Lab of
New Generation Network
Techonology & Applications

# 第12讲 模板和容器概述

- 函数和类模板
- 容器
- 小学期综合程序训练安排
- 考试安排

黄永峰
yfhuang@tsinghua.edu.cn
62792710

# 12.1 模板的基本概念

```
//求两个不同类型数的极大值
double GetMax( double a, double b )
{return( a > b ) ? a : b;
}
char GettMax( char a, char b )
{return( a > b ) ? a : b;
}
int GetMax( int a, int b )
{     return( a > b ) ? a : b;
}
long GetMax( long a, long b )
{return( a > b ) ? a : b;
}
```

- 是否可将上述四个函数合成为一个函数呢？
- Type用前述的int、long、double、char替换可得到上述四个函数。

```
1   Type GetMax( Type a, Type b )
2   {
3       return( a > b ) ? a, b;
4   }
```

难道可以自动生产函数吗？

# 12.1 模板的基本概念

- 模板是C++支持多态性的一种工具，体现C++ 泛化（通用）编程思想。
- 模板是一种使用“数据类型”作为参数来产生一系列函数或类的机制；采用“类型参数”来完成不同的功能。
- 模板可让用户得到类或函数声明的一种通用模式。使得类中的某些数据成员或者成员函数的参数、返回值取得不同类型。
- 模板方便了更大规模的软件开发；减少了程序员编写代码的工作量。

## 函数模版与模版函数

- 模板分为类模板（class template）和函数模板（function template）两种。
- 说明了一个函数模板后，当系统发现有一个对应函数调用时，将根据实参中的数据类型来确认是否匹配函数模板中对应的形参，然后生成一个模版函数（template function）；
- 生成的模版函数的函数体与函数模板的函数体相同；但其中数据类型参数实例化了。

## 类模板与模版类

- 同理，在说明了一个类模板之后，可以创建类模板的实例，该实例称为模板类。
- 或者说，将数据类型作为形式参数就得到了类模板；如果将类型参数实例化就得到了模板类。
- 总结：函数或类模版是将数据类型参数化；模版函数或类是模版中数据类型参数实例化。

# 12.2 函数模板

- 函数模板：本质是建立一个通用函数。其函数的数据类型和形参类型不具体指定，用虚拟类型表示。
- 函数调用时，系统会根据实参类型来取代模板中虚拟类型，从而实现不同功能。
- 凡是多个函数体相同的函数都可合并为函数模板表示。
- 格式：

template(typename/class T)

通用函数定义

T是虚拟类型名

```
1
2  #include <iostream>
3  using namespace std;
4  template <typename T>
5  T max(T a,T b,T c)
6   {if (b>a) a=b;
7    if(c>a) a=c;
8    return  a;}
9
10 int main()
11 {int i1=15, i2=19, i3=10, i;
12  float j1=1.5, j2=1.9, j3=1.0, j;
13  i=max( i1,i2,i3); //模板中T被int取代
14  j=max( j1, j2,j3); //模板中T被？ 取代
15 cout<<"i_max="<<i<<endl;
16 cout<<"j_max="<<j<<endl;
17 return 0;}
18
```

```
C:\WINDOWS\sys
i_max=19
j_max=1.9
请按任意键继续.
```

# 12.2 函数模板

```
1  # include   <iostream>
2  using namespace std;
3  template<class T>
4  void outputArray(const T *P_array, const  int count)
5  {for (int i=0; i<count; i++)
6   cout<<P_array[i]<<" ";
7   cout <<endl;
8  }
9
10 int main()
11 { const int aCount =8,bCount=8,cCount=20;
12 int aArray[aCount]={1,2,3,4,5,6,7,8};
13 double bArray[bCount]={1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8};
14 char cArray[cCount]="Welcome to see you";
15 cout<<"a Array contains:"<<endl;
16 outputArray(aArray,aCount);
17 cout<<"b Array contains:"<<endl;
18 outputArray(bArray,bCount);
19 cout<<"c Array contains:"<<endl;
20 outputArray(cArray,cCount);
21 }
```

```
C:\WINDOWS\system32\cmd.exe
a Array contains:
1 2 3 4 5 6 7 8
b Array contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8
c Array contains:
W e l c o m e   t o   s e e   y o u
请按任意键继续. . .
```

# 注意事项

- 函数模板的说明（定义）必须在全局作用域。
- 函数模板不能说明为类的成员函数。
- 模板类型参数不具有隐式类型转换的作用。
- 模板函数也可以重载。匹配规则：

