问题的提出

```
int  abs(int x)  {return  x>0?x:-x;}

double  abs(double x)  {return  x>0?x:-x;}

long abs(long x)  {return  x>0?x:-x;}
```

作业中的vector类的定义

第11讲

模板

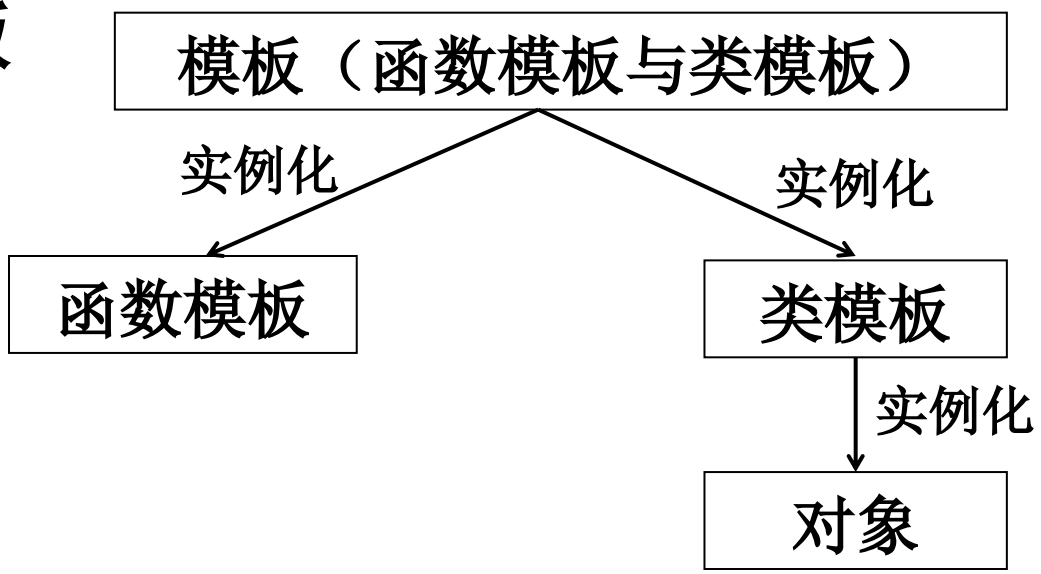
主讲人：赵文彬

本章主要内容

- 模板的概念
- 函数模板
  - 函数模板的定义
  - 函数模板的实例化
  - 函数模板与重载
- 类模板
  - 类模板的定义
  - 类模板的实例化
- C++标准模板库

模板的概念

- 模板是实现**代码重用**机制的一种工具，它可以实现**参数类型化**，即**把类型定义为参数**，从而实现代码重用。
- 包括
  - 函数模板
  - 类模板



函数模板

- 函数模板的定义
  - 格式

关键字

```
template <类型参数表>
返回值类型  函数名  （函数形参表）
{…}    //函数体
```

类型参数表可列举一个或多个类型参数项（用逗号分隔），每个参数项由关键字class后跟一个标识符组成。

```
template <class T>
```

```
template <class T1, class T2>
```

函数模板

- 函数模板的定义
  - 举例

```
template <class T>
T max(T x, T y)
{
    return x>y?x:y;
}
```

函数模板

- 函数模板的实例化

```
// function template
#include <iostream>
using namespace std;

template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}

int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

```
6
10
```

```
double x1(3.4),y1(5.6);
char x2='a', y2='b';
cout<<GetMax(x1,y1)<<endl;
cout<<GetMax(x2,y2)<<endl;
```

```
cout<<GetMax(x1,i)<<endl;
```

函数模板

- 说明：
  - 函数模板实例化时，可以显式实例化也可以隐式实例化。

    显式实例化　GetMax<int> (I,j);
    隐式实例化　GetMax(x1,y1)

  - 函数模板中的类型参数只有到该函数真正被调用时才能确定其实际类型。在函数调用时，编译器按最先遇到的实参类型隐含生成一个模板函数（注意，并没有隐含的类型转换），并对所有相应参数进行类型一致性检测。

函数模板

函数模板与重载

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

template <class T>
T sum(T * array, int size=0)
{
T total =0;
for (int i=0;i<size;i++)
total+=array[i];
return total;
}
```

```
template <class T1,class T2>
T2 sum(T1 * array1, T2 * array2, int size=0)
{
T2 total=0;
for (int i=0;i<size;i++)
total+=array1[i]+array2[i];
return total;
}
```

函数模板

函数模板与重载

```
char * sum(char * s1, cha
{
char* str = new char[strle
strcpy(str,s1);
return strcat(str,s2);
}
```

```
void main()
{
int iArr[] = {1,2,3,4,5};
double dArr[]={1.1,2.2,3.3,4.4,5.5};
char *p1="Hello,";
char *p2="World";
int iTotal=sum(iArr,5);
double dTotal = sum(dArr,5);
double idTotal=sum(iArr,dArr,5);
p1=sum(p1,p2);
cout<<iTotal<<endl;
cout<<dTotal<<endl;
cout<<idTotal<<endl;
cout<<p1<<endl;
}
```

```
0
0
31.5
Hello,World
```

