

LESSON 8:

Overloading



- If any class have multiple functions with same names or we have multiple global function with same name but different parameters then they are said to be overloaded.

Nếu một tệp có nhiều hàm với cùng tên hoặc chúng ta có nhiều hàm toàn cục với cùng tên nhưng khác tham số thì chúng ta gọi là hàm quá tải.

- Different ways to Overload a Function
 - ✓ By changing number of Arguments.
 - ✓ By having different types of argument.

Các cách khác nhau để quá tải hàm:
- Bằng cách thay đổi số lượng tham số.
- Bằng cách có các loại tham số khác nhau.

Note: change return type of a function is not called overload.

Lưu ý: Thay đổi kiểu trả về của hàm không gọi là quá tải hàm.

■ What is Purpose?

- ✓ Function overloading is usually used to enhance the readability of the program and support developer auto analyze which function is called depend on parameters. Developer don't need remember too much function name.

giat ng

tính d c

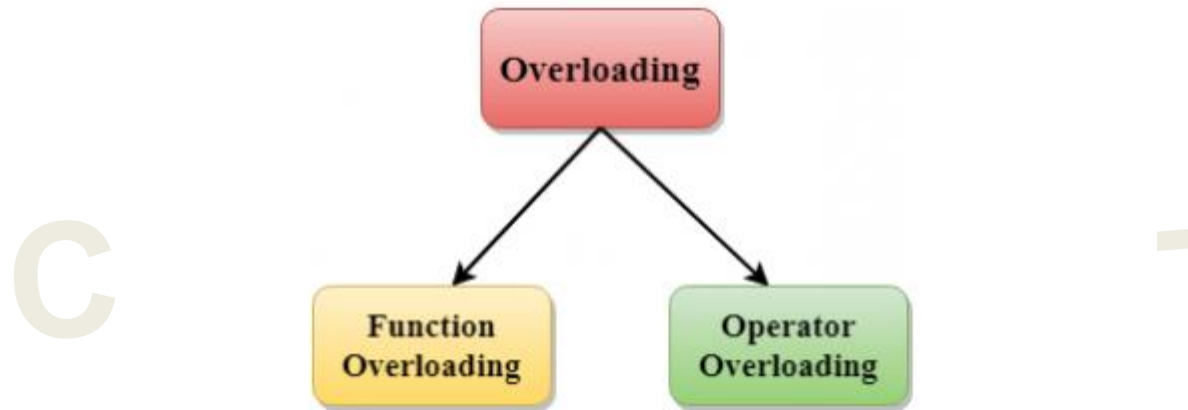
Quát i hàm th ng cs d ng t ng tính d c c cách ng trình và tr l p trình viên ng phân tích
hàm nào c g i d a trên các tham s .

```
// first definition
int sum(int x, int y)
{
    cout<< x+y;
}

// second overloaded defintion
double sum(double x, double y)
{
    cout << x+y;
}

int main()
{
    sum (10,20);
    sum(10.5,20.5);
}
```

- How many type of overloading ?



What is operator overloading ?

khái niệm

- ✓ Operator overloading is an important concept in C++.
- ✓ Overloaded operator is used to perform operation like +, -, /, * on user-defined data type.

Which operator can overloading ?

Toán tử phép toán có thể dùng để thực hiện các phép toán nh +, -, /, * trên các kiểu dữ liệu dùng để

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

Which operator can't overloading ?

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

toán tử thành viên lớp toán tử thành viên cấu trúc toán tử phạm vi toán tử kiểu union

■ What is purpose ?

✓ Using operator symbol for operands is user-defined. The expression become look clearly. Sử dụng ký hiệu toán học cho các toán hạng là ký hiệu dùng để giúp cho biểu thức trở nên rõ ràng hơn.

✓ Ex + operator work on operands of type char, int, float, double. If s1, s2, s3 are objects of class that user defined, then statement $s3 = s1 + s2$ will can't compile. If we overload operator+ for this class then we can build successfully.

Ví dụ: toán học + hoạt động trên các toán hạng có kiểu char, int, float, double. Nếu s1, s2, s3 là các đối tượng của lớp mà người dùng định nghĩa, thì câu lệnh $s3 = s1 + s2$ không thể biên dịch được. Nếu chúng ta quá tải toán tử + cho lớp này thì có thể biên dịch và thực thi thành công.

tri n khai

■ Two ways:

✓ Member function of user defined class Hàm thành viên có thể dùng như sau

- **Note:** Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function, Ex: >>, << operator.

