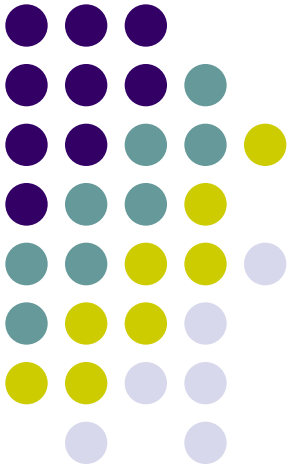
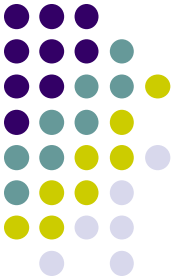
Chương 4

Quá tải toán tử

(Operator Overloading)

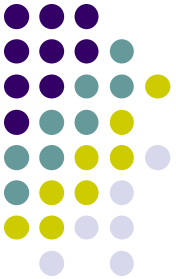


Nội dung



- Giới thiệu
- Khả năng và hạn chế của quá tải toán tử
- Cài đặt quá tải toán tử

Giới thiệu



```
1  #pragma once
2  class Fraction
3  {
4  private:
5      int numerator;
6      int denominator;
7  public:
8      Fraction( int num, int denom );
9      Fraction add(const Fraction &f);
10     int getNum() const { return numerator; }
11     int getDenom() const { retur int Fraction::numerator; }
12      ~Fraction(void);
13 };
```

```
11 int main()
12 {
13     //demo fraction
14     Fraction f1(3,2), f2(1,6);
15     f1 = f1.add(f2);
16     cout<<f1.getNum()<<"/"<<f1.getDenom()<<endl;
17
```

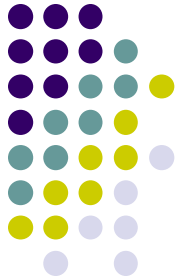
Giới thiệu

f1 = f1.add(f2);

→ f1 = f1 + f2;

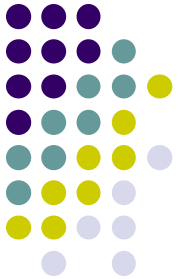
- Toán tử “+” trong ngôn ngữ C++ chỉ hỗ trợ cho các kiểu dữ liệu cơ bản: int, float, double, char,…
- C++ hỗ trợ khả năng xây dựng trên lớp các toán tử cần thiết(hàm toán tử)

Giới thiệu



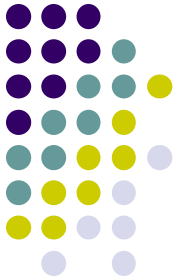
- Là một dạng của quá tải hàm
- Sử dụng các toán tử hiện có trong C++ cho các đối tượng do người dùng định nghĩa thay vì gọi hàm
  - Tự nhiên
  - Ngắn gọn, có ý nghĩa hơn
  - Ý nghĩa ban đầu của toán tử vẫn giữ nguyên
- Định nghĩa giống với định nghĩa hàm
- Hàm toán tử có thể là hàm thành viên hoặc là hàm bạn của lớp hoặc là hàm tự do

Ví dụ quá tải toán tử



```
2  class Fraction
3  {
4  private:
5      int numerator;
6      int denominator;
7  public:
8      Fraction( int num, int denom );
9      //Fraction add(const Fraction &f);
10     Fraction operator+( const Fraction& other ) const;
11     Fraction operator+( int val ) const;
12     Fraction& operator+=( const Fraction& other );
13     Fraction& operator+=( int val );
14
15     int getNum() const { return numerator; }
16     int getDenom() const { return denominator; }
17      ~Fraction(void);
18 };
```

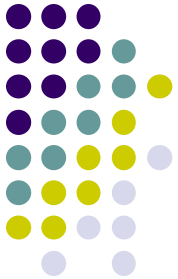
Ví dụ quá tải toán tử



```
15  Fraction Fraction::operator+( const Fraction& other ) const
16  {
17        // Tim boi so chung nho nhat của mau so
18        int lcm = other.denominator;
19        while( lcm % denominator != 0 ) {
20           lcm += other.denominator;
21        }
22        // Return a new fraction
23        return Fraction(
24           (lcm / denominator) * numerator +
25           (lcm / other.denominator) * other.numerator, lcm );
26
27  }
```

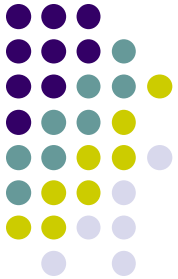
```
28  Fraction Fraction::operator+( int val ) const
29  {
30        // Return a new fraction
31        return Fraction( numerator + denominator * val, denominator );
32  }
33
```

