C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

教材介绍

C++程序设计

（2008年版）

编著：刘振安

机械工业出版社

考试题型

单选题　　1分×20题 = 20分

填空题　　1分×20题 = 20分

程序改错题　4分×5题 = 20分

完成程序题　4分×5题 = 20分

程序分析题　5分×2题 = 10分

程序设计题 10分×1题 = 10分

第七章　类模板与向量

目录

第7章

类模板与向量

§7.1 类模板

类模板概念：

如将类看做是包含某些数据类型的框架，然后将这些数据类型从类中分离出来形成一个通用的数据类型T，为其设计一个操作集，并允许原来那些数据类型的类都能使用这个操作集，这就能避免因为类的数据类型不同而产生的重复性设计，这种对类的描述的类型T称为类模板。

在编译时，由编译器将类模板与某种特定数据类型联系起来，就产生一个特定的类，称为模板类。由此可见，利用类模板能大大简化程序设计。

一、类模板基础知识

1、类模板的成分及语法

类模板声明的一般形式：template <类模板参数> class 类名 { //类体};

类模板参数中的class意为“任意内部类型或用户定义类型”，T是结构或类。

由此可见，类模板与函数模板的声明方法及参数格式是相同的，不同的是在类模板参数表之后还有类声明。在类中可以像使用其他参数类型（如int、double等）那样使用模板参数。例如，可以把模板参数用作数据成员的类型、成员函数的类型或成员函数的参数类型等。

例1：使用类模板的实例

```
template <class T>  //带参数T的类模板声明，可用typename代替class
class TAnyTemp{    //类声明
    T x,y;    //声明类型为T的私有数据成员
Public:
    TAnyTemp(T X,T Y):x(X),y(Y) { }  //类TAnyTemp的构造函数，实参类型为T
    T getx( ){return x;}  //返回类型为T的内联成员函数
    T gety( ){return y;}  //返回类型为T的内联成员函数
};
```

2、类模板的对象

类模板也称参数化类，初始化类模板时，只要传给它指定的数据类型，编译器就会用指定类型替代模板参数产生相应的模板类。

类模板定义对象的一般格式：

类名 <模板实例化参数类型> 对象名； //默认或无参数构造函数

类名 <模板实例化参数类型> 对象名（构造函数实参列表）；//构造函数

例2：求4个数中最大值的类模板程序

```
template <class T>
class Max4 {
                T a,b,c,d;  //四个类型为T的私有数据成员
                T Max(T a,T b){return (a>b)?a:b;}
                                        //类型为T，参数类型为T，返回a、b二者最大值的私有成员函数
            public:
                Max4(T,T,T,T);  //声明构造函数，含4个类型为T的参数
                T Max(void);     //声明返回值类型为void的公有成员函数
};
template <class T>          //定义成员函数必须再次声明类模板
Max4<T>::Max4(T x1,T x2,T x3,T x4):a(x1),b(x2),c(x3),d(x4) { }
template <class T>      //定义成员函数必须再次声明类模板
T Max4<T>::Max(void)
 {return Max(Max(a,b),Max(c,d));}
                                        //定义类Max4的成员函数Max(void)，定义时要将Max<T>看作整体
void main( ){
            Max4 <char> C('W','w','a','A');  //比较字符
            Max4 <int> A(-25,-67,-66,-256);   //比较整数
            Max4 <double> B(1.25,4.3,-8.6,3.5);   //比较双精度实数
            cout<<C.Max( )<<" "<<A.Max( )<<" "<<B.Max( )<<endl;}
```