（1）匹配类型完全相同的重载函数。

（2）寻求函数模板来匹配。

Lab of New Generation Network Techonology & Applications

```
#include <iostream>
using namespace std;
int GetMax( int a, int b ) //求两个整型数的最大值
{  cout << "调用int， maxValue =  ";
   return ( a > b ) ? a : b;}
long GetMax( long a, long b )          //求两个长整型数的最大值
{  cout << "调用long， maxValue =  ";
   return ( a > b ) ? a : b;}
double GetMax( double a, double b )        //求两个双精度型数的最大值
{  cout << "调用double， maxValue =  ";
   return ( a > b ) ? a : b;}

//char GetMax( char a, char b )          //求两个字符型数的最大值
//{ cout << "调用char， maxValue =  ";
// return ( a > b ) ? a : b;}

//定义函数模板1,在函数模板中添加第三参数char *， 是为了与函数模板2区分开
template <class Type>
Type GetMax( Type a[ ], int iCnt, char *lpszArrayName )
{  int i;
   Type tMaxValue = a[0];              //定义Type类型的变量
   for ( i=1; i<iCnt; i++ )           //在循环中寻找数组中最大的值
   {  if ( tMaxValue < a[i] )
      {tMaxValue = a[i];}
   }
   cout << "使用 函数模板1， " << lpszArrayName << "的最大值， maxValue =  ";
   return tMaxValue;}

//定义函数模板2
template <class TypeX, class TypeY>
TypeX GetMax( TypeX tX, TypeY tY )
{  TypeX tMaxValue = 0;       //定义一个TypeX类型变量
   if ( tX > ( TypeX )tY )      //比较前， 首先将TypeY类型变量转化为TypeX类型变量
   {tMaxValue = tX;}
   else
   {tMaxValue = ( TypeX )tY;}
   cout << "调用 函数模板2， maxValue =  ";
   return tMaxValue;}
```

```
41  void main( )
42  {   int a[ ] = {1, 3, 5, 7, 9, 6, 4, 8, 2, 10};
43      double b[ ] = {3.2, -6.4, 6.0, 9.9, 8.6, 2.1};
44      char c[ ] = {'A', 'C', '1', 'a', 'c'};
45
46      cout << "   " << GetMax( a, 10, "数组a" ) << endl;//使用函数模板1
47      cout << "   " << GetMax( b, 5, "数组b" ) << endl;  //使用函数模板1
48      cout << "   " << GetMax( c, 5, "数组c" ) << endl;  //使用函数模板1
49
50      cout << "   " << GetMax( 10, 20 ) << endl;          //调用重载函数
51      cout << "   " << GetMax( 101L, 201L ) << endl;      //调用重载函数
52      cout << "   " << GetMax( 1.0, 2.0 ) << endl;      //调用重载函数
53
54      cout << "char=" << GetMax( 'A', '2' ) << endl;     //使用函数模板2
55
56      cout << "   " << GetMax( 10, 5.0 ) << endl;         //使用函数模板2
57      cout << "   " << GetMax( 11.1, 5 ) << endl;         //使用函数模板2
58      cout << "   " << GetMax( 22, 10L ) << endl;         //使用函数模板2
59      cout << "   " << GetMax( 'A', 2L ) << endl;         //使用函数模板2
60      cout << "   " << GetMax( 1.0, 200L ) << endl;       //使用函数模板2
61      cout << "   " << GetMax( 100.0, 'A' ) << endl;      //使用函数模板2
62
63      cin.get( );
64  }
```

- 使用函数模板1，数组a的最大值，maxValue ＝ 10
- 使用函数模板1，数组b的最大值，maxValue ＝ 9.9
- 使用函数模板1，数组c的最大值，maxValue ＝ c
- 调用int，maxValue ＝ 20
- 调用long，maxValue ＝ 201
- 调用double，maxValue ＝ 2
- 调用函数模板2，maxValue ＝ char=A
- 调用函数模板2，maxValue ＝ 10
- 调用函数模板2，maxValue ＝ 11.1
- 调用函数模板2，maxValue ＝ 22
- 调用函数模板2，maxValue ＝ A
- 调用函数模板2，maxValue ＝ 200
- 调用函数模板2，maxValue ＝ 100

- 结果分析

（1）GetMax(10,20)调用直接调用int GetMax(int a, int b)；

（2）注释掉char GetMax( char a, char b )前，GetMax( ‘A’, ‘2’ )调用该重载函数；

（3）注释掉char GetMax( char a, char b )以后， GetMax( ‘A’, ‘2’ )调用了模板函数2；在匹配模板函数时，系统不会进行隐式类型转换以匹配重载函数，否则它就应该调用：int GetMax( int a, int b )；

（4）GetMax( 10, 5.0 )调用函数模板2。在重载函数中没有匹配版本，在函数模板中匹配，调用函数模板2。

# 12.3 类模板的说明及生成模板类

- 类模版的说明

```
1 template  < 虚拟类型参数表>
2 class  <类名>
3 {
4     //类说明体
5 };
```

- 如若干个类的功能相同，仅仅是数据类型不同，则可以声明一个通用的类模板。或者说，类模板可使类中的某些数据成员、成员函数的参数，或返回值能取任意值；
- 如果说类是对象的抽象，对象是类的实例，则类模板是类的抽象，类是类模板的实例。类模板也称为“参数化类”。

```
class Compare_float
{public:
Compare(float a,float b)
{x=a;y=b;}
float max( )
{return(x>y)?x:y;}
float min( )
{return(x<y)?x:y;}
private:
float x,y;}
```

```
class Compare_int
{public:
Compare(int a,int b)
{x=a;y=b;}
int max( )
{return(x>y)?x:y;}
int min( )
{return(x<y)?x:y;}
private:
int x,y;};
```

```
template<class numtype>
//声明一个模板，虚拟类型名为numty
class Compare
//类模板名为Compare
{public:
Compare(numtype a,numtype b)
{x=a;y=b;}
numtype max( )
{return (x>y)?x:y;}
numtype min( )
{return (x<y)?x:y;}
 private:
numtype x,y;};
```

