为什么使用函数模板？

举例，比较两个形参的大小，返回较大的一个。

首先使用函数重载，编写形参是int，double，char等类型的一系列函数，

charcompare(chara,charb){

returna>b?a:b;

}

intcompare(inta,intb){

returna>b?a:b;

}

如果使用函数模板，只要编写一个函数模板，然后直接调用模板，根据形参自动推出数据类型。

template<typename T>Tcompare(Ta,Tb){

returna>b?a:b;

}

由此可以得出函数模板的优点：

1. 一个模板可以生成一系列“特殊的”重载函数，避免逐个编写只是形参数据类型不同的一系列函数，减少了程序员工作量。
2. 函数模板生成模板函数，被调用时才生成相应的模板函数。不使用模板函数的方式，会把所有潜在可能调用的函数都编写了，而实际未必在程序中调用。可见，函数模板，减小了代码的体量。

如何定义函数模板？

定义：

template<typenameT1,typenameT2,typenameT3...>

voidfun(T1t1,T2t2,T3t3){}

声明：

template<typenameT1,typenameT2,typenameT3...> voidfun(T1t1,T2t2,T3t3);

原则：template指定的类型，都要在函数中出现！

如何调用函数模板？

函数模板的所有typename类型，调用该函数模板时，必须全部能推断出。

推断的依据：函数名<>指定 或 实参类型

原则：类型是根据传入的实参类型推出的，传入的实参要满足规则，不能产生矛盾。

1. 函数返回类型和形参有关联

调用时，只需要传入实参即可，无需<>指定类型。但是，根据传入实参的类型推断template时，不能出现矛盾！

template<typenameT>

voidSwap(T&a,T&b){

Tt;

t=a;

a=b;

b=t;

}

//error

inta=0;

doubleb=1;

Swap(a,b);

//error

inta=0;

doubleb=1;

Swap(a,(int)b);

//OK

inta=0;

intb=1;

Swap(a,b);

//OK

template<typenameT>

voidSwap(Ta,Tb){

Tt;

t=a;

a=b;

b=t;

}

inta=0;

doubleb=1;

Swap<int>(a,b);

--------------------------------------------------------------------------

template<typenameT1,typenameT2>

voidfun(T1a,T2b,T2c){

cout<<a<<b<<<<c<<endl;

}

//error

inta=0;

doubleb=1;

intc=100;

fun(a,b,c);

//OK

inta=0;

doubleb=1;

intc=100;

fun<double,int>(a,b,c);

//OK

inta=0;

doubleb=1;

intc=100;

fun(a,b,(double)c);

1. 无法根据形参推出函数返回类型

template<typenameT1,typenameT2,typenameT3>

T1Sum(T2v1,T3v2){

return(v1+v2);

}

调用时，无法确定函数返回类型T1

Sum<int>(1.1,2.2)

函数名<>内指定T1（函数返回类型）

-----------------------------------------------------------------

template<typenameT1,typenameT2,typenameT3>

T3Sum(T1v1,T3v2){

return(v1+v2);

}

Sum<double,long,int>(1.1,2.2);

函数名<>指定T3，然而，指定T3必须按顺序指定，先指定T1，T2.和默认形参原理一样。

-----------------------------------------------------------------------------------------------------

函数模板和函数重载？

函数模板也能当做重载函数。

intMax(inta,intb)

{

cout<<"int Max(int a, int b)"<<endl;

returna>b?a:b;

}

template<typenameT>

TMax(Ta,Tb)

{

cout<<"T Max(T a, T b)"<<endl;

returna>b?a:b;

}

template<typenameT>

TMax(Ta,Tb,Tc)

{

cout<<"T Max(T a, T b, T c)"<<endl;

returnMax(Max(a,b),c);

}

intmain()

{

inta=1;

intb=2;

cout<<Max(a,b)<<endl;//当函数模板和普通函数都符合调用时,优先选择普通函数

cout<<Max<>(a,b)<<endl;//若显示使用函数模板,则使用<> 类型列表

cout<<Max(3.0,4.0)<<endl;//如果 函数模板产生更好的匹配 使用函数模板

cout<<Max(5.0,6.0,7.0)<<endl;//重载

cout<<Max('a',100)<<endl;//调用普通函数 可以隐式类型转换

return0;

}

类模板

/\*类模板的接口和实现都在头文件中

\*客户源代码如果想使用类模板，包含头文件即可

\*类模板的成员函数可能是函数模板。

\*/

#ifndefSTACK\_H

#defineSTACK\_H

#include<deque>

template<typenameT>

classStack{

public:

Stack(intn){

N=n;

}

boolisEmpty()const{

returnstack.empty()?true:false;

}

size\_tsize()const{

returnstack.size();

}

constT&top()const{

returnstack.back();

}

voidpush(constT&elem){

if(stack.size()>=N)

cout<<"stack overflow\n";

else

stack.push\_back(elem);

}

voidpop(){

if(Stack.isEmpty())

cout<<"stack is empty\n";

else

stack.pop\_back();

}

private:

std::deque<T>stack;

size\_tN;

};

#endif

模板的非类型形参

**表示一个固定类型的常量而不是一个类型。**

**这里要强调一点，我们对于非类型形参的限定要分两个方面看**

**1.对模板形参的限定，即template<>里面的参数**

**2.对模板实参的限定，即实例化时<>里面的参数**