第一章 开始

函数的四个部分：返回类型、函数名、形参列表、函数体

Main函数的返回值，大多用来表示运行状态，非0之外往往表示异常

C++使用iostream库来输入输出：cin输入、cout输出、cerr输出警告、clog输出日志

Endl的双重作用：换行；刷新缓冲，保证将缓冲区的内容真正写入输出流

标准库都在命名空间std中，注意作用域符号：：

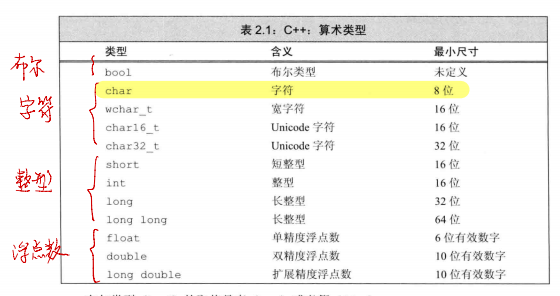
注释：//单行注释；/\* \*/多行注释（不要嵌套）

区别++i（先加再输出）与i++（先输出再加）

控制流语句：while,for,if

第二章 基础

算术类型：



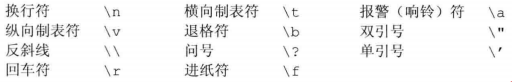
如何选择类型：

明确不为负数，则用无符号类型；使用int和long long而不用long和short；不用char和bool进行运算；用double而不用float；不要混用无符号和有符号类型

赋值时的类型转换：

非布尔转布尔，非0即true；布尔转算数值，true1false0；浮点转整型，抛去小数；无符号类型超范围，取最大数的余；有符号类型超范围，未定义；

字面值常量：整数与进制，八进制0开头，十六进制0x开头，二进制0b开头；字符串结尾多一个空字符；转义序列，见下表：



变量与对象：对象是一块特定类型的存储空间；变量是命名之后的对象

初始化：区别初始化与赋值；列表初始化（括号花括号），不会丢失信息；默认初始化，函数体内对象不会默认初始化。

声明与定义：声明规定变量类型与名字；定义：申请存储空间；使用extern来只声明而不定义，在别的文件去查找定义，若extern时初始化了，则也定义了；只能定义一次而可以声明多次；

作用域大多用花括号表示，作用域能嵌套，新建局部变量会覆盖同名全局变量

引用： int a = 1;int &b = a;引用必须初始化，引用并非对象，而是原来对象的别名；

指针：int a = 1;int\* p = &a;指针本身是一个对象，且无须在定义时赋初值（为nullptr），指针存放某个对象的地址；指针的值有下列四种状态：指向对象；指向对象邻近下一位置；空指针；无效指针。解引用符\*p即代表p所指对象。如果两个指针的地址值相等，则指针相等。

注意int\* p1,p2;p1是int指针，p2是int

常量限定符：如果想在多个文件共享const对象，则在定义和声明前面加extern

顶层与底层const：顶层表示指针本身是个常量，如int\* const p1=&a;底层表示指针所指对象是常量，如: const int\* p2 = &a；声明引用的const都是底层，引用绑定和对象都不能改，如：const int &b = a;

常量表达式：constexpr指值不会改变且在编译阶段就确定了；

类型别名：typedef a b;等效于using b =a; a可以是基础类型，也可以是自定义类，b是其别名

auto类型说明符：让编译器自动推断类型，必须有初始值才能推断。

decltype类型指示符：decltype(b) a; 则a的类型就和b一样

预处理变量与指令：

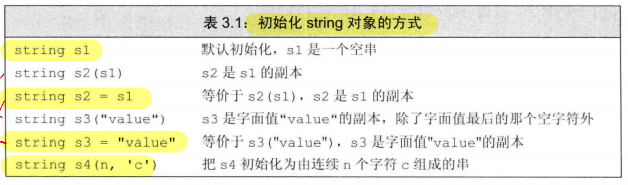
#ifdef 检查是否已定义；#ifