第3章　模　　板

使用模板可以建立具有通用类型的函数库或类库，为一系列逻辑功能相同而数据类型不同的函数或类创建框架。模板提供了一种重用程序源代码的有效方法，使大规模软件开发更加方便。

3.1　模板的概念

模板的本质就是将所处理的数据类型说明为参数，是对具有相同特性的函数或类的再抽象。它将程序处理的数据类型参数化，可使一段程序代码处理多种不同类型的数据。C++ 程序由类和函数组成，类对应类模板，函数对应函数模板。

为了更好地理解，先来考察 3 个 Swap()函数，是如何交换两个整型数、两个单精度实型数以及两个双精度实型数的。这 3 个 Swap()函数的功能完全一样，只是所处理的数据类型不同，下面是这 3 个函数的具体实现：

```
void Swap(int &x,int &y)                    //交换整型数 x、y
{
    int temp=x; x=y; y=temp;                //通过循环赋值交换 x、y
}

void Swap(float &x,float &y)                //交换单精度实型数 x、y
{
    float temp=x; x=y; y=temp;              //通过循环赋值交换 x、y
}

void Swap(double &x,double &y)              //交换双精度实型数 x、y
{
    double temp=x; x=y; y=temp;             //通过循环赋值交换 x、y
}
```

上面通过函数重载虽然能让 Swap()函数处理多种类型的数据，但不是任意的，只能交换一对相同类型的数据。最好的解决方法是将类型参数化后得到函数模板。使用函数模板定义如下：

```
template <class Type>       //此处的 class 不是定义类的标识，只表明 Type 是一个类型参数
void Swap(Type &x,Type &y)                  //交换 x、y
{
    Type temp=x; x=y; y=temp;               //通过循环赋值交换 x、y
}
```

这样得到可以将任意一种类型 Type 的两个数据进行互换的函数模板。

下面的 3 个类 Integer、Float 和 Double 分别用来处理整型数、单精度实型数以及双精度实型数。这 3 个类的处理功能完全一样，只是处理的数据类型不同，下面是这 3 个类的声明：

```
//声明整型类
class Integer
{
private:
//数据成员
    int num;                                    //数据值

public:
//公有函数
    Integer(int n=0): num(n){ }                 //构造函数
    void Set(int n) { num=n; }                  //设置数据值
    int Get() const { return num; }             //返回数据值
};

//声明单精度实型数类
class Float
{
private:
//数据成员
    float num;                                  //数据值

public:
//公有函数
    Float(float n=0): num(n){ }                 //构造函数
    void Set(float n) { num=n; }                //设置数据值
    float Get() const { return num; }           //返回数据值
};

//声明双精度实型数类
class Double
{
private:
//数据成员
    double num;                                 //数据值

public:
//公有函数
    Double(double n=0): num(n){ }               //构造函数
    void Set(double n) { num=n; }               //设置数据值
    double Get() const { return num; }          //返回数据值
};
```

上面实现的 3 个类的功能相同，只是处理的数据类型不同，都不能处理任意类型的数据，但采用类型参数化后就可以处理任何类型的数据，这样就得到了类模板。使用类模板定义如下：

```
template <class Type>
class Number
{
private:
//数据成员
    Type num;                                   //数据值

public:
//公有函数
    Number(Type n=0): num(n){}                  //构造函数
    void Set(Type n) { num=n; }                 //设置数据值
    Type Get() const { return num; }            //返回数据值
};
```

注意：函数模板或类模板是对一组函数或类的描述，模板是一种由通用代码构成，使用类型参数产生一组函数或类的机制。

3.2 函数模板及模板函数

函数模板是对一批功能相同的函数的说明，它不是某一个具体的函数，而是带有类型参数的一种描述。模板函数是将函数模板内的数据类型参数取某一个具体的数据类型后得到的具体函数。

3.2.1 函数模板的声明及生成模板函数

使用函数模板的方法是先声明函数模板，然后将其类型参数具体化，形成相应的模板函数后，才能调用模板函数。

函数模板的一般声明格式如下。

格式 1：

```
template <class 类型参数名 1, class 类型参数名 2, …>
返回值类型 函数模板名(形参表)
{
    …                                           //函数模板体
}
```