Ví dụ quá tải toán tử



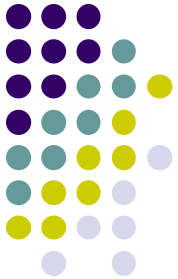
```
34  Fraction& Fraction::operator +=(const Fraction& other ){
35
36          Fraction temp( *this );
37          // su dung toan tu "+" da dinh nghia
38          temp = temp + other;
39          numerator = temp.numerator;
40          denominator = temp.denominator;
41
42          // Return a reference to ourself
43          return *this;
44  }
46 Fraction& Fraction::operator+=( int val ) {
47
48          // su dung toan tu "+" da dinh nghia
49          Fraction temp( *this );
50          temp = temp + val;
51
52          numerator = temp.numerator;
53          denominator = temp.denominator;
54
55          // Return a reference to ourself
56          return *this;
57  }
```

Hạn chế của quá tải toán tử



- Không thể tạo ra toán tử mới
- Không thể thay đổi
  - Cách thức hoạt động của toán tử trên các kiểu dữ liệu nguyên thủy
  - Thứ tự ưu tiên của toán tử
  - Kết hợp (từ trái sang phải hoặc từ phải sang trái)
  - Số lượng toán hạng
  - Kiểu trả về của toán tử

Hạn chế của quá tải toán tử



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

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

Cú pháp

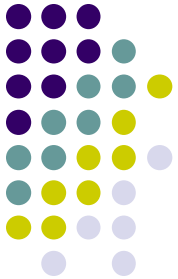
<*returnType*> operator<@>(*parameters*);

↑ ↑ ↑

any type *keyword* *operator symbol*

- Trong đó, @: kí tự toán tử trong C++(+,-,*,=…)

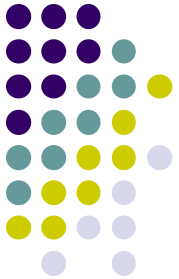
Gọi hàm toán tử



```
Fraction Fraction::operator+( const Fraction& other ) const
```

```
//demo fraction
Fraction f1(3,2), f2(1,6);

Fraction f3 = f1 + f2;//f3 = f1.operator+(f2)

cout<<f3.getNum()<<"/"<<f3.getDenom()<<endl;
```

Cài đặt toán tử được quá tải



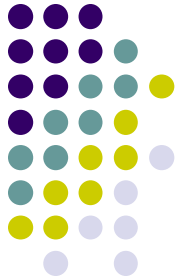
- Có 3 cách cài đặt toán tử được quá tải
  - Hàm thành viên
  - Hàm không thành viên toàn cục
  - Hàm bạn

Số lượng tham số

	Unary	Binary
Member*	0	1**
Non-member	1	2

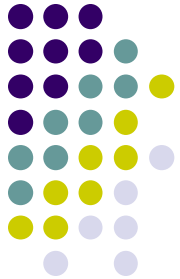
- Khi hàm toán tử là hàm thành viên của lớp
  - → Số lượng tham số giảm đi 1, vì đã có 1 tham số ngầm định gọi hàm toán tử
- Khi hàm toán tử không là hàm thành viên thì phải bao hàm đầy đủ số tham số

Cài đặt toán tử được quá tải



```
class complex {
private:
    double    real;
    double    imag;

public:
                  complex(double r=0, double i=0);

        complex   operator+(complex);
        complex   operator+(double);
friend  complex   operator+(double, complex);

friend  complex   operator+(complex, complex);
};
```

Cài đặt toán tử bằng hàm thành viên

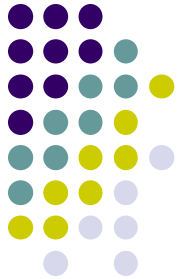


```
complex complex::operator+(complex b)
{
    double r = real + b.real;
    double i = imag + b.imag;

    return complex(r, i);
}

complex complex::operator+(double b)
{
    double r = real + b;
    double i = imag;

    return complex(r, i);
}
```

```
complex c1(3,4);
complex c2(2,5);
complex c3;

c3 = c1 + c2;
```

```
complex c1(1,4);
complex c2;
double      d = 3.5;

c2 = c1 + d;
```

Cài đặt toán tử bằng hàm bạn



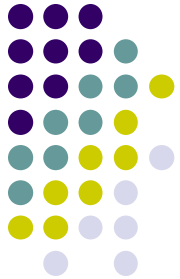
```
complex operator+(double a, complex b)
{
    double r = a + b.real;
    double i = b.imag;

    return complex(r, i);
}
```

```
complex c1(1,4);
complex c2;
double      d = 3.5;

c2 = d + c1;
```

```
complex operator+(complex a, complex b)
{
    double r = a.real + b.real;
    double i = a.imag + b.imag;

    return complex(r, i);
}
```

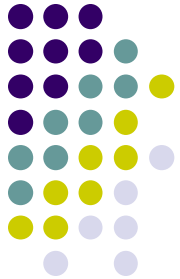
Does all 3 operations – Assumes a double to complex conversion ctor