**类模板的实例化：**在类模板名之后在尖括号内指定实际的类型名，在进行编译时，编译系统就用int取代类模板中的类型参数numtype

```
1  #include <iostream>
2  using namespace std;
3  template<class numtype>                    //定义类模板
4  class Compare
5  {public:
6  Compare(numtype a,numtype b)
7  {x=a;y=b;}
8  numtype max( )
9  {return (x>y)?x:y;}
10 numtype min( )
11 {return (x<y)?x:y;}
12 private:
13 numtype x,y;};
14
15 int main( )
16 {Compare<int> cmp1(3,7);          //定义对象cmp1，用于两个整数的比较
17 cout<<cmp1.max( )<<" is the Maximum of two integer numbers."<<endl;
18 cout<<cmp1.min( )<<" is the Minimum of two integer numbers."<<endl<<endl;
19 Compare<float> cmp2(45.78,93.6);  //定义对象cmp2，用于两个浮点数的比较
20 cout<<cmp2.max( )<<" is the Maximum of two float numbers."<<endl;
21 cout<<cmp2.min( )<<" is the Minimum of two float numbers."<<endl<<endl;
22 Compare<char> cmp3('a','A');       //定义对象cmp3，用于两个字符的比较
23 cout<<cmp3.max( )<<" is the Maximum of two characters."<<endl;
24 cout<<cmp3.min( )<<" is the Minimum of two characters."<<endl;
25 return 0;}
```

```
C:\WINDOWS\system32\cmd.exe
7 is the Maximum of two integer numbers.
3 is the Minimum of two integer numbers.

93.6 is the Maximum of two float numbers.
45.78 is the Minimum of two float numbers.

a is the Maximum of two characters.
A is the Minimum of two characters.
请按任意键继续. . .
```

- 用类模板定义对象时的一般形式：

  类模板名<实际类型名>　对象名;

  或：类模板名<实际类型名>　对象名(实参表列);

  如：Compare<int> cmp; 或 Compare<int> cmp(3,7);

- 在类模板外定义成员函数的一般格式：

  template<class 虚拟类型参数>

  函数类型 类模板名<虚拟类型参数>::函数名(形参表列)

  {函数体…}

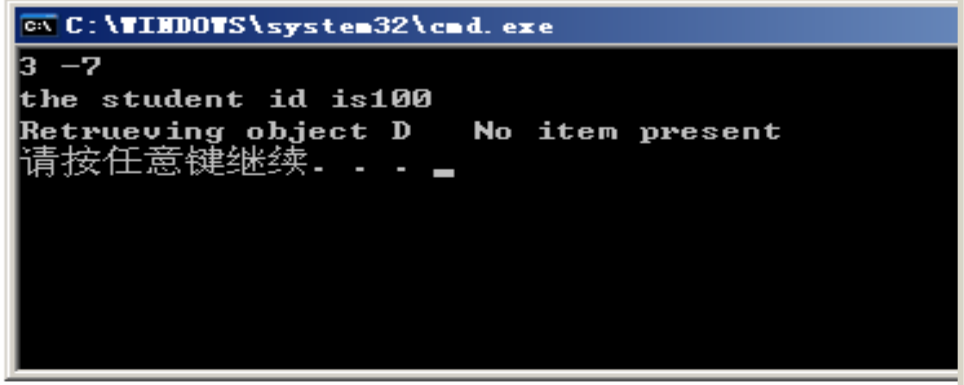
例:template<class numtype>

numtype Compare<numtype>::max( )

{{return (x>y)?x:y;}

Lab of New Generation Network Techonology & Applications

```
1  # include   <iostream>
2  #include <cstdlib>
3  using namespace std;
4  struct Student
5  { int id; float gpa;};
6  template<class T>
7  class Store
8  {private:
9     T item;
10    int haveValue;
11 public:
12    Store(void);
13    T GetElem(void);
14    void PutElem(T x);};
15 template<class T>
16 Store<T>:: Store(void):haveValue(0)
17 {}
18 template<class T>
19 T Store<T>:: GetElem(void)
20  { if (haveValue==0) {cout<<"No item present"<<endl;exit(1);}
21    return item;}
22 template<class T>
23 void Store<T>:: PutElem(T x)
24 {haveValue ++; item=x;}
25
26 int main()
27 {Student g={100,22};
28  Store<int> S1,S2;
29  Store<Student> S3;
30  Store<double> D;
31  S1.PutElem(3);
32  S2.PutElem(-7);
33  cout<<S1.GetElem()<<" "<<S2.GetElem()<<endl;
34  S3.PutElem(g);
35  cout<<"the student id is"<<S3.GetElem().id<<endl;
36  cout<<"Retrueving object D   ";
37  cout<<D.GetElem()<<endl;}
```



# 类模板／模版类的派生

- 用**类模板**派生出**新的类模板**

```
template <class Type>
class TSet : public TList<Type>
{派生类类模板定义};
```

