

泛型（模板）程序设计

Generic Programming

徐延宁 xyn@sdu.edu.cn

数字媒体技术教育部工程研究中心

山东大学软件学院

1



主要内容

- 泛型的基本概念
- 函数模板
- **类模板**
- C++标准模板库简介
- ~~λ表达式 (匿名函数)~~



问题的提出

- 在程序设计中，经常需要用到一些功能、实现相同，但所涉及数据类型不同的函数，例如，排序：

```
void int_sort(int x[],int num);
```

```
void double_sort(double x[],int num);
```

```
void A_sort(A x[],int num);
```

- 三个sort函数的实现是一样的（如都采用冒泡法），因而希望只写一个 sort函数。



问题的提出

- 栈、链表等容器类（数据结构），所有栈都有push, pop等操作，所有链表都是遍历、插入、删除等操作。

```
class IntStack
{
    int buf[100];
public:
    bool push(int);
    bool pop(int&);
};
```

```
class AStack
{
    A buf[100];
public:
    bool push(A);
    bool pop(A&);
};
```

```
class DoubleStack
{
    double buf[100];
public:
    bool push(double);
    bool pop(double&);
};
```

希望只写一个通用的Stack



泛型程序设计

- 在程序设计中，以类型作为参数。
 - `int x;` vs `<T1> x;`
 - T1可以是int, double或者magazine
- 泛型函数-包含类型参数的函数
- 泛型类-包含类型参数的类
- 相关程序设计技术称为：泛型程序设计、类属程序设计、Generic Programming



- C++提供了两种实现泛型的机制：1、通用指针类型；2、模板
- 例子：C语言库函数**qsort**，快速排序，可以对任意类型的数组数据进行排序
void qsort(void *base, int nelem, unsigned int width,
int (* pfCompare)(const void *, const void *));
 - 1) 数组起始地址：base
 - 2) 数组元素的个数：nelem
 - 3) 每个元素的大小（字节为单位）：width

万能的**void ***
不是本章重点





```
int cmp1(const void *a, const void *b){  
    return *(unsigned int *)a - *(unsigned int *)b; // 从小到大排序  
}  
int cmp2(const void *a, const void *b){  
    return *(float *)b - *(float *)a; // 从大到小排序  
}
```

```
#include <stdio.h>  
#include <stdlib.h> // qsort  
int main(){  
    unsigned int a[] = { 8, 123, 11, 10, 4 };  
    float f[] = { 1.0, 2.7, 3.6, 0.8, 4.1 }  
    qsort( a, 5, sizeof(unsigned int), cmp1);  
    qsort( f, 5, sizeof(float), cmp2);  
}
```



泛型函数

- C++提供了两种实现泛型函数的机制：1、通用指针类型；2、模板

```
template <class T>
void sort(T elements[], unsigned int count)
{
    ...
}
```

```
int a[100];
sort(a,100); //对int类型数组进行排序
```

```
double b[200];
sort(b,200); //对double类型数组进行排序
```

```
A c[300]; //类A中需重载操作符：<，需要
          时还应自定义拷贝构造函数和重载操作符=
sort(c,300); //对A类型数组进行排序
```

编译器实际生成了3个sort函数

```
void sort(int elements[], unsigned int count);
void sort(double elements[], unsigned int count);
void sort(A elements[], unsigned int count);
```




泛型函数

函数模板

- 函数模板定义:
 - 首先, 定义函数中用到的类型参数
template <class T1, class T2, ...> //T1、T2等是类型参数
 - 然后, 应用类型参数定义函数, 函数的**返回值类型**、**参数列表**、以及**函数体**内的局部变量类型可以使用类型参数T1、T2
- 函数模板定义了一系列函数。
- 函数模板的使用: 对函数模板进行**实例化** (明确T1, T2的具体类型, 生成具体的函数)。
 - 编译程序根据实参的类型**自动地**推演T1,T2的类型 (template argument deduction)。

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

```
int a[3];
a[2]=max (a[0],a[1]);
实例化, 推断T为int:
Int max(int,int)
Dog a[3];
a[2]=max<Dog> (a[0],a[1]);
实例化, T为Dog:
Dog max(Dog,Dog)
```



- 除了类型参数外，函数模板也可以带有非类型参数，一般配合数组使用（因为数组的长度一旦确定，不能更改）。例如：

```
template <class T, int size> //size为一个int型的普通参数。  
void f()  
{ T temp[size]; ..... }  
f<int,10>(); //生成函数f(), 该函数体中 int temp[10]。  
f<int,100>(); //生成函数f(), 该函数体中 int temp[100]。
```

- 实际生成的函数名不会是f，由编译器确定(f_10, f_100)，程序员不需要关心



- 如果一个类的成员的类型可参，则称该类为泛型类（模板类），用类模板实现。

```
template <class T1,class T2,...>
class 类名
{
    类成员说明
}
```

其中，T1、T2等为类模板的类型参数，在类成员的说明中可以用T1、T2等来说明它们的类型。

```
template <class T>
class Stack{
    T buffer[100];
    int top;
public:
    Stack() { top = -1; }
    bool push(const T &x);
    bool pop(T &x);
    const T &peek() const;    //访问栈顶元素
    bool isEmpty() const;    //测试是否栈满
    bool isFull() const;    //测试是否栈空
};
```



类模板的使用

非类型参数

- 除了类型参数外，类模板也可以包括非类型参数。例如：

1、声明

```
template <class T, int size>
class Stack
{
    T buffer[size];
    int top;
public:
    Stack() { top = -1; }
    bool push(const T &x);
    bool pop(T &x);
};
```

2、实现

```
template <class T, int size>
bool Stack<T, size>::push(const T &x) {
    if (top == size - 1) { resize(); }
    buffer[++ top] = value;
}
```

3、使用

```
Stack<int, 100> st1; //st1为元素个数最多
                    为100的int型栈
st1.push(50);
```



类模板的使用

类模板

- 编译程序将会对模板类进行实例化，生成若干类。
- 模板类的实例化需要在程序中显式地指出，编译器不能推演。例如：

```
Stack<int> st1; //创建一个元素为int型的栈对象。
```

```
int x; st1.push(10); st1.pop(x);
```

```
Stack<double> st2; //创建一个元素为double型的栈对象。
```

```
double y; st2.push(1.2); st2.pop(y);
```

```
Stack<A> st3; //创建一个元素为A类型的栈对象（A为定义的一个类）。
```

```
A a,b; st3.push(a); st3.pop(b);
```

使用模板类，定义Stack、Queue/Vector等
End of Course,
Beginning of C++ Practice

14



课程复习考试

- 选择题2*10: 细节
- 程序阅读 4*5: 有一定难度
- 编程-控制流程 10*3: 循环、分支、递归
- 编程-面向对象 10*3: 运算符重载、继承、类模板

- 编程题目。
 - 面向对象编程30分, 基本套路; 重点练习, 复习收益最高;
 - 控制流程30分, 基本功, 没有难度, 几乎和课件、课程无关 (不学也会); 例如, 递归就是斐波那契数列的难度, 分支竟然没有循环, 远远小于实验作业

- 阅读, 选择。难度中等: 逻辑不难, 但涉及大量细节
 - 看课件 (Optional的都不涉及)

15