

高级语言程序设计II

四川大学计算机学院

四川大学计算机学院

四川大学计算机学院

陈良银

cly_6666@sina.com

<http://cs.scu.edu.cn/~chenliangyin>

2005-7-1

星期一、第4大节、N1-A509。

面向对象程序设计语言C++

- 第1章 [引论](#)
- 第2章 [C++语言与C语言的不同](#)
- 第3章 [类类型](#)
- 第4章 [运算符重载](#)
- 第5章 [派生类](#)
- 第6章 [流库](#)
- 第7章 [模板](#)
- 第8章 面向对象设计技术（自学）
- 第9章 命名空间和例外处理（自学）

第六章

文件 I/O

- C++ 中，要进行文件I/O，首先必须创建一个流，然后将这个流与文件相关联（称为**打开文件**），
- 这时才能进行**读和写**操作，
- 使用完后，需**关闭**文件。

6.1 文件的打开和关闭

- C++有三种类型的文件流：**输入文件流、输出文件流和输入/输出文件流。**
- 要打开一个输入文件流，必须定义类型为**ifstream**的对象；
- 要打开一个输出文件流，必须定义一个类型为**ofstream**的对象；
- 要建立输入和输出的流，必须定义一个类型为**fstream**的对象。

- 例如，下一程序段建立一个输入流、一个输出流和一个输入/输出流：
- **ifstream in; //input**
- **ofstream out; //output**
- **fstream both; //input and output**

- 一旦建立了一个流，将它与文件相关联的一种方法是使用函数 **open()**。

- **ofstream ofile; // 创建输出文件流**

- **ofile.open("payroll");**

- **ofile.close();**

- **ofile.open("employee"); // 重用 ofile**

- 上述程序段也可写为
- **ofstream ofile("payroll");**
- **ofile.close();**
- **ofstream ofile("payroll");**



创立流并与文件关联。

- **ofstream**类 **自动打开文件的构造函数**，
- 该构造函数的参数和缺省值与**open()**函数相同。

- **open(···)**函数的函数原型为

- **void open(const char * ,
int filemode,
int = filebuf::openprot)**

- 第一个参数表示相**关联的文件名**,
- 第二个参数表示文件的**打开方式**,
- 第三个参数是文件的**保护方式**, 与操作系统有关, 用户一般只使用缺省值。

文件的打开方式

方式 动作

ios::app 追加数据（总是写到文件尾）

ios::ate 在原打开文件上找到文件尾

ios::in 打开输入（对 **ifstream** 适用）

ios::out 打开输出（对 **ofstream** 适用）

ios::binary 以二进制方式打开文件，

（**缺省时为文本方式**）

- **ios::trunc** 若文件存在则清除原内容
(若指定 **ios::out**,
但未指定 **ios::ate**
或 **ios::app**,
则这种方式为隐含的)
- **ios::nocreate** 若文件不存在, 则打开失败
- **ios::noreplace** 若文件存在, 除非设置 **ate**
或 **app**, 否则打开输出失败

- **filemode** 表示文件的打开方式，可以将几种方式通过“或”操作结合起来，
- 例如，打开一个供读和写的文件，其方式可以定义为 **ios::in | ios::out**。
- **filemode** 是一个缺省参数，打开输入文件时，它缺省为 **ios::in**；当打开输出文件时，缺省为 **ios::out**。

- 缺省情况下，文件用**正文方式**打开。
- 这就是说，在输入时，回车 / 换行序列要转换为字符 ‘\n’。
- 在输出时，字符 ‘\n’ 转换为回车 / 换行序列。
- 这些转换在二进制方式下是不进行的。
- 这是**正文方式和二进制方式主要的区别**。

6.2 文件的读写

文件**读**操作是从流中**取**一个元素，

文件**写**操作是向流中**写(存)**一个元素。

ofstream类从**ostream**中继承了输出操作，

而 **ifstream** 类从 **istream** 中继承了输入操作。

当一个输入文件流、输出文件流或输入 / 输出文件流建立后，对文件的读写就像控制台读写一样的方便。

- 将文件file_from拷贝到文件 file_to。
- [fileCopy.cpp](#)

- 将一个整数、一个浮点数和一个串写到 **test** 文件中。
- [writeFile.cpp](#)
- 则**E:\\text**文件的内容为：
- **10 123.23 This is a short text file.**

- 从上面程序建立的文件中读入一个整数、一个 float、一个字符和一个串。
- [readFile.cpp](#)
- 该程序的输出为
- **10 123.23 T**
- **his**
- 在缺省情况，**>>** 运算符跳过空白，然后读入对应于输入对象类型的字符。

第七章

模板

- C++有两种模板：

函数模板和类模板。

函数模板与模板函数

- 考虑返回两参数中较大者的函数 $\max(x, y)$ 。
- x 和 y 为具有可比较次序的任何类型。
- 但是，因为 C++ 是强类型语言，它希望参数 x 和 y 的类型在编译时就声明。
- 需要 $\max()$ 的许多重载版本，在这些重载版本中，**每个版本的代码是相同的，但是形参代表的数据类型却不相同。**

```
int max(int x, int y)
{
    return (x > y) ? x : y;
}
```

```
long max(long x, long y)
{
    return (x > y) ? x : y;
}
```

...

