模板

Tempalte

Why Templates?

- 泛型编程的需要。

  设想你对整数类型实现了一个排序算法：

  ```
  void sort(int *is,int n);
  ```

  用该函数可以对实、复数或工资单排序吗？

- 模板可以复用源代码-泛型编程.

```
inline void Swap( int &x, int &y){
  int t = x; x = y; y  =t;
}

inline void Swap(double &x, double &y){
  double t = x; x = y; y  =t;
}
```

Why Templates?

```
void f(){
   int a=3,b = 5;
   double  x= 2.4 ，  y= 9；
   char c =‘C’,, e = ‘L’;
   Swap(a,b);
   Swap(x,y);
   Swap(c,e);   //错 !
 }
```

Function Templates

```
inline void Swap(double &x, double &y){
   double t = x; x = y; y  =t;
}
```

```
template <class T>
 void Swap( T &x, T &y){
    T t = x; x = y; y  =t;
 }
```

Function Templates

```
struct  student{
    string name;
    int age;
};
void f(){
   string s1(“Li”),s2(“Wang”);
   student stu1,stu2;
   swap(s1,s2);
   swap(stu1,stu2);
}
```

Function Templates

- The format for declaring a function template is:

```
template <class identifier> function_declaration;
```

or

```
template <typename identifier> function_declaration;
```

```
template <class T>
  void Swap( T &x, T &y){
    T t = x; x = y; y  =t;
  }
```

模板头

Function Templates

- To invoke a function template, we use:

```
function_name <type> (parameters);
```

```
template <typename T>
T sum(const T a, const T b) {
    return a + b;
}

int main() {
    cout << sum<int>(1, 2) << endl;
    cout << sum<float>(1.21, 2.43) << endl;
    return 0;
}
```

Function Templates

- It is also possible to invoke a function template without giving an explicit type. 模板类型参数可以从实际参数类型推断出来。

```
int main() {
    cout << sum(1, 2) << endl;
    cout << sum(1.21, 2.43) << endl;
    return 0;
}
```

Function Templates

- Templates can also specify more than one type parameter. For example:

```
#include <iostream>
using namespace std;

template <typename T, typename U>
U sum(const T a, const U b) {
    return a + b;
}

int main() {
    cout << sum<int, float>(1, 2.5) << endl;
    return 0;
}
```

Class templates

- Class templates are also possible, in much the same way we have written function templates:

```
#include <iostream>
using namespace std;
template <typename T>
class Point {
private:     T x, y;
public:
    Point(const T u, const T v) : x(u), y(v) {}
    T getX() { return x; }
    T getY() { return y; }
};

int main() {
    Point<float> fpoint(2.5, 3.5);
    cout << fpoint.getX() << ", " << fpoint.getY() << endl;
    return 0;
}
```

Class templates

- To declare member functions externally, we use the following syntax:

```
template <typename T>
T classname<T>::function_name()
```

- for example, getX could have been declared in the following way:

```
template <typename T>
T Point<T>::getX() { return x; }
```

Templates

- 在函数和类的前面加上模板头：关键字template和<模板参数表>，即：template <模板参数表>
- 简单的方法是先写出普通函数或类，然后将其转换为模板：1）加模板头 2）代码中的元素类型换成模板中的参数类型。
- 注意在类模板体外成员函数定义时，一定要说明模板类型。类模板也称为模板类。

向量(数组)

```
class Vector{
public:
     Vector( int cElements );
     ~Vector() {delete[] _iElements;}
     int& operator[]( int nSubscript );
private:
      int *_iElements;
     int _iUpperBound;
};
Vector:: Vector(int cElements ){
   _iElements = new int[cElements]; _iUpperBound = cElements;
}
int& Vector:: operator[]( int nSubscript ){
   if(nSubscript >=0&&nSubscript <_iUpperBound)
     return _iElements[nSubscript ];
}
```

向量(数组)

```
int main(){
  Vector v( 10 );        //整数向量（数组）
  for( int i = 0; i < 10; ++i )
      v[i] = i;
  v[2] = 34.56; //错：Vector只能存放整数
  return v[0];
}
```

