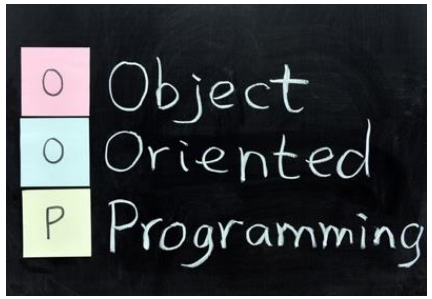



Môn học

Lập trình hướng đối tượng

Đỗ Phúc Hảo

Khoa Công nghệ thông tin




Nội dung

❖Chương 6: Hàm và lớp template

- Khái niệm
- Các hàm template
- Các lớp template




1. Khái niệm

❖Template (khuôn mẫu)

- Là một từ khoá trong C++
- Là một kiểu dữ liệu trừu tượng
- Báo cho trình biên dịch rằng đoạn mã đó định nghĩa cho nhiều kiểu dữ liệu và mã nguồn của nó sẽ được compile sinh ra cho từng kiểu dữ liệu trong quá trình biên dịch




1. Khái niệm (t)

❖Có 2 loại “template” cơ bản:

- Function template: khuôn mẫu hàm, định nghĩa các hàm tổng quát thao tác cho nhiều kiểu dữ liệu
- Class template: khuôn mẫu lớp, định nghĩa lớp tổng quát cho nhiều kiểu dữ liệu




2. Các hàm template

- Các hàm template được định nghĩa bắt đầu với từ khoá template, theo sau đó là một danh sách các tham số hình thức với hàm template vây quanh trong các ngoặc nhọn (< và >);
- Mỗi tham số hình thức phải được đặt trước bởi từ khoá class như:

```
template <class T>
Hoặc
template<class T1, class T2>
```




2. Các hàm template (t)

```
int MyAbs(int X) {
        return X>=0?X:-X;
}
long MyAbs(long X) {
        return X>=0?X:-X;
}
double MyAbs(double X) {
        return X>=0?X:-X;
}
```

```
template<class T>
T MyAbs(T x){
        return (x >= 0) ? x : -x;
}
int main(){
        cout << MyAbs(3) << endl;
        cout << MyAbs(-3.2) << endl;
        return 0;
}
```




2. Các hàm template (t)

```
#include <iostream>
using namespace std;
template<class T> T Max(T a, T b){
	return a > b ? a : b;
}
int main(){
	cout << Max(3, 4) << endl;
	cout << Max(32.33, 3333.333) << endl;
	cout << Max('a', 'r') << endl;
}
```




3. Các lớp template

❖Lớp template:

- Bên cạnh các hàm template, C++ còn trang bị thêm lớp template, nó mang đầy đủ ý tưởng của hàm template.
- Giả sử, để cài đặt một lớp Stack, thường chúng ta phải định nghĩa trước một kiểu dữ liệu cho từng phần tử của nó.
- Nhưng trong trường hợp tổng quát nó sẽ không tối ưu. Nên chúng ta sẽ sử dụng lớp template để cài đặt.




3. Các lớp template (t)

❖File tstack.h

```
template <class T>
class Stack {
        private:
                int sz; int top; T *a;
        public:
                Stack(int sz);
                ~Stack();
                bool isEmpty() const;
                bool isFull() const;
                void push(T item);
                T pop();
};
```




3. Các lớp template (t)

```
template <class T>
T Stack<T>::pop(){
        if(!isEmpty())
        return a[top--];
}
template <class T>
void Stack<T>::push(T item){
        if(!isFull())
        a[++top] = item;
}
```

```
template <class T>
Stack<T>::~Stack(){
        delete []a;
        a = NULL;
}
template <class T>
bool Stack<T>::isEmpty() const {
        return top == -1;
}
```




3. Các lớp template (t)

```
template <class T> bool Stack<T>::isFull() const {
        return top == sz - 1;
}
template <class T> Stack<T>::Stack(int sz){
        this->sz = sz;
        this->top = -1;
        this->a = new T [sz];
}
// Các kiểu dữ liệu sẽ được dùng cho template của class (*)
template class Stack<int>;
template class Stack<float>;
template class Stack<double>;
```




3. Các lớp template (t)

```
int main(){
        Stack<double> s(4);
        s.push(3); s.push(1); s.push(4); s.push(5); s.push(3333);
        while(!s.isEmpty()){
                cout << s.pop() << endl;
        }
        return 0;
}
```




Thank you!

Any questions?