# Phần 2: Ngôn ngữ lập trình C++

Chương 6: Mẫu (template)

# Các nội dung chính

1. Giới thiệu
2. Mẫu hàm
3. Mẫu lớp

# 1. Giới thiệu

- Khái niệm Mẫu (template):
  - Là một kỹ thuật cho phép một thành phần chỉ cần được định nghĩa một lần hoặc một số ít lần, nhưng có thể được sử dụng lại nhiều lần cho nhiều đối tượng khác
  - Là kỹ thuật cho phép tham số hóa kiểu dữ liệu; như cho phép định nghĩa cấu trúc **Stack<T>**, với **T** là tham số kiểu, đại diện cho kiểu DL của các phần tử của Stack. Sau đó T có thể được thay thế bằng một kiểu DL cụ thể, ví dụ **int**, và C++ sẽ tự động tạo ra code để định nghĩa **Stack<int>**
  - Nó có thể dùng để thay thế cho việc định nghĩa chồng hàm
  - Trong C++, các thành phần mà ta có thể tạo Mẫu là Hàm và Lớp

# 2. Mẫu hàm

- Khái niệm mẫu hàm
- Tạo mẫu hàm
- Sử dụng mẫu hàm

# Khái niệm mẫu hàm

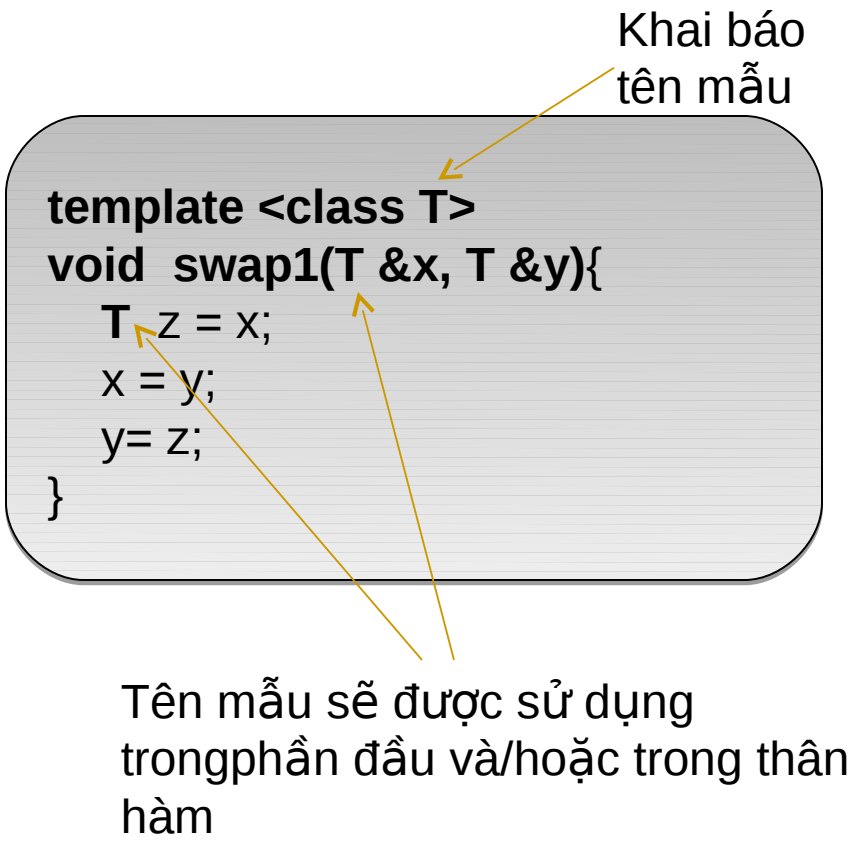
- Là hàm mà khi định nghĩa có sử dụng một hoặc nhiều mẫu
- Mẫu hàm được dùng để cho phép định nghĩa hàm một lần, nhưng có thể được gọi nhiều lần với tham số là các kiểu dữ liệu khác nhau

```
template <class T>
void  swap (T &x, T &y){
    T  z = x;
    x = y;
    y= z;
}
```

```
int  i, j; char  a,b;
float  x, y;
swap(i, j);
swap(a, b);
swap(x, y);
```