w -25 4.3

3、定义模板类的成员函数

在类体外面定义成员函数时，必须用template重写类模板说明，一般格式为：

template <模板参数>

返回类型 类名 <模板类型参数>::成员函数名（函数参数列表）{ //函数体}

式中<模板类型参数>是指template后的”< >”内使用class（或typename）声明的类型参数，构造函数和析构函数没有返回类型。

模板实例化参数类型包括数据类型和值，因为编译器不能从构造函数参数列表推断出模板实例化参数类型，所以必须显式的给出对象的参数类型。

例3：演示对四个数字求和的类模板程序

```
#include <iostream>
using namespace std;
template <class T,int size=4>  //可以传递程序中的整数参数值
class Sum {
        T m[size];  //数据成员
public:
        Sum(T a,T b,T c,T d)  {m[0]=a;m[1]=b;m[2]=c;m[3]=d;}  //构造函数
        T S( )  {return m[0]+m[1]+m[2]+m[3];}  //求和成员函数
};
void main( ){
        Sum<int,4>num1(-23,5,8,-2);  //整数求和
        Sum<float,4>f1(3.4f,-8.5f,8.8f,9.7f);  //单精度求和，使用f显式说明
        Sum<double,4>d1(355.4,253.8,456.7,-67.8);  //双精度求和
        Sum<char,4>c1('W',-2,-1,-1);  //字符减，等效于'W'-4，结果为'S'
        cout<<num1.S( )<<" "<<f1.S( )<<","<<d1.S( )<<","<<c1.S( )<<endl;
}
```