这个Vector只能存放整数，加入需要一个存放student的向量呢？

向量(数组)模板

```
class Vector{
public:
    Vector( int cElements );
    ~Vector() { delete _iElements; }
    int& operator[]( int nSubscript );
private:
    int *_iElements;
    int _iUpperBound;
};
```

向量(数组)模板

```
template <class T>
class Vector{
public:
    Vector( int cElements );
    ~Vector() { delete _iElements; }
     T & operator[]( int nSubscript );
private:
     T *_iElements;
    int _iUpperBound;
};
```

向量(数组)模板

```
Vector::Vector( int cElements ){
    _iElements = new int[cElements];
    _iUpperBound = cElements;
}
```

```
template <class T>
Vector <T> ::Vector( int cElements ){
    _iElements = new T[cElements];
    _iUpperBound = cElements;
}
```

类模板的类型是*Vector<T>*

向量(数组)模板

```
int& Vector ::operator[]( int nSubscript ){
    if ( nSubscript >= 0&&nSubscript<iUpperBound )
        return _iElements[nSubscript];
    else{ throw std::out_of_range("out_of_range]");}
}
```

```
template <class T>
T & Vector<T> ::operator[]( int nSubscript ){
        if ( nSubscript >= 0&&nSubscript<iUpperBound )
            return _iElements[nSubscript];
        else{ throw std::out_of_range("out_of_range]"); }
}
```

向量(数组)模板

```
#include <string>
int main(){
    Vector<int>  v( 10 );       //实例化模板
    for( int i = 0; i < 10; ++i )
            v[i] = i;

   Vector<string>  s( 10 );
   s[2] = “hello!”;
   s[4] = s[2];
   return v[0];
}
```

模板实例化/ Template Instantiation

- 由类模板（模板类）通过指定模板参数类型，得到一个具体的类的过程。如

  Vector<int>

- 模板实例化是由编译器完成的，与程序员无关
- 如果**模板实例化**过程中，发现某个模板实参不能提供模板所要的语义时，将产生编译错误。如：

  max(stu1,stu2);　//如果stu1,stu2对应类型没有重载>运算符，则编译出错

模板参数/ template parameters

- Template Parameters/模板参数分为类型模板参数和非类型模板参数。

  如上2个例子中的T.

- 类型模板参数用来参数化一个类型，非类型模板参数用来参数化一个常量。类型模板参数由关键字typename或class和参数名构成，非类型模板参数与一般函数参数一样，由普通类型和参数名构成

模板参数/ template parameters

```
template <typename T，int size=20>
class Vector{
public:
    Vector(): _iUpperBound(size){};
    T & operator[]( int nSubscript );
private:
     T _iElements[size];
     int _iUpperBound;
};
```

非类型模板参数必须是常量表达式

- 因为模板实例化是编译期行为。

```
int i= 6;
Vector<int , 45> intVec; //OK
Vector<int , i> intVec2;   //Error
```

