附录A 其他性能

在写这本书时，C++的标准还没有制定出来。虽然实际上所有这些特征最终都会被加入到这个标准中，但有些并没有在所有的编译器中出现。这个附录中简单地介绍了一些其他特征，这些应该在编译器中（或在编译器的未来版本中）去查找。

A.1 bool、true、false

实际上每个人都在用布尔形变量，而且每个人定义它们的方式都不相同[1]，有些人用枚举，另一些人用typedef。 typedef是一个特殊的问题，因为不能重载它（一个typedef对于一个int还是一个int），也不能用它初始化一个唯一的模板。

在标准库中，可能为bool类型创建了一个类，但这也不能很好地工作，因为我们只能有了一个自动类型转换运算符，而没有解决重载问题。

对于这样一个有用的类型，最好的方法是把它建在语言内部。一个 bool型有两种状态，它们由内部常量true（它转化成一个整数1）和false(它转化成一个整数0)表示，这三个名字都是关键词，另外对部分语言成分作了修改：

成 分	bool型的用法
&&‖！	取bool型参数，返回bool值
< > <= >= == !=	产生bool型结果
if ,for while,do	条件表达式转换为一个bool值
?:	第一个操作数转换为bool值

因为已有的许多代码常用一个int表示一个标志，编译器将一个int隐式转换成一个bool型。理想情况下，编译器将给我们一个警告，以建议我们改正这种情况。

一种俗话称“低劣的编程风格”的情况是用 ++来把一个标志置为true。这是允许的，但不赞成这样做，这意味着在将来某个时候它会变成不合法的。这个问题与 enum的增运算一样：我们正在做从bool 型转化成int型的隐式类型转换，增加这个值（可能越出一般的 bool值0~1的范围），然后隐式地映射回来。

在必要时指针也可以自动转换成bool型的。

A.2 新的包含格式

随着C++的不断发展，不同的编译器开发商选择了不同的文件扩展名。另外，不同的操作系统对文件名有不同的限制，特别是在文件名的长度上。为了适应各种不同的情况， C++标准采纳了一个新的格式，它允许文件名突破那很不好的八个字符的限制，并且取消了扩展名。比如，包含IOSTREAM.H就变成了：

```
# include  <iostream>
```

[1] 见“Josée Lajoie，"The new cast notation and the bool data type”C++报告，1994年9月。

解释器按特定的编译器和操作系统去实现文件的包含，必要时缩短文件名并加上一个扩展名。如果我们想在编译器开发商支持这一特性之前使用这一风格的包含文件，也可以把开发商提供给我们的头文件拷贝到不带扩展名的文件中去。

所有从标准C中继承来的库在我们包含它们时还是用 .h作为扩展名，这使读者很容易从我们使用的C++库中识别出C库。

A.3 标准C++库

标准的C++不仅包含了全部的标准C库（做了一点小的增补和改动，以支持安全类型），而且还增加了一些它自己的库。这些库比标准的 C库功能更强，从中获得的益处与从 C向C++转变获得的益处类似。对这些库的最好的参考文献就是标准本身（写这本书时还只能得到非正式版），可以在 Internet或BBS上找到它们。

输入输出流库已在本书第6章做了介绍，下面简要介绍一下C++中其他常用的库：

语言支持：包括继承到本语言中来的成分，像 <climits>和<cfloat>中的实现限制；<new>中的动态内存声明，例如bad_alloc（当我们超出内存范围时抛出的异常）和 set_new_handler；用于RTTI的<typeinfo>头文件和声明了terminate()和unexpected() 函数的<exception>头文件。

诊断库：一些组件，C++程序能够用以发现和报告错误。<stdexcept>头文件声明了标准异常类，<cassert>与C中ASSERT.H的作用相同。

通用实用库：这些组件被标准C++库的其他部分使用，我们也可以在我们的程序中使用它们。包括运算符!=、>、<=和>=（为防止多余的定义）的模板化版本、带 tuple模板函数的pair模板类、支持STL（在本附录的下一节中介绍）的一套函数对象和与 STL一起使用的能使我们可以很容易地修改存储分配机制的内存分配函数。

