chapter16 string类和标准模板库

1）string类

（1）string类的输入

【1】cin >> str;
暂停方式：
不断读取，直到遇见空白字符（空格、换行符、制表符）并将其留在输入队列中

【2】getline(cin,str);
会自动调整目标string对象的大小，使之刚好能存储输入的字符。不需要指定读取多少字符的数值参数
暂停方式：
[1] 到达文件尾：输入流的eofbit将被设置，意味着fail()与eof()都将返回true
[2] 遇到分界字符（默认为\n）：此时将把分界字符从输入流中删除，但不存储它
[3] 读取的字符数到达最大允许值：将设置输入流的failbit，这意味着fail()将返回true

（2）使用字符串

【1】比较字符串（字典序）：</=/>即可
【2】确定字符串长度：str.length()；str.size()
【3】内存块分配：

string 库提供了很多其他的工具，包括完成下述功能的函数：删除字符串的部分或全部内容、用一个字符串的部分或全部内容替换另一个字符串的部分或全部内容、将数据插入到字符串中或删除字符串中的数据、将一个字符串的部分或全部内容与另一个字符串的部分或全部内容进行比较、从字符串中提取子字符串、将一个字符串中的内容复制到另一个字符串中、交换两个字符串的内容。这些函数中的大多数都被重载，以便能够同时处理 C-风格字符串和 string 对象。附录 F 简要地介绍了 string 库中的函数。

首先来看自动调整大小的功能。在程序清单 16.3 中，每当程序将一个字母附加到字符串末尾时将发生什么呢？不能仅仅将已有的字符串加大，因为相邻的内存可能被占用了。因此，可能需要分配一个新的内存块，并将原来的内容复制到新的内存单元中。如果执行大量这样的操作，效率将非常低，因此很多 C++ 实现分配一个比实际字符串大的内存块，为字符串提供了增大空间。然而，如果字符串不断增大，超过了内存块的大小，程序将分配一个大小为原来两倍的新内存块，以提供足够的增大空间，避免不断地分配新的内存块。方法 capacity( )返回当前分配给字符串的内存块的大小，而 reserve( )方法让您能够请求内存块的最小长度。程序清单 16.4 是一个使用这些方法的示例。

2）智能指针模板类

[1] 三种智能指针模板：auto_ptr/ unique_ptr/ shared_ptr
[2] 可将new获得的(直接或间接)的地址赋给这种对象，当智能指针过期时，其析构函数将使用delete来释放内存
[3] 使用格式：

```
(1) 包含头文件<memory>
#include<memory>

(2) 调用方式：
auto_ptr<double> pd(new double);
auto_ptr<string> pd(new string);

unique_ptr<double> pd(new double);
shared_ptr<double> pd(new double);
```

```
(3)不需要使用delete语句
```

ps：unique_ptr 优于 auto_ptr

3）标准模板库(standard template library)

1）基础：review Vector即可

2）基于范围的for循环：

> （1）for循环中，括号内的代码声明一个类型与容器存储的内容相同的变量，然后指出容器的名称

```
vector<Review> books;
for(auto x : books) ShowReview(x);

根据books的类型(vector<Review>)，编译器将自动推断出x的类型为Review，而循环将依次将books中的每个Review对象传递给 ShowReview()
```

> （2）如果想修改容器的内容，需要在括号内加上 *引用参数*

```
vector<Review> books;
void Change(Review& r) { r.rating++; }
......
for(auto& x : books) Change(x);
```

4）泛式编程（generic programming）

（1）begin( ) 返回指向第一个元素的迭代器，end( ) 返回一个指向超尾位置的迭代器即可

（2）auto简化迭代器表示：

```
vector<double>:: iterator pr;
for(pr=scores.begin(); pr!=scores.end(); pr++)
        cout << *pr <<endl;
-------------------------------------------------
for(auto pr=scores.begin(); pr!=scores.end(); pr++)
        cout << *pr <<endl;
```

（3）迭代器类型：

【1】输入迭代器：

> [1]解引用能够读取容器内的值，但不一定能让程序修改值
> [2]必须能够访问容器内所有值，通过++运算符实现
> [3]是单向迭代器，可以++，但不能倒退

【2】输出迭代器：

> [1]解引用能让程序修改容器值，而不能读取（eg：cout可以修改发送到显示器的字符流，却不能读取屏幕上的内容）

【3】正向迭代器：

> [1]仅使用++运算符来遍历容器
> [2]将正向迭代器递增后，仍可以对前面的迭代器解引用，并可以得到相同的值

【4】双向迭代器：

> 具备正向迭代器的全部特性，同时支持(前缀、后缀)递减运算符

【5】随机访问迭代器：

> 能够直接跳到容器中的任何一个元素
> 具备特性：双向迭代器+支持随机访问的操作+对元素进行排序的关系运算符

（4）迭代器层次结构：

表 16.4　　迭代器性能

迭代器功能	输　入	输　出	正　向	双　向	随机访问
解除引用读取	有	无	有	有	有
解除引用写入	无	有	有	有	有
固定和可重复排序	无	无	有	有	有
++i　i++	有	有	有	有	有
−−i　i−−	无	无	无	有	有
i[n]	无	无	无	无	有
i + n	无	无	无	无	有
i - n	无	无	无	无	有
i + = n	无	无	无	无	有
i − = n	无	无	无	无	有

（5）迭代器中的概念、改进和模型：

【1】概念（concept）：描述迭代器需要满足的一系列要求
【2】改进（refinement）：概念上的继承（例如双向迭代器是对单向迭代器的继承）
【3】模型（model）：概念的具体实现

（6）容器种类：

【1】容器概念：具有名称（eg:容器、序列容器、关联容器）的通用类别
【2】容器类型：用于创建具体容器对象的模板（deque、list、queue、priority_queue...）
【3】基本容器特征：

表 16.5　　一些基本的容器特征

表　达　式	返 回 类 型	说　明	复　杂　度
X :: iterator	指向 T 的迭代器类型	满足正向迭代器要求的任何迭代器	编译时间
X :: value_type	T	T 的类型	编译时间
X u;		创建一个名为 u 的空容器	固定
X( );		创建一个匿名的空容器	固定
X u(a);		调用复制构造函数后 u == a	线性
X u = a;		作用同 X u(a);	线性
r = a;	X&	调用赋值运算符后 r == a	线性
(&a)->~X( )	void	对容器中每个元素应用析构函数	线性
a.begin( )	迭代器	返回指向容器第一个元素的迭代器	固定
a.end( )	迭代器	返回超尾值迭代器	固定
a.size( )	无符号整型	返回元素个数，等价于 a.end( )− a.begin( )	固定
a.swap(b)	void	交换 a 和 b 的内容	固定
a = = b	可转换为 bool	如果 a 和 b 的长度相同，且 a 中每个元素都等于（= = 为真）b 中相应的元素，则为真	线性
a != b	可转换为 bool	返回!(a= =b)	线性

【4】时间复杂度解释：
1）编译时间：操作将在编译时执行，执行时间为0

2）固定时间：操作将在运行阶段进行，但独立于对象中的元素个数
3）线性时间：时间与元素数目成正比

【5】c++11 新增基本容器要求：

表 16.6 C++11 新增的基本容器要求

表 达 式	返 回 类 型	说 明	复 杂 度
X u(rv);		调用移动构造函数后，u 的值与 rv 的原始值相同	线性
X u = rv;		作用同 X u(rv);	
a = rv;	X&	调用移动赋值运算符后，u 的值与 rv 的原始值相同	线性
a.cbegin( )	const_iterator	返回指向容器第一个元素的 const 迭代器	固定
a.cend( )	const_iterator	返回超尾值 const 迭代器	固定

【6】序列：元素按照严格的线性顺序排列

表 16.7 序列的要求

表 达 式	返 回 类 型	说 明
X a(n, t);		声明一个名为 a 的由 n 个 t 值组成的序列
X(n, t)		创建一个由 n 个 t 值组成的匿名序列
X a(i, j)		声明一个名为 a 的序列，并将其初始化为区间[i，j)的内容
X(i, j)		创建一个匿名序列，并将其初始化为区间[i，j)的内容
a.insert(p, t)	迭代器	将 t 插入到 p 的前面
a.insert(p, n, t)	void	将 n 个 t 插入到 p 的前面
a.insert(p, i, j)	void	将区间[i，j)中的元素插入到 p 的前面
a.erase(p)	迭代器	删除 p 指向的元素
a.erase(p, q)	迭代器	删除区间[p，q)中的元素
a.clear( )	void	等价于 erase(begin( ), end( ))

表 16.8 序列的可选要求

表 达 式	返 回 类 型	含 义	容 器
a.front( )	T&	*a.begin( )	vector、list、deque
a.back( )	T&	*--a.end( )	vector、list、deque
a.push_front(t)	void	a.insert(a.begin( ), t)	list、deque
a.push_back(t)	void	a.insert(a.end( ), t)	vector、list、deque
a.pop_front(t)	void	a.erase(a.begin( ))	list、deque
a.pop_back(t)	void	a.erase(--a.end( ))	vector、list、deque
a[n]	T&	*(a.begin( )+ n)	vector、deque
a.at(t)	T&	*(a.begin( )+ n)	vector、deque

（7）常见STL容器使用：

> 略

（8）关联容器（associative container）：

【1】关联容器将键值匹配，形成 键-值 对，并使用键来查找值
【2】它提供了对元素的快速访问。与序列相似，关联容器允许插入新元素，但是不能指定插入位置
【3】set（头文件 #include < set > ）：值 与 键值 的类型相同，键是唯一的
【4】multiset（头文件 #include < set > ）：值 与 键值 的类型相同，一键可以对多个值
【5】map（头文件 #include < map >）：值 与 键值 的类型可以不同，键是唯一的
【6】multimap（头文件 #include < map >）:值 与 键值 的类型可以不同，一键可以对多个值

> 具体使用方法略

【7】无序关联对象：unordered_set/ unordered_multiset/ unordered_map/ unordered_multimap

5）函数对象

（1）函数符 (functor)：可以以函数方式与()结合使用的任意对象
【1】生成器（generator）：不使用参数就可以调用的函数符
【2】一元函数（unary function）：一个参数就可以调用的函数符
【3】二元函数（binary function）：两个参数可以调用的函数符
【4】谓词（predicate）：返回值为bool值的一元函数
【5】二元谓词（binary predicate）：返回值为bool值的二元函数

（2）自适应函数符和函数适配器：
特征：它携带了标识参数类型和返回类型的typedef成员，这些成员分别是result_type、first_argument_type、second_argument_type
目的：函数适配器对象可以使用函数对象，并认为存在这些typedef成员
例如：plus< int >对象的返回类型被标识为 plus< int >::result_type，这是int的typedef

（3）使用STL：略

6）其他库

（1）complex库：复数模板
（2）random库：提供随机数