函数模板

- 函数模板与重载
  - 说明，函数模板与同名的非模板函数重载时，
    - 编译器优先调用对应的一般函数（非模板函数）；
    - 若找不到，则编译器检查是否可以用模板函数产生符合函数名和函数参数的模板函数，若有则实例化
    - 若不能完全匹配，则检查是否可以通过类型转换进行匹配；
    - 若均不能或多于一种选择，则产生编译错误

类模板

- 类模板的定义
  - 格式

template <类型参数表>
class 类名
{…
};

```
template <class T>
class Test
{
    T a;
     int b;
Public:
     Test(): b(0) {}
     Test(T x, int y): a(x), b(y) {}
      int Getb() {return b;}
      void Print() {cout<<a<<b<<endl;}
}
```

类模板

- 类模板的定义
  - 成员函数的定义也可以放在类外定义，但必须

```
template <class T>
class Test
{
    T a;
    int b;
Public:
    Test(): b(0) {}
    Test(T x, int y);
    int Getb() {return b;}
    void Print() {cout<<a<<b<<endl;}
}
```

```
templat <class T>
Test<T>: :Test(T x, int y): a(x), b(y) {}
```

类模板

- 类模板的定义

```
// class templates
#include <iostream>
using namespace std;

template <class T>
class mypair {
    T a, b;
  public:
    mypair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};

template <class T>
T mypair<T>::getmax ()
{
  T retval;
  retval = a>b? a : b;
  return retval;
}

int main () {
  mypair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

100

类模板

类模板的实例化

格式

类名<参数类型表> 对象1,…,对象n

```
template <class T>
class Test
{
    T a;
     int b;
Public:
     Test(): b(0) {}
     Test(T x, int y): a(x), b(y) {}
      int Getb() {return b;}
      void Print() {cout<<a<<b<<endl;}
}
```

101
A2

```
#include <iostream>
#include “Test.h”
using namespace std;
void main()
{
     Test<int> obj1(10,1);
     Test<char> obj2(‘A’,2);
      cout<<obj1.Print();
      cout<<obj2.Print();
}
```

- 类模板的实例化

```
100
3.1416
```

```
// sequence template
#include <iostream>
using namespace std;

template <class T, int N>
class mysequence {
    T memblock [N];
  public:
    void setmember (int x, T value);
    T getmember (int x);
};

template <class T, int N>
void mysequence<T,N>::setmember (int x, T value)
{
  memblock[x]=value;
}

template <class T, int N>
T mysequence<T,N>::getmember (int x) {
  return memblock[x];
}

int main () {
  mysequence <int,5> myints;
  mysequence <double,5> myfloats;
  myints.setmember (0,100);
  myfloats.setmember (3,3.1416);
  cout << myints.getmember(0) << '\n';
  cout << myfloats.getmember(3) << '\n';
  return 0;
}
```

类模板

- 类模板的实例化
  - 类模板必须显式实例化
  - 类模板中函数的实现（定义）可以在类外进行，但必须以函数模板格式进行，且与类模板的定义放在同一文件中。

C++标准模板库

- C++标准模板库
  - Standard Template Library, STL
  - 包含
    - 容器模板
    - 算法模板
    - 迭代器模板

C++标准模板库

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
using namespace std;
int main()
{
   vector<int> v;  //创建一个vector类容器对象v，其元素为int型
      //生成容器v中的元素
   int x;
   cin>>x;
   while(x>0)  //不断向容器中添加正int型元素
   {
      v.push_back(x); //向容器尾部添加一个元素
      cin>>x;
   }
//创建容器v的一个迭代器it1使其指向容器v中的第一个元素
   vector<int>::iterator it1= v.begin();
//创建容器v的一个迭代器it2使其指向容器v中最后一个元素的下一个位置
   vector<int>::iterator it2= v.end();
//利用算法模板max_element计算容器v中的最大元素
   cout<<"Max="<<*max_element(it1,it2)<<endl;
//利用算法模板accumulate计算容器v中所有元素的和
   cout<<"Sum="<<accumulate(it1,it2,0)<<endl;
//利用算法模板sort对容器v中的元素进行升序排列
   sort(it1,it2);
//输出排序结果
   cout<<"Sorted result is:\n";
   while(it1!=it2)
   {
      cout<<*it1<<' ';
      ++it1;
   }
   cout<<endl;
   return 0;
}
```

C++标准模板库

- 容器vector
  - 头文件#include<vector>.
  - 创建vector对象，vector<int> vec;
  - 尾部插入数字：vec.push_back(a);
  - 使用下标访问元素，cout<<vec[0]<<endl;记住下标是从0开始的。
  - 使用迭代器访问元素.
    - vector<int>::iterator it;
      for(it=vec.begin();it!=vec.end();it++)
      cout<<*it<<endl;
  - 插入元素： vec.insert(vec.begin()+i,a);在第i+1个元素前面插入a;

C++标准模板库

- 容器vector
  - 删除元素：
    - vec.erase(vec.begin()+2);删除第3个元素
    - vec.erase(vec.begin()+i,vec.end()-j);删除区间[i+1,N-1-j];区间从0开始
  - 向量大小:vec.size();
  - 清空:vec.clear();

本章小结

- 函数模板的定义和实例化
- 类模板的定义和实例化
- 容器vector