格式 2：

```
template <typename 类型参数名 1, typename 类型参数名 2, …>
返回值类型 函数模板名(形参表)
{
```

```
    …                                    //函数模板体
}
```

其中，template 是一个声明模板的关键字，class 在此处并不表示类，只是借用此关键字表示其后是一个类型参数。“class 类型参数名 1，class 类型参数名 2，…”称为类型形参表，类型参数名可以用任何实际的类型（包括类类型）进行具体化（也称为实例化）得到模板函数。

class 和 typename 的作用相同，都是表示类型名，二者可以互换。大部分 C++ 程序员都喜欢用 class，这是因为 class 更容易输入，typename 是新加入标准 C++ 中的，使用 typename 时，含义非常清楚。

在使用函数模板时，用实际的数据类型具体化（实例化）类型形式参数，再根据实际参数类型生成一个具体的模板函数，模板函数的函数体与函数模板的函数模板体完全相同，在程序中真正执行的代码是模板函数的代码。

在使用函数模板生成模板函数时，有两种使用方式。

方式 1：

```
函数模板名(实参表)
```

方式 2：

```
函数模板名 <类型 1, 类型 2, …>(实参表)
```

方式 1 将根据实参类型确定类型形式参数的具体类型，方式 2 中，“<类型 1，类型 2，…>”称为类型实参表，用类型实参表中的类型来确定类型形式参数的具体类型。

说明：类型形参表与类型实参表通常只包含一个类型，这时函数模板的一般声明格式如下。

格式 1：

```
template <class 类型参数名>
返回值类型 函数模板名(形参表)
{
    …                                    //函数模板体
}
```

格式 2：

```
template <typename 类型参数名>
返回值类型 函数模板名(形参表)
{
    …                                    //函数模板体
}
```

使用函数模板生成模板函数的两种方式。

方式 1：

```
函数模板名(实参表)
```

方式 2：

函数模板名 <类型> (实参表)

例 3.1 函数模板的声明与模板函数的调用示例,程序演示见 3011.mp4～3013.mp4。


```
//文件路径名:e3_1\main.cpp
#include <iostream>                                          //编译预处理命令
using namespace std;                                         //使用命名空间 std

template <class Type>
Type Max(Type x,Type y)                                      //求 x、y 的最大值
{
    return x<y? y:x;                                         //返回 x、y 的最大值
}

int main()                                                   //主函数 main()
{
    cout<<"2 和 3 的最大值为"<<Max(2, 3)<<endl;               //输出 2、3 的最大值
    //cout<<"2 和 3.0 的最大值为"<<Max(2, 3.0)<<endl;
    //错,无法根据 2 和 3.0 确定类型形式参数的具体类型
    cout<<"2 和 3.0 的最大值为"<<Max<int>(2, 3.0)<<endl;     //输出 2、3.0 的最大值

    return 0;                                                //返回值 0, 返回操作系统
}
```

程序运行时屏幕输出如下:

```
2 和 3 的最大值为 3
2 和 3.0 的最大值为 3
```

从上面的例题可以看出,采用方式 1 使用函数模板生成模板函数时,同一模板类型参数的各参数之间必须保持完全一致的类型。

3.2.2 重载函数模板

模板函数类似于重载函数,但是同一个函数模板类型形式参数具体化(实例化)后的所有模板函数必须执行相同的代码,而函数重载时在每个函数体中可以执行不同的代码,当遇到执行的代码不同时,不能简单地套用函数模板,而应像重载普通函数那样进行重载。

重载函数模板后,编译器首先匹配类型完全相同的函数,如果匹配失败,再寻求函数模板进行匹配。

例 3.2 重载函数模板与匹配过程示例,程序演示见 3021.mp4～3023.mp4。



```
//文件路径名:e3_2\main.cpp
#include <iostream>                                    //编译预处理命令
#include <string.h>                                    //编译预处理命令
using namespace std;                                   //使用命名空间 std

template<class Type>
Type Max(Type x,Type y)                                //求 x、y 的最大值
{
    return x<y? y:x;                                   //返回 x、y 的最大值
}

char * Max(char * str1, char * str2)                   //求 str1、str2 的最大值
{
    return strcmp(str1, str2)<0 ? str2 : str1;         //返回 str1、str2 的最大值
}

int main()                                             //主函数 main()
{
    cout<<"2 和 3 的最大值为"<<Max(2, 3)<<endl;      //输出 2、3 的最大值, 匹配函数模板
    cout<<"China 与 American 的最大值为"<<Max("China", "American")<<endl;
                                          //输出"China","American"的最大值, 匹配函数

    return 0;                                          //返回值 0, 返回操作系统
}
```

