Introduction to C++ Programming

Hàm bạn, Biến tĩnh và Chồng toán tử

Hàm bạn và lớp bạn - `friend`Functions and `friend` Classes

- **`friend`** function
  - Định nghĩa ngoài phạm vi của lớp
  - Có quyền truy cập tới cả thành phần không phải public
- Khai báo **`friend`**s
  - friend function
    - Thêm keyword friend trước khuôn mẫu hàm
  - Tất cả các hàm của class **`ClassTwo`** là **`friend`**s của class **`ClassOne`**
    - Thêm khai báo

      ```
      friend class ClassTwo;
      ```

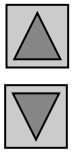
      trong định nghĩa của **`ClassOne`**

Hàm bạn và lớp bạn

- Tính chất của quan hệ friend:

  Class **B** là **friend** của class **A**

  - Không đối xứng
    - Class **B** **friend** của class **A**
    - Class **A** không nhất thiết là **friend** của class **B**
  - Không bắc cầu
    - Class **A** **friend** của class **B**
    - Class **B** **friend** của class **C**
    - Class **A** không nhất thiết là **friend** của Class **C**



Outline

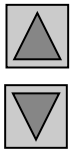
fig07_11.cpp
(1 of 3)

```
// Fig. 7.11: fig07_11.cpp
//Dich boi NgocDTB
// Friends co the truy cap vào cac thanh phan private cua 1 class
#include <iostream>
using std::cout;
using std::endl;
// Dinh nghia Count class
class Count {
   friend void setX( Count &, int );
public:
// ham tao
Count()  : x( 0 )  // khoi tao x bang 0
   {
      // than ham rong
       } // end constructor Count
// hien thi x
   void print() const
   {
      cout << x << endl;
   } // end function print
private:
   int x;  // du lieu
}; // end class Count
// ham setX co the thay doi du lieu private cua Count
// vi setX duoc dinh nghia la friend cua Count
void setX( Count &c, int val )
{
   c.x = val;  // cho phep vi: setX la friend cua Count
} // end function setX
int main(){
   Count counter;       // tao doi tuong Count
   cout << "counter.x sau khi khoi tao: ";
   counter.print();
   setX( counter, 8 );  // khoi tao x bang 1 friend
   cout << "counter.x sau khi goi ham ban setX: ";
  counter.print();
return 0;
```

Precede function prototype with keyword **`friend`**.

Dùng con trỏ this - `this` Pointer

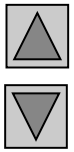
- **`this`** pointer
  - Cho phép đối tượng truy cập tới chính địa chỉ của nó
  - Kiểu của con trỏ **`this`** phụ thuộc vào
    - Kiểu đối tượng
    - **`this`** có kiểu **`Employee * const`**



Outline

fig07_13.cpp
(1 of 3)

```
// Fig. 7.13: fig07_13.cpp
//Dich boi NgocDTB
// Su dung con tro this de truy cau toi cac doi tuong thanh vien.
#include <iostream>
using std::cout;
using std::endl;
class Test {
public:
   Test( int = 0 );    // ham tao mac dinh
   void print() const;
private:
   int x;
}; // end class Test
// ham tao
Test::Test( int value )
   : x( value )  // khoi tao x = value
{
   // than ham rong
} // end constructor Test
// hien thi x su dung con tro this ngam va ro rang;
// can co dau ngoac quanh *this
```



Outline

fig07_13.cpp
(2 of 3)

```
void Test::print() const
{   // su dung con tro this ngam truy cap toi thanh phan x
  cout << "        x = " << x;
   // su dung con tro this de truy cap toi thanh phan x
   cout << "\n  this->x = " << this->x;
   // su dung con tro this de truy cap toi thanh phan x dung * va .
   cout << "\n(*this).x = " << ( *this ).x << endl;
} // end function print
int main()
{   Test testObject( 12 );
   testObject.print();
  return 0;
} // end main
```

Dùng con trỏ this

- Thay thế lời gọi hàm thành viên
  - Nhiều hàm được gọi trong cùng câu lệnh
  - Hàm trả về con trỏ tham chiếu tới chính đối tượng

```
{ return *this; }
```

  - Hàm khác thao tác trên con trỏ đó
  - Hàm không trả về tham chiếu phải được gọi cuối