- 用**模板类**派生派生类

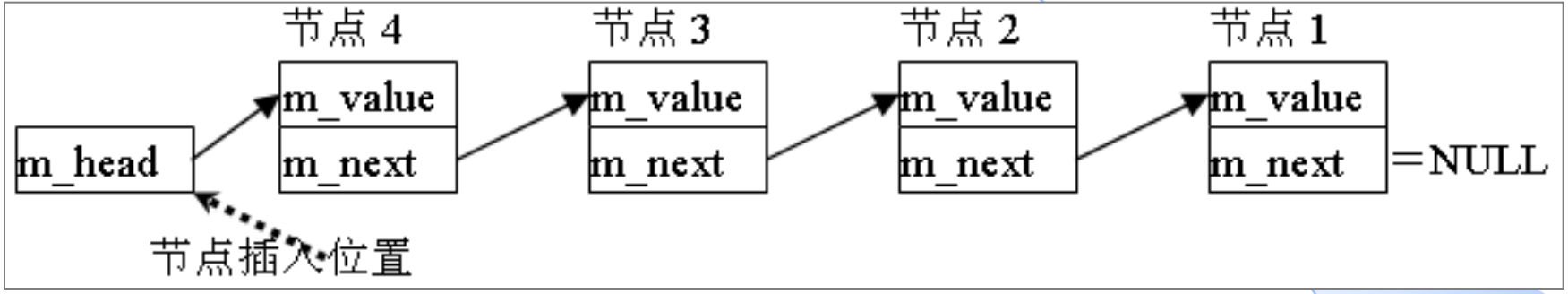
```
template <class Type>
class TSet : public TList<实参类型>
{派生类定义};
```

```
#include <iostream>      //包含头文件。使用iostream库
using namespace std;      //使用std命名空间
class SNode //定义链表的结点结构类
{public:
   SNode( int value );
   ~SNode( ){ };
   int m_value;     //结点值
   SNode *m_next;};    //结点后继，指向下一个结点的指针
template <class Type> //定义链表的类模板
class TList
{public:
   TList( );                  //构造函数
   ~TList( );                 //析构函数
   virtual bool Insert( Type value );  //在链表头部插入一个结点
   bool Delete( Type value );          //从链表中删除值为value的一个结点
   bool Contain( Type value );         //判断链表中是否包含某结点
   void Print( );              //输出链表结点的值
protected:
   SNode *m_head;  }; //设置为只有头指针的单向链表

template<class Type>//定义集合类模板–集合与链表采用同样的组织方式;
//但是链表和集合的概念不同：集合中不允许有重复的结点
class TSet : public TList<Type>
{public:
   bool Insert( Type value );}; //在集合中重载插入方法，插入前先判断结点是否已经存在
SNode::SNode( int value )//结点类构造函数
{  m_value = value;        //结点值
   m_next = NULL;   }          //结点后继
template <class Type>//构造函数
TList<Type>::TList( )
{  m_head = NULL;}    //链表头

template <class Type> //链表析构函数中需要delete所有还在链表中的结点
TList<Type>::~TList( )
{  SNode *p = m_head;          //
   for ( ; p != NULL; )      //直到结点不为空
   {m_head = p->m_next;    //头结点指向下一个结点，作为新的头结点
      delete p;          //释放p所指向的结点
      p = m_head;          //p指向新的头结点
   }}
```

```
43  bool TList<Type>::Insert( Type value )
44  {  SNode  *pTemp = new SNode( value );   //构造一个新结点
45     if ( pTemp == NULL )                  //结点空间申请不成功,退出
46     {return false; }
47        pTemp->m_next = m_head;      //新结点的m_next指针指向头结点原来所指的结点
48     m_head = pTemp;          //头结点改为新生成的结点
49     return true;}
50
51  template <class Type>
52  bool TList<Type>::Delete( Type value )
53  {  SNode *p1, *p2;      //申请两个结点指针，用于结点操作时要遍历整个链表以查找
54  //其中是否包含有结点值为value的结点,但是一次遍历只能够删除一个值为value结点
55  //如果在链表中有多个值相同的结点，需要分别删除。
56     if ( m_head->m_value == value )      //如果头结点就是要找的结点，直接删除
57     {  p1 = m_head->m_next;          //p1指向头结点的后继结点
58        delete m_head;          //释放头结点
59        m_head = p1;          //p1成为新的头结点，即原头结点的后继成为新的头结点
60        return true;          //返回真
61     }
62     else                  //要删除的结点非头结点
63     {  for ( p1 = m_head, p2 = m_head->m_next; p2 != NULL;  )//遍历链表
64        {if ( p2->m_value == value )   //如果找到，则释放该结点，并结束遍历
65           {  p1->m_next = p2->m_next; //p1指向p2的下一个结点
66              delete p2;          //释放p2
67              p2 = NULL;          //让p2指向NULL，该步骤通常很有必要
68              return true;}          //结束遍历。
69           else               //如果该结点不是要找的结点，则p1,p2向后遍历
70           {p1 = p1->m_next;       //p1指向其后继
71           p2 = p2->m_next;     //p2指向其后继
72           }}}
73     return false;  }
74
75  template <class Type>
76  bool TList<Type>::Contain( Type value )
77  {  //遍历链表以查找其中是否包含有结点值为value的结点，有返回true，没有返回false
78     for ( SNode  *p = m_head; p != NULL; p = p->m_next )
79     {
80        if ( p->m_value == value )          //找到该结点，返回true
81        {return true;}
82     }
83     return false;}
```

```
85  template <class Type>//显示表中的结点数据
86  void TList<Type>::Print( )
87  {  cout << "结点的值依次为： "; //遍历链表以读出每一个结点值并显示
88     for ( SNode  *p = m_head; p != NULL; p = p->m_next )
89     {cout << " " << p->m_value << "; ";   }
90     cout << endl; }
91
92  template <class Type>
93  bool TSet<Type>::Insert( Type value )//集合类的结点插入方法
94  {  //集合中无值为value的结点，才可以作插入操作，否则直接返回false。插入失败也返回false
95     if ( !( TList<Type>::Contain( value ) )&&( TList<Type>::Insert( value ) ) )
96     {return true;  }
97     return false;}
98
99  //主测试程序
100 void main( )
101 {  TList<int> sIntList;
102    sIntList.Insert( 12 );
103    sIntList.Insert( 24 );   //在链表sIntList中， 两次插入24
104    sIntList.Insert( 48 );
105    sIntList.Insert( 96 );
106    sIntList.Insert( 24 );   //在链表sIntList中， 两次插入24
107    sIntList.Print( );
108
109    sIntList.Delete( 24 );     //删除一次24
110    sIntList.Print( );
111
112    TSet<int> sIntSet;
113    sIntSet.Insert( 12 );
114    sIntSet.Insert( 24 );   //在集合sIntList中， 两次插入24
115    sIntSet.Insert( 48 );
116    sIntSet.Insert( 96 );
117    sIntSet.Insert( 24 );   //在集合sIntList中， 两次插入24
118    sIntSet.Print( );
119
120    sIntSet.Delete( 24 );   //删除一次24
121    sIntSet.Print( );
122
123    cin.get( );}
```

