C++函数模板（Function Template）

在《C++函数重载》一节中，为了交换不同类型的变量的值，我们通过函数重载定义了四个名字相同、参数列表不同的函数，如下所示：

```
//交换 int 变量的值
void Swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

//交换 float 变量的值
void Swap(float *a, float *b){
    float temp = *a;
    *a = *b;
    *b = temp;
}

//交换 char 变量的值
void Swap(char *a, char *b){
    char temp = *a;
    *a = *b;
    *b = temp;
}

//交换 bool 变量的值
void Swap(bool *a, bool *b){
    char temp = *a;
    *a = *b;
    *b = temp;
}
```

这些函数虽然在调用时方便了一些，但从本质上说还是定义了三个功能相同、函数体相同的函数，只是数据的类型不同而已，这看起来有点浪费代码，能不能把它们压缩成一个函数呢？

能！可以借助本节讲的函数模板。

我们知道，数据的值可以通过函数参数传递，在函数定义时数据的值是未知的，只有等到函数调用时接收了实参才能确定其值。这就是值的参数化。

在C++中，数据的类型也可以通过参数来传递，在函数定义时可以不指明具体的数据类型，当发生函数调用时，编译器可以根据传入的实参自动推断数据类型。这就是类型的参数化。

值（Value）和类型（Type）是数据的两个主要特征，它们在C++中都可以被参数化。

所谓函数模板，实际上是建立一个通用函数，它所用到的数据的类型（包括返回值类型、形参类型、局部变量类型）可以不具体指定，而是用一个虚拟的类型来代替（实际上是用一个标识符来占位），等发生函数调用时再根据传入的实参来逆推出真正的类型。这个通用函数就称为函数模板（Function Template）。

在函数模板中，数据的值和类型都被参数化了，发生函数调用时编译器会根据传入的实参来推演形参的值和类型。换个角度说，函数模板除了支持值的参数化，还支持类型的参数化。

一但定义了函数模板，就可以将类型参数用于函数定义和函数声明了。说得直白一点，原来使用 int、float、char 等内置类型的地方，都可以用类型参数来代替。

下面我们就来实践一下，将上面的四个Swap() 函数压缩为一个函数模板：

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

template<typename T> void Swap(T *a, T *b){
    T temp = *a;
    *a = *b;
    *b = temp;
}
```

```
09.
10.  int main(){
11.      //交换 int 变量的值
12.      int n1 = 100, n2 = 200;
13.      Swap(&n1, &n2);
14.      cout<<n1<<", "<<n2<<endl;
15.
16.      //交换 float 变量的值
17.      float f1 = 12.5, f2 = 56.93;
18.      Swap(&f1, &f2);
19.      cout<<f1<<", "<<f2<<endl;
20.
21.      //交换 char 变量的值
22.      char c1 = 'A', c2 = 'B';
23.      Swap(&c1, &c2);
24.      cout<<c1<<", "<<c2<<endl;
25.
26.      //交换 bool 变量的值
27.      bool b1 = false, b2 = true;
28.      Swap(&b1, &b2);
29.      cout<<b1<<", "<<b2<<endl;
30.
31.      return 0;
32.  }
```

运行结果：
200, 100
56.93, 12.5
B, A
1, 0

请读者重点关注第 4 行代码。`template`是定义函数模板的关键字，它后面紧跟尖括号`<>`，尖括号包围的是类型参数（也可以说是虚拟的类型，或者说是类型占位符）。`typename`是另外一个关键字，用来声明具体的类型参数，这里的类型参数就是`T`。从整体上看，`template<typename T>`被称为模板头。

模板头中包含的类型参数可以用在函数定义的各个位置，包括返回值、形参列表和函数体；本例我们在形参列表和函数体中使用了类型参数`T`。

类型参数的命名规则跟其他标识符的命名规则一样，不过使用 T、T1、T2、Type 等已经成为了一种惯例。

定义了函数模板后，就可以像调用普通函数一样来调用它们了。

在讲解C++函数重载时我们还没有学到引用（Reference），为了达到交换两个变量的值的目的只能使用指针，而现在我们已经对引用进行了深入讲解，不妨趁此机会来实践一把，使用引用重新实现 Swap() 这个函数模板：

```
01.  #include <iostream>
02.  using namespace std;
03.
04.  template<typename T> void Swap(T &a, T &b){
05.      T temp = a;
06.      a = b;
07.      b = temp;
08.  }
09.
10.  int main(){
11.      //交换 int 变量的值
12.      int n1 = 100, n2 = 200;
13.      Swap(n1, n2);
14.      cout<<n1<<", "<<n2<<endl;
15.
16.      //交换 float 变量的值
17.      float f1 = 12.5, f2 = 56.93;
18.      Swap(f1, f2);
19.      cout<<f1<<", "<<f2<<endl;
20.
```

```
    //交换 char 变量的值
    char c1 = 'A', c2 = 'B';
    Swap(c1, c2);
    cout<<c1<<", "<<c2<<endl;

    //交换 bool 变量的值
    bool b1 = false, b2 = true;
    Swap(b1, b2);
    cout<<b1<<", "<<b2<<endl;

    return 0;
}
```

引用不但使得函数定义简洁明了，也使得调用函数方便了很多。整体来看，引用让编码更加漂亮。

下面我们来总结一下定义模板函数的语法：

```
template <typename 类型参数1 , typename 类型参数2 , ...> 返回值类型  函数名(形参列表){
  //在函数体中可以使用类型参数
}
```

类型参数可以有多个，它们之间以逗号`,`分隔。类型参数列表以`< >`包围，形式参数列表以`()`包围。

`typename`关键字也可以使用`class`关键字替代，它们没有任何区别。C++ 早期对模板的支持并不严谨，没有引入新的关键字，而是用 class 来指明类型参数，但是 class 关键字本来已经用在类的定义中了，这样做显得不太友好，所以后来 C++ 又引入了一个新的关键字 typename，专门用来定义类型参数。不过至今仍然有很多代码在使用 class 关键字，包括 C++ 标准库、一些开源程序等。

本教程会交替使用 typename 和 class，旨在让读者在别的地方遇到它们时不会感觉陌生。更改上面的 Swap() 函数，使用 class 来指明类型参数：

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

除了将 typename 替换为 class，其他都是一样的。

为了加深对函数模板的理解，我们再来看一个求三个数的最大值的例子：

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

//声明函数模板
template<typename T> T max(T a, T b, T c);

int main( ){
    //求三个整数的最大值
    int i1, i2, i3, i_max;
    cin >> i1 >> i2 >> i3;
    i_max = max(i1,i2,i3);
    cout << "i_max=" << i_max << endl;

    //求三个浮点数的最大值
    double d1, d2, d3, d_max;
    cin >> d1 >> d2 >> d3;
    d_max = max(d1,d2,d3);
    cout << "d_max=" << d_max << endl;

    //求三个长整型数的最大值
    long g1, g2, g3, g_max;
    cin >> g1 >> g2 >> g3;
    g_max = max(g1,g2,g3);
    cout << "g_max=" << g_max << endl;

```

```
    return 0;
}

//定义函数模板
template<typename T>  //模板头，这里不能有分号
T max(T a, T b, T c){ //函数头
    T max_num = a;
    if(b > max_num) max_num = b;
    if(c > max_num) max_num = c;
    return max_num;
}
```

运行结果：
12 34 100↙
i_max=100
73.234 90.2 878.23↙
d_max=878.23
344 900 1000↙
g_max=1000

函数模板也可以提前声明，不过声明时需要带上模板头，并且模板头和函数定义（声明）是一个不可分割的整体，它们可以换行，但中间不能有分号。