字符串库：字符串类可能是我们曾经见过的最完整的字符串处理工具。我们在 C中用了数行代码来完成的工作都可以用字符串类中的一个成员函数来代替，包括 append()、assign()、insert()、remove()、replace()、resize()、copy()、find()、rfind（）、find_first_of()、find_last_of()、find_first_not_of()、find_last_not_of()、substr()和compare()。此外还重载了运算符=、+=和[]使这些运算更直观。另外有一个“宽字符”的 wstring类，用来支持国际字符集。string和wstring (在<string>中声明，不要和C中的STRING.H弄混)都是从一个叫basic_string的通用模板类中产生的。注意字符串类和输入输出流已经无缝地结合在一起了，所以我们甚至无需用strstream了（也不用担心第6章中描述的有关内存管理了）。

本地化库：这个库用来调整字符集以使我们的程序能在不同国家使用，包括货币、数字、日期、时间等等。

包容器库：这包括标准的模板库（在本附录的下一节中介绍）以及 <bits>和<bitstring>中的bits类和bit_string类。bits和bit_string都更完整地实现了第5章中介绍的位向量的概念。bits模板产生一个固定大小的位数组，可以用所有的位运算符对其运算，同时包含 set() 、reset() 、count()、length()、test()、any() 和none()等成员函数。另外还有几个转换运算符 to_ushort()、to_ulong()和to_string()。

bit_string则相反，它是一个动态长度的位数组，它不仅包含 bits同样的运算符，还包含一些其他运算使它看上去像string类。bits和bit_string在位的权重上有一个根本的区别：对于 bits，最右的位（第 0位）是最不重要的位，但在 bit_string中，最右的位是最有意义的位。 bits和bit_string之间不能互相转换。我们可以把 bits用于一组节省空间的开关变量，而用 bit_string来管理二进制值的数组（如像表素）。

循环子库：包括用于STL的工具（下一节中介绍）、流及流缓冲区。

算法库：这些都是一些模板函数，用循环子来完成 STL包容器上的运算，它包括adjacent_find、prev_permutation、binary_search、push_heap、copy、random_shuffle、copy_backward、remove、count、remove_copy、count_if、remove_copy_if、equal,、remove_if、equal_range、replace、fill、replace_copy、fill_n、replace_copy_if、find、replace_if、find_if、reverse、for_each、reverse_copy、generate、rotate、generate_n、rotate_copy、includes、search、inplace_merge、set_difference、lexicographical_compare、set_intersection、lower_bound、set_symmetric_difference、make_heap、set_union、max、sort、max_element、sort_heap、merge、stable_partition、min、stable_sort、min_element、swap、mismatch、swap_ranges、next_permutation、transform、nth_element、unique、partial_sort、unique_copy、partial_sort_copy、upper_bound和partition。

数字库：这个库的目的是允许编译器的实现者在用数字运算时充分利用低层机器结构。这样更高层的数字库可以写到这个数字库中，以产生更高效的代码，而不必为每种可能的机器都编写代码。这个数字库也包括复杂的数字类（这些在C++的第一个版本中是以例子出现的，现在已经是这个库的一部分了），这些类以float、double和long double的形式出现。

A.4 标准模板库（STL）

第15章提供的模板说明了C++中模板的强大功能和灵活性，但它们并不仅仅是作为一般目的的工具，虽然它们确实可以解决一些问题。C++的STL [1]是一个功能强大的库，它可以满足我们对包容器和算法的巨大需求，而且是一种完全可移植的方式。这意味着程序不仅可以很容易地移植到其他平台上，而且我们的知识本身并不依赖某个特定编译器开发商提供的库。因此在寻找特定开发商的解决方案之前应该首先在STL中寻找包容器和算法。

软件设计有一个基本原则：就是所有的问题都可以通过引进一个间接层来简化。这种简化在STL中是用循环子来完成的，它在尽可能少地知道某种数据结构的情况下完成对这一结构的运算，因此它使得数据结构与STL相独立，这意味着所有运用于内部类型上的运算也可以在用户定义的类型上完成，反之亦然。如果我们学会了这个库，就可以运用于每种情况。

这种独立性的缺点是我们得花时间去熟悉STL中处理事情的方法。当然，STL用的是统一的模式，所以我们一旦适应了它，从一个STL工具到另一个工具，就不会有什么变化。。

请看下面用STL的set类的一个例子，一个简单的set可以创建与int一起工作：

```
//: INTSET.CPP -- Simple use of STL set
#include <set.h>

void main() {
  set<int, less<int> > intset;
  for(int i = 0; i < 25; i++)
    for(int j = 0; j < 10; j++)
      // Try to insert multiple copies:
      intset.insert(j);
  // Print to output:
  copy(intset.begin(), intset.end(),
```

