3.1：using 命名空间声名

使用 using 命名空间声名可以简化程序结构，每个名字都需要独立的 using 声名，头文件应不包含 using 声名。

3.2：标准库类型 string

①：STL 中的 string 表示可变长的字符序列，使用时必须包含头文件，string 也是在 std 命名空间中，所以也必须在命名空间下使用。

②：需注意 string 是一个类，声名在类下的都是对象，对这个对象来说有许多成员函数可以使用。

初始化：string 提供了多种初始化的方式，详见 p76。

String 对象上的操作：可以写入，输出，判断是否为空，大小，相加，赋值，判断是否相等，判断大小关系等。这些都是成员函数和重定义的操作符完成的。具体参见 p77

读：>>操作符读到空白就停止。

Getline 可以读取一行，遇见回车为止（换行符也被读入，但是在存入对象的时候把换行符丢掉）。

```
String line;
Getline(cin,line);
Cout<<line;
```

Empty 和 size 操作：empty()返回的是一个对应的布尔值。Size()返回的却不是 int 类型的，实际上返回的是一个 string::size_type 的类型，这是一个无符号的值，用的时候可以用 auto 来推导：

```
Auto len=line.size();
```

比较 string 对象：这些比较规则都依据字典顺序（大小写敏感）

String 对象赋值：允许把一个对象的值赋值给另外一个对象，

String 相加：两个 string 对象相加会得到一个新的对象。赋值给左边的对象。

String 和字面值相加：允许 string 对象和字面值相加，结果转化为对象，但不允许字面值和字面值相加。

处理 string 对象中的字符：比较好的方法是利用 range for 语句。结合 cctype 头文件里的一些函数进行处理。结合下标运算符可以对对象中的一部分进行修改。

3.3：标准库类型 vector。

①：vector 是模板而非类型，所以在使用时必须包含其中元素的类型，eg：vector<int>

某些编译器如果要把 vector 的元素还是 vector 的话要在外层右尖括号添加一个空格。

Vector<vector<int> >　这样才可以。

②：vector 的一些操作。

初始化：vector 是一个类模板，也提供了许多初始化的方法，详见 p87.

特殊地：c++11 提供了一种为 vector 初始化的方法：列表初始化，即把所要初始化的元素放在大括号里列出，赋值给 vector 对象。

创建制定数量的元素：vector<int>　i1(10,1)　创建大小为 10 的 int 型的 vector，每个元素都是 1.其他的类型也可这样写。

向 vector 中添加元素：利用 push_back(t)成员函数。可以向末尾添加一个元素。

另外，vector 也提供了 string 类似的一些成员函数和重载运算符，比如 v.empty(),v.size(),v[n]（返回引用）,比较（==,！=，<……）等。

用 range for 语句进行元素访问：可以用 range for 语句对 vector 的元素进行访问。

v.size()的返回类型：要使用 size_type 必须指定是哪种类型定义的。

```
Vector<int>::size_type;
```

Vector::size_type;　这样写是不对的。

不能用下标形式添加元素：只能对已经存在的元素进行下标操作，不能以下标形式添加新的元素。要添加元素用 push_back()成员函数。下标越界是一种常见的错误，但是编译器不会发现，保证下标合法的一种有效方法是使用 range for 语句。

3.4：迭代器

当然可以用下标运算符来访问 string 或者 vector 的元素，迭代器是一种更通用的方法，除了 vector 之外，STL 还定义了其他集中容器，所有的标准库容器都可以使用迭代器，但只有少数的几种同时支持下标运算。

① 使用迭代器：

和指针不同的是，获取迭代器使用的不是取地址符，有迭代器的类型同时拥有返回迭代器的成员。

比如这些类型都拥有名为 begin 和 end 的成员，分别指向第一个和最后一个元素后面的位置（尾后 off the end）。特殊情况下如果容器为空，则 begin 和 end 返回的是同一个迭代器。