Cài đặt toán tử bằng hàm thành viên



- Cài đặt toán tử bằng hàm thành viên của lớp A khi:
  - Toán hạng bên trái nhất phải có kiểu thuộc lớp A
  - Toán tử **()**, **[ ]**, **->**, **=** phải là hàm thành viên của lớp
- Con trỏ this được sử dụng làm tham số ngầm định

Cài đặt toán tử bằng hàm không thành viên



- Có một đối tượng thuộc lớp khác
- Định nghĩa là hàm bạn nếu cần truy cập đến dữ liệu thành viên của lớp
- Toán tử << có toán hạng bên trái thuộc lớp **ostream**: **cout << classObject**
- Toán tử >> có toán hạng bên trái thuộc lớp **istream** : **cin >> classObject**

Operator	Name or Category	Method or Global Friend Function	When to Overload	Sample Prototype

`operator+`
`operator-`
`operator*`
`operator/`
`operator%`
	Binary arithmetic	Global friend function recommended	Whenever you want to provide these operations for your class	`friend const T operator+(const T&, const T&);`

`operator-`
`operator+`
`operator~`
	Unary arithmetic and bitwise operators	Method recommended	Whenever you want to provide these operations for your class	`const T operator-() const;`

`operator++` `operator--`	Increment and decrement	Method recommended	Whenever you overload binary + and -	`T& operator++();` `const T operator++(int);`
`operator=`	Assignment operator	Method required	Whenever you have dynamically allocated memory in the object or want to prevent assignment, as described in Chapter 9	`T& operator=(const T&);`

`operator+=`
`operator-=`
`operator*/`
`operator/=`
`operator%=`
	Shorthand arithmetic operator assignments	Method recommended	Whenever you overload the binary arithmetic operators	`T& operator+=(const T&);`

`operator<<`	Binary bitwise	Global friend	Whenever you want	`friend const T operator<<(const T&, const`

`operator>>`
`operator&`
`operator|`
`operator^`
	operators	function recommended	to provide these operations	`T&);`

`operator<<=`
`operator>>=`
`operator&=`
`operator|=`
`operator^=`
	Shorthand bitwise operator assignments	Method recommended	Whenever you overload the binary bitwise operators	`T& operator<<=(const T&);`

`operator<`
`operator>`
`operator<=`
`operator>=`
`operator==`
	Binary comparison operators	Global friend function recommended	Whenever you want to provide these operations	`friend bool operator<(const T&, const T&);`

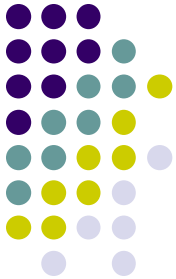
`operator<<` `operator>>`	I/O stream operators (insertion and extraction)	Global friend function recommended	Whenever you want to provide these operations	`friend ostream &operator<<(ostream&, const T&);` `friend istream &operator>>(istream&, T&);`
`operator!`	Boolean negation operator	Member function recommended	Rarely; use `bool` or `void*` conversion instead	`bool operator!() const;`
`operator&&` `operator||`	Binary Boolean operators	Global friend function recommended	Rarely	`friend bool operator&&(const T& lhs, const T& rhs);` `friend bool operator||(const T& lhs, const T& rhs);`
`operator[]`	Subscripting (array index)	Method required	When you want to support subscripting:	`E& operator[](int);`

	operator		in array-like classes	`const E& operator[](int) const;`
`operator()`	Function call operator	Method required	When you want objects to behave like function pointers	Return type and arguments can vary; see examples in this chapter
`operator new` `operator new[]`	Memory allocation routines	Method recommended	When you want to control memory allocation for your classes (rarely)	`void* operator new(size t size) throw(bad_alloc);` `void* operator new[](size t size) throw(bad_alloc);`
`operator delete` `operator delete[]`	Memory deallocation routines	Method recommended	Whenever you overload the memory allocation routines	`void operator delete(void* ptr) throw();` `void operator delete[](void* ptr) throw();`
`operator*` `operator->`	Dereferencing operators	Method required for operator-> Method recommended for operator*	Useful for smart pointers	`E& operator*() const;` `E* operator->() const;`
`operator&`	Address-of operator	N/A	Never	N/A
`operator->*`	Dereference pointer-to-member	N/A	Never	N/A
`operator,`	Comma operator	N/A	Never	N/A
`operator type()`	Conversion, or cast, operators (separate per type)	Method required	When you want to provide conversions from your class to other types	`operator type() const;`

Ví dụ: quá tải toán tử “=“

```
time&  time::operator=(time& t)
{
    hours = t.hours;
    minutes =t.minutes;
    seconds = t.seconds;

    return *this;
}
```

Ví dụ: quá tải toán tử "+="

