第六讲 模版

一、 函数模版

二、 类模版

1.1 函数模版

有很多函数仅仅是参数类型不同，但是实现过程很类似，此时可以用一个通用函数表示它们，这种通用函数称为函数模板。在C++中使用关键字template定义模板。

【例1-1】把所有的代码都写在main函数里，没有任何封装和模块。

```
/**测试文件**/
#include <iostream>
using namespace std;

int main(int argc, const char * argv[]) {

    const int max = 5;
    int iArray[max] = {10, 20, 30, 40, 50};
    float fArray[max] = {1.1, 1.2, 1.3, 1.4, 1.5};
    char* cArray[max] = {"one", "two", "three", "four","five"};
```

```
    for(int i=0; i<max; i++){
        cout << iArray[i] << ", ";
    }
    cout<<endl;

    for(int i=0; i<max; i++){
        cout << fArray[i] << ", ";
    }
    cout<<endl;

    for (int i=0; i<max; i++){
        cout << cArray[i] << ", ";
    }
    cout<<endl;

    return 0;
}
```

【例1-2】 把打印部分封装成函数，初步复用。

```
/**测试文件**/
#include <iostream>
using namespace std;
void printArray(int *array, int size){
    for (int i=0; i<size; i++){
        cout << array[i];
        if(i < (size - 1)){
            cout << ", ";
        }
    }
}

void printArray(float *array, int size){
    for (int i=0; i<size; i++){
        cout << array[i];
        if (i < (size - 1) ){
            cout << ", ";
        }
    }
}
```

```
//这里不要写成(char*)*array:这种形式被编译器理解成类型转换
void printArray(char **array, int size) {

    for (int i=0; i<size; i++){
        cout << array[i];
        if (i < (size - 1) ){
            cout << ", ";
        }
    }
}

int main(int argc, const char * argv[]) {
    const int max = 5;
    int iArray[max] = {10, 20, 30, 40, 50};
    float fArray[max] = {1.1, 1.2, 1.3, 1.4, 1.5};
    char *cArray[max] = {"one", "two", "three", "four", “five"};

    printArray(iArray, max);
    printArray(fArray, max);
    printArray(cArray, max);
    return 0;
}
```

在示例1-2中，printArray()函数执行的任务是一样的，只有函数的参数类型不同，此时可以定义一个函数模板表示这三个打印函数。

普通函数与函数模版的区别：

(1) 普通函数：参数值未定，但参数类型确定；

(2) 函数模板：不仅参数的值不确定，参数的类型也不确定；

【例1-3】 把打印部分封装成函数模版，更高级复用。

```
#include <iostream>
using namespace std;

//T是类型形参，表明T本身是个类型的名字，T是一个未实例化的类型。
//T在函数调用时，自动初始化类型。

template <class T>
void printArray(T* array, int size){

    for (int i=0; i<size; i++){
        cout << array[i];

        //array数组元素的类型，未定。
        //在调用该函数模板时，确定T的类型
        if (i < (size - 1) ){
            cout << ", ";
        }
    }
}
```

```
int main(int argc, const char * argv[]) {

    const int max = 5;
    int iArray[max] = {10, 20, 30, 40, 50};
    float fArray[max] = {1.1, 1.2, 1.3, 1.4, 1.5};
    char *cArray[max] = {"one", "two", "three", "four", "five"};

    //调用函数模板时，自动对模板形参进行实例化,模板形参的类型就是传入的函数实参的类型。
    printArray( iArray, max);  //T被实例化为int型,T = int;
    cout << "\n";

    printArray( fArray, max);
    cout << "\n";

    printArray(cArray, max);
    cout << "\n";

    return 0;
}
```

2.1 类模版

实际编程过程中，不仅可以使用函数模板，也可以使用类模板，类模板可以用于表示一组类，这些类仅仅是成员变量类型不同，但是对这些成员变量的操作类似。

【例2-1】编写一个类，使用类模版。注：模板的声明与实现需要在同一个文件中。

```
#include <iostream>
using namespace std;
template <class T> //T是一个模板参数
class Stack
{
public:
    Stack(int s = 10);
    ~Stack();
    int push(const T&); //入栈(添加元素)
    int pop(T&); //出栈(删除元素)
    int isEmpty() const;
    int isFull() const;
    void printOn(ostream& output);

private:
    int size;
    int top;
    T* stackPtr;  //使用模板参数T 声明成员变量
};
```

```
template <class T>
Stack<T>::~Stack() {
        delete [] stackPtr;
}

template <class T>
Stack<T>::Stack(int s) {
    size = s;
    top = -1;
    stackPtr = new T[size];//在堆上开辟空间
}

template <class T>
int Stack<T>::push(const T& item) {
    if (!isFull()){//判断容器是否已满，避免溢出
        stackPtr[++top] = item;
        return 1;
    }
    return 0;
}
```

```
template <class T>
int Stack<T>::pop(T& item) {
    if (!isEmpty()){ //判断是否为空，避免操作空容器
        item = stackPtr[top--];
        return 1;
    }
    return 0;
}
```

```
template <class T>
int Stack<T>::isEmpty() const {//判断容器是否为空
    return top == -1;
}

template <class T>
int Stack<T>::isFull() const {//判断容器是否已满
    return top == size - 1;
}
```

```
template <class T>
void Stack<T>::printOn(ostream& output){
    output << "Stack( ";
    for (int i=0; i<=top; i++){ //从栈底打印到栈顶所有元素
        output << stackPtr[i] << ", ";
    }
    output << ")\n";
}

/**第一个测试主函数**/
int main(int argc, const char * argv[]) {
    //类模板必须在声明该模板的对象时，明确的对模板参数实例化。
    Stack<int> stack; //实例化T为int类型
    for (int i=0; i<5; i++){
        stack.push(i*10);//添加元素
    }
    stack.printOn(cout);

    return 0;
}
```

程序运行结果如下：

Stack(0, 10, 20, 30, 40,)

```
/**第二个测试主函数**/
int main(int argc, const char * argv[]) {

    Stack<float> stack; //实例化T为int类型
    for (int i=0; i<5; i++){
        stack.push(i*10.321);//添加元素
    }
    stack.printOn(cout);

    for(int i=0;i<2;i++){
        float f;
        stack.pop(f);//删除元素
        cout<<f<<endl;
    }

    stack.printOn(cout);

    return 0;
}
```

程序运行结果如下：

```
Stack( 0, 10.321, 20.642, 30.963, 41.284, )
======删除元素======
41.284
30.963
======删除元素结束======
Stack( 0, 10.321, 20.642, )
```

```
/**第三个测试主函数**/
int main(int argc, const char * argv[]) {
    char *cArray[5] = {"one", "two", "three", "four", "five"};

    Stack<char*> stack;
    for (int i=0; i<5; i++){
        stack.push(cArray[i]);
    }
    stack.printOn(cout);

    return 0;
}
```

程序运行结果如下：

Stack(one, two, three, four, five,)

示例分析：

(1) 模板中的每个成员函数，都是函数模板。实现时，每个成员函数前面，都要加上模板声明template<class T>;

(2) 模板的定义与实现只能在同一个文件中；

2.2 类模版默认值

模版参数也可以有默认值。

【例2-2】带默认值的模板参数

```
#include <iostream>
using namespace std;

template <class T = int> //类型默认初始化为int类型
class A{
    T t;
public:
    A(){
        t = 10;
    }
    void print(){
        cout<<t<<endl;
    }
};
int main(int argc, const char * argv[]) {
    A<> a;//未填写类型，T将使用默认类型int
    a.print();
    return 0;
}
```

程序运行结果如下：

10

2.3 多个模版参数

```
#include <iostream>
using namespace std;

//多个参数(未实例化参数T,未实例化参数S)
template  <class T, class S>

void test(T t, S s){
    cout<<t<<endl;
}

int main(int argc, const char * argv[]) {
    int a = 10;
    char * b = "hello";
    test(a,b);

    return 0;
}
```

程序运行结果如下:

10

hello