16 string类和标准模板库

前言

- 学习string类，更深入地讨论它
- “智能指针”模板类，它们让管理动态内存更容易
- 标准模板库(STL)，它是一组用于处理各种容器对象的模板
- STL演示了一种编程模式–泛型编程

string类

1 string类

- 由string.h(C)和cstring(C++)支持C库字符串函数，但不支持string类

- string类
  - 头文件string
  - 若干构造函数，将字符串赋给变量
  - 合并字符串、比较字符串和访问各个元素的重载运算符
  - 以及用于在字符串中查找字符和子字符串的工具等

1.1 构造字符串

- string的构造函数
- 表格中使用缩写NBTS（null-terminated string)来表示以空字符结束的字符串(传统的C字符串）

构 造 函 数	描 述
string(const char * s)	将string对象初始化为s指向的NBTS
string(size_type n, char c)	创建一个包含n个元素的string对象，其中每个元素都被初始化为字符c
string(const string & str)	将一个string对象初始化为string对象str(复制构造函数)
string( )	创建一个默认的sting对象，长度为0(默认构造函数)
string(const char * s, size_type n)	将string对象初始化为s指向的NBTS的前n个字符，即使超过了NBTS结尾
template<class Iter> string(Iter begin, Iter end)	将string对象初始化为区间[begin，end)内的字符，其中begin和end的行为就像指针，用于指定位置，范围包括begin在内，但不包括end
string(const string & str, string size_type pos = 0, size_type n = npos)	将一个string对象初始化为对象str中从位置pos开始到结尾的字符，或从位置pos开始的n个字符
string(string && str) noexcept	这是C++11新增的，它将一个string对象初始化为string对象str，并可能修改str(移动构造函数)
string(initializer_list<char> il)	这是C++11新增的，它将一个string对象初始化为初始化列表il中的字符

1.2 string类输入

- C-风格字符串3种输入方式：

```
char info[100];
cin»info;                 // read a word
cin.getline(info, 100); // read a line, discard \n
cin.get(info, 100);     // read a line, leave \n in queue
```

- string 对象两种方式：

```
string stuff;
cin » stuff;           // read a word
getline(cin, stuff); // read a line, discard \n
```

- string版本的getline()自动调整目标string 对象的大小，使之刚好能够存储输入的字符
  - 到达文件尾
  - 遇到分界字符
  - 读取字符数达到最大允许值

P16.2 strfile.cpp

1.3 使用字符串

- 比较字符串
  - <, <=, ==, >, >=, !=
- 确定字符串长度
  - size(), length()
- 搜索给定子字符串或字符
  - find()
  - rfind(), find_first_of(), find_first_not_of()
- find_last_not_of()

- P16.3 hangman.cpp

方 法 原 型	描 述
size_type find(const string & str, size_type pos = 0)const	从字符串的pos位置开始，查找子字符串str。如果找到，则返回该子字符串首次出现时其首字符的索引；否则，返回string :: npos
size_type find(const char * s, size_type pos = 0)const	从字符串的pos位置开始，查找子字符串s。如果找到，则返回该子字符串首次出现时其首字符的索引否则，返回string :: npos
size_type find(const char * s, size_type pos = 0, size_type n	从字符串的pos位置开始，查找s的前n个字符组成的子字符串。如果找到，则返回该子字符串首次出现时其首字符的索引；否则，返回string :: npos
size_type find(char ch, size_type pos = 0)const	从字符串的pos位置开始，查找字符ch。如果找到，则返回该字符首次出现的位置；否则，返回string :: npos

1.4 string还提供了哪些功能

- 删除字符串的部分或全部内容
- 字符串取代
- 数据插入或删除
- 字符串内容比较
- 提取子字符串
- 字符串内容复制
- 交换字符串内容

P16.4 str2.cpp

1.5 字符串种类

- string库基于模板类
  - 可使用wchar_t、char16_t、char32_t、char类型字符串，也可开发相关类

```
typedef basic_string<char> string;
typedef basic_string<wchar_t> wstring;
typedef basic_string<char16_t> u16string; // C++11
typedef basic_string<char32_t> u32string; // C++11
```

