

Kỹ thuật lập trình

Phần III: Lập trình tổng quát

Chương 9:

Khuôn mẫu hàm và khuôn mẫu lớp



Nội dung chương 9



9.1 Khuôn mẫu hàm

- Vai trò của khuôn mẫu hàm
- Định nghĩa khuôn mẫu hàm
- Sử dụng khuôn mẫu hàm

9.2 Khuôn mẫu lớp

- Định nghĩa khuôn mẫu lớp
- Dẫn xuất khuôn mẫu lớp
- Ví dụ khuôn mẫu lớp Vector

9.1 Khuôn mẫu hàm (function template)

- Vấn đề: Nhiều hàm chỉ khác nhau về kiểu dữ liệu tham số áp dụng, không khác nhau về thuật toán
- Ví dụ:

```
int max(int a, int b) {  
    return (a > b)? a : b;  
}  
double max(double a, double b) {  
    return (a > b)? a : b;  
}  
...
```
- Các ví dụ khác: các hàm swap, sort, find, select,...
- Bản chất của vấn đề? Nằm ở ngôn ngữ lập trình còn thấp, chưa gần với tư duy của con người!
- Giải pháp: Tổng quát hóa các hàm chỉ khác nhau về kiểu dữ liệu áp dụng thành **khuôn mẫu hàm**.

Định nghĩa khuôn mẫu hàm

- Ví dụ tổng quát hóa hàm max để có thể áp dụng cho nhiều kiểu dữ liệu khác nhau:

```
template <typename T>  
T max(T a, T b) {  
    return (a > b)? a : b;  
}
```

- Ví dụ tổng quát hóa hàm swap:

```
template <class X>  
void (X& a, X& b) {  
    X temp = a;  
    a = b;  
    b = temp;  
}
```

Sử dụng từ khóa **typename** hoặc **class** để khai báo **tham số khuôn mẫu**

- Một khuôn mẫu hàm inline:

```
template <typename T>  
inline T max(T a, T b) { return (a > b)? a : b;}
```

Khai báo và sử dụng khuôn mẫu hàm

- Ví dụ sử dụng khuôn mẫu hàm max

```
template <class T> T max(T a, T b);  
template <class T> void swap(T&, T&);
```

Khuôn mẫu hàm

```
void main() {
```

```
    int N1 = 5, N2 = 7;
```

```
    double D1 = 5.0, D2 = 7.0;
```

```
    int N = max(N1,N2);    // max<int>(int,int)
```

```
    char c = max('c','a'); // max<char>(char, char)
```

```
    double D = max(D1,D2); // max<double>(double, double)
```

```
    swap(N1,N2);           // swap<int>(int&,int&)
```

```
    swap(D1,D2);           // swap<double>(double&,double&)
```

```
    D = max(D1,A1);         // error: ambiguous
```

```
    N = max('c',A1);        // error: ambiguous
```

```
    D = max<double>(D1,A1); // OK: explicit qualification
```

```
    N = max<int>('c',A);     // OK: explicit qualification
```

```
}
```

Hàm khuôn mẫu

Khả năng áp dụng khuôn mẫu hàm

- Khả năng áp dụng một khuôn mẫu hàm là vô tận, nhưng không phải áp dụng được cho tất cả các đối số khuôn mẫu

Ví dụ: Điều kiện ràng buộc đối với kiểu dữ liệu có thể áp dụng trong khuôn mẫu hàm max là phải có phép so sánh lớn hơn (>):

```
template <class T>
```

```
inline T max(T a, T b) { return (a > b)? a : b;}
```

=> Đối với các kiểu dữ liệu mới, muốn áp dụng được thì cần phải nạp chồng toán tử so sánh >

- **Tuy nhiên**, khả năng áp dụng được chưa chắc đã có ý nghĩa
- Ví dụ: Xác định chuỗi ký tự đứng sau trong hai chuỗi cho trước theo vần ABC

```
char city1[] = "Ha Noi", city2[] = "Hai Phong";
```

```
char* city = max(city1,city2); // ???
```

```
// max<char*>(char*,char*)
```