程序运行时屏幕输出如下：

```
2 和 3 的最大值为 3
China 与 American 的最大值为 China
```

char * Max(char * str1, char * str2)函数中的函数名与函数模板的函数模板名相同，但操作代码不同，函数体中的比较采用了字符串比较函数，所以要用重载的方法把它们区分开，遵循的原则是首选函数名、参数类型都匹配的函数，再找模板。本例在调用 Max(2,3)函数时，由于实参是整型，与 char * Max(char * str1, char * str2)函数的形参类型不匹配，因此只能匹配函数模板，在调用 Max("China", "American")函数时，实参为字符串，与 char * Max(char * str1, char * str2)函数的形参类型相匹配，因此匹配此函数。

例 3.3 重载函数模板示例，程序演示见 3031.mp4～3033.mp4。



```
//文件路径名:e3_3\main.cpp
#include <iostream>                                    //编译预处理命令
using namespace std;                                   //使用命名空间 std

template <class Type>
Type Max(Type x,Type y)                                //求 x、y 的最大值
{
    return x<y? y:x;                                   //返回 x、y 的最大值
}

template <class Type>
Type Max(Type x,Type y,Type z)                         //求 x、y、z 的最大值
{
    Type m=x<y? y:x;                                   //m 为 x、y 的最大值
    m=m<z? z:m;                                        //m、z 的最大值
    return m;                                          //返回最大值
}

template <class Type>
Type Max(Type a[], int n)                              //求 a[0],a[1],…,a[n-1]的最大值
{
    Type m=a[0];                                       //假设 a[0]为最大值
    for (int i=1; i<n; i++)
        if (m<a[i]) m=a[i];                            //如 a[i]更大,则将 a[i]赋值给 m
    return m;                                          //返回最大值
}

int main()                                             //主函数 main()
{
    int a[]={1,9,7,5,6,3};                             //定义数组 a
    cout<<"数组 a 的最大元素值为"<<Max(a,6)<<endl;     //输出数组 a 的最大元素值
    cout<<"2 和 3 的最大值为"<<Max(2,3)<<endl;         //输出 2、3 的最大值
    cout<<"2,3 和 8 的最大值为"<<Max(2,3,8)<<endl;     //输出 2、3、8 的最大值

    return 0;                                          //返回值 0, 返回操作系统
}
```

程序运行时屏幕输出如下：

```
数组 a 的最大元素值为 9
2 和 3 的最大值为 3
2,3 和 8 的最大值为 8
```

本例中根据实参的个数与类型来匹配不同的函数模板。

3.3 类模板及模板类

类模板与函数模板类似，它可以为任意数据类型定义一种类模板，使用不同的数据类型具体化（实例化）类模板生成具体的模板类，模板类可以用于生成具体的对象。

3.3.1 类模板的声明及生成模板类

声明一个类模板与声明函数模板的格式类似，必须以关键字 template 开始，类模板的一般声明形式如下。

形式 1：

```
template <class 类型参数名 1, class 类型参数名 2, … >
class 类模板名
{
    …                                                    //类模板体
};
```

形式 2：

```
template <typename 类型参数名 1, typename 类型参数名 2, … >
class 类模板名
{
    …                                                    //类模板体
};
```

类模板的成员函数模板不但可以在类模板体内定义，也可以在类模板体外定义。在类模板体内定义时，与一般类的成员函数的定义方法完全一样；在类模板体外定义时，需要采用下面的形式。

形式 1：

```
template <class 类型参数名 1, class 类型参数名 2, … >
返回值类型 类模板名<类型参数名 1, 类型参数名 2, … >::成员函数模板名(形参表)
{
    …                                                    //成员函数模板体
}
```

形式 2：

```
template <typename 类型参数名 1, typename 类型参数名 2, … >
返回值类型 类模板名<类型参数名 1, 类型参数名 2, … >::成员函数名(形参表)
{
    …                                                    //成员函数模板体
}
```