-12 13.4,998.1,S

课堂练习

以下类模版定义正确的为（ ）

A:template<class T>

B:template<class T,class int i>

C:template<class T,typename T>

D:template <class T1,T2>

课堂练习

以下类模版定义正确的为（ ）

A:template<class T>

B:template<class T,class int i>

C:template<class T,typename T>

D:template <class T1,T2>

答案：A

课堂练习

允许用户为类定义一种模式，使得类中的某些数据成员及某些成员函数的返回值能取任意类型，这是一个（ ）

A:类模版

B:模版类

C:函数模版

D:模版函数

课堂练习

允许用户为类定义一种模式，使得类中的某些数据成员及某些成员函数的返回值能取任意类型，这是一个（ ）

A:类模版

B:模版类

C:函数模版

D:模版函数

答案：B

课堂练习

关于类模板的说法正确的是（ ）。

A:类模板的主要作用是生成抽象类

B:类模板实例化时，编译器将根据给出的模板实参生成一个类

C:在类模板中的数据成员具有同样类型

D:类模板中的成员函数没有返回值

课堂练习

关于类模板的说法正确的是（ ）。

A:类模板的主要作用是生成抽象类

B:类模板实例化时，编译器将根据给出的模板实参生成一个类

C:在类模板中的数据成员具有同样类型

D:类模板中的成员函数没有返回值

答案：B

课堂练习

关于函数模板，描述错误的是（ ）。

A:函数模板必须由程序员实例化为可执行的函数模板

B:函数模板的实例化由编译器实现

C:一个类定义中，只要有一个函数模板，则这个类是类模板

D:类模板的成员函数都是函数模板，类模板实例化后，成员函数也随之实例化

课堂练习

关于函数模板，描述错误的是（ ）。

A:函数模板必须由程序员实例化为可执行的函数模板

B:函数模板的实例化由编译器实现

C:一个类定义中，只要有一个函数模板，则这个类是类模板

D:类模板的成员函数都是函数模板，类模板实例化后，成员函数也随之实例化

答案：C

二、类模板的派生和继承

类模板也可以继承，继承的方法和普通的类一样。声明模板继承之前，必须重新声明类模板。模板类的基类和派生类都可以是模板类或非模板类。

例4：类模板Line继承非模板类Point

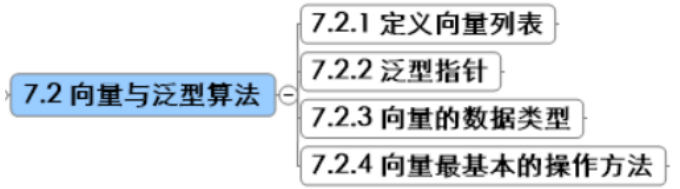
```
#include <iostream>
using namespace std;
class Point{
        int x,y;
public:
        Point(int a,int b){x=a;y=b;}  //类Point的构造函数
        void display(){cout<<x<<","<<y<<endl;}  //类Point的公有成员函数
};
template <typename T>  //声明继承之前，需重新声明类模板
class Line:public Point{  //模板类Line公有继承非模板类Point
        T x2,y2;
public:
        Line(int a,int b,T c,T d):Point(a,b)  {x2=c;y2=d;}   //类Line的构造函数
        void display(){Point::display();  cout<<x2<<","<<y2<<endl;}};
void main(){
        Point a(3,8);
        a.display();  //输出3，8
        Line<int> ab(4,5,6,7);   //线段ab两个点的坐标均是整数
        ab.display();  //输出4，5  6，7
        Line<double> ad(4,5,6.5,7.8);   //线段ad一个点的坐标是整数，另一个是实数
        ad.display();  //输出4，5  6.5，7.8
}
```

```
3,8
4,5
6,7
4,5
6.5,7.8
```

```
#include <iostream>
using namespace std;
template <typename T>
class Point{
        T x,y;
public:
        Point(T a,T b){x=a;y=b;}  //模板类Point的构造函数
        void display(){cout<<x<<","<<y<<endl;}  //模板类Point的公有成员函数
};
template <typename T>  //声明继承之前，需重新声明类模板
class Line:public Point<T>{  //模板类Line公有继承模板类Point
        T x2,y2;
public:
        Line(T a,T b,T c,T d):Point<T>(a,b){x2=c;y2=d;}  //模板类Line的构造函数
        void display() {Point<T>::display(); cout<<x2<<","<<y2<<endl;}};
void main(){
        Point <double> a(3.5,8.8);
        a.display();  //输出3.5，8.8
        Line<int> ab(4,5,6,7);   //全部使用整数
        ab.display();  //输出4，5  6，7
        Line<double> ad(4.5,5.5,6.5,7.5);   //全部使用实数
        ad.display(); } //输出4.5，5.5  6.5，7.5
```

```
3.5,8.8
4,5
6,7
4.5,5.5
6.5,7.5
```

§7.2 向量与泛型算法



向量是一维数组的类版本，与数组类似，其中的元素项是连续存储的，不同的是：

第一，在数组生存期内，数组的大小是不变的，而向量中存储元素的多少可以在运行中根据需要动态的增长或缩小；

第二，向量是类模板，具有成员函数，例如可以使用向量名.size（）的方法动态地获得vector对象当前存储元素地数量。

一、定义向量列表

1、向量的构造函数

向量（vector）类模板定义在头文件vector中，它提供4种构造函数，用来定义由各元素组成的列表。

如果用length表示长度，用type表示数据类型，用name表示对象名，则有：

vector <type> name;　　//定义元素类型为type的向量空表

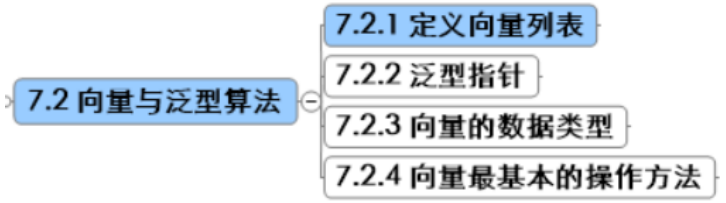
vector <type> name（length）; //定义具有length个类型为type的元素的向量，元素均初始化为0

vector <type> name（length,a）;//定义具有length个类型为type的元素的向量，元素均初始化为a

vector <type> name1（name）;　//用已定义的向量name1构造向量name

空表没有元素，它使用成员函数获取元素。

2、向量赋值的典型方法：




```
vector <char> A;  //定义空的char向量A
vector <int> B(20);  //定义含20个值均为0的int型元素的向量B
vector <int> C(20,1);  //定义含20个值均为1的int型元素的向量C
vector <int> D(C);  //用向量C初始化D，使D与C一样
vector <char> E(20,’t’);  //定义含20个值均为t的char型元素的向量E
vector <int> F(2*2,5);  //定义含4个值均为5的int型元素的向量F
vector <int> G(F);  //用向量F初始化G，使G与F一样
```

2、向量赋值的典型方法：

D=F; //整体赋值，使D与F相同

G=C; //整体赋值，使G与C相同

式中通过赋值运算符“=”可使同类型的向量列表相互赋值，而不管它们的长度如何，向量可以改变赋值目标的大小，使它的元素数目与赋值源的元素数目相同。

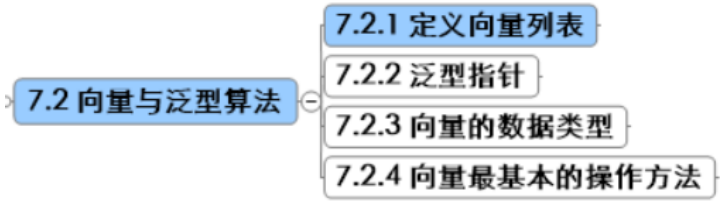
对于上面的定义，通过使用语句

cout<<B.size( )<<”,” <<C.size( )<<”,” <<D.size( )<<”,”

<<E.size( )<<”,” <<F.size( )<<”,” <<G.size( )<<endl;” ,

可得到关于每个向量长度的信息“20,20,4,20,4,20”。

3、利用函数size（）循环输出向量B的内容




```
for（int i = 0； i<B.size( );i++）
    cout<<B[i]<<endl;
```

4、利用数组初始化向量

不能使用列表初始化向量，但可通过定义一个数组，然后把数组的内容复制给向量的方法来初始化向量。

```
int IA[10]={1,2,3,4,5,6,7,8,9,0};
vector <int> VB(IA,IA+10);
```

式中IA是数组名，所以可以代表数组首地址，IA+10是向量VB的结束标志位，VB的长度为10。因为向量自动产生一个结束标志，所以VB不需要与IA等长。当定义的VB比IA长时，第10个以后的元素值将不确定。这是定义向量的又一种方式，而且是在定义向量的同时完成初始化。不过要注意的是，不能使用这种方法去初始化一个已经声明或定义过的向量。

二、泛型指针

与操作对象的数据类型相互独立的算法称为泛型算法，泛型算法通过一对泛型指针begin和end实现，元素存在范围是半开区间[begin,end 。

向量a和泛型指针示意图

1、iterator声明正向泛型指针

在向量中，**泛型指针**是在底层指针的行为之上提供一层抽象化机制，是使用类实现的，取代程序原来的“指针直接操作方式”。

对向量的访问可以是双向的，即可正向，也可逆向。iterator用于声明向量的正向泛型指针，它在STL里面是一种通用指针，如用T表示向量的参数化数据类型，则iterator在向量中的作用相当于T *，其声明形式为：

vector <type>::iterator 泛型指针名；

例如“vector <char>::iterator p;”表示向量的数据类型为char，指针名为p，注意不能使用*p，这是因为iterator相当于char *，如果用*p，则表示指针指向的元素值，这样来定义指针就是错误的。

2、reverse_iterator声明逆向泛型指针

逆向泛型指针的开始和结束标志是如上图中所示的rbegin和rend，加操作时向rend方向移动，减操作时向rbegin方向移动。

逆向泛型指针的声明使用reverse_iterator，声明形式为：

vector <type>::reverse_iterator 泛型指针名；

3、使用typedef声明或定义泛型指针标识符

使用typedef声明或定义泛型指针标识符后，可直接用该标识符定义泛型指针。

例如语句“typedef vector <double>::iterator iterator;”