Nạp chồng khuôn mẫu hàm

- Một khuôn mẫu hàm có thể được nạp chồng bằng hàm cùng tên...

```
char* max(char* a, char* b) { if (strcmp(a,b))... }  
void f() {  
    char c = max('H', 'K');           // max<char>(char, char)  
    char city1[] = "Ha Noi", city2[] = "Hai Phong";  
    char* city = max(city1, city2);    // max(char*, char*)  
    ...  
}
```

- ...hoặc bằng một khuôn mẫu hàm cùng tên (khác số lượng các tham số hoặc kiểu của ít nhất một tham số), ví dụ:

```
template <class T> T max(T a, T b, T c)    {...}  
template <class T> T max(T* a, int n)     {...}
```

nhưng không được như thế này:

```
template <class X> X max(X a, X b)        {...}
```

Tham số khuôn mẫu

- Tham số khuôn mẫu hàm có thể là một kiểu cơ bản hoặc một kiểu dẫn xuất, nhưng không thể là một biến hoặc một hằng số:

```
template <class T> max(T a, T b) { ... }    // OK  
template <int N> max(int* a)    { ... }    // error
```

- Một khuôn mẫu hàm có thể có hơn một tham số kiểu:

```
template <class A, class B> void swap(A& a, B& b) {  
    A t = a;  
    a = b;    // valid as long as B is compatible to A  
    b = t;    // valid as long as A is compatible to B  
}  
void f() {  
    double a = 2.0;  
    int b = 3;  
    swap(a,b);    // swap<double,int>(double&,int&)  
    swap(b,a);    // swap<int,double>(int&, double&)  
}
```


- Thông thường, tham số khuôn mẫu xuất hiện ít nhất một lần là kiểu hoặc kiểu dẫn xuất trực tiếp của các tham biến:
template <class X> void f1(X a, int b) {...}
template <class X> void f2(X* b) {...}
template <class X, class Y> void f3(Y& a, X b) {...}
- Theo chuẩn ANSI/ISO C++, tham số khuôn mẫu không bắt buộc phải xuất hiện trong danh sách tham biến, nhưng cần lưu ý khi sử dụng. Ví dụ

```
#include <stdlib.h>
template <class X> X* array_alloc(int nelem) {
    return (X*) malloc(nelem*sizeof(X));
}
void main() {
    double* p1 = array_alloc(5);           // error!
    double* p2 = array_alloc<double>(5);    // OK!
    ...
    free(p2);
}
```

Khuôn mẫu hàm và hàm khuôn mẫu

- Khuôn mẫu hàm chỉ đưa ra cách thức thực hiện và sử dụng một thuật toán nào đó một cách tổng quát
- Trong khi biên dịch khuôn mẫu hàm, compiler chỉ kiểm tra về cú pháp, không dịch sang mã đích
- Mã hàm khuôn mẫu được compiler tạo ra (dựa trên khuôn mẫu hàm) khi và chỉ khi khuôn mẫu hàm được sử dụng với kiểu cụ thể
- Nếu một khuôn mẫu hàm được sử dụng nhiều lần với các kiểu khác nhau, nhiều hàm khuôn mẫu sẽ được tạo ra tương ứng
- Nếu một khuôn mẫu hàm được sử dụng nhiều lần với các kiểu tương ứng giống nhau, compiler chỉ tạo ra một hàm khuôn mẫu.

Ưu điểm của khuôn mẫu hàm

- Tiết kiệm được mã nguồn => dễ bao quát, dễ kiểm soát lỗi, nâng cao hiệu quả lập trình
- Đảm bảo được tính chặt chẽ về kiểm tra kiểu mạnh trong ngôn ngữ lập trình (hơn hẳn sử dụng macro trong C)
- Tính mở, nâng cao giá trị sử dụng lại của phần mềm: thuật toán viết một lần, sử dụng vô số lần
- Đảm bảo hiệu suất tương đương như viết tách thành từng hàm riêng biệt
- Cho phép xây dựng các thư viện chuẩn rất mạnh (các thuật toán thông dụng như sao chép, tìm kiếm, sắp xếp, lựa chọn,)