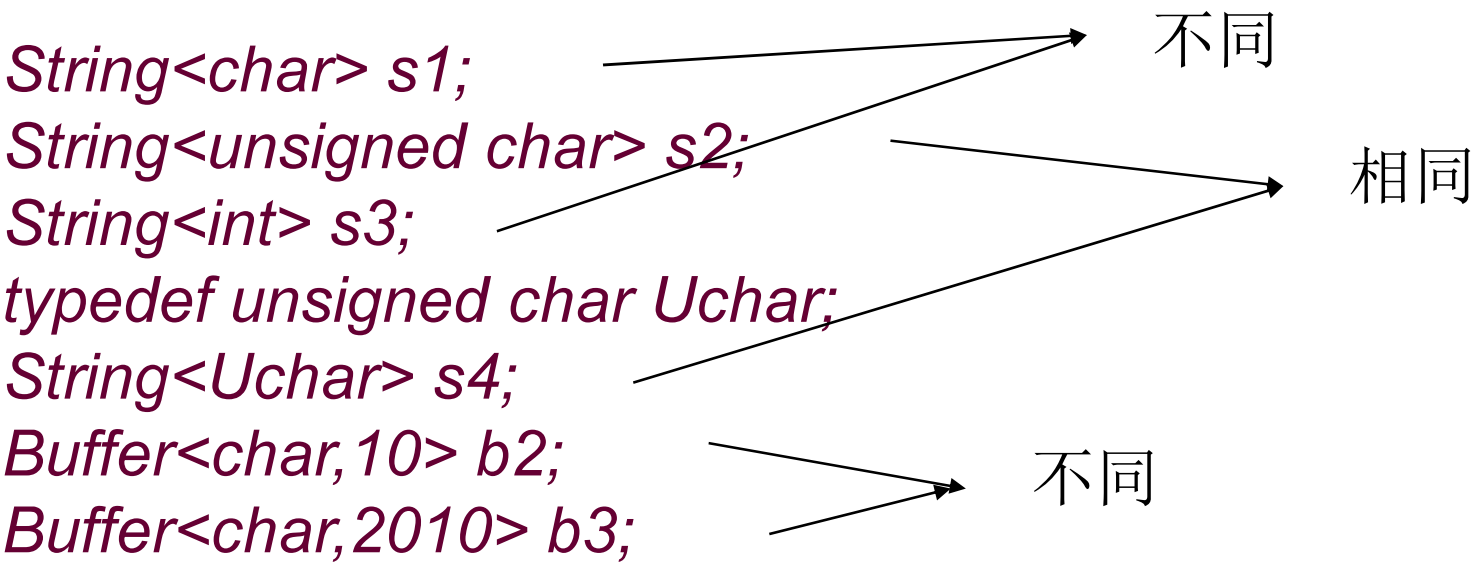
缺省模板参数

- 参数象普通函数参数有缺省参数一样，模板也有缺省模板

```
template <typename T = int>
class Vector{
public:
        Vector( int cElements );
        ~Vector() { delete[] _iElements; }
         T & operator[]( int nSubscript );
private:
         T *_iElements;
         int _iUpperBound;
};
```

类型等价/ Type Equivalence

- 对于一个模板，给它不同的模板参数，将产生不同的模板实例化类型。
- 同样的模板参数将生成同样的类型。



类型等价/ Type Equivalence

- 同一模板生成的不同类型的变量当然不能相互赋值.

```
String<unsigned char> s2;
String<Uchar> s4;
Buffer<char,10> b2;
Buffer<char,2010> b3;

s4 = s2 ; //可以
b3 = b2 ;  //错!
```

模板专门化/template specialization

- 通过提供一个模板的不同定义,并由编译器在使用时根据提供的模板参数,选择一个合适的定义. 称为模板的专门化或特化.

模板专门化/template specialization

```
#include <iostream>
using namespace std ;
template <class T>
T max(T a, T b) {    return a > b ? a : b ;}

int main(){
    cout << "max(10, 15) = " << max(10, 15) << endl ;
    cout << "max('k', 's') = " << max('k', 's') << endl ;
    cout << "max(10.1, 15.2) = " << max(10.1, 15.2) << endl ;
    cout << "max(\"Aladdin\", \"Jasmine\") = "
     << max("Aladdin", "Jasmine") <<endl ;
    return 0 ;
}
```

```
max(10, 15) = 15
max('k', 's') = s
max(10.1, 15.2) = 15.2
max("Aladdin", "Jasmine") = Aladdin
```