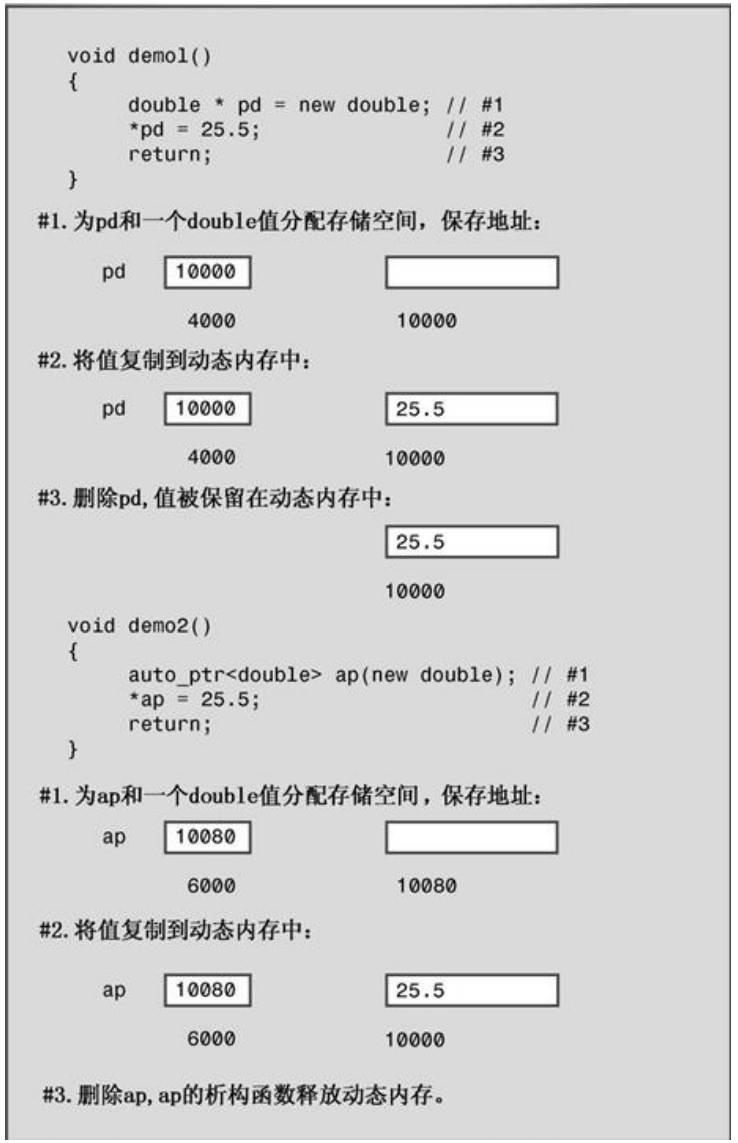
2 智能指针模板类

- 智能指针是行为类似于指针的类对象
  - 目标：不要delete
- 可帮助管理动态内存分配
  - auto_ptr(不推荐)
  - unique_ptr
  - shared_ptr

2.1 使用智能指针

P16.5 smrtptrs.cpp，头文件memory

```
1. class Report{
2. private:
3.     std::string str;
4. public:
5.     Report(const std::string s) : str(s) {}
6.     void comment() const { std::cout << str << "\n"; }
7. };
8. int main(){
9.     {   std::auto_ptr<Report> ps(new Report("auto_ptr"));
10.        ps->comment(); // use -> to invoke a member function
11.    }
12.    {   std::shared_ptr<Report> ps(new Report("shared_ptr"));
13.        ps->comment();
14.    }
15.    {   std::unique_ptr<Report> ps(new Report("unique_ptr"));
16.        ps->comment();
17.    }
18. }
```



