2024年春

程序设计实习：C++程序设计

第九讲 函数模板-上

贾川民
北京大学

函数模板

- **函数模板**
  - 模板函数的重载
- **类模板**
  - 继承
  - static成员
  - 友元
- **String类**

模板的基本概念

- **排序问题**
  - 实际应用中经常遇到的问题
    - 对学生按成绩**排序**
    - 对事件按发生的时间**排序**
    - 对产品按销售量和顾客的投诉率**排序**
  - 采用的**排序算法完全相同**

模板的基本概念

排序问题

不同类型的对象, 排序时的不同: **比较两个对象的大小**

- 函数sortA用于对类A的一组对象排序
- 函数sortB用于对类B的一组对象排序
- 类A和类B都重载了关系运算符 ">"
- sortA和sortB中, 除了被排序数组, 存储被排序数组元素的变量的**类型声明不同**之外, 其它部分可以完全相同

模板的基本概念

有两种解决办法:

- 函数重载
- 函数模板

模板的基本概念

- **函数重载的办法**

  sort(…classA …){…}

  sort(…classB …){…}

  - 分别写**两个同名的函数**
  - 由编译系统根据函数**调用时实参的类型**，确定实际执行哪个函数

- **函数模板**

  template<class T>

  ***return-type* sort(…T…)**

  - 由编译系统根据sort函数**调用时实参的类型**，自动生成相应的模板函数

输出全部数组元素的函数模板

函数模板的类型参数

```
template<class T>
void print( const T array[], int size ){
    int i;
    for ( i =0; i<size; i++) cout<<array[i];
    return;
}
Cstudent undergraduates[number1], graduates[number2];
string  telephoneNubmers[nubmer3];

…
print(undergraduates, number1);

…
print(telephoneNumbers, number3);

…
print(graduates, number2);
…
```

输出全部数组元素的函数模板

```
void print( const CStudent array[], int size){
    int i;
    for ( i =0; i<size; i++) cout<<array[i];
    return;
} // 编译到print(undergraduates, number1)时自动产生

void print( const string array[], int size){
    int i;
    for ( i =0; i<size; i++) cout<<array[i];
    return;
} // 编译到print(telephoneNumbers, number3)时自动产生
```

要能编译通过，要求对 << 有适当的重载

一个函数模板可以有多个类型参数

```
template<class T1, class T2>
void print(T1 arg1, T2 arg2)
        {   cout<<arg1<<"   "<<arg2<<endl;  return;   }
```

- **函数模板的参数类型**
  - 可以用类型参数说明
  - 也可以用**基本数据类型**, 其他的类说明

```
template<class T1, class T2>
void print(T1 arg1, T2 arg2, string s, int k)
{     cout<<arg1<<s<<arg2<<k<<endl;  return;     }
```

一个函数模板可以有多个类型参数

- 函数模板的类型参数可以用于函数模板的局部变量声明

```
template<class T1, class T2>
void print(T1 arg1, T2 arg2)
{
    T1  locVar=arg1;
    cout<<locVar<<"    "<<arg2<<endl;
    return;
 }
```

一个函数模板可以有多个类型参数

- 赋值兼容原则引起函数模板中类型参数的二义性

```
template<class T>
T myFunction( T arg1, T arg2)
{ cout<<arg1<<“   ”<<arg2<<“\n”; return arg1;} …
myFunction(5, 7); //ok: replace T withint
myFunction(5.8, 8.4); //ok: replace T with double
myFunction(5, 8.4); //error: replace T withintor double? 二义性
```

- 可以在函数模板中使用多个类型参数,可以避免二义性

```
template<class T1, class T2>
T1 myFunction( T1 arg1, T2 arg2)
{ cout<<arg1<<“   ”<<arg2<<“\n”; return arg1;} …
myFunction(5, 7); //ok: replace T1 and T2 withint
myFunction(5.8, 8.4); //ok: replace T1 and T2 with double
myFunction(5, 8.4); //ok: replace T1 withint, T2 with double
```

重载与函数模板

- 函数模板与函数模板的重载：同一函数名，参数的数量不同

```
template<class T>
T myFunction( T arg )
{ cout<< “one argument\n”; return arg;}

template<class T1, class T2>
T1 myFunction( T1 arg1, T2 arg2 )
{ cout<< “two arguments\n”; return arg1;}
   …
myFunction(5); //ok: replace with int
myFunction(5.8, 8.4); //ok: replace T1 and T2 with double
myFunction(5, 8.4); //ok: replace T1 with int, T2 with double
```

函数和模板的匹配顺序

1) 先找一个参数完全匹配的**函数**

2) 再找一个参数完全匹配的**模板**

3) 在没有二义性的前提下, 再找一个参数经过自动转换后能够**匹配的函数**

4) 都找不到, 则报错

函数和模板的匹配顺序

```
template <class T>
void Max( T a, T b ) {
        cout << "TemplateMax" <<endl;
        return 0;
}
void Max(double a, double b){
        cout << "MyMax" << endl;
        return 0;
}
main(){
    int i=4, j=5;
    Max(1.2, 3.4); //输出MyMax
    Max(i, j);       //输出TemplateMax
    Max(1.2, 3);   //二义性, 强制类型转换3为double, 调用Max函数
}
```

例：函数模板调用顺序

```
template <class T>
void Max(T a, T b){
        cout << "Template Max 1" <<endl;
        return 0;
}
template <class T, class T2>
void Max(T a, T2 b){
        cout << "Template Max 2" <<endl;
        return 0;
}
```

```
void Max(double a, double b){
        cout << "MyMax" << endl;
        return 0;
}
int main()
{
        int i=4, j=5;
        Max(1.2, 3.4);  //调用Max(double, double)函数
        Max(i, j);      //调用第一个T Max(T a, T b)模板生成的函数
        Max(1.2, 3);    //调用第二个T Max(T a, T2 b)模板生成的函数
        return 0;
}
```

运行结果:

```
MyMax
Template Max 1
Template Max 2
```

泛型程序设计

- **Generic Programming**
- 算法实现时不指定具体要操作的数据的类型
- 泛型 — 算法实现一遍　　适用于多种数据结构
- **优势:** 减少重复代码的编写
- 大量编写模板, 使用模板的程序设计
  - 函数模板
  - 类模板

函数模板

- 用 函数模板 解决

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

Thanks!