定义标识符iterator，在程序中就可直接使用iterator声明或定义泛型指针，如语句“iterator p;”声明了泛型指针p。

例6：演示泛型指针和copy函数的实例

```
1.1 4.4 3.3 2.2 2.2 3.3 4.4 1.1 1.1 4.4 3.3 2.2
2.2 3.3 4.4 1.1 1.1 4.4 3.3 2.2 2.2 3.3 4.4 1.1
```

```
#include <iostream>
#include <algorithm>     //升幂需含此头文件
#include <vector>
using namespace std;
void main(){
        double a[]={1.1,4.4,3.3,2.2};     //定义数组a
        vector <double> va(a,a+4),vb(4);  //定义向量va及vb，并用数组a初始化va
        typedef vector <double>::iterator iterator; //自定义正向泛型指针标识符iterator
        iterator first=va.begin();  //用标识符iterator定义正向泛型指针first并指向va的首元素
        for(first;first<va.end();first++)   cout<<*first<<" ";  //循环正向输出va
        for(--first;first>va.begin()-1;first--)   cout<<*first<<" ";    //语句1
        copy(va.begin(),va.end(),ostream_iterator<double>(cout," "));  //整体正向输出va
        cout<<endl;    //换行
        typedef vector<double>::reverse_iterator reverse_iterator;  //语句2
        reverse_iterator last=va.rbegin();  //语句3，定义逆向泛型指针last并指向va的尾元素
        for(last;last<va.rend();last++)   cout<<*last<<" ";  //循环逆向输出va
        for(--last;last>va.rbegin()-1;last--)   cout<<* last<<" ";  //语句4
        copy(va.rbegin(),va.rend(),ostream_iterator<double>(cout," "));  //整体逆向输出va
}
```

课堂练习

根据“vector<char>：： iterator p；”可知，向量的数据类型为（）

A:vector

B:p

C:char

D:iterator

课堂练习

根据“vector<char>：： iterator p；”可知，向量的数据类型为（）

A:vector

B:p

C:char

D:iterator

答案：C

课堂练习

根据“vector<char>：： iterator p；”可知，指针名称为（ ）

A:*p

B:p

C:char

D:vector

课堂练习

根据“vector<char>：： iterator p；”可知，指针名称为（ ）

A:*p

B:p

C:char

D:vector

答案：B

三、向量的数据类型

向量除了可使用基本数据类型，还可使用构造类型，只要符合构成法则即可。

例7、演示向量使用实数类型的例子

```
#include <iostream>
#include <algorithm>
#include <functional>
#include <vector>
using namespace std;
```

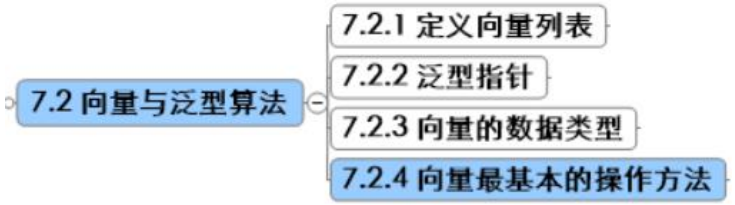
```
void main(){
        double a[]={1.1,4.4,3.3,2.2};
        vector <double> va(a,a+4),vb(4);  //定义实数向量va、vb，并用数组a初始化va
        copy(va.begin(),va.end(),ostream_iterator<double>(cout," "));  //正向输出va
        cout<<endl;
        reverse_copy(va.begin(),va.end(),vb.begin());  //逆向复制va至vb
        copy(vb.begin(),vb.end(),ostream_iterator<double>(cout," "));  //正向输出vb
        cout<<endl;
        sort(va.begin(),va.end());  //升幂排序va
        sort(vb.begin(),vb.end(),greater<double>());  //降幂排序vb
        copy(va.begin(),va.end(),ostream_iterator<double>(cout," "));  //正向输出va
        cout<<endl;
        copy(vb.begin(),vb.end(),ostream_iterator<double>(cout," "));  //正向输出vb
        cout<<endl;
        va.swap(vb);  //交换va和vb的内容
        copy(va.begin(),va.end(),ostream_iterator<double>(cout," "));  //正向输出va
        cout<<endl;
        copy(vb.begin(),vb.end(),ostream_iterator<double>(cout," "));  //正向输出vb
        cout<<endl;
        cout<<*find(va.begin(),va.end(),4.4);  //在va中查找4.4
}
```

```
1.1 4.4 3.3 2.2
2.2 3.3 4.4 1.1
1.1 2.2 3.3 4.4
4.4 3.3 2.2 1.1
4.4 3.3 2.2 1.1
1.1 2.2 3.3 4.4
```

演示使用复数类和结构作为向量数据元素的实例

```
#include <iostream>
#include <complex>
#include <vector>
using namespace std;
struct st{
        int a,b;
} a[]={{2,5},{4,8}};  //声明结构的对象数组a
void main(){
        complex<float>num[]={complex<float>(2,3),complex<float>(3.5,4.5)};
        vector<complex<float> * >vnum(2);  //复数类的指针作为向量的数据类型
        vnum[0]=&num[0];
        vnum[1]=&num[1];
        for(int i=0;i<2;i++)
            cout<<"real is"<<vnum[i]->real()<<",image is"<<vnum[i]->imag()<<endl;
        vector<st * >cp(2);  //结构指针作为向量的数据类型
        cp[0]=&a[0];
        cp[1]=&a[1];
        for(i=0;i<2;i++)
            cout<<"a="<<cp[i]->a<<",b="<<cp[i]->b<<endl;
}
```

```
real is2,image is3
real is3.5,image
is4.5
a=2,b=5
a=4,b=8
```



四、向量最基本的操作方法

向量有许多成员函数用来提供不同的操作，如：

1、访问向量容量信息的方法

① size（）：返回当前向量中已经存放的对象个数；

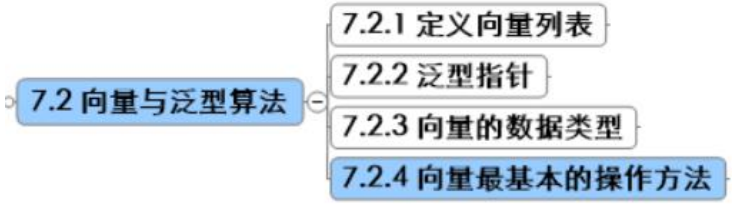
② max_size（）：返回向量最多可以容纳的对象个数，此参数取决于硬件结构，不由用户指定；

③ capacity（）：返回无需再次分配内存就能容纳的对象个数，其初始值为程序员最初申请的元素个数，即已申请的空间。实际操作时，当存放空间已满，又增加一个元素时，其值在原来的基础上自动翻倍，扩充空间以便存放更多的元素。

④ empty（）：当前向量为空时，返回true值，内容不空时，返回0值。

⑤ 前三者关系：max_size( )>=capacity( )>=size( )