2.2 有关智能指针的注意事项

- 不推荐使用auto_ptr

```
auto_ptr<string> ps(new string("I reigned lonely as a cloud."));
auto_ptr<string> vocation;
vocation = ps;
```

- 有时不适合使用auto_ptr，可以用shared_ptr代替
- unique_ptr
  - 对于特定的对象，只能有一个智能指针可拥有它
- shared_ptr
  - 跟踪引用特定对象的智能指针数，引用计数（reference counting）
    - 例如，赋值时计数将加1，指针过期时，计数将减 1

2.3 unique_ptr为何优于auto_ptr

- 程序在编译环节就会出错，而不会在运行时

```
unique_ptr<string> pu1(new string "Hi ho!");
unique_ptr<string> pu2;
pu2 = pu1; //#1 not allowed
unique_ptr<string> pu3;
pu3 = unique_ptr<string>(new string "Yo!"); //#2 allowed
```

2.4 选择智能指针

- 多个指针指向同一对象：shared_ptr
  - 很多STL 算法都支持复制和赋值操作，这些操作可用于shared_ptr，但不能用于unique_ptr（编译器发出警告)和auto_ptr（行为不确定）
- 如果程序不需要多个指向同一个对象的指针，则可使用unique_ptr

```
unique_ptr<int> make_int(int n) { return unique_ptr<int>(new int(n)); }
void show(unique_ptr<int> &pi){ // pass by reference
    cout << *a << ' ';
}
int main(){
    vector<unique_ptr<int>> vp(size);
    for (int i = 0; i < vp.size(); i++)
        vp[i] = make_int(rand() % 1000);   // copy temporary unique_ptr
    vp.push_back(make_int(rand() % 1000)); // ok because arg is temporary
    for_each(vp.begin(), vp.end(), show);  // use for_each()
}
```

- 其中的push_back()调用没有问题，因为它返回一个临时uique_ptr，该uique_ptr被赋给vp中的一个uique_ptr
  - 如果按值而不是按引用给show()传递对象， for_each语句将非法，因为这将导致使用一个来自vp 的非临时uique_ptr初始化pi，而这是不允许的

标准模板库

3 标准模板库

- STL提供一组表示容器、迭代器、函数对象和算法的模板
  - 容器，是一个与数组类似的单元，可以存储若干个值
    - ST 容器是同质的，即存储的数据类型相同
  - 算法，是完成特定任务(如对数组进行排序或在链表中查找特定值)的方法
  - 迭代器，能够用来遍历容器的对象，与能够遍历数组的指针类似，是广义指针
  - 函数对象，是类似于函数的对象，可以是类对象或函数指针(包括函数名，因为函数名被用作指针)
  - 算法，STL使得能够构造各种容器和执行各种操作(包括搜索、排序和随机排列)

- STL不是面向对象，而是泛型编程

3.1 模板类vector

P16.7 vect1 .cpp

```
#include <vector>
using namespace std;
vector<int> ratings(5); // a vector of 5 ints
int n;
cin >> n;
vector<double> scores(n); // a vector of n doubles
```