- **类模版派生实例**。从一个链表类模板派生出了集合类模板。为了简单，只在表头插入，可以删除指定值的节点，但限定一次只能够删除一个节点。

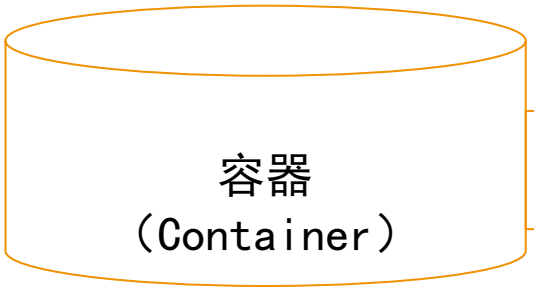



节点的值依次为： 24； 96； 48； 24； 12；
节点的值依次为： 96； 48； 24； 12；
节点的值依次为： 96； 48； 24； 12；
节点的值依次为： 96； 48； 12；

- 从运行结果看：在集合sIntSet中第二次插入24节点是无效的，而在sIntList中两次插入24都有效。

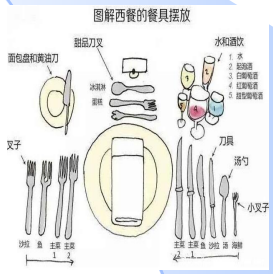
# 12.4 STL 和容器

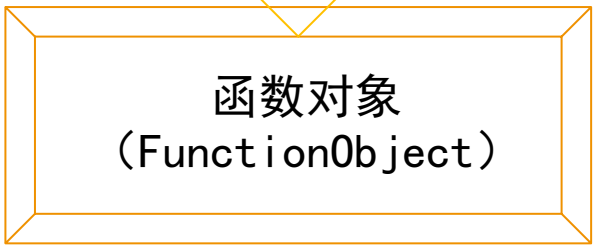
- 标准模板库(Standard Template Library, STL)。采用模板和可重用组件，实现了通用数据结构以及处理这些数据的算法。STL是一套程序库，也是软件重用技术发展史上突破，提出了一种泛型（通用）程序设计模式。




迭代器
（Iterator）






# 12.4 STL 和容器

1. 容器:常用数据结构的模板化，可表示各种数据结构对象；每个容器表现为类模版。例：Vector、List、Map等。

2. 算法:常用操作函数的通用模板，适用于不同类型的数据，以迭代器作为函数参数，可灵活地处理不同长度数据集合。例：sort、search、copy等。

3. 迭代器：算法和容器间的“桥梁”，是一种泛型指针，保存它所操作的特定容器状态；指向容器中某位置，用运算符函数“++”或“--”前后移动，用“*迭代器”表示指向数据。

4. 函数对象：行为类似函数，可作为演化算法的某种策略(policy)，传递算法操作的特定规则。

# 12.4 STL 和容器

- 容器类别：顺序性容器，关联式容器和容器适配器

- 顺序容器：可变长动态数组 vector、双端队列 deque、双向链表 list。其特点是元素在容器中的位置同元素的值无关，即容器是不排序的，便于扩展。
- 关联容器：set、multiset、map、multimap。特点是元素是排序的。默认关联容器中的元素是从小到大排序；具有很好的查找性能。
- 容器适配器：在两类容器的基础上屏蔽一部分功能，突出或增加另一部分功能，实现了3种适配器：栈 stack、队列 queue、优先级队列 priority_queue。

- vector<int> a;//vector<int> 容器类名；定义容器对象 a，即a变长数组，每个元素都是int 变量；
- vector<double> b;//定义了容器对象 b

# 12.4 STL 和容器

STL中定义容器模板类的头文件有8个

| 头文件 | 描　　述 |
| --- | --- |
| <vector> | 作用是**向量**，一个大小可以重新设置的数组类型，比普通该数据更安全、灵活。在尾端增加删除元素具有较佳的性能。 |
| <list> | 作用是**链表**，可以在任意位置插入和删除元素。 |
| <deque> | 作用是**大小可以重新设置的数据类型**，性能仅次于vector。在两端增加删除元素都有较佳的性能。 |
| <queue> | 作用为**队列**，按照先进先出的原则，插入只能在尾部进行；删除、检索和修改只能从头部进行。 |
| <stack> | 作用为**栈**，按照后进先出的原则插入、删除、检索和修改项。 |
| <map> | 作用为**图**，存放成对的key/value，并按照key元素进行排序，可以快速检索元素。 |
| <set> | 作用为**集合**，有两个类set和multiset。set中不允许有相同的元素，而multiset允许相同元素。 |
| <bitset> | 为**固定长度的位序列定义bitset模板**，可以看成是固定长度的bool数组。 |

| | |
|---|---|
| 默认构造函数 | 提供容器默认初始化的构造函数。 |
| 复制构造函数 | 将容器初始化为现有同类容器副本的构造函数 |
| 析构函数 | 不再需要容器时进行内存整理的析构函数 |
| empty | 容器中没有元素时返回true,否则返回false |
| max_size | 返回容器中最大元素个数 |
| size | 返回容器中当前元素个数 |
| operator= | 将一个容器赋给另一个容器 |
| operator< | 如果第一个容器小于第二个容器，返回true，否则返回false， |
| operator<= | 如果第一个容器小于或等于第二个容器，返回true，否则返回false |
| operator> | 如果第一个容器大于第二个容器，返回true，否则返回false |
| operator> | 如果第一个容器大于第二个容器，返回true，否则返回false |
| operator>= | 如果第一个容器大于或等于第二个容器，返回true，否则返回false |
| operator== | 如果第一个容器等于第二个容器，返回true，否则返回false |
| operator!= | 如果第一个容器不等于第二个容器，返回true，否则返回false |
| swap | 交换两个容器的元素 |