Nhược điểm của khuôn mẫu hàm

- Nếu muốn đảm bảo tính mở hoàn toàn thì người sử dụng khuôn mẫu hàm cũng phải có mã nguồn thực thi
 - Mã nguồn thực thi cần được đặt trong header file
 - Khó bảo vệ chất xám
- Việc theo dõi, tìm lỗi biên dịch nhiều khi gặp khó khăn
 - Lỗi nhiều khi nằm ở mã sử dụng, nhưng lại được báo trong mã định nghĩa khuôn mẫu hàm
 - Ví dụ: Compiler không báo lỗi ở dòng lệnh sau đây, mà báo lỗi ở phần định nghĩa hàm max, tại phép toán so sánh lớn hơn không được định nghĩa cho kiểu Complex:
Complex a, b;
...
Complex c = max(a,b);
- Định nghĩa và sử dụng không đúng cách có thể dẫn tới gia tăng lớn về mã đích, bởi số lượng hàm khuôn mẫu có thể được tạo ra quá nhiều không cần thiết.

Ví dụ: khuôn mẫu hàm copy

```
template <class S, class D>
void copy(const S * s, D* d, int n) {
    while (n--)
        *d++ = *s++;
}

void main() {
    int a[] = {1,2,3,4,5,6,7};
    double b[10];
    float c[5];
    copy(a,b,7);           // copy<int,double>(a,b,7)
    copy(b,c,5);           // copy<double,float>(b,c,5);
    ...
}
```

9.2 Khuôn mẫu lớp (class template)

- Nhiều cấu trúc dữ liệu như Point, Complex, Vector, List, Map,... trước kia vẫn phải được định nghĩa riêng cho từng kiểu dữ liệu phần tử cụ thể, ví dụ DoubleComplex, FloatComplex, DoubleVector, IntVector, ComplexVector, DateList, MessageList, ...
- Cách thực hiện mỗi cấu trúc thực ra giống nhau, nói chung không phụ thuộc vào kiểu phần tử cụ thể

```
class IntPoint { int x,y;  
public: IntPoint(int x0, int y0) : x(x0), y(y0) {}  
    ...  
};  
class DoublePoint { double x,y;  
public: DoublePoint(double x0, double y0) : x(x0), y(y0) {}  
    ...  
};
```

Định nghĩa khuôn mẫu lớp

```
// Point.h
template <typename T>
class Point {
    T x, y;
public:
    Point(): x(0), y(0) {}
    Point(T x0, T y0) : x(x0), y(y0) {}
    Point(const Point&);
    void move(T dx, T dy) { x += dx; y += dy; }
    bool inRect(Point p1, Point p2);
    //...
};

template <class T>
Point<T>::Point(const Point<T>& b) : x(b.x), y(b.y) {}

template <class Coord>
bool Point<Coord>::inRect(Point<Coord> p1, Point<Coord> p2) {
    return (x >= p1.x && x <= p2.x || x >= p2.x && x <= p1.x) &&
           (y >= p1.y && y <= p2.y || y >= p2.y && x <= p1.y);
}
```

Tham số khuôn mẫu:
Kiểu hoặc hằng số

Mỗi hàm thành viên của một khuôn mẫu lớp là một khuôn mẫu hàm

Sử dụng khuôn mẫu lớp: Lớp khuôn mẫu

```
#include "Point.h"
void main() {
    Point<int> A1(5,5), A2(10,10);
    Point<int> A3(A1);
    while (A3.inRect(A1,A2))
        A3.move(2,3);
    typedef Point<float> FPoint;
    FPoint B1(5.0,5.0), B2(10.0,10.0);
    FPoint B3(B1);
    while (B3.inRect(B1,B2))
        B3.move(2,3);
    // ...
    Point<double> C1(B1);    // error
    if (A3.inRect(B1,B2))    // error
        ; // ...
}
```


Những kiểu nào có thể áp dụng?

