



# NGÔN NGỮ LẬP TRÌNH

## Bài 7: Khuôn mẫu

Giảng viên: Lý Anh Tuấn

Email: [tuanla@tlu.edu.vn](mailto:tuanla@tlu.edu.vn)

# Nội dung

## 1. Khuôn mẫu hàm

- Cú pháp, định nghĩa
- Sự biên dịch

## 2. Khuôn mẫu lớp

- Cú pháp
- Ví dụ: lớp khuôn mẫu mảng

## 3. Khuôn mẫu và kế thừa

- Ví dụ: lớp khuôn mẫu mảng nhập giá trị một phần

# Giới thiệu

- Khuôn mẫu C++
  - Cho phép các định nghĩa tổng quát cho hàm và lớp
  - Tên kiểu làm tham số thay vì kiểu thực sự
  - Định nghĩa chính xác được quyết định ở thời điểm chạy

- Nhắc lại hàm swapValues:

```
void swapValues(int& var1, int& var2)
{
    int temp;
    temp = var1;
    var1 = var2;
    var2 = temp;
}
```

- Chỉ áp dụng cho các biến kiểu int
- Nhưng phần mã lệnh làm việc với bất kỳ kiểu nào

# Khuôn mẫu hàm vs. Nạp chồng

- Có thể nạp chồng hàm cho kiểu char:  

```
void swapValues(char& var1, char& var2)
{
    char temp;
    temp = var1;
    var1 = var2;
    var2 = temp;
}
```
- Lưu ý: Mã lệnh gần giống nhau
  - Chỉ khác nhau về kiểu được sử dụng ở 3 vị trí

# Cú pháp khuôn mẫu hàm

- Cho phép “hoán đổi giá trị” cho bất kỳ kiểu biến nào:

```
template<class T>
void swapValues(T& var1, T& var2)
{
    T temp;
    temp = var1;
    var1 = var2;
    var2 = temp;
}
```

- Dòng đầu tiên là tiền tố khuôn mẫu:
  - Báo cho bộ biên dịch biết đằng sau là khuôn mẫu
  - Và T là một tham số kiểu

# Tiền tổ khuôn mẫu

`template<class T>`

- Ở đây, class nghĩa là kiểu, hoặc sự phân lớp
- Dễ bị nhầm lẫn với từ class được sử dụng rộng rãi
  - C++ cho phép sử dụng từ khóa “typename” ở vị trí từ khóa class
  - Tuy nhiên trong mọi trường hợp nên sử dụng class
- T có thể được thay bằng bất kỳ kiểu nào
  - Kiểu được định nghĩa trước hoặc người dùng định nghĩa
- Trong thân định nghĩa hàm
  - T được sử dụng giống như một kiểu bất kỳ

# Định nghĩa khuôn mẫu hàm

- Khuôn mẫu hàm `swapValues()` thực sự là một tập hợp các định nghĩa
  - Một định nghĩa cho mỗi kiểu có thể có
- Bộ biên dịch chỉ phát sinh các định nghĩa khi được yêu cầu
  - Với điều kiện bạn đã định nghĩa cho tất cả các kiểu
- Viết một định nghĩa → làm việc cho tất cả các kiểu có thể có

# Gọi khuôn mẫu hàm

- Xét lời gọi hàm sau đây  
`swapValues(int1, int2);`
  - Bộ biên dịch C++ sử dụng khuôn mẫu để khởi tạo định nghĩa hàm cho hai tham số int
- Tương tự như tất cả các kiểu khác
- Không cần làm điều gì đặc biệt trong lời gọi
  - Định nghĩa cần thiết được phát sinh tự động



# Một khuôn mẫu hàm khác

- Khai báo/nguyên mẫu:

```
template<class T>  
void showStuff(int stuff1, T stuff2, T stuff3);
```

- Định nghĩa

```
template<class T>  
void showStuff(int stuff1, T stuff2, T stuff3)  
{  
    cout << stuff1 << endl  
        << stuff2 << endl  
        << stuff3 << endl;  
}
```