# 12.4 STL 和容器

顺序容器和关联容器共有函数

| | |
|---|---|
| begin | 该函数两个版本返回iterator或const_iterator，引用容器第一个元素 |
| end | 该函数两个版本返回iterator或const_iterator,引用容器最后一个元素后面一位 |
| rbegin | 该函数两个版本返回reverse_iterator或const_reverse_iterator,引用容器最后一个元素 |
| rend | 该函数两个版本返回reverse_iterator或const_reverse_iterator，引用容器第一个元素前面一位 |
| erase | 从容器中清除一个或几个元素 |
| clear | 清除容器中所有元素 |

- c.begin() 返回一个迭代器，它指向容器c的第一个元素；c.end() 返回一个迭代器，它指向容器c的最后一个元素的下一个位置。

```cpp
#include<vector>
int main()
{
      vector<int> v(3,2);//3个元素，初值为2
      v[0]=100; //第一个元素赋值为100
      v.at(1) = 200;//第二个元素赋值为200
      for(int i=0;i<3;i++)    //输出所有元素值
         cout<<v.at(i)<<' ';
     cout<<endl;
     system("pause");
     return 0;
      }
```

- c++的Vector一样提供[]下标操作，并从0项开始。还提供at()的只读操作,at()比[]更快，因为它不会导致深度复制。

# 第12次作业要求（必做题）

第1道题：设计一个函数模板，其中包括数据成员T a[n]以及对其进行排序的成员函数sort()，模板参数T可实例化成字符串。

第2道题：设计一个类模板，其中包括数据成员T a[n]以及在其中进行查找数据元素的函数int search(T)，模板参数T可实例化成字符串。

# 第12次作业要求（选做题）

- 要求如下：

（1）参考课件中“类模版派生实例代码”，用模板类派生新的派生类方法（即p18第2种派生方式），重新改写p19-21的程序代码。

（2）链表插入要求是在指定的结点之后插入新结点。结点指定是指输入结点序号值。例如，在3号结点之后插入等。

# 12.5 笔试安排

- 笔试考核内容覆盖了本学期的学习内容。包括但不限于：运算符重载、文件输入输出、虚函数与多态性、构造与析构函数等；讲课内容不考有：多继承、模板、异常处理、标准模板库（STL）
- 题型：概念填空、完成程序填空和分析程序结果3类；
- 考试地点：6C201、6C202
- 考试时间：15周周6（6月3号晚上7:00-9:00）
- 考试时长：考试时间为120分钟，题量为40分；
- 答疑时间：考前2天在电子馆5层104实验室。平时也可以去答疑，事先电话联系。也可通过EMAIL答疑；或找助教答疑。

# 小学期安排

- 小学期时间6月26日-7月7日（2周）；第一周以听python讲座为主；第2周上机完成C++大作业。
- C++大作业具体任务见《 C++课程设计任务书》、完成“学生成绩管理系统”或“ 公司人事管理系统”。

## 第1周Python语言课程安排

- 3次讲座，分2个班，时间：
  周一（6月26日）9:50到12:15
  周三（6月28日）9:50到12:15
  周五（6月30日）9:50到12:15
- 杨昉老师课，每次3学时（对接黄永峰班+杨昉班同学）
  周二（6月27日）9:50到12:15
  周四（6月29日）9:50到12:15
  周六（7月1日） 9:50到12:15
- 杨毅老师课，每次3学时（对接孙老师班同学）
- 地点：罗姆楼三层报告厅

# 第2周C++大作业上机安排

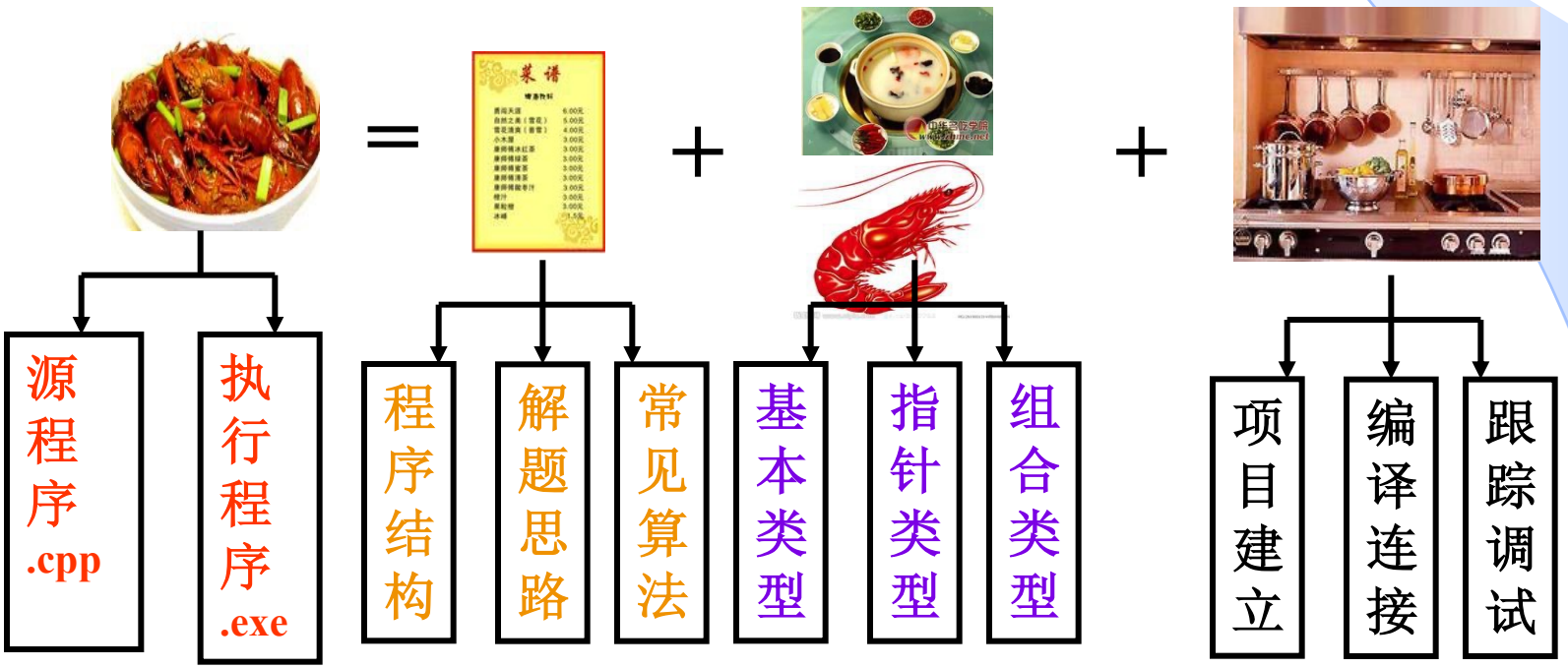
- 实验时间安排与要求：大体分成四组：
- 第一组：无21、无22、无23
- 第二组：无24、无25、无26
- 第三组：无27、无28、车辆13
- 第四组：其他选修同学
- 要求：每人每天一时间单位上机。除非有特殊情况跟授课教师请假外，未在网上提交实验报告之前，须按照时间表去主楼九楼机房上机(用自己笔记本也要带着去机房上机)