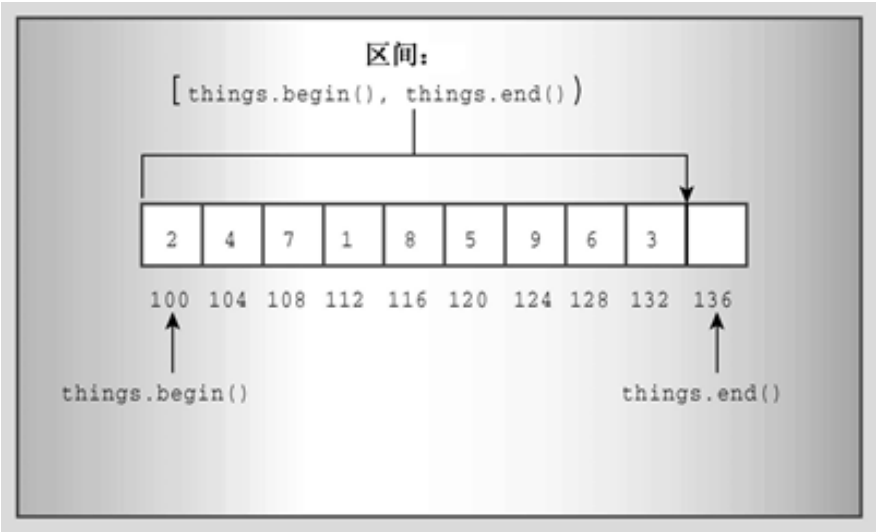
//由于运算符[ ]被重载，因此创建vector对象后，可以使用通常的数组表示法来访问各个元素：

```
ratings[0] = 9;
for (int i = 0; i < n; i++)
    cout << scores[i] << endl;
```

3.2 可对矢量执行的操作

P16.8 vect2.cpp

- size(), swap(), begin(), end(), erase(), insert(), push_back()
- 迭代器--广义指针
  - 它可以是指针
  - 也可以是一个可对其执行类似指针的操作的对象
    - 如，解除引用(如 operator*()) ，递增(如 operator++())




```
vector<double>::iterator pd = scores.begin();
auto pd = scores.begin(); // C++11 automatic

for (pd = scores.begin(); pd != scores.end(); pd++)
    cout << *pd << endl;
```

3.3 对向量可执行的其他操作

P16.9 vect3.cpp

- STL从更广泛的角度定义非成员函数执行相关操作，节省大量重复工作
  - for_each()
  - random_shuffer()
  - sort()

3.4 基于范围的for循环

- `for_each(books.begin(), books.end(), ShowReview);`

泛型编程

4 泛型编程

- 与面向对象编程存在共同点，是抽象和可重用，但理念不同
- 泛型编程更关注算法，旨在编写独立于数据类型的代码
  - 模板：完成通用程序的工具
- 模板使算法独立于存储的数据类型

4.1 为何使用迭代器

- 迭代器使算法独立于使用的容器类型，是遍历容器中的值的通用表示
- 要实现 find 函数，迭代器应具备特征
  - 对*p进行定义
  - 对p=q进行定义
  - 对p==q和p!=q进行定义
  - 遍历所有元素，p++和++p
- C++将operator++作为前缀版本，将operator++(int)作为后缀版本
  - 其中的参数不会被用到，所以不必指定其名称
- 其begin()返回一个指向第一个元素的迭代器， end()返回一个指向超尾位置
- 使用C++11新增的自动类型推断可进一步简化：对于矢量或列表，都可使用如下代码：

```
for (auto pr = scores.begin(); pr != scores.end(); pr++)
    cout << *pr << endl;
```

4.2 迭代器类型

- 输入迭代器
- 输出迭代器
- 正向迭代器
- 双向迭代器
- 随机访问迭代器

4.3 迭代器层次结构

- 迭代器类型形成了一个层次结构
  - 正向迭代器具有输入迭代器和输出迭代器的全部功能，同时还有自己的功能
  - 双向迭代器具有正向迭代器的全部功能，同时还有自己的功能
  - 随机访问迭代器具有正向迭代器的全部功能，同时还有自己的功能

4.4 概念、改进和模型

- 迭代器是要求，而非类型，STL算法可以使用任何满足其要求的迭代器实现
- STL 文献使用术语概念用来描述一系列要求
- 概念可以具有类似继承的关系，用改进来表示概念上的继承
- 概念的具体实现被称为模型