Time6.h

Time6.cpp

fig07_16.cpp

Introduction to C++ Programming

Biến tĩnh - Static variable

`static` Class Members

- **`static`** class variable
  - "Class-wide" data
    - Dữ liệu chung của class, chứ không phải của đối tượng cụ thể
  - Dùng khi chỉ cần duy nhất 1 dữ liệu
    - Chỉ biến **`static`** cần được cập nhật
  - Tương tự như biến toàn cục nhưng có phạm vi class
    - Chỉ có thể truy cập được từ các đối tượng trong cùng class
  - Khởi tạo duy nhất 1 lần trong file
  - Có thể tồn tại ngay cả khi không có đối tượng nào
  - Có thể **`public`**, **`private`** or **`protected`**

`static` Class Members

- **Truy cập vào các biến `static` class**
  - Có thể truy cập từ bất kì đối tượng nào của class
  - **`public static`** variables
    - Có thể truy cập thông qua toán tử phạm vi (`::`)

      `Employee::count`
  - **`private static`** variables
    - Khi không có đối tượng nào
      - Chỉ có thể truy cập thông qua các phương thức **`public static`**
      - Để gọi phương thức **`public static`** , ta dùng cả tên lớp và toán tử phạm vi (`::`)

        `Employee::getCount()`

`static` Class Members

- **`static`** member functions
  - Không thể truy cập các phương thức/dữ liệu không phải là static
  - Không có con trỏ this cho **`static`** functions
    - **Thuộc tính và phương thức (dữ liệu và hàm)** **`static`** tồn tại độc lập với các đối tượng
    - Employee2.h
    - Employee2.cpp
    - fig07_19.cpp

Introduction to C++ Programming

Chồng toán tử- Operator Overloading

Giới thiệu

Sử dụng phép toán với các đối tượng (chồng toán tử - operator overloading)

Sáng sủa, dễ hiểu hơn là lời gọi hàm

Ví dụ

- +
  - Phép cộng trên các kiểu khác nhau (integers, floats, ...)

Khi nào thì sử dụng chồng toán tử?

Chồng toán tử

Kiểu

Có sẵn(**`int`**, **`char`**) hoặc người dùng định nghĩa

Có thể sử dụng các phép toán có sẵn với kiểu do người dùng định nghĩa

- Không được phép tạo phép toán mới

Chồng toán tử

Tạo ra 1 hàm cho class

Đặt têm hàm có chữ **`operator`** ngay trước ký hiệu phép toán

- **`Operator+`** cho phép **`+`**

Chồng toán tử

Sử dụng toán tử với các đối tượng của class

Phải được thực hiện trong class

Ngoại lệ:

- Phép gán, **=**
  - Có thể dùng mà không cần phải định nghĩa lại
  - Thực hiện gán từng thành phần giữa 2 đối tượng
- Phép lấy địa chỉ, **&**
  - Có thể dùng cho bất kỳ lớp mà không phải định nghĩa
  - Trả về địa chỉ của đối tượng
- Cả 2 đều có thể nạp chồng

Chồng toán tử dễ hiểu và ngắn gọn hơn hàm

- `object2 = object1.add(object2);`
- `object2 = object2 + object1;`

Ví dụ: chồng toán tử + trong class PhanSo

```
class PhanSo{
private:
    int tu, mau;
public:
    PhanSo(){tu= 0; mau= 1;}
    void nhap(){cout<<"Nhap phan so "; cin>>tu>>mau;}//chua
kiem tra mau!=0
    void hienthi(){cout<<tu<<"/"<<mau;}
    PhanSo operator+( PhanSo);
};
PhanSo PhanSo::operator+(PhanSo p){
    PhanSo kq;
    kq.tu = this->tu * p.mau + this->mau * p.tu;
    kq.mau = this->mau *  p.mau;
    return kq;
}
int main(){
    PhanSo a,b,kq;
    a.nhap(); b.nhap();
    kq = a + b;
    a.hienthi(); cout<< " + "; b.hienthi();
    cout<< " = "; kq.hienthi();
}
```

Lưu ý

Không thể thay đổi

Phép toán trên các kiểu có sẵn

- Ví dụ, định nghĩa lại phép cộng integer

Thứ tự ưu tiên của các phép toán

- Sử dụng dấu ngoặc () để thay đổi thứ tự ưu tiên

Tính kết hợp(trái sang phải hoặc phải sang trái)