Hàm quá tải toán tử có thể làm thành viên của lớp toán tử bên trái làm thì ngược lại phải. Tuy nhiên, nếu toán tử bên trái khác kiểu, thì hàm quá tải toán tử phải làm thành viên của lớp, ví dụ toán tử >>, <<.

✓ Non-member “stand-alone” function

Hàm không phải thành viên (stand-alone function)

✓ Operator overloading function can be made friend function if it needs access to the private and protected members of class.

Hàm quá tải toán tử có thể khai báo là bạn (friend function) nếu cần truy cập vào các thành viên riêng tư (private) và bảo vệ (protected) của lớp.

- **Note:** Some operator can't non-member function, ex: (), =, [], -> or any of assignment operator then it must be member function of class.

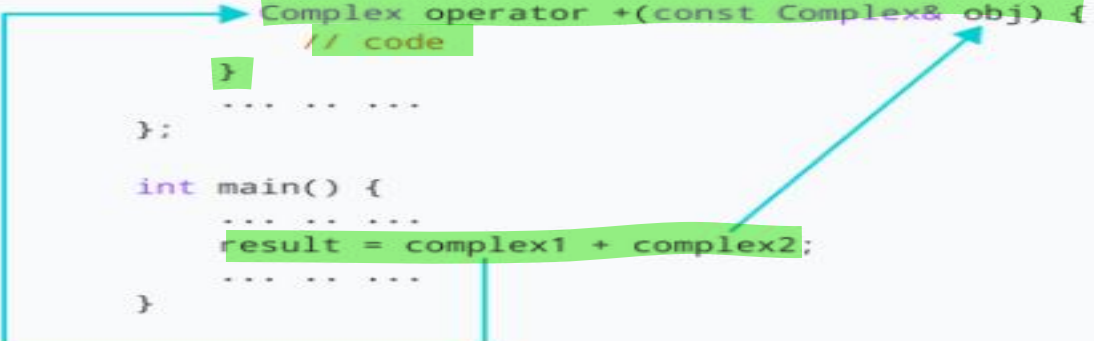
Lưu ý: Một số toán tử nhất định phải là thành viên của lớp, ví dụ như (), =, [], -> hoặc bất kỳ toán tử gán nào.

- Member function of user defined class

Hàm thành viên áp dụng nhúng

```
class Complex {  
    ...  
public:  
    ...  
    Complex operator +(const Complex& obj) {  
        // code  
    }  
    ...  
};  
  
int main() {  
    ...  
    result = complex1 + complex2;  
    ...  
}
```

function call from complex1



- Non-member “stand-alone” function

Hàm "stand-alone" không phải thành viên

ReturnType *operator* @(argument list)

```
friend phanso operator +(phanso a, phanso b)  
{  
    phanso c;  
    c.tu = a.tu * b.mau + a.mau * b.tu;  
    c.mau = a.mau * b.mau;  
    return c;  
}
```


- Precedence and Associativity of an operator cannot be changed.
Thứ tự ưu tiên và thứ tự phép toán không thay đổi: VD luôn là nhân chia trước cộng trừ sau
- Numbers of Operands cannot be changed. Unary operator remains unary, binary remains binary etc.
Số lượng toán hạng không thay đổi
- No new operators can be created, only existing operators can be overloaded.
Không thể tạo ra các toán tử mới
- Cannot redefine the meaning of a procedure. You cannot change how integers are added.
Không thể thay đổi ý nghĩa của hàm tính toán
không thay đổi cách mà các toán tử đã có nghĩa hoạt động Ví dụ, bạn không thể thay đổi cách cộng nguyên các số nguyên

Operator Overloading: I/O operator

- We can overload output operator << to print values for user defined data types. Quá trình toán xu t <<
- We can overload ⁱⁿput operator >> to input values for user defined data types. Quá trình toán nh p >>
- In case of input/output operator overloading, **left operand** will be of types ostream& and istream&
- Overloading these operators, we must **make sure that the functions must be a Non-Member function because left operand is not an object of the class.** Trong trường hợp quá trình toán nh p/xu t, toán bên trái s là ki u ostream& và istream&.

```
friend ostream &operator<<( ostream &output, const Distance &D ) {  
    output << "F : " << D.feet << " I : " << D.inches;  
    return output;  
}
```

```
friend istream &operator>>( istream &input, Distance &D ) {  
    input >> D.feet >> D.inches;  
    return input;  
}
```

```
int main() {  
    Distance D1(11, 10), D2(5, 11), D3;  
  
    cout << "Enter the value of object : " << endl;  
    cin >> D3;  
    cout << "First Distance : " << D1 << endl;  
    cout << "Second Distance : " << D2 << endl;  
}
```

Khi quá trình toán này, chúng ta phải nhớ rằng các hàm phải là hàm không phải thành viên vì toán bên trái không phải là một đối tượng của lớp.

left operand: bên trái của toán tử (phép + thì bên trái của toán tử s là 1 đối tượng => cần 1 member function)

Operator Overloading: Binary operator

- *objectA@objectB* -> Non-member "stand-alone" function

```
friend phanso operator +(phanso a, phanso b)
{
    phanso c;
    c.tu = a.tu * b.mau + a.mau * b.tu;
    c.mau = a.mau * b.mau;
    return c;
}
```

Non-member "stand-alone" function

↔ friend

phép cộng 2 phân số và tính toán

- *objectA.operator@(objectB)*

```
phanso operator +(phanso b)
{
    phanso c;
    c.tu = this->tu * b.mau + this->mau * b.tu;
    c.mau = this->mau * b.mau;
    return c;
}
```

member function

minh họa rõ ràng cho contr this

Operator Overloading: Assignment operator

toán t gán

- **Assignment operator** is used to copy the values from one object to another already existing object.

Toán t gán cs d ng sao chép các giá trị từ một đối tượng này sang một đối tượng khác cùng loại.

- Must use a member function *Phải dùng hàm thành viên:*

```
0
1 MyClass& MyClass::operator=(const MyClass &rhs)
2 {
3     if (this == &rhs)        // kiểm tra có cùng đối tượng?
4         return *this;        // Nếu trùng thì bỏ qua và return chính nó
5     //Xử lý... (Cấp phát vùng nhớ mới, sao chép giá trị, ...)
6     return *this;
7 }
```

1 cách khác s d ng contr this

các cách s d ng contr this, khi nào c n s d ng contr this

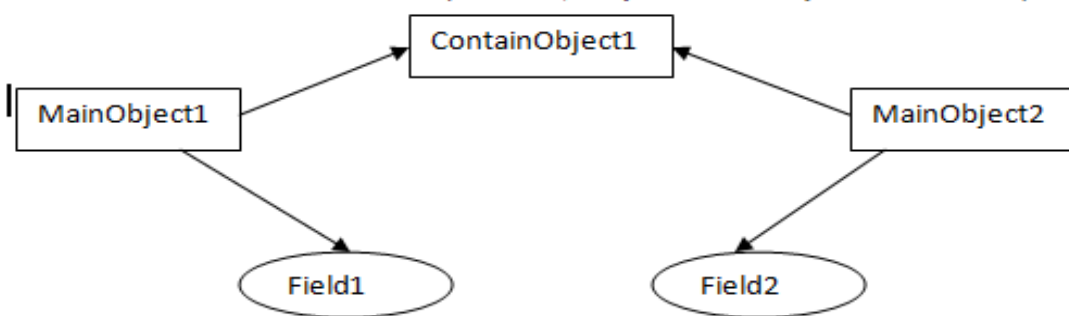
tránh xung đột tên: Khi tên hàm toán tử trùng với tên của thành viên hoặc biến khác thì giúp phân biệt rõ ràng giữa các thành viên của đối tượng và các biến khác.

Khi cần tham chiếu đến đối tượng: Nếu muốn tham chiếu đến đối tượng thì cần dùng `this` để chỉ đến đối tượng đang gọi.

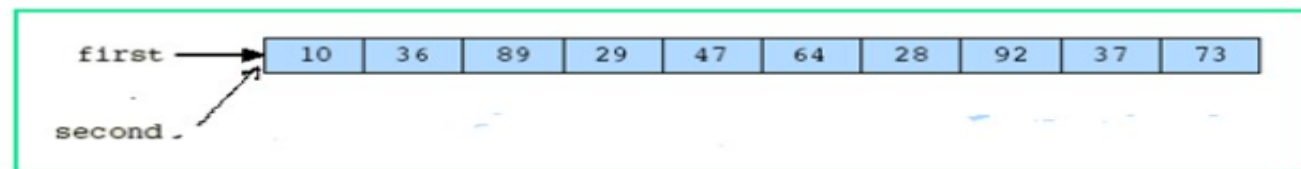
Trong các hàm thành viên non-static: `this` chỉ tồn tại trong các hàm thành viên non-static, vì nó phụ thuộc vào một đối tượng cụ thể.

có thể custom (\approx chainable)
`return *this;`

- Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied



```
int *first;  
int *second;  
first = new int[10]; // suppose that it is filled out  
second = first; // shallow copy
```