2、访问向量中对象的方法



2、访问向量中对象的方法

① front（）：返回向量中的第一个对象；

② back（）：返回向量中的最后一个对象；

③ operator[ ]（size_type,n）:返回向量中下标为n（第n + 1个）的元素。

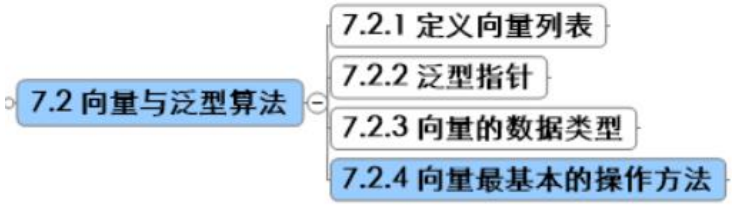
3、在向量中插入或删除对象的方法

1、push_back（const T&）：向向量尾部插入一个对象。

2、insert(iterator it,const T&)：向it所指的向量位置前插入一个对象。

3、insert(iterator it,size_type n,const T&X)：

向it所指向量位置前插入n个值为x的对象。



4、在向量中删除对象的方法

1、pop_back（const T&）:删除向量中最后一个对象。

2、erase(iterator it)： 删除it所指向的容器对象

3、clear():删除向量中的所有对象，empty()返回true值。

例9：访问向量容量信息及对象的实例

```
#include <iostream>
#include <algorithm>
#include <functional>
#include <vector>
using namespace std;
void main(){
        vector <char>a(10),b(10);  //产生向量a、b，内容均为0
        cout<<a.empty()<<","<<sizeof(a)<<",";  //输出0，16
        for (char i='a',j=0;j<10;j++) a[j]=i+j;
        cout<<a.max_size()<<","<<a.capacity()<<","<<a.size()<<endl;  //输出4294967295，10，10
        for (j=0;j<10;j++) cout<<a[j]<<" ";  //输出向量a内容“a b c d e f g h i j”
        cout<<endl;
        copy(a.begin(),a.end(),b.begin());  //复制向量a内容到b
        copy(b.begin(),b.end(),ostream_iterator<char> (cout," "));  //输出“a b c d e f g h i j”
        cout<<endl;
        reverse_copy(b.begin(),b.end(),ostream_iterator<char>(cout," "));  //逆向输出b
        cout<<endl;
        cout<<a.front()<<","<<a.back()<<","<<a.operator[](5)<<endl;  //输出“a,j,f”
        sort(b.begin(),b.end(),greater<char>());  //降幂排序b,
        copy(b.begin(),b.end(),ostream_iterator<char>(cout," "));  //输出向量b
}
```

```
0,16,4294967295,10,10
a b c d e f g h i j
a b c d e f g h i j
j i h g f e d c b a
a,j,f
j i h g f e d c b a
```

例11：使用泛型指针进行插入和删除的实例

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
void main(){
        char st[11]="abcdefghij";
        vector <char>a(st,st+10);   //将数组st的前10个元素复制给向量a，不复制标志“\0”
        vector <char>::iterator p;  //定义泛型指针p
        p=a.begin();  //p指向第一个元素的地址
        a.insert(p+3,'X');  //a[3]='X'
        copy(a.begin(),a.end(),ostream_iterator<char> (cout," "));   //输出a
        cout<<endl;
        p=a.begin();  //指针p返回首位置
        a.insert(p,3,'A');  //在a[0]前插入3个A
        copy(a.begin(),a.end(),ostream_iterator<char> (cout," "));   //输出a
        cout<<endl;
        a.erase(p+8);  //删除a[8]，即第九个元素e
        copy(a.begin(),a.end(),ostream_iterator<char> (cout," "));   //输出a
}
```

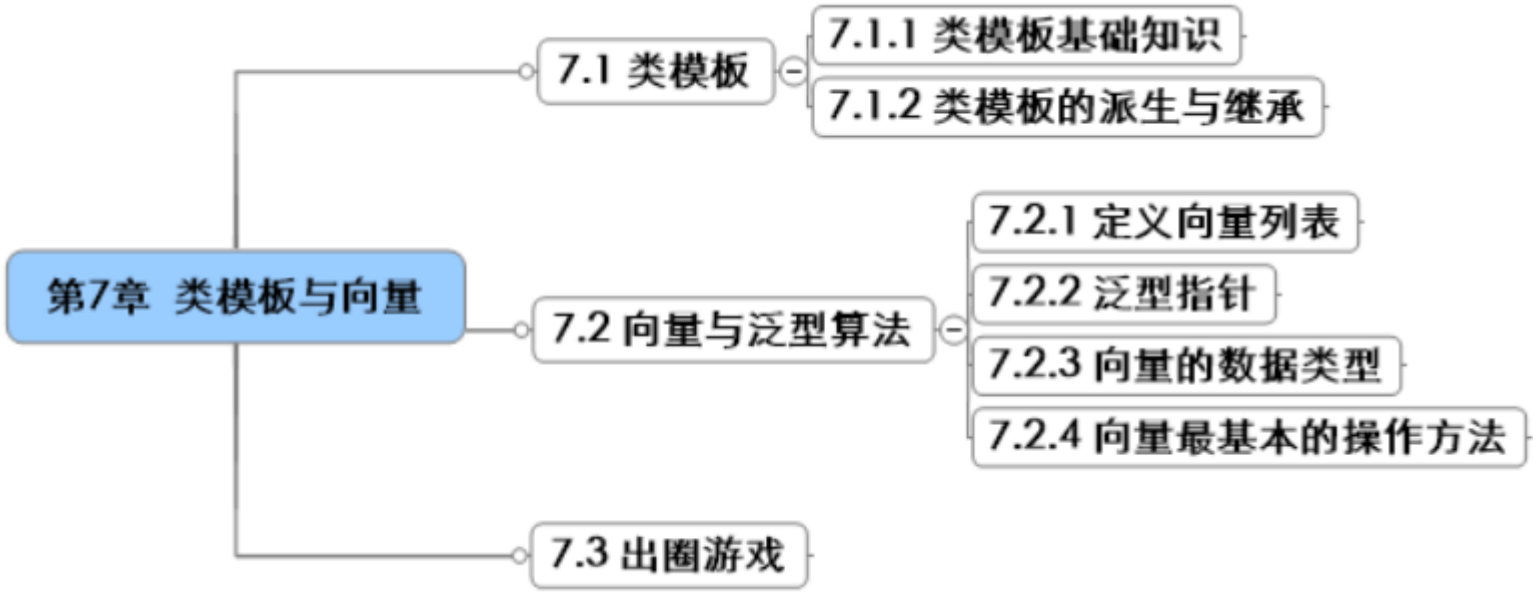
```
a b c X d e f g h i j
A A A a b c X d e f g h i j
A A A a b c X d f g h i j
```

例12：双向访问的实例