1. 将指针用作迭代器
2. 其他有用的迭代器

P16.10 copyit.cpp P16.11 inserts.cpp

4.5 容器种类

- STL具有容器概念和容器类型。
  - 概念是具有名称的通用类别，容器概念指定了一系列必须满足的要求【抽象的】
  - 容器类型是创建具体容器对象的模板【C++ 里实现的具体的】
- 11 个容器类型
  - vector, deque, list, queue, priority_queue, stack等
- 容器概念
  - 容器概念指定了所有STL 容器类都必须满足的一系列要求。
- C++11新增的容器要求
- 序列
  - 序列概念增加了迭代器至少是正向迭代器这样的要求，保证了元素将按特定顺序排列，不会在两次迭代之间发生变化

4.5 容器种类

- vector
  - vector 是数组的一种类表示
  - 提供了自动内存管理功能，可以动态地改变 vector 对象的长度，并随着元素的添加和删除而增大和缩小
  - 提供了对元素的随机访问。在尾部添加和删除元素的时间是固定的，但在头部或中间插入和删除元素的复杂度为线性时间
- deque
  - 表示双端队列（double-ended queue）
- list
  - 表示双向链表
- queue
  - queue 模板的限制比 deque 更多。它不仅不允许随机访问队列元素，甚至不允许遍历队列
- priority_queue
  - 在priority_queue中，最大的元素被移到队首
  - 内部区别在于，默认的底层类是vector。可以修改用于确定哪个元素放到队首的比较方式
- Stack
  - 给底层类(默认情况下为 vector）提供了典型的栈接口。

4.6 关联容器

- 关联容器(associative container)是对容器概念的另一个改进
  - 关联容器将值与键关联在一起，并使用键来查找值
  - 基本元素：<键，值>
- 关联容器的优点
  - 对元素快速访问
  - 允许插入，但不能指定位置
- 关联容器一般用树实现
- 4种关联容器

  set：其值类型与键相同，键是唯一的，这意味着集合中不会有多个相同的键

  multiset：可能有多个值的键相同

  map：值与键的类型不同，键是唯一的，每个键只对应一个值

  multimap：一个键可以与多个值相关联

P16.13 setops.cpp P16.14 multimap.cpp

4.7 无序关联容器

- 无序关联容器是对容器概念的另一种改进
  - 基于哈希表(提高添加、删除和查找效率)
- 4种无序关联容器
  - unordered_set, unordered_mutiset, unorderd_map, unordered_multimap

函数对象【不做要求】

5 函数对象

- 函数对象(也叫函数符，functor)
  - 函数符，可以以函数方式与( )结合使用的任意对象
  - 包括函数名、指向函数的指针和重载了( )运算符的类对象(即定义了函数operator( )的类)
    - 形式上像把对象当函数用

```
1.  class Linear{
2.  private:
3.      double slope;
4.      double y0;
5.  public:
6.      Linear(double sl_ = 1, double y_ = 0) : slope(sl_), y0(y_) {}
7.      double operator()(double x) { return y0 + slope * x; }
8.  };
9.  Linear f1;
10. Linear f2(2.5, 10.0);
11. double y1 = f1(12.5); // right-hand side is f1.operator()(12.5)
12. double y2 = f2(0.4);
```

5.1 函数符概念

- STL定义了函数符概念
  - 生成器(generator) 是不用参数就可以调用的函数符
  - 一元函数(unary function) 是用一个参数可以调用的函数符
  - 二元函数(binary function) 是用两个参数可以调用的函数符

P16.15 functor.cpp

5.2 预定义的函数符

- STL定义了多个基本函数符，为支持将函数作为参数的STL函数
- 对于所有内置的算术运算符、关系运算符和逻辑运算符都有等价的STL函数符

5.3 自适应函数符和函数适配器

- 函数符自适应：携带标识参数类型和返回类型的typedef成员
- 其意义在于函数适配器对象可以使用函数对象，并承认typedef成员

P16.16 funadap.cpp

算法

6 算法

- 算法函数设计，有两个主要的通用部分
  - 使用模板来提供泛型
  - 使用迭代器来提供访问容器中数据的通用表示

- 统一的容器设计，使得不同类型的容器之间具有明显关系
  - 如，可使用copy()将常规数组中的值复制到vector对象中，将vector对象中的值复制到list对象中，将list对象中的值复制到set对象中
  - 可以用＝＝来比较不同类型的容器，如deque和vector
  - 之所以能够这样做，是因为容器重载的＝＝运算符使用迭代器来比较内容
    - 因此如果deque对象和vector对象的内容相同，并且排列顺序也相同，则它们是相等的