# Tạo một mẫu hàm

- Cú pháp
  - Một mẫu hàm có thể sử dụng một hoặc nhiều tên mẫu



# Tạo một mẫu hàm

- Mẫu hàm có hai tên mẫu

```
template <class T, class U>
void  swap2 (T &x, U &y){
    T  z = x;
    x = (T) y;
    y= (U) z;
}
```

# Sử dụng mẫu hàm

- Việc gọi mẫu hàm cũng giống như gọi hàm thông thường. Hàm được gọi này, khi đó được gọi là ***hàm thể hiện***
- Khi gọi hàm mẫu, thì tùy theo kiểu dữ liệu của hàm thể hiện, mà chương trình dịch sẽ tự động tạo ra định nghĩa phù hợp cho hàm này.

# Ví dụ áp dụng mẫu hàm

```
1.  #include <iostream>
2.  using namespace std;

3.  //Định nghĩa mẫu hàm
4.  template <class T>
5.  void swap1(T &a,T &b) {
6.      T c;
7.      c=a; a=b; b=c;
8.  }
```

```
9.  int main(int argc, char* argv[]) {
10.     int i=20,j=30;
11.     char c1='A',c2='B';
12.     float x=20.15, y=35.5;
13.     //Gọi mẫu hàm
14.     swap1(i,j);
15.     swap1(c1,c2);
16.     swap1(x,y);
17.     cout<<"i="<<i<<" j="<<j<<endl;
18.     cout<<"c1="<<c1<<" c2="<<c2<<endl;
19.     cout<<"x="<<x<<" y="<<y<<endl;
20.     return 0;
21. }
```

# Ví dụ áp dụng mẫu hàm

- Kết quả chạy chương trình trên

Output

```
i=30  j=20
c1=B  c2=A
x=35.5  y=20.15
```

# Mẫu hàm và sự chồng hàm

- Mẫu hàm là một công cụ hỗ trợ cho việc chồng hàm, chứ không hoàn toàn thay thế được cho chồng hàm
- Ví dụ hàm swap1 ở trên không thực hiện được việc hoán đổi 2 chuỗi ký tự, khi đó ta phải chồng hàm này.

```
1.  #include <string.h>
2.  #include <iostream>
3.  using namespace std;

4.  //Định nghĩa mẫu hàm
5.  template <class T>
6.  void swap1(T &a,T &b) {
7.      T c;
8.      c=a; a=b; b=c;
9.  }
9.  void swap1(char a[], char
    b[]){
10.     char st[1000];
11.     strcpy(st,a);
12.     strcpy(a,b);
13.     strcpy(b,st);
14. }
```

```
15. int main(int argc, char **argv)
16. {
17.     int i=10,j=20;
18.     swap1(i,j);
19.     cout<<"i="<<i<<"; j="<<j<<endl;
20.     char name1[]="Gone With The Wind";
21.     char name2[]="Mission Impossible";
22.     swap1(name1,name2);
23.     cout<<"Name 1:"<<name1<<endl;
24.     cout<<"Name 2:"<<name2;
25.     return 0;
26. }
```

# Kết quả chạy chương trình

Output

```
i=20;  j=10
Name 1: Mission Impossible
Name 2: Gone With The Wind
```

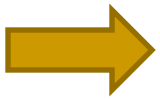
# 3. Mẫu lớp

- Khái niệm mẫu lớp
- Tạo mẫu lớp
- Sử dụng mẫu lớp

# Khái niệm mẫu lớp

- Là lớp mà khi định nghĩa có sử dụng một hoặc nhiều mẫu
- Mẫu lớp được dùng để cho phép định nghĩa lớp một lần, nhưng có thể tạo ra nhiều lớp khác nhau với tham số là các kiểu dữ liệu khác nhau

```
template <class T>
class  Stack {
    Stack() ;
    ~Stack() ;
    int push(const T& x);
    int pop(T& x) ;
    int isEmpty() const;
    int isFull() const;
};
```