其中，“class 类型参数名 1，class 类型参数名 2，…”与“typename 类型参数名 1，typename 类型参数名 2，…”为类型形参表，与函数模板中的类型形参表意义一样。

类模板必须用实际的数据类型具体化(实例化)类型形式参数，再根据实际参数类型，生成一个具体的模板类，然后才能用来定义具体对象。一般语法格式如下：

```
类模板名<类型 1, 类型 2, … >对象名;
```

例 3.4 使用类模板的示例，程序演示见 3041.mp4～3043.mp4。


```
//文件路径名:e3_4\main.cpp
#include <iostream>                                          //编译预处理命令
using namespace std;                                         //使用命名空间 std

//声明数组类模板
template <class Type>
class Array
{
private:
//数据成员
    Type *elem;                                              //存储数据元素值
    int size;                                                //数组元素个数

public:
//公有函数模板
    Array(int sz): size(sz) { elem=new Type[size]; }         //构造函数
    ~ Array(){ delete elem; }                                //析构函数
    void SetElem(Type e, int i);                             //设置元素值
    Type GetElem(int i) const;                               //求元素值
};

template<class Type>
void Array <Type>::SetElem(Type e,int i)                     //设置元素值
{
    if (i<0||i>=size)
    {
        cout<<"元素位置错!"<<endl;
        exit(1);                                             //退出程序的运行,返回到操作系统
    }
    elem[i]=e;                                               //设置元素值为 e
}

template<class Type>
Type Array<Type >::GetElem(int i) const                      //求元素值
{
    if (i<0||i>=size)
    {
        cout<<"元素位置错!"<<endl;
        exit(2);                                             //退出程序的运行,返回到操作系统
    }
    return elem[i];                                          //返回元素值 elem[i]
}

int main()                                                   //主函数 main()
{
```

```
    int a[]={1,9,7,5,6,3};                          //定义数组 a
    int n=6;                                        //数组元素个数
    Array <int> obj(n);                             //定义数组对象
    int i;                                          //定义临时变量
    for (i=0; i<n; i++)
        obj.SetElem(a[i],i);                        //设置数组元素值
    for (i=0; i<n; i++)
        cout<<obj.GetElem(i)<<" ";                  //输出元素值
    cout<<endl;                                     //换行

    return 0;                                       //返回值 0, 返回操作系统
}
```

程序运行时屏幕输出如下：

```
1 9 7 5 6 3
```

怎样选择是将成员函数在类模板体内定义还是在类模板体外定义呢？建议对函数模板体实现较简单、行数较少的类模板的成员函数模板都在类模板体内定义；对函数模板体实现较复杂、行数较多的类模板的成员函数模板在类模板体外定义。

3.3.2 在类型形参表中包含常规参数的类模板

在声明类模板的类型形参表中还可以包含常规参数，常规参数经常是数值。对类模板进行具体化(实例化)为模板类时，给这些参数所提供的表达式必须是常量表达式。

例 3.5 使用在类型形参表中包含常规参数的示例，程序演示见 3051.mp4～3053.mp4。


```
//文件路径名:e3_5\main.cpp
#include <iostream>                                 //编译预处理命令
using namespace std;                                //使用命名空间 std

//声明数组类模板
template <class Type,int size>
class Array
{
private:
//数据成员
    Type elem[size];                                //存储数据元素值

public:
//公有函数模板
    void SetElem(Type e,int i);                     //设置元素值
```

```
    Type GetElem(int i) const;                          //求元素值
};

template <class Type, int size>
void Array <Type,size>::SetElem(Type e,int i)    //设置元素值
{
    if (i<0||i>=size)
    {
        cout<<"元素位置错!"<<endl;
        exit(1);                                        //退出程序的运行,返回到操作系统
    }
    elem[i]=e;                                          //设置元素值为 e
}

template <class Type,int size>
Type Array <Type, size>::GetElem(int i) const     //求元素值
{
    if (i<0||i>=size)
    {
        cout<<"元素位置错!"<<endl;
        exit(2);                                        //退出程序的运行,返回到操作系统
    }
    return elem[i];                                     //返回元素值 elem[i]
}

int main()                                              //主函数 main()
{
    int a[]={1,9,7,5,6,3};                              //定义数组 a
    const int n=6;                                      //数组元素个数
    Array<int,n>obj;                                    //定义数组对象
    int i;                                              //定义临时变量
    for (i=0; i<n; i++)
        obj.SetElem(a[i],i);                            //设置数组元素值
    for (i=0; i<n; i++)
        cout<<obj.GetElem(i)<<" ";                      //输出元素值
    cout<<endl;                                         //换行

    return 0;                                           //返回值 0, 返回操作系统
}
```