[1] 由Alexander Stepanov和Meng Lee在惠普促成为C++标准。

```
        ostream_iterator<int>(cout, "\n"));
}
```

set的第一个参数是包含在这个集合中的类型，第二个参数产生一个运算用于在集合中插入元素（用insert()函数）。这个集合将允许不同的关键字插入到集合中。

copy()函数显示了循环子的使用。set的成员函数begin()和end()产生循环子作为它们的返回值。copy()函数就用这些指针作为它运算的开始点和结束点，并在由它们产生的边界之间简单移动并把元素拷贝到第三个参数上，这第三个参数也是一个循环子，但它是为输入输出流而产生的特殊类型。这就将一个int对象放到了cout上并用一个新行来分割它们。

copy()函数在一个流上打印是不受限制的。它实际上可用于任何场合：它只需要三个循环子来交流数据即可。所有的算法都遵从copy()的这种形式，并简单地管理循环子（这就是额外的间接层）。

现在来看看INTSET.CPP的形式，并改写它来解决第15章中出现的协调性问题，这种解决方法相当简单。

```
//: CONCORD.CPP -- Concordance with STL
#include <set.h>
#include <fstream.h>
#include "..\allege.h"
#include "..\14\sstring.h"
const char* delimiters =
  " \t;()\"<>:{}[]+-=&*#.,/\\"
  "0123456789";

main(int argc, char* argv[]) {
  allege(argc == 2, "usage: concord filename");
  ifstream in(argv[1]);
  allegefile(in);
  set<Hstring, less<Hstring> > concordance;
  const sz = 1024;
  char buf[sz];
  while(in.getline(buf,sz)) {
    // Capture individual words:
    char* s = strtok(buf, delimiters);
    while(s) {
      concordance.insert(s); // Auto type conv.
      s = strtok(0, delimiters);
    }
  }
  // output results:
  copy(concordance.begin(), concordance.end(),
       ostream_iterator<Hstring>(cout, "\n"));
}
```

这里真正的唯一的区别在于用Hstring而不是int。被插入的词是从一个文件中读入的并用strtok()函数作为标志。其他的和INTSET.CPP中完全一样。更方便的是，这种协调性是自动排

序的。

STL中包含的其他类如下：

包 容 器	使 用
vector	随机访问循环，在尾部插入 /删除运算的时间相同。在中间插入 /删除运算将花线性时间，也有用于 bool型的规范说明
list	双向循环和在任何地方顺序插入 /删除运算有一致时间
deque	随机访问循环，在头部、尾部插入 /删除运算的时间相同。在中间插入/删除运算将花线性时间
stack	我们可以选择 vector、list或deque作为模板参数来创建它
queue	可以用list或 deque从模板中创建
priority_queue	随机访问循环，提供 top(),push()和pop()运算，可以从 vector或deque模板中产生
multiset	一个允许有相同关键词的集合
map	一个带唯一关键词的一个联合包容器
multimap	一个允许有相同关键词的联合包容器，像一个哈希 (hash)表

本节只是简单地介绍了STL的功能，我们花点时间来学习STL是值得的。

A.5 asm关键字

这个关键字允许我们在C++程序中用汇编代码来控制我们的硬件。通常可以在汇编代码中引用C++变量，这意味着可以很容易地和我们的 C++代码通信，并且只用在必须高效运行或运用特殊的处理器指令的情况下才使用汇编代码。汇编语言的确切语法是依赖特定编译器的，这可以在编译器文档中找到。

A.6 明确的运算符

这些是位运算和逻辑运算运算符。美国以外地区的键盘上可能没有 &、|、……之类的键，只好使用C中可怕的特殊符号，这不仅难以打印，而且阅读起来非常难懂。这在 C++中用了一些新的关键词来代替。

关 键 词	含 义
and	&&(逻辑与)
or	|| （逻辑或）
not	!（逻辑反）
not_eq	!=（逻辑不等于）
bitand	&（按位与）
and_eq	&=（按位与赋值）
bit or	|（按位或）
or_eq	|=（按位或赋值）
xor	^（按位异或）
xor_eq	^=（按位异或赋值）
compl	~（1的补）