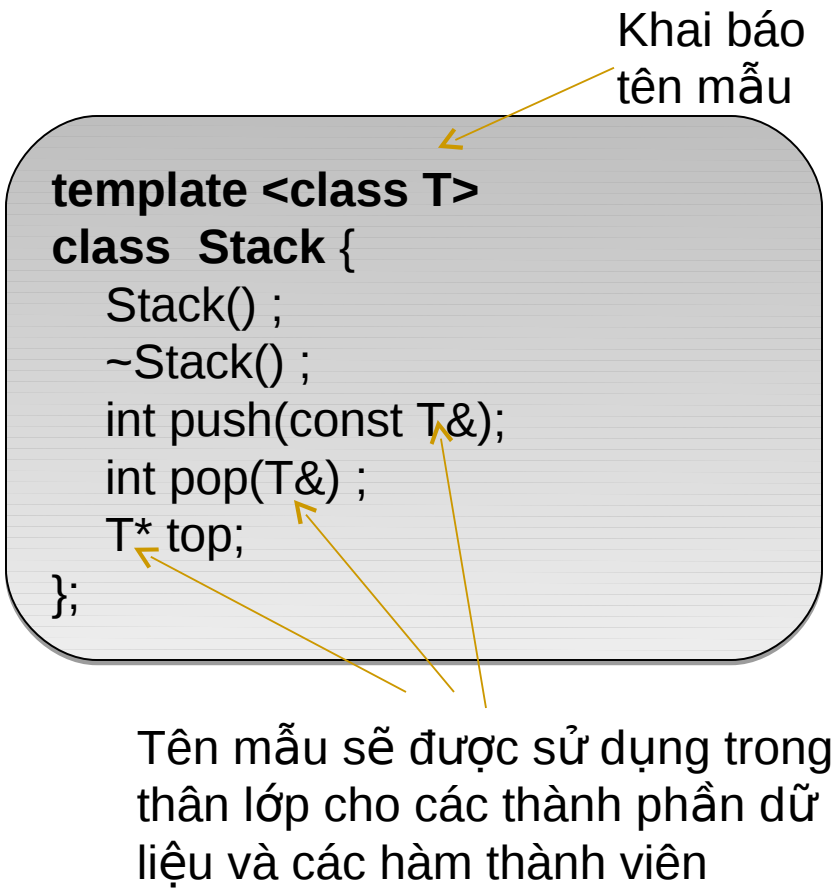


```
typedef  Stack<int> IntStack;
typedef  Stack<float> FloatStack;
typedef  Stack<string> StringStack;

IntStack  s1;
FloatStack s2;
```

# Tạo mẫu lớp

- Cú pháp:



# Sử dụng mẫu lớp

- **Lớp thể hiện**: là lớp được tạo ra từ mẫu lớp với các mẫu được thay thế bằng các kiểu dữ liệu cụ thể
- Có 2 cách để tạo ra lớp thể hiện:
  - Cách 1: định nghĩa tường minh một lớp thể hiện cho một kiểu dữ liệu cụ thể từ mẫu lớp (với từ khóa ***typedef***), rồi sau đó khai báo các đối tượng thuộc lớp thể hiện này.
  - Cách 2: Khai báo luôn các đối tượng thuộc lớp thể hiện ngầm định (không tường minh)

# 2 cách sử dụng mẫu lớp

Cách 1:
tường minh

```
typedef  Stack<int> IntStack;
typedef  Stack<float> FloatStack;

IntStack  s1;
FloatStack s2;
```