程序运行时屏幕输出如下：

```
1 9 7 5 6 3
```

由于在类模板进行具体化(实例化)为模板类时,类型形参表中包含的表达式参数必须是常量表达式,因此可以直接用数组储存数据元素值,比例 3.4 采用指针方式存储数据元素值更简捷。

**3.4 实例研究：快速排序

快速排序(quick sort)有着广泛的应用，典型应用是 UNIX 系统库函数例程中的 qsort()函数，快速排序的基本思想是任选序列中的一个数据元素(通常选取第一个数据元素)作为枢轴，并以它和所有剩余数据元素进行比较，将所有较它小的数据元素都排在它前面，将所有较它大的数据元素都排在它之后，经过一遍排序后，可按此数据元素所在位置为界，将序列划分为两部分，再对这两部分重复上述过程，直至每一部分中只剩一个数据元素或没有数据元素为止。

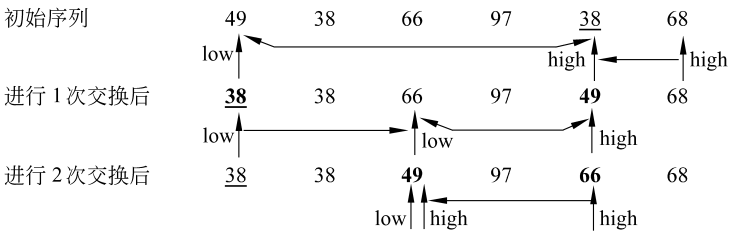
假设待排序的序列为(a[low]，a[low+1]，…，a[high])，首先任选取序列中的一数据元素(通常可选第一个数据元素)作为枢轴(pivot)，再将所有比 pivot 小的数据元素排序在 pivot 的左边，将所有比 pivot 大的数据元素排序在 pivot 的右边，设由此得到的枢轴的位置是 i，则以 i 作为分界线，将原序列作一次如下的划分：

```
(a[low],a[low+1],…,a[i-1]) a[i] (a[i+1],a[i+2],…,a[high])
```

这样划分后得到两个子序列(a[low]，a[low+1]，…，a[i-1])和(a[i+1]，a[i+2]，…，a[high])，并且枢轴 a[i](也就是 pivot)已被放到正确的位置上，然后再分别对(a[low]，a[low]，…，a[i-1])和(a[i+1]，a[i+2]，…，a[high])子序列进行划分，直到每个子序列的长度都为 1 或 0 时为止。

一遍快速排序的具体方法是，设枢轴是 a[low]，首先从 high 所指位置开始向左搜索到第一个小于 a[low]的数据元素 (此数据元素仍假设为 a[high])为止，然后交换 a[low]和 a[high]，这时枢轴为 a[high]，再从 low 所指位置开始向右搜索到第一个大于 a[high]的数据元素(此数据元素仍假设为 a[low])为止，然后交换 a[low]和 a[high]，这时枢轴为 a[low]；重复这两步直到 low=high 时为止，这时枢轴的位置为 low。

图 3.1(a)是对关键字序列(49，38，66，97，38，68)第一遍快速排序的过程，图 3.1(b)是排序全过程。



(a) 一遍快速排序

初始序列	49	38	66	97	38	68
一次划分后	(38	38)	**49**	(97	66	68)
继续划分	**38**	(38)		(68	66)	**97**
		结束		(66)	**68**	
				结束		
有序序列	38	38	49	66	68	97