- 解决这个问题的一种方法是使用宏：

```
#define max(x,y)  
(((x)>(y))?(x):(y))
```

- 然而，使用 #define **避开了类型检查机制**，使得宏替换在用户不希望其发生的地方出现，

- 例如，使用宏可能会导致在一个 int 参数和一个 struct 参数之**间类型不匹配，无法比较。**

- 另一种解决办法是使用模板。如果使用模板，数据类型本身就是一个参数：

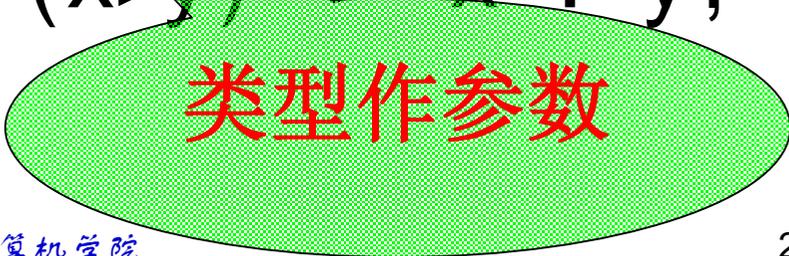
- `template <class T>`

- `T max(T x, T y)`

- `{`

- `return (x > y) ? x : y;`

- `}`



类型作参数

- 关键字 `template` 表示正在声明一个模板，数据类型参数T由模板参数 `<class T>` 给出。
- 该模板的含义为，无论模板参数T实例为 `int`、`char` 或任意其他类型，包括类类型时，函数 `max` 就为实例化了的类型的参数求最大值。

- max函数并不是一个完全的函数。称它为**函数模板**。
- 因此，**函数模板代表了一类函数**，而且它不是一个完全的函数，**必须将其模板参数T实例化后**，才能完成具体函数的功能。

- 将T实例化的参数常常称为**模板实参**。
- 用模板实参实例化的函数称为**模板函数**。

- 一个函数模板提供一类函数的抽象，它以任意类型 T 为参数。
- 由一个函数模板产生的函数称为模板函数，它是函数模板的具体实例。

已经是一个完全函数。

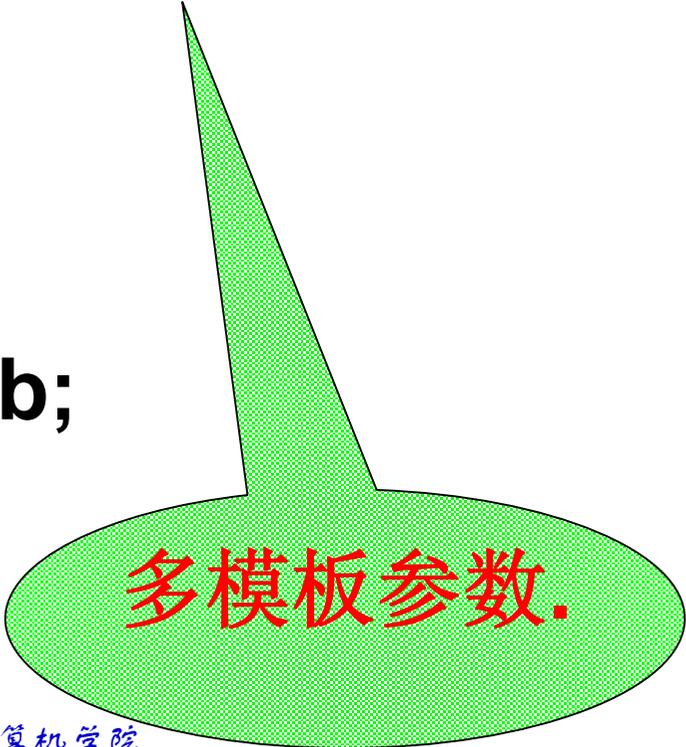
- **函数模板**将具有相同程序正文的一类函数抽象出来，可以适应任意类型T。
- 函数模板对某一特定类型的实例就是**模板函数**。
- 函数模板代表了一类函数，模板函数表示某一**具体的函数**。

重载模板函数

- 有些特殊情况需要函数模板参与重载，
- [reLoadFuncTemplate.cpp](#)

- 为了解决这个问题,
- **C++** 允许一个函数模板可以使用 **多个模板参数** 或者 **超载** 一个函数模板。

```
template <class T, class D>  
T max(T a, D b)  
{  
    return (a>b)? a : b;  
}
```



多模板参数.

- C++ 允许函数模板被一个或多个同名的**非模板函数**重载。
- [reLoadFuncTemplate2.cpp](#)

- 在C++中，函数模板与同名的非模板函数的重载方法遵循下述**约定**：

- 1. **寻找**一个参数完全匹配的**函数**，如果找到了，就调用它。
- 2. 寻找一个**函数模板**，将其实例化产生一个匹配的模板函数，如果找到了，就调用它。
- 3. 试一试低一级的对**函数的重载方法**，如通过**类型转换**可产生参数匹配等，如果找到了，就调用它。

- 如果(1)(2)(3)均未找到匹配的函数，那么这个调用是一个**错误**。

类模板与模板类

- 定义一个整数类:
- [integerClass.cpp](#)
- 再定义一个浮点类:
- [realClass.cpp](#)

- 如果还需要定义字符数据类等，那就还要定义许多的类，但它们都很类似，
- 仅仅是处理的数据的类型不同而已。
- 可以定义类模板，解决这个重复现象。 [classTemplate.cpp](#)

- 考虑如下**向量类**（一维数组）的例子。
- 不管整型向量还是任何其他类型的向量，在所有类型上进行的**基本操作是相同的**（如插入、删除、检索等）。
- [vectorTemplate.cpp](#)

- **Vector**被称为**类模板**。
- **Vector<T>**是该**类模板的名字**。
- **Vector<int>**则是一个**模板类**。

课程结束。

谢谢大家。