1. 模板
2. 函数模板与模板函数
3. 应用场景：形参类型不同但处理算法相同的函数
4. 使用方法：

首先需要定义函数模板，再将其实例化为相应的模板函数，最后才能使用模板函数。

模板函数可以与重载函数共存。在此情况下，

首先匹配类型完全相同的重载函数。

其次才寻求函数模板来匹配，

注意，当实参类型不同于形参类型时，不能进行隐式类型转换。

1. 声明格式：

template < class 类型形参名>

<函数返回值类型> <函数名>（形参表）

{

//函数定义体

}

//形参列表可以是自定义类，可以是不同的类，也可以是标准化类型

1. 作用：

定义了一个通用性的算法，可以对不同类型的参数使用。

将形参列表中不确定的参数在实例中用确定的参数类型替代即为函数实例化。

1. 类模板与模板类
2. 应用场景：处理类型不同但类对象内部构造相同的各个类
3. 关系：类模板实例化之后成为模板类
4. 声明格式：

Template<class type>

Class T-Number{

public:

T-Number(Type);

~T-Number(){ };

void Set-Item(Type);

Type Get-Item( );

//…

private:

Type m\_item;

……

}

1. 使用方法：

如 T\_number< int > a;

T\_number< double > b;

可以创建参数为int/double的模板类

1. 使用注意：

1类模板不能生成具体对象，实例化之后才可以

2如果类型形参多于一个，则每个形参前都要使用关键词class。