模板专门化/template specialization

```
#include <iostream>
#include <cstring>
using namespace std ;

//max returns the maximum of the two elements
template <class T>
T max(T a, T b){     return a > b ? a : b ;}

// Specialization of max for char*
template <>
char* max(char* a, char* b){
    return strcmp(a, b) > 0 ? a : b ;
}
int main(){
   //…
}
```

```
max(10, 15) = 15
max('k', 's') = s
max(10.1, 15.2) = 15.2
max("Aladdin", "Jasmine") = Jasmine
```

```
#include <iostream>
#include <cctype>
using namespace std;

template <typename T>
class Container {
   private:
         T elt;
   public:
         Container(const T arg) : elt(arg) {}
         T inc() { return elt+1; }
};

template <>
class Container <char> {
   private:
         char elt;
   public:
         Container(const char arg) : elt(arg) {}
         char uppercase() { return toupper(elt); }
};

int main() {
   Container<int> icont(5);
   Container<char> ccont('r');
   cout << icont.inc() << endl;
   cout << ccont.uppercase() << endl;
   return 0;
}
```

Standard Template Library (STL)

- container classes : lists, maps, queues, sets, stacks, and vectors.
- Algorithms : sequence operations, sorts, searches, merges, heap operations, and min/max operations.

```
#include <iostream>
#include <set>
#include <algorithm>
using namespace std;

int main() {
   set<int> iset;
   iset.insert(5); iset.insert(9);  iset.insert(1);
   iset.insert(8); iset.insert(3);

   cout << "iset contains:";
   set<int>::iterator it;
   for (it=iset.begin(); it != iset.end(); it++)
       cout << " " << *it;cout << endl;
   int searchFor;
   cin >> searchFor;
   if (binary_search(iset.begin(), iset.end(), searchFor))
       cout << "Found " << searchFor << endl;
   else
       cout << "Did not find " << searchFor << endl;
   return 0;
}
```

```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

main()
{
   vector<string> SS;

   SS.push_back("The number is 10");
   SS.push_back("The number is 20");
   SS.push_back("The number is 30");

   cout << "Loop by index:" << endl;

   int ii;
   for(ii=0; ii < SS.size(); ii++)
   {
      cout << SS[ii] << endl;
   }

   cout << endl << "Constant Iterator:" << endl;
   vector<string>::const_iterator cii;
   for(cii=SS.begin(); cii!=SS.end(); cii++)
   {
      cout << *cii << endl;
   }

   cout << endl << "Reverse Iterator:" << endl;

   vector<string>::reverse_iterator rii;
   for(rii=SS.rbegin(); rii!=SS.rend(); ++rii)
   {
      cout << *rii << endl;
   }

   cout << endl << "Sample Output:" << endl;

   cout << SS.size() << endl;
   cout << SS[2] << endl;

   swap(SS[0], SS[2]);
   cout << SS[2] << endl;
}
```

```cpp
#include <iostream>
#include <vector>


using namespace std;


main()
{
   // Declare size of two dimensional array and initialize.
   vector< vector<int> > vI2Matrix(3, vector<int>(2,0));


   vI2Matrix[0][0] = 0;
   vI2Matrix[0][1] = 1;
   vI2Matrix[1][0] = 10;
   vI2Matrix[1][1] = 11;
   vI2Matrix[2][0] = 20;
   vI2Matrix[2][1] = 21;


   cout << "Loop by index:" << endl;


   int ii, jj;
   for(ii=0; ii < 3; ii++)
   {
      for(jj=0; jj < 2; jj++)
      {
         cout << vI2Matrix[ii][jj] << endl;
      }
   }
}
```

```
#include <iostream>
#include <algorithm>
using namespace std;
void printArray( const int arr[], const int len) {
    for ( int i=0; i < len; i++)   cout << " " << arr[i];
   cout << endl;
}
int main() {
  int a[] = {5, 7, 2, 1, 4, 3, 6};
  sort(a, a+7);
  printArray(a, 7);
  rotate(a,a+3,a+7);
  printArray(a, 7);
  reverse(a, a+7);
  printArray(a, 7);
  return 0;
}
```

This program prints out:

```
1 2 3 4 5 6 7
4 5 6 7 1 2 3
3 2 1 7 6 5 4
```

The **rotate** algorithm rotates the elements in the range [*First*, *Last*) to the right by *n* positions, where *n* = *Middle* - *First*.

作业

- 实现一个较为完整的向量类模板Vector。
- 实现一个函数模板，求三个可以比较大小的变量的最大值。