| 小学期第2周 | 7月3日 | 7月4日 | 7月5日 | 7月6日 | 7月7日 |
|---|---|---|---|---|---|
| | 星期一 | 星期二 | 星期三 | 星期四 | 星期五 |
| 上午 | 第一组 | 第二组 | 第三组 | 第四组 | 第一组 |
| 中午 | 第二组 | 第三组 | 第四组 | 第一组 | 第二组 |
| 下午 | 第三组 | 第四组 | 第一组 | 第二组 | 第三组 |
| 晚上 | 第四组 | 第一组 | 第二组 | 第三组 | 第四组 |
| 上午：8:00~11:30、 | | | 中午：11:30~15:00 | | |
| 下午：15:00～18:30 | | | 晚上：18:30~22:00 | | |

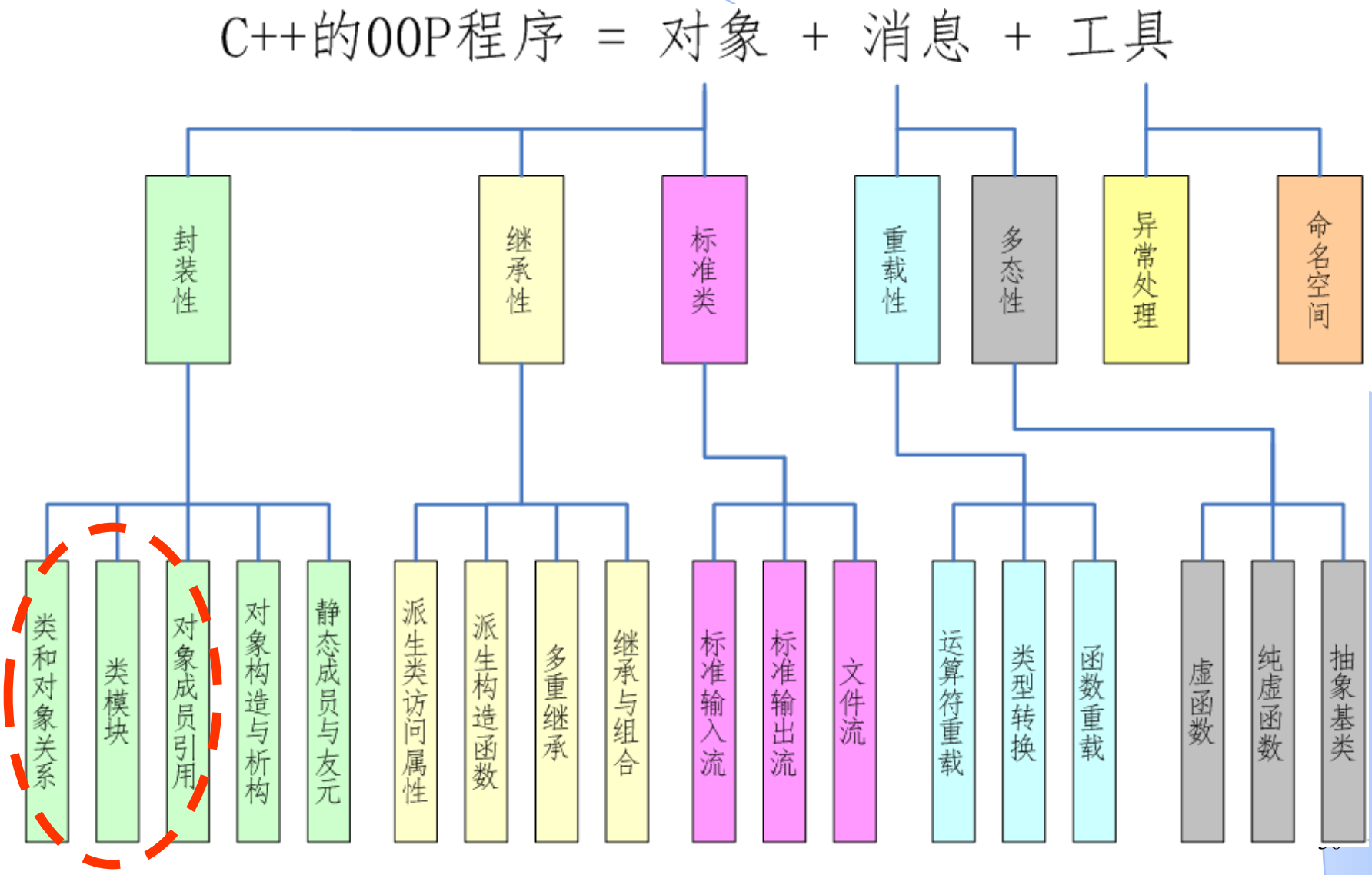
# 计算机程序设计(1)知识体系

- Nikiklaus wirth著名公式：程序＝算法＋数据结构

- 编程 ＝ 算法 ＋ 数据类型 ＋ 工具

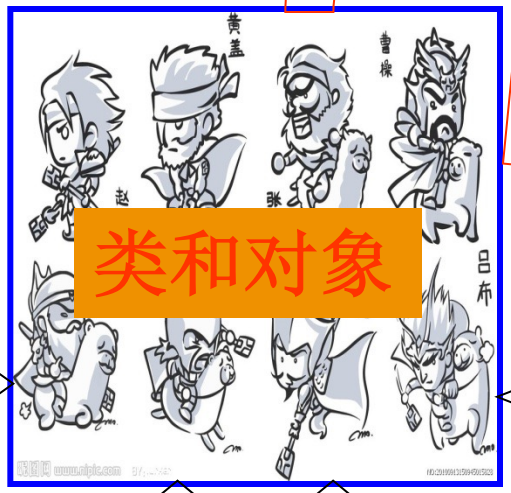


# 本学期的知识点MAP


C++的OOP程序 = 对象 + 消息 + 工具
封装性
继承性
标准类
重载性
多态性
异常处理
命名空间
类和对象关系
类模块
对象成员引用
对象构造与析构
静态成员与友元
派生类访问属性
派生构造函数
多重继承
继承与组合
标准输入流
标准输出流
文件流