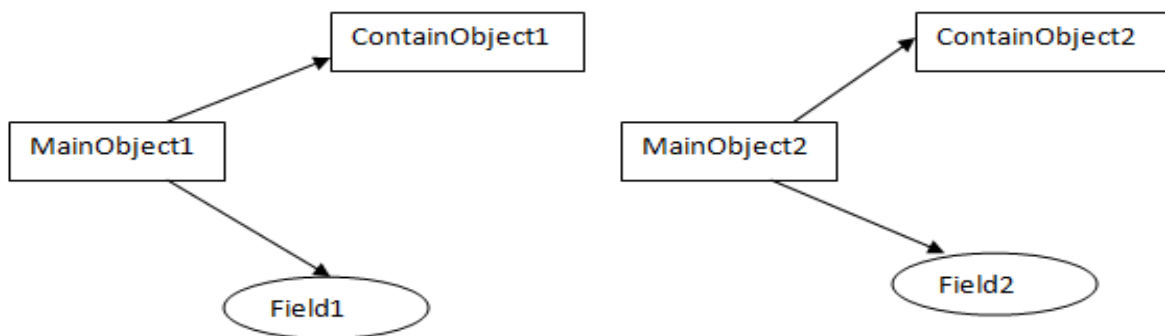
Shallow copy là vi c sao chép bit-by-bit c am t it ng M t it ng m i ct or a v i m t b n sao chính xác c a các giá tr trong it ng g c. N u b t k tr ng nào c a it ng là tham chi u t i các it ng khác, ch ách tham chi u c sao chép, nghĩa là ch sao chép ách b nh

i u này có nghĩa là hai it ng s tr n cùng m t vùng d li u trong b nh, vì c thay i giá tr trong ách này làm thay i giá tr t t c các it ng c shallow copy

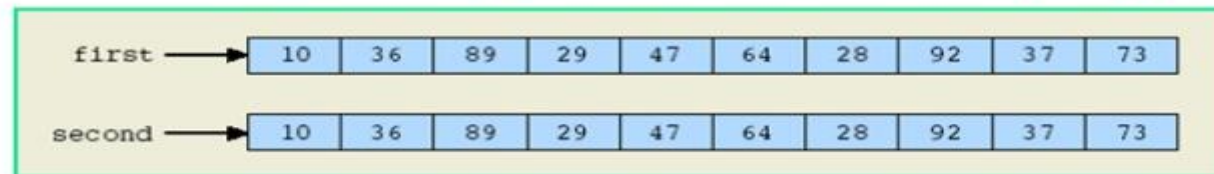
.Default copy constructor và default assignment operator ct ng t or a b i C++ n u b n không t nh ngh áchúng v i m c nh là Shallow copy, t c là các thành vi n ch tham chi u n cùng m t vùng nh ..

- A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers

Vì sao chép sâu (deep copy) sao chép tất cả các trường dữ liệu và các đối tượng mà nó tham chiếu. Do đó, khi tạo một bản sao sâu của một đối tượng, nó sẽ tạo ra một đối tượng mới và sao chép tất cả các trường dữ liệu và các đối tượng mà nó tham chiếu. Điều này đảm bảo rằng bản sao mới là một bản sao độc lập của đối tượng gốc.



```
int *first;  
int *second;  
first = new int[10]; // suppose that it is filled out  
second = new int[10];  
for (int j = 0; j < 10; j++)  
    second[j] = first[j]; // it produces a deep copy,
```



Deep copy (sao chép sâu) là việc tạo ra một bản sao mới của một đối tượng, sao chép tất cả các thành viên và các đối tượng mà nó tham chiếu. Điều này đảm bảo rằng bản sao mới là một bản sao độc lập của đối tượng gốc.

phần này mô tả về default copy constructor và default assignment operator trong class. Nó có tính là các phương thức con tr,

```
class MyClass {
public:
    int* data;

    // Default constructor
    MyClass() : data(new int(0)) {}

    // Deep copy assignment operator
    MyClass& operator=(const MyClass& other) {
        if (this != &other) {
            delete data; // Free existing data
            data = new int(*(other.data));
        }
        return *this;
    }

    // Destructor
    ~MyClass() { delete data; }
};
```

```
class MyClass {
public:
    int* data;

    // Default constructor
    MyClass() : data(new int(0)) {}

    // Deep copy constructor
    MyClass(const MyClass& other) : data(new int(*(other.data))) {}

    // Destructor
    ~MyClass() { delete data; }
};
```

Thank you