# Lời gọi showStuff

- Xét lời gọi hàm:  
showStuff(2, 3.3, 4.4);
- Bộ biên dịch phát sinh định nghĩa hàm
  - Thay T bằng double
  - Vì tham số thứ hai có kiểu double
- Hiển thị:  
2  
3.3  
4.4

# Sự biên dịch

- Khai báo và định nghĩa hàm
  - Chúng ta thường tách rời chúng
  - Với các khuôn mẫu → việc này không được hỗ trợ trong hầu hết các bộ biên dịch
- An toàn nhất là đặt định nghĩa hàm khuôn mẫu trong file mà nó được gọi
  - Nhiều bộ biên dịch yêu cầu nó xuất hiện ở vị trí đầu tiên
  - Chúng ta thường `#include` tất cả các định nghĩa khuôn mẫu

# Khuôn mẫu đa tham số kiểu

- Có thể có:  
`template<class T1, class T2>`
- Không đặc thù:
  - Thường chỉ cần một kiểu có thể thay thế
  - Không cho phép có tham số khuôn mẫu không được sử dụng
    - Mỗi tham số khuôn mẫu cần được sử dụng trong định nghĩa
    - Bằng không chương trình dịch sẽ báo lỗi

# Trừu tượng hóa thuật toán

- Liên quan đến việc thi hành khuôn mẫu
- Biểu diễn thuật toán theo cách chung nhất:
  - Thuật toán áp dụng cho các biến thuộc bất kỳ kiểu nào
  - Bỏ qua chi tiết không thiết yếu
  - Tập trung vào các phần trọng yếu của thuật toán
- Khuôn mẫu hàm là một cách C++ hỗ trợ trừu tượng hóa thuật toán

# Chiến lược định nghĩa khuôn mẫu

- Phát triển hàm như thông thường
  - Sử dụng các kiểu dữ liệu thật
- Hoàn thành việc gỡ lỗi hàm nguyên bản
- Sau đó chuyển đổi thành khuôn mẫu
  - Thay thế các tên kiểu bằng tham số kiểu khi cần
- Ưu điểm:
  - Dễ giải quyết trường hợp cụ thể
  - Tập trung vào thuật toán, thay vì cú pháp khuôn mẫu

# Các kiểu không phù hợp trong khuôn mẫu

- Có thể sử dụng bất kỳ kiểu nào trong khuôn mẫu làm cho mã lệnh có nghĩa
  - Mã lệnh phải vận hành theo cách phù hợp
- Ví dụ, hàm khuôn mẫu `swapValues()`
  - Không thể sử dụng kiểu mà toán tử gán chưa được định nghĩa cho nó
  - Ví dụ: một mảng:  
`int a[10], b[10];`  
`swapValues(a, b);`
  - Các mảng không được phép gán

# Khuôn mẫu lớp

- Cũng có thể “khái quát hóa” các lớp `template<class T>`
  - Có thể áp dụng cho định nghĩa lớp
  - Tất cả các bản thể của T trong định nghĩa lớp được thay thế bằng tham số kiểu
  - Giống như với các khuôn mẫu hàm
- Một khi khuôn mẫu được định nghĩa, có thể khai báo các đối tượng của lớp



# Định nghĩa khuôn mẫu lớp

- ```
template<class T>
class Pair
{
public:
    Pair();
    Pair(T firstVal, T secondVal);
    void setFirst(T newVal);
    void setSecond(T newVal);
    T getFirst() const;
    T getSecond() const;
private:
    T first; T second;
};
```

# Các thành viên lớp khuôn mẫu Pair

- ```
template<class T>
Pair<T>::Pair(T firstVal, T secondVal)
{
    first = firstVal;
    second = secondVal;
}
template<class T>
void Pair<T>::setFirst(T newVal)
{
    first = newVal;
}
```

# Lớp khuôn mẫu Pair

- Các đối tượng của lớp có “cặp” giá trị kiểu T
- Sau đó có thể khai báo các đối tượng:  
Pair<int> score;  
Pair<char> seats;
  - Sau đó có thể sử dụng các đối tượng giống như các đối tượng bất kỳ
- Ví dụ sử dụng:  
score.setFirst(3);  
score.setSecond(0);