运算符重载
类型转换
函数重载
虚函数
纯虚函数
抽象基类

# 核心概念

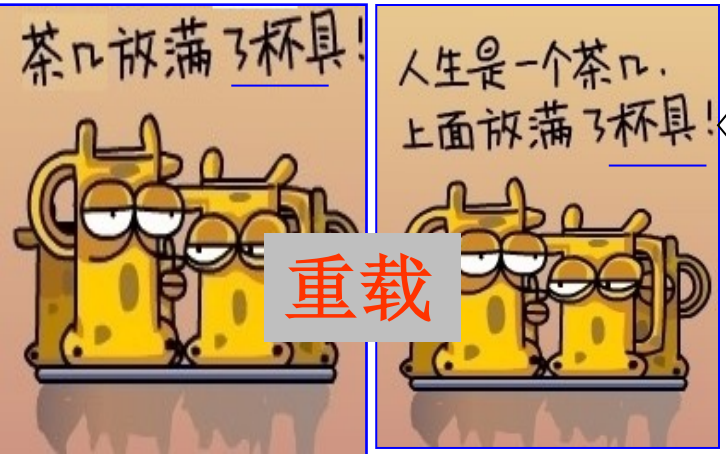

流





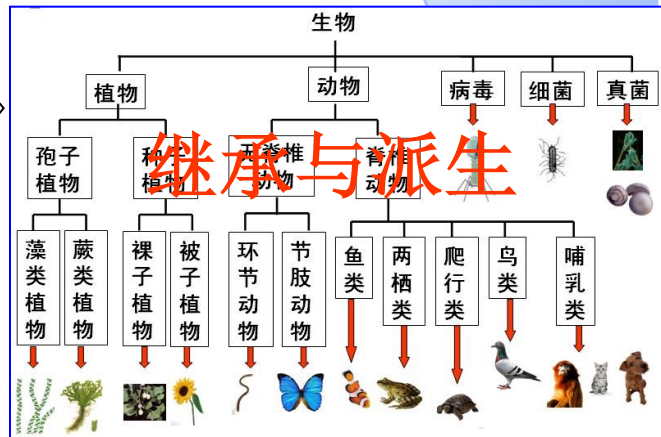
数据私有性

类族统一调用




运算

重用



# 面向对象程序设计特点总结

- 抽象性：类是对象的抽象，对象是类的具体实例；类模板是类的抽象，类是类模板的具体实例；基类是派生类的抽象，派生类是基类的具体实例；抽象类等；
- 封装性：类成员的3种访问属性；公有接口与私有实现的分离；类申明与成员函数定义的分离；构造函数；数据保护（6常）；
- 继承性：3种继承方式，派生类成员的4种访问属性；派生类的构造函数（3类）；多重继承中函数同名问题与虚基类；
- 多态性：函数重载、运算符重载、多层派生的函数同名问题与虚函数；纯虚函数与抽象类。

# 面向对象程序设计关键技术总结

- 对象成员的3种访问方式；6种构造函数；对象指针（this）；数据保护的6种常类型；对象动态建立与释放；对象的复制（浅拷贝与深拷贝问题）；静态成员，友元函数；
- 运算重载规则；成员函数与友元函数重载；类型转换函数与转换构造函数；
- 派生类的构造函数的实现方法；基类与派生类的兼容性问题
- 虚函数的引用方法；虚析构函数；
- Cin和cout;文件操作与文件流；
- 异常处理。

# 祝愿大家考出好成绩！

# 希望都成为编程高手！

罗联系