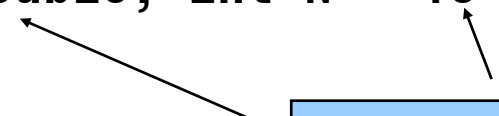
- Khả năng áp dụng của kiểu là vô tận, tuy nhiên không có nghĩa là áp dụng được cho tất cả các kiểu
- Một kiểu muốn áp dụng được phải hỗ trợ các phép toán được sử dụng trong mã thực thi khuôn mẫu lớp.
- Ví dụ khuôn mẫu lớp Point yêu cầu kiểu tọa độ phải áp dụng được các phép toán sau đây:
 - Chuyển đổi từ số nguyên (trong hàm tạo mặc định)
 - Sao chép (trong hàm tạo thứ hai và hàm tạo bản sao)
 - Toán tử += (trong hàm move)
 - Các phép so sánh >=, <= (trong hàm inRect)
- Việc kiểm tra kiểu được tiến hành khi sử dụng hàm thành viên của lớp khuôn mẫu, nếu có lỗi thì sẽ được báo tại **mã nguồn thực thi khuôn mẫu lớp**

Tham số khuôn mẫu: kiểu hoặc hằng số

```
template <class T = double, int N = 10>
class Array {
    T data[N];
public:
    Array(const T& x = T(0));
    int size() const { return N; }
    T operator[](int i) const { return data[i]; }
    T& operator[](int i)      { return data[i]; }
    //...
};

template <class T, int N> Array<T,N>::Array(const T& x) {
    for (int i=0; i < N; ++ i) data[i] = x;
}

void main() {
    Array<double,10> a;
    Array<double>    b; // same as above
    Array<>          c; // same as above
    //...
}
```



Tham số mặc định

Dẫn xuất từ khuôn mẫu lớp

```
template <class T> class IOBuffer : public Array<T,8> {
public:
    IOBuffer(T x) : Array<T,8>(x) {}
    //...
};
class DigitalIO : public IOBuffer<bool> {
public:
    DigitalIO(bool x) : IOBuffer<bool>(x) {}
    //...
};
class AnalogIO : public IOBuffer<unsigned short> {
    typedef IOBuffer<unsigned short> BaseClass;
public:
    AnalogIO(unsigned short x) : BaseClass(x) {}
    //...
};
void main() {
    IOBuffer<double> delayBuf(0);
    DigitalIO di(false);
    AnalogIO ao(0); //...
}
```

Ví dụ khuôn mẫu lớp Vector

```
template <class T>
class Vector {
    int nelem;
    T* data;
public:
    Vector() : nelem(0), data(0) {}
    Vector(int n, T d);
    Vector(int n, T *array);
    Vector(const Vector<T>&);
    ~Vector();
    int size() const { return nelem; }
    T operator[](int i) const      { return data[i]; }
    T& operator[](int i)           { return data[i]; }
private:
    void create(int n) { data = new T[nelem=n]; }
    void destroy()     { if (data != 0) delete [] data; }
};
```

```

template <class T> Vector<T>::Vector(int n, T d) {
    create(n);
    while (n-- > 0)
        data[n] = d;
}
template <class T> Vector<T>::Vector(int n, T* p) {
    create(n);
    while (n-- > 0)
        data[n] = p[n];
}
template <class T> Vector<T>::~~Vector() { destroy(); }

template <class T>
Vector<T>::Vector(const Vector<T>& a) {
    create(a.nelem);
    for (int i=0; i < nelem; ++i)
        data[i] = a.data[i];
}

```

```
#include "Point.h"
#include "Vector.h"
void main()
    Vector<double> v(5,1.0);
    double d = v[0];
    v[1] = d + 2.0;
    Vector<double> v2(v);
    //...
    int b[] = {1,2,3,4,5};
    Vector<int> a(5,b);
    int n = a[0];
    a[1] = n + 2;
    Vector<int> a2(a);
    //...
    typedef Vector<Point<double> > Points;
    Points lines(5,Point<double>(0.0,0.0));
    lines[0] = Point<double>(5.0,5.0);
    //...
}
```

Bài tập về nhà

- Xây dựng một khuôn mẫu hàm xác định vị trí (địa chỉ) của phần tử có giá trị lớn nhất xuất hiện đầu tiên trong một dãy số. Viết chương trình minh họa sử dụng cho hai kiểu số liệu cụ thể.
- Từ bài tập xây dựng lớp MessageList, tổng quát hóa thành một khuôn mẫu lớp có tên là List với các phép toán (hàm thành viên) cần thiết