# Định nghĩa hàm thành viên Pair

- Lưu ý trong định nghĩa hàm thành viên:
  - Bản thân mỗi định nghĩa là một khuôn mẫu
  - Đòi hỏi tiền tố khuôn mẫu trước mỗi định nghĩa
  - Tên lớp trước :: là `Pair<T>` thay vì chỉ là `Pair`
  - Nhưng tên hàm tạo chỉ là `Pair`
  - Tên hàm hủy cũng chỉ là `~Pair`

# Khuôn mẫu lớp làm tham số

- Xét:

`int addUP(const Pair<int>& thePair);`

- Kiểu (int) được cung cấp để sử dụng cho T trong định nghĩa tham số kiểu lớp này
- Ở đây xảy ra truyền tham chiếu
- Kiểu khuôn mẫu có thể được sử dụng bất cứ chỗ nào cho phép sử dụng các kiểu chuẩn

# Khuôn mẫu lớp trong khuôn mẫu hàm

- Thay vì định nghĩa nạp chồng mới:  
template<class T>  
T addUp(const Pair<T>& thePair);  
//Tiền điều kiện:Toán tử + được định nghĩa  
cho các giá trị kiểu T  
//Trả về tổng của hai giá trị trong thePair
- Hàm bây giờ áp dụng cho tất cả các kiểu số

# Các hạn chế trên tham số kiểu

- Chỉ các kiểu hợp lý có thể thay thế cho T
- Xét:
  - Toán tử gán phải hoạt động tốt
  - Hàm tạo sao chép cũng phải hoạt động tốt
  - Nếu T bao gồm con trỏ thì hàm hủy phải phù hợp
- Các vấn đề tương tự như khuôn mẫu hàm

# Các định nghĩa kiểu

- Có thể định nghĩa tên kiểu lớp mới
  - Để biểu diễn tên khuôn mẫu lớp được đặc tả
- Ví dụ:  
`typedef Pair<int> PairOfInt;`
- Tên “PairOfInt” bây giờ được sử dụng để khai báo các đối tượng kiểu Pair<int>:  
`PairOfInt pair1, pair2;`
- Tên cũng có thể được sử dụng làm tham số, hoặc ở bất kỳ chỗ nào cho phép các tên kiểu



# Hàm bạn và Khuôn mẫu

- Hàm bạn có thể được sử dụng với các lớp khuôn mẫu
  - Giống như các lớp nguyên bản
  - Chỉ đòi hỏi tham số kiểu ở vị trí phù hợp
- Việc khuôn mẫu lớp có hàm bạn là rất phổ biến
  - Đặc biệt trong việc nạp chồng toán tử

# Lớp khuôn mẫu định nghĩa trước

- Lớp vector là một lớp khuôn mẫu
- Một ví dụ khác: Lớp khuôn mẫu `basic_string`
  - Xử lý các chuỗi phần tử có kiểu bất kỳ
  - Ví dụ:

<code>basic_string&lt;char&gt;</code>	làm việc với kiểu <code>char</code>
<code>basic_string&lt;double&gt;</code>	làm việc với kiểu <code>double</code>
<code>basic_string&lt;YourClass&gt;</code>	làm việc với các đối tượng <code>YourClass</code>
  - `string` là tên thay thế của `basic_string<char>`
  - `basic_string` được định nghĩa trong thư viện `<string>`

# Khuôn mẫu và kế thừa

- Các lớp khuôn mẫu dẫn xuất
  - Có thể dẫn xuất từ lớp khuôn mẫu hoặc không khuôn mẫu
  - Lớp dẫn xuất sau đó về bản chất là một lớp khuôn mẫu
- Cú pháp tương tự như lớp nguyên bản được dẫn xuất từ lớp nguyên bản

# Tóm tắt

- Khuôn mẫu hàm
  - Định nghĩa các hàm với tham số của một kiểu
- Khuôn mẫu lớp
  - Định nghĩa lớp với tham số của các phần con của lớp
- Các lớp có sẵn `vector` và `basic_string` là các lớp khuôn mẫu
- Có thể định nghĩa lớp khuôn mẫu được dẫn xuất từ một lớp cơ sở khuôn mẫu