Số toán hạng

- **&** là phép toán 1 toạn hạng (KHÔNG được nạp chồng 2 toán hạng)

Không được tạo phép toán mới (ký hiệu mới)

Phép toán phải nạp chồng chính xác

– **Phép +** không thể dùng cho **+=**

Các phép toán có thể nạp chồng

Phép toán có thể nạp chồng							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Phép toán KHÔNG thể nạp chồng				
.	.*	::	?:	sizeof

Hàm phép toán, thành phần class, hàm bạn

Hàm toán hạng

Hàm thành viên

- **`Sử dụng từ khoá this`** để lấy các thuộc tính của đối tượng hiện tại
- Lấy toán hạng phía trái của phép toán nhị phân (ví dụ +)
- Đối tượng bên trái phép toán phải là cùng class như toán hạng

Hàm không phải thành viên (không ở trong class)

- Cần tham số ở cả 2 bên phép toán
- Có thể có đối tượng khác class với phép toán
- Phải là **`friend`** để truy cập được vào dữ liệu **`private`** hay **`protected`**

Hàm phép toán, thành phần class, hàm bạn

- **Nạp chồng phép <<**

Toán hạng bên trái có kiểu **ostream &**

- **Ví dụ đối tượng cout** trong **cout << classObject**

– **Tương tự, nạp chồng >>** cần **istream &**

Do vậy, cả hai đều không là hàm thành viên của class nào cả

Hàm phép toán, thành phần class, hàm bạn

Phép toán giao hoán

Ta muốn **+** có tính giao hoán

- Nghĩa là "**a + b**" và "**b + a**" có kết quả như nhau

Giả thiết ta có phép toán trên 2 lớp khác nhau

Chồng toán tử chỉ có thể là hàm thành viên của lớp ở bên trái

- **HugeIntClass + Long int**
- Có thể là hàm thành viên

Nếu viết như dưới, ta cần chồng toán tử không phải hàm thành viên

- **Long int + HugeIntClass**

Overloading Stream-Insertion and Stream-Extraction Operators

- **<<** và **>>**

Đã được nạp chồng để xử lý các kiểu có sẵn

Có thể xử lý các class do người dùng định nghĩa

Ví dụ

Class **`PhoneNumber`**

- Lưu 1 số điện thoại

In ra định dạng sau

- **`(123) 456-7890`**

```cpp
#include <iostream>
#include <iomanip>
using namespace std;
using std::setw;
class PhoneNumber {
    friend ostream &operator<<( ostream&, const PhoneNumber & );
    friend istream &operator>>( istream&, PhoneNumber & );
private:
    char areaCode[ 4 ];  // 3-ki tu cho ma vung va 1 ky tu null
    char exchange[ 4 ];  // 3-ki tu sdt va ky tu null
    char line[ 5 ];      // 4-ki tu cuoi sdt va ky tu null

}; // ket thuc class PhoneNumber

// nap chong toan tu << khong the la
// mot ham thanh vien neu ta muon goi voi cout: cout << somePhoneNumber;
ostream &operator<<( ostream &output, const PhoneNumber &num )
{
    output << "(" << num.areaCode << ") "
           << num.exchange << "-" << num.line;
    return output;
} // ket thuc ham operator<<
```

```
istream &operator>>( istream &input, PhoneNumber &num )
{
        input.ignore();                         // bo qua (
        input >> setw( 4 ) >> num.areaCode; // ma vung
        input.ignore( 2 );                      // bo qua ) va dau cach
        input >> setw( 4 ) >> num.exchange; // nhap 3 so dau va bo qua dau
-
        input >> setw( 5 ) >> num.line;     // nhap 4 so cuoi
        return input;       // cho phep cin >> a >> b >> c;

} // Ket thuc ham operator>>
int main()
{
        PhoneNumber phone; // tao doi tuong phone
        cout << "Nhập số điện thoại theo định dạng (123) 456-7890:\n";
        // cin >> phone goi  operator>>( cin, phone )
        cin >> phone;
        cout << "Số điện thoại đã nhập: " ;
        // cout << phone goi operator<<( cout, phone )
        cout << phone << endl;
        return 0;
} // end main
```