ this trỏ vào dữ liệu thành viên của chính nó. Điều này có nghĩa là con trỏ this chỉ có phạm vi tác dụng trong một lớp. Một điều cực kì quan trọng, là con trỏ this chỉ hoạt động với các dữ liệu thành viên và các hàm thành viên được khai báo là không tĩnh (non-static). Các dữ liệu thành viên và hàm thành viên tĩnh (static) không hỗ trợ con trỏ this.

Ví dụ

4.2 Trị trả về của hàm là đối tượng, con trỏ *this

```
class Complex{
        float real;
        float img;
        public   Complex(float real, float img)
        {
                this->real = real;
                this->img = img;
        }
        bool isMe(const Complex &c)
        {
                if(&c==this) return true;
                else return false;
        }
        public static void main(){
                Complex a = Complex(3, 2);
                Complex b=Complex b(2, 2);
                Complex *c = &a;
                Console.WriteLine( a.isMe(b));// cho giá trị false
                Console.WriteLine( a.isMe(*c));// cho giá trị true
```

4.2 Trị trả về của hàm là đối tượng, con trỏ *this

Với việc sử dụng con trỏ this trong hàm tạo, bạn có thể đặt tên các tham số trong hàm tạo trùng với tên các dữ liệu của lớp. Để truy cập đến các thuộc tính của lớp, ta sử dụng con trỏ this. Hàm thành viên isMe sẽ kiểm tra một đối tượng có phải là chính nó hay không (có cùng địa chỉ trên bộ nhớ). Dù là một bản sao của nó (có dữ liệu thành viên giống nhau) thì kết quả nhận được cũng là sai (0). Trong hàm main, ta khởi tạo hai đối tượng a và b. Đối tượng con trỏ c sẽ trỏ vào địa chỉ của đối tượng a. Điều này có nghĩa là c sẽ có cùng vùng địa chỉ với a, còn b thì không. Khi gọi hàm a.isMe(b) sẽ cho kết quả là sai (0) và a.isMe(*c) sẽ cho kết quả là đúng (1).

4.3 Cách sử dụng thành phần tĩnh (static)

Xét ví dụ class TinhToan có phương thức
Class TinhToan{
public int Cong(int a, int b); }
Nếu bạn muốn xài phương thức này thì phải có 1 object TinhToan, ví dụ TinhToan tt = new TinhToan(); rồi gọi tt.Cong(1, 2); mới xài được.
Ở đây tạo object hơi thừa vì với bất kì object nào của class TinhToan thì hàm Cong() cũng như nhau cả, vậy cần tạo 1 object để làm gì. Chỉ cần thêm từ khóa "static" vào public static int Cong(int a, int b); thì không cần tạo object làm gì nữa, gọi TinhToan.Cong(1, 2); là được.

Tóm lại khi sử dụng từ khóa static thì không cần khởi tạo đối tượng (object)

XIN CÁM ƠN !

www.lytc.edu.vn