模板

实例 instantiation

未定义 声明 只是写个格式 没有具体类型 如 void Swap( T &a, T &b);

调用时 传递的形参 是有具体的类型 这种是隐式实例化 implicit instantiation

工作机制 ：// 是由编译器根据形参来 定义 因为传递的时候调用的值是由具体类型的

显示实例化 explicit instantiation 有具体类型 如 Swap<int>

语法 声明所需的种类 用 < > 符号指示类型 ，并在声明前加上关键字 template

正确格式 template void Swap<int>( int, int);

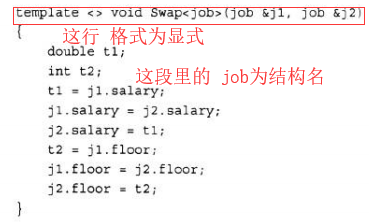
显示具体化

template <> void Swap <int>(int &, int &); 或者

template<> void Swap(int &, int &); //两种写法的意思一样

工作机制 ：// 注意 具体化 对比实例化的 在template 后面有<> 声明的意思为

在代码中 应查找到 更加具体的模板 此模板 必须是 使用专门为<>中类型所定义过的函数比如结构 这种的 而不是去使用Swap( )模板

 调用 类型为 结构 job

选择函数版本 多个函数模块编译器的优先选择

优先权

：char 优先 模块 优先 int 优先 float

完全匹配 和 最佳匹配 //叙述的是 当多个函数版本中 有至少2个函数符合，则出现两种更细的匹配制度

术语 Type表示任意类型 如： int实参 与 int&形参完全匹配

char & 到 const char& 的转换

下图表格意思 ：

// 实参的函数名与用作形参的函数指针只要返回类型和参数列表相同，就是匹配到。



完全匹配中的区分 **：**

**// 如果两个函数都完全匹配则可以完成重载解析**

**但 只在以下情况 // const 和非const 之间完全匹配只在 指针 和引用 指向的数据**

**否则 即使 完全匹配也会报错 ambiguous （二义性）**

优先级

**非模板函数 优先 模板函数（包括显示具体化）**

如果两个完全匹配的函数都是模板函数 **则**

**优先全是 显示具体化 优先 显示实例化 优先 隐式实例化**

**其中**

**template <class Type> void recycle (type t); // #1**

**template <class Type> void recycle (type \* t); //#2**

**struct blot (int a, char b[10];);**

**blot ink = (25, “spots”);**

**recycle(&ink);**

**其中 #1 #2中模板 #1 Type解释为 blot\* #2 recycle（&ink）解释为 ink**

**recycle<blot \*>(blot \*) 和 recycle<blot >(blot \*) 发送到可行函数池中**

**则 #2更具体** 这种规则被称为函数模板的部分排序规则 partial ordering rules

部分排序规则

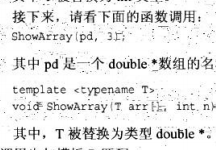
语法 ShowArray（things ,6）;

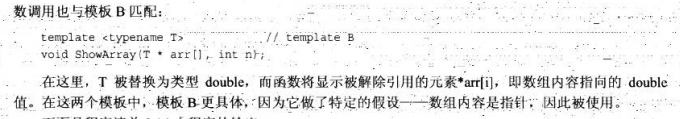
标识符things 是一个int数组的名称 所以模板匹配是

template<typename T> void ShowArray(T arr[], int n);

// T 被替换为 int类型

同理 图1模板A 图2模板B

 原型 double \* pd[3];



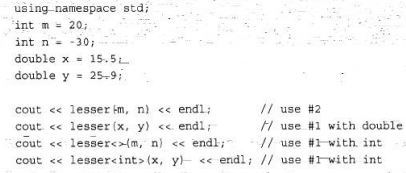
其中模板A 显示的是地址 模板B 显示的是值因为模板B 被解除了

将模板函数定义在main函数前面

函数原型

 // 模板函数

// 非模板函数



其中如果 函数调用 模板函数和非模板函数匹配则 非模板函数优先

其中 lesser(x,y); 由于 x 和 y类型为 double 选择 模板函数

接下来 lesser<>(x,y); 此时 由于<>指出 选择模板函数，而不是非模板函数 所以强制转换 double 转换成 int 对模板进行实例化

lesser<int>( x,y) 语句要求进行显式实例化 使用int替代T将使用显式实例化得到的函数 x和y 的值被强制转换成int 该函数返回一个int值

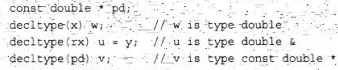
关键字 decltype //后置返回类型

工作机制：// 在编写模板函数时没有确定是什么类型的前提下来用来提供参数确定类型

语法： decltype(x+y) xpy ; xpy = x + y; 合并 decltype(x+y) xpy =x+y;

用来判断类型的四个步骤 **decltype(expression) var;**

第一步：如果 expression 是一个没有用括号括起来的标识符，则var的类型与该标识符的类型相同，包括const等限定符 x是左值 所以w是double

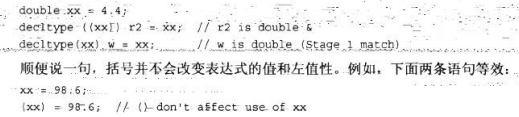


第二步：如果expression是一个函数调用，则var的类型与函数的返回类型相同

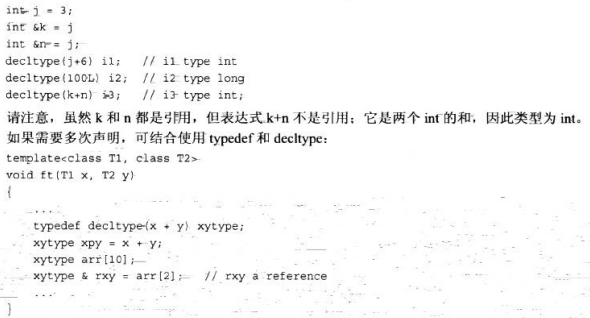
并不会去调用函数 至少查看函数的原型返回类型给decltype



第三步expression不能是未用括号括起来的标识符 ，当expresion是用括号括起来的标识符



第四步 如果前面的都不满足 则var的类型与expression的类型相同



auto 用于函数定义

double h(int x, float y);

auto h (int x, float y) -> double;