(b) 排序全过程

图 3.1 快速排序示意图

程序演示见 3cs1.mp4～3cs3.mp4。

下面是快速排序算法及测试程序上机操作步骤。

(1) 建立工程 quick_sort。

(2) 建立头文件 quick_sort.h,实现快速排序算法,具体内容如下:

```
#ifndef __QUICK_SORT_H__                    //如果没有定义__QUICK_SORT_H__
#define __QUICK_SORT_H__                    //那么定义__QUICK_SORT_H__

template <class Type>
int Partition(Type a[], int low, int high)
//交换 a[low..high]中的元素,使枢轴移动到适当位置,要求在枢轴之前的元素
//不大于枢轴,在枢轴之后的元素不小于枢轴的,并返回枢轴的位置
{
    while (low<high)
    {
        while (low<high&&a[high]>=a[low])
        {    //a[low]为枢轴,使 high 右边的元素不小于 a[low]
             high--;
        }
        Type tem=a[low]; a[low]=a[high]; a[high]=tem;    //交抽 a[low]、a[high]

        while (low<high&&a[low]<=a[high])
        {    //a[high]为枢轴,使 low 左边的元素不大于 a[high]
             low++;
        }
        tem=a[low]; a[low]=a[high]; a[high]=tem;       //交抽 a[low]、a[high]
    }
    return low;                                        //返回枢轴位置
}

template <class Type>
void QuickSortHelp(Type a[],int low,int high)
                                          //对数组 a[low..high]中的元素进行快速排序
{
    if (low<high)
    {    //子序列 a[low..high]长度大于 1
         int pivotLoc=Partition(a,low,high);   //进行一遍找分
         QuickSortHelp(a,low, pivotLoc -1);    //对序列 a[low, pivotLoc-1]递归排序
         QuickSortHelp(a,pivotLoc+1, high);    //对序列 a[pivotLoc+1,high]递归排序
    }
```

```
}

template <class Type>
void QuickSort(Type a[],int n)                    //对数组 a 进行快速排序
{ QuickSortHelp(a,0,n - 1); }

#endif
```

(3) 建立源程序文件 main.cpp,实现 main()函数,具体代码如下:

```
//文件路径名: quick_sort\main.cpp
#include <iostream>                               //编译预处理命令
using namespace std;                              //使用命名空间 std
#include "quick_sort.h"                           //快速排序

int main()                                        //主函数 main()
{
    int a[]={49,38,66,97,38,68};                  //数组
    int i,n=6;                                    //定义变量
    cout<<"排序前:";
    for (i=0; i<n; i++) cout<<a[i]<<" ";          //输出 a
    cout<<endl;                                   //换行

    QuickSort(a, n);                              //快带排序

    cout<<"排序后:";
    for (i=0; i<n; i++) cout<<a[i]<<" ";          //输出 a
    cout<<endl;                                   //换行

    return 0;                                     //返回值 0, 返回操作系统
}
```

程序运行时屏幕输出如下:

```
排序前:49 38 66 97 38 68
排序后:38 38 49 66 68 97
```

3.5 程序陷阱

模板是标准 C++ 新加入的功能,各 C++ 编译器对模板处理细节可能所有区别,例如对于类模板的友元函数模板,各 C++ 编译器就有不同的使用格式。

对于 Visual C++ 6.0 和 Visual C++ 2022,可采用如下格式:

```
//本程序适合于 Visual C++6.0 和 Visual C++2022
//文件路径名:trap 3_1\main.cpp
```

```
#include <iostream>                                    //编译预处理命令
using namespace std;                                   //使用命名空间 std

//声明类模板 MyCalss
template <class Type>
class MyCalss
{
private:
//数据成员
    Type value;                                        //存储数据元素值

public:
//公有函数
    MyCalss(const Type &v): value(v){ }                //构造函数
    template <class Type>
    friend void Show(const MyCalss<Type>&obj);         //输出对象的信息
};

template <class Type>
void Show(const MyCalss <Type>&obj)                    //输出对象的信息
{ cout<<obj.value<<endl; }

int main()                                             //主函数 main()
{
    MyCalss <int> obj(8);                              //定义对象
    Show(obj);                                         //输出对象的信息

    return 0;                                          //返回值 0, 返回操作系统
}
```

对于 Visual C++ 6.0,还可去掉类体模板中的“template <class Type>”,也就是采用如下格式:

```
//本程序适合于 Visual C++6.0
//文件路径名:trap 3_2\main.cpp
#include <iostream>                                    //编译预处理命令
using namespace std;                                   //使用命名空间 std

//声明类模板 MyCalss
template <class Type>
class MyCalss
{
private:
//数据成员
    Type value;                                        //存储数据元素值
```

```
public:
//公有函数模板
    MyCalss(const Type &v): value(v){ }                    //构造函数模板
    friend void Show(const MyCalss<Type>&obj);             //输出对象的信息
};

template <class Type>
void Show(const MyCalss<Type>&obj)                         //输出对象的信息
{ cout<<obj.value<<endl; }
…
```