Cách  2:
ngầm định

```
Stack<int>   s1;
Stack<float>  s2;
```

# Ví dụ áp dụng: xây dựng mẫu lớp Stack, tệp Stack.h

```
//stack.h
1.  #pragma once
2.  template <class T>
3.  class Stack
4.  {
5.  public:
6.          Stack(unsigned int msize=10) ;
7.          ~Stack() { delete [] top ; }
8.          int push(const T&);
9.          T pop() ;  // pop an element off the stack
10.         int isEmpty()const { return size == 0 ; }
11.         int isFull() const { return size == maxsize ; }
12. private:
13.         int size ;  // Number of elements on Stack
14.         int maxsize;
15.         T* top ;
16. } ;
```

# Tệp Stack.h

```
//stack.h (tiếp)

17.//constructor with the default size 10
18.template <class T>
19.Stack<T>::Stack(unsigned int msize)
20.{
21.         size = 0;
22.         maxsize=msize;
23.         top = msize>0?(new T[msize]):NULL;
24.}
```

# Tệp Stack.h

```
//stack.h (tiếp)

25.// push an element onto the Stack
26.template <class T>
27.int Stack<T>::push(const T& item)
28.{
29.         if (!isFull())
30.         {
31.                   top[size] = item ;
32.                   size++;
33.                   return 1 ;  // push successful
34.         }
35.         return 0 ;  // push unsuccessful
36.}
```

# Tệp Stack.h

```
//stack.h (tiếp)

37.// pop an element off the Stack
38.template <class T>
39.T Stack<T>::pop()
40.{
41.        if (!isEmpty())
42.        {
43.                size--;
44.                return top[size];  // pop successful
45.        }
46.        return NULL ;  // pop unsuccessful
47.}
```

# Sử dụng Stack theo cách 1. Hàm main

```
1.  #include <iostream>
2.  #include "Stack.h"
3.  using namespace std;

4.  typedef Stack<int> IntStack;
5.  int main(int argc, char **argv)
6.  {
7.          IntStack si(20);
8.          for (int i=1;i<10;i++) si.push(i);
9.          cout<<"Stack of integers: ";
10.         while (!si.isEmpty()) {
11.                 cout<<si.pop()<<" ";
12.         }
13.         return 0;
14.}
```

Output:

**Stack of integers:**
**9 8 7 6 5 4 3 2 1**

# Sử dụng Stack theo cách 2. Hàm main

```
1.  #include <iostream>
2.  #include "Stack.h"
3.  using namespace std;

4.  int main(int argc, char **argv)
5.  {
6.          Stack<char> sc(30);

7.          for (int i=0;i<10;i++) sc.push('A'+i);
8.          cout<<"Stack of characters: ";
9.          while (!sc.isEmpty()) {
10.                 cout<<sc.pop()<<" ";
11.         }
12.         return 0;
13.}
```

Output:

**Stack of characters:**
**J I H G F E D C B A**

# Các câu hỏi tóm tắt

- Thế nào là một mẫu ?
- Mục đích của việc xây dựng mẫu là gì ?
- Thế nào là mẫu hàm ?
- So sánh giữa mẫu hàm và sự chồng hàm
- Mẫu lớp là gì ?
- Nêu các cách sử dụng mẫu lớp

# Bài tập

- **Bài 1**: Viết mẫu hàm tính tổng của một dãy N phần tử
- **Bài 2**: Viết mẫu hàm cho phép tìm một phần tử K trong một dãy A có N phần tử
- **Bài 3**: Xây dựng mẫu lớp cho cấu trúc hàng đợi
- **Bài 4**: Xây dựng mẫu lớp cho danh sách tổng quát, sử dụng cấu trúc lưu trữ móc nối đơn, mà có các thao tác sau:
  - Khởi tạo: tạo một danh sách rỗng
  - Lấy kích thước của danh sách
  - Bổ sung: bổ sung một phần tử vào đầu, vào cuối, và vào một vị trí bất kỳ trong danh sách
  - Lấy ra: lấy ra một phần tử ở đầu, ở cuối và ở vị trí bất kỳ trong danh sách

# Thank you!