② 迭代器的一些运算符：

详见 P96

③ 泛型编程。

在使用迭代器时，常用!=而非<，这是 c++的习惯，因为并非所有的容器都提供了< 运算符，但是都提供了==和!=运算符，在使用迭代器时，常常使用！=来进行判断。

④ 迭代器的类型：

一般来说我们不需要知道迭代器的类型，实际上迭代器一般是 iterator 或者 const_iterator 这样的类型。

Vector<int>::iterator it;

Vector<int>::const_iterator;

这两种类型实际上是迭代器的类型，其中带 const 的类型只能读而不能写对象。如果元对象是一个常量的话就只能使用 const 型的迭代器。

一般来说我们用 auto 来推断即可不用关心迭代器的类型。

但凡使用了迭代器的循环体，都不要向迭代器所属的容器添加元素，会带来意想不到的错误。

⑤ 对迭代器的解引用：

迭代器是类似与指针的一种类型，所以要通过迭代器访问对象需要对迭代器进行解引用操作。如果这个对象恰好是一个对象的话，需要访问其数据成员或者成员函数，要如下写。

Eg：(*it).empty()

　　*it.empty()

前面的括号是必不可少的，需要先解引用再访问成员，如果没有括号的话就会报错，c++提供了简单的写法：箭头。

　　It->empt()

这种写法和指向对象的指针访问对象成员的写法是一致的，但是需要注意迭代器并非指针。

⑥ 迭代器的运算：

主要是加减运算，可以加减一个整数，表示迭代器移动，另外两个迭代器也可以比较或者作减法，作减法的结果是 diffreence_type 类型的带符号整型数。表示从

右侧迭代器要往前移动多少才能到左侧迭代器。（it1-it2）

注意：迭代器不支持加法操作!!

3.5：数组

简介：

① 数组不同于 vector，需通过位置访问，且数组大小固定，相对于 vector 来说损失了一些灵活性，但是在某些特殊应用下性能良好。

② 不允许通过 auto 关键字由初始值列表对端数组类型。

初始化：

① 数组不允许拷贝和赋值，不能将数组的内容拷贝给其他数组作为其初始值，也不能用数组为其他数组赋值。（某些编译器支持数组赋值）

数组和指针：

① 指针和数组有很大的关系，实际上，数组名就是指针，指向数组的第一个元素。

② 另外，指针也是迭代器，可能使用起来没有迭代器那么简单。

```
Int a[10]={};
Int *p=&a[10];      //这里是不存在的，但是可以取到这个指针，尾后指针
for(int *b=a;b!=p;b++)
cout<<b[i]<<endl;
```

这种方法也可以把数组中的元素都取出来。

③ 新标准引入了 begin 和 end 函数，但是数组毕竟不是一个类，所以不是以成员函数的形式引入的，在具体使用的时候，和 vector 及 string 的用法稍有区别。

```
Int a[]={12, 3, 3,3,4,5,3,5,5,4,5,6,4 };
Int *beg=begin(a);
Int *end=end(a);
```

这种方法可以得到起始指针和尾后指针。

④ 和迭代器一样，两个指针也可进行相减运算，得到的结果是 ptrdiff_t 类型，这是定义在标准库中的机器相关的类型，是一种带符号整型。只要两个指针指向同一个数组的元素，或者指向该数组的末尾元素的下一位，那么就能利用关系运算符进行比较，否则则不能使用关系比较符。

⑤ 数组下标和指针：实际上下标的实质就是指针，在进行下标操作时，编译器会默认转换为指针进行操作。

Eg：int a[5]={1,2,3,4,5};

a[2];

相当于：

```
Int *p1=a;
*(p1+2)             //这两者是等价的。
```

另外，内置的下标运算并不是无符号类型，可以有负的。

```
Eg: int *p2=&a[2];
    Int i=p2[2];     //这个实际上是 a[4];
    Int j=p2[-2];    //这个实际上是 a[0];
```

虽然介绍了数组，但是在现代 c++编程中，尽量使用标准库中的容器而非数组或者 c 风格的字符串，这些操作太过底层容易出错。