对于 Dev-C++ 5.11 只能将模板函数的定义放在类模板体内，并且不能加“template <class Type>”，也就是采用如下格式：

```
//本程序适合于 Dev-C++
//文件路径名:trap3_3\main.cpp
#include <iostream>                                        //编译预处理命令
using namespace std;                                       //使用命名空间 std

//声明类模板 MyCalss
template <class Type>
class MyCalss
{
private:
//数据成员
    Type value;                                            //存储数据元素值

public:
//公有函数模板
    MyCalss(const Type &v): value(v){ }                    //构造函数模板
    friend void Show(const MyCalss<Type>&obj){ cout<<obj.value<<endl; }
                                                           //输出对象的信息
};
…
```

习　题　3

一、选择题

1. 下列关于模板的叙述中，错误的是_____。

A. 模板声明中的第一个符号总是关键字 template

B. 在模板声明中用“<”和“>”括起来的部分是模板的类型形参表

C. 类模板不能有数据成员

D. 在一定条件下函数模板的类型实参可以省略

2. 有以下函数模板定义：

```
template <class Type>
Type Fun(const Type &x, const Type &y) { return x * x+y * y; }
```

在下列对 Fun()的调用中，错误的是_____。

A. Fun(3，5)　　B. Fun(3.0，5.5)

C. Fun(3，5.5)　　D. Fun<int>(3，5.5)

3. 关于关键字 class 和 typename，下列表述中正确的是_____。

A. 程序中 typename 都可以替换为 class

B. 程序中的 class 都可以替换为 typename

C. 在模板类型形参表中只能用 typename 声明参数的类型

D. 在模板类型形参表中只能用 class 声明参数的类型

4. 有以下函数模板：

```
template <class Type>
Type Square(const Type &x) { return x * x; }
```

其中，Type 是_____。

A. 函数形参　　B. 函数实参

C. 模板类型形参　　D. 模板类型实参

5. C++ 中的模板包括_____。

A. 对象模板和函数模板　　B. 对象模板和类模板

C. 函数模板和类模板　　D. 变量模板和对象模板

二、填空题

1. 已知一个函数模板的声明如下：

```
template <typename T1,typename T2>
T1 Fun(T2 n) { return n*5.0; }
```

若要求以 int 型数 7 为函数实参调用该模板函数，并返回一个 double 型数，则该调用应表示为_____。

2. 已知 int dbl(int n) { return n+n; }和 long dbl(long n) { return n+n; }是一个函数模板的两个模板函数，则该函数模板的声明是

```
template <typename Type>

_____
```

3. 下面程序的运行结果是_____。

```
//文件路径名:ex3_2_3\main.cpp
#include <iostream>                                    //编译预处理命令
using namespace std;                                   //使用命名空间 std

template <class Type>
```

```
Type Min(const Type &a, const Type &b)                    //求 a、b 的最小值
{
    if (a<b) return a;                                    //a 更小
    else return b;                                        //b 更小
}

int main()                                                //主函数 main()
{
    int n1=4,n2=5;                                        //定义整型变量
    double d1=0.35,d2=4.4;                                //定义实型变量
    cout<<"最小整数="<<Min(n1, n2)<<","<<"最小实型="<<Min(d1, d2)<<endl;
                                                          //输出信息

    return 0;                                             //返回值 0, 返回操作系统
}
```

三、编程题

1. 试使用函数模板实现输出一个数组各元素的值，要求编写测试程序。

2. 编写一个复数类模板 Complex，其数据成员 real 和 image 的类型未知，定义相应的成员函数模板，包括构造函数模板、输出复数值的函数模板、求复数和的函数模板和求复数差的函数模板，主函数中定义模板类对象，分别以 int 和 double 实例化类型参数。

*3. 编写一个使用数组类模板 Array 对数组求最大值和求元素和的程序，要求编写出测试程序。

*4. 对数组求最大值和求元素和的算法都编写为函数模板，要求编写出测试程序。

**5. 对数组求最大值和求元素和的函数采用静态成员函数的方式封装成数组算法类模板 ArrayAlg，要求编写出测试程序。