```
a b c d e f g h i j
j i h g f e d c b a
j i h g f e d c b a
a b c d e f g h i j
```

```
#include <iostream>
#include <vector>
using namespace std;
void main(){
        char st[11]="abcdefghij";
        vector<char>a(st,st+10);
        vector<char>::iterator p=a.begin();  //定义正向泛型指针并初始化
        vector<char>::reverse_iterator ps;  //定义逆向泛型指针
        for ( p=a.begin();p!=a.end();++p) cout<<*p<<" ";  //正向访问，输出a～j
        cout<<endl;
        for ( p=a.end()-1;p!=a.begin()-1;--p) cout<<*p<<" ";  //使用正向泛型指针逆向访问，输出j～a
        cout<<endl;
        for ( ps=a.rbegin();ps!=a.rend();++ps) cout<<*ps<<" ";
                        //使用逆向泛型指针正向访问，使用＋＋运算，输出j～a
        cout<<endl;
        for ( --ps;ps!=a.rbegin()-1;--ps) cout<<*ps<<" ";
                        //使用逆向泛型指针逆向访问，使用－－运算，输出a～j
}
```

本章总结


第7章 类模板与向量
7.1 类模板
7.1.1 类模板基础知识
7.1.2 类模板的派生与继承
7.2 向量与泛型算法
7.2.1 定义向量列表
7.2.2 泛型指针
7.2.3 向量的数据类型
7.2.4 向量最基本的操作方法
7.3 出圈游戏

真题演练

13. 下列有关模板的描述中，错误的是

A. 模板把数据类型作为一个设计参数，称为参数化程序设计

B. 使用时，模板参数与函数参数相同，是按位置而不是名称对应的

C. 模板实例化参数类型包括数据类型和值

D. 类模板与模板类是同一个概念

D

真题演练

38. 在 C++ 中，利用向量类模板定义一个具有 10 个 int 的向量 A，其元素均被置为 1，实现此操作的语句是 _____。

vector <int> A(10,1)

真题演练

假设声明了以下的函数模板，错误的调用语句是

```
template <class T>
T max(T x,T y) { return (x>y)? x:y;}
并定义了 int i;char c;
A. max(i,i)        B. max(c,c)        C. max((int)c,i)    D. max(i,c)
```

D

真题演练

9. 实现两个相同类型数加法的函数模板的声明是

A. add(T x,T y)　　B. T add(x,y)

C. T add(T x,y)　　D. T add(T x,T y)

D

真题演练-程序改错

45.

```
#include <iostream.h>
template <typename AT>
AT max(AT x,AT y)
{
  return (x>y)? x:y;
void main()
{
  int i1=10,i2=56;
  double d1=50.344,d2=56.346;
  char c1='k',c2='n';
  cout<<"较大的整数是:"<<max(i1,i2)<<endl;
  cout<<"较大的双精度型数是:"<<max(d1,d2);
  cout<<endl;
  cout<<"较大的字符是:"<<max(c1,c2)<<endl;
}
```

祝大家顺利通过考试！