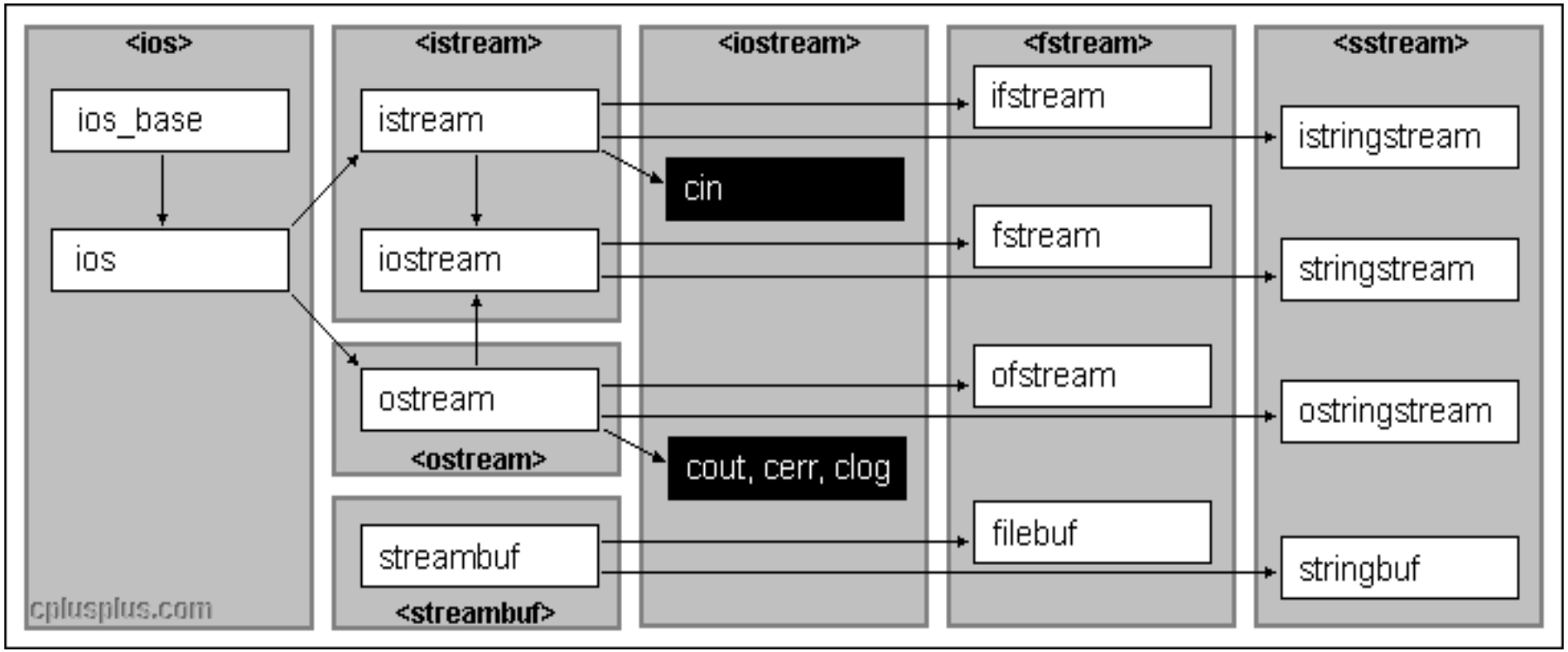


```
time&  time::operator+=(time t2)
{
   int   i1 = hours * 3600 + minutes * 60 + seconds;
   int   i2 = t2.hours * 3600 + t2.minutes * 60 + t2.seconds;
   int   s = i1 + i2;

   hours = s / 3600;
   s = s % 3600;
   minutes = s / 60;
   seconds = s % 60;

   return *this;
}
```

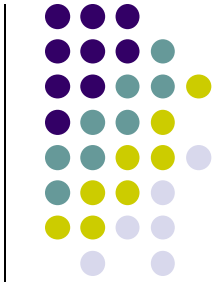
Ví dụ: quá tải toán tử “<<,>>“

```
friend ostream &operator<<(ostream &out, Complex c)      //output
{
        out<<"real part: "<<real<<"\n";
        out<<"imag part: "<<imag<<"\n";
        return out;
}
friend istream &operator>>(istream &in, Complex &c)      //input
{
        cout<<"enter real part:\n";
        in>>c.real;
        cout<<"enter imag part: \n";
        in>>c.imag;
        return in;
}
```

Các stream trong C++



Bài tập



- Viết class thể hiện Phân số, có thể +, -, *, /, các phép so sánh logic và sử dụng toán tử nhập xuất.
- Viết class thể hiện Số phức, quá tải các toán tử:
  - * nhân hai số phức
  - <<, >> xuất nhập số phức
  - ==, != so sánh hai số phức