6.1 算法组

- 非修改式序列操作
  - 非修改式序列操作对区间中的每个元素进行操作。这些操作不修改容器的内容。如，find()和for_each()
- 修改式序列操作
  - 可以修改值，也可以修改值的排列顺序。如，transform() 、random_shuffle()和copy()
- 排序和相关操作
  - 包括多个排序函数(包括sort()) 和其他各种函数，包括集合操作
- 通用数字运算
  - 数字操作包括将区间的内容累积、计算两个容器的内部乘积、计算小计、计算相邻对象差的函数
    - 通常，这些都是数组的操作特性，因此vector是最有可能使用这些操作的容器

- 前3组在头文件algorithm 中描述，第4组专用于数值数据的，头文件为numeric

6.2 算法的通用特征

- STL函数使用迭代器和迭代器区间
- 根据结果放置位置进行分类
  - 就地算法(sort函数)
  - 复制算法(copy函数)，复制版本的名称将以_copy 结尾

```
template <class InputIterator, class OutputIterator, class T>
OutputIterator replace_copy(InputIterator first, InputIterator last,
                            OutputIterator result, const T &old_value, const T &new_value);
```

- 根据结果执行操作的函数版本，通常以_if结尾
- 另一个常见的变体：有些函数，根据将函数应用于容器元素得到的结果来执行操作。这些版本的名称通常以_if结尾

```
template <class ForwardIterator, class Predicate class T>
void replace_if(ForwardIterator first, ForwardIterator last,
                Predicate pred, const T &new_value);
```

6.3 STL和string类

- string类不是STL组成部分，但考虑到了STL
- 包含begin(), end(), rbegin(), rend(), 可以使用STL接口

- P16.17 strgst1 .cpp
  - 用STL 显示了使用一个词的字母可以得到的所有排列组合

6.4 函数和容器方法

- 通常选择STL方法(STL容器里的方法)优于STL函数(函数)
  - 容器里的方法，更适合特定容器
  - 作为成员函数，它可以使用模板类的内存管理工具，从而在需要时调整容器的长度。

P16.18 listrmv.cpp

6.5 使用STL

- STL是一个库，组成部分协同工作
- STL组件是工具，创建其他工具的基本部件

P16.19 usealgo.cpp

其他库

7 其他库

- C++提供了其他一些更为专用的类库
- complex, valarray

7.1 vector、valarray和array

P16.20 valvect.cpp

- vector支持面向容器的操作，是一个容器类和算法系统的一部分
- valarray是面向数值计算的，不是STL的一部分
  - 没有push_back()和inset()方法，但为很多数学运算提供了一个简单、直观的接口
- array表示长度固定的数组，为代替内置数组设置
  - 长度固定的数组，因此不支持push_back()和insert()
  - 但提供了多个STL方法，包括begin() 、end() 、rbegin()和rend()，这使得很容易将STL算法用于array对象

7.2 模板initializer_list(C++11)

7.3 使用initializer_list

P16.22 ilist.cpp

8 总结

- C++提供了一组功能强大的库，这些库提供了很多常见编程问题的解决方案以及简化其他问题的工具
- string类为将字符串作为对象来处理提供了一种方便的方法
- C++11新增的share_ptr和unique_ptr等智能指针模板管理由new分配的内存
- STL 是一个容器类模板、迭代器类模板、函数对象模板和算法函数模板的集合，它们的设计是一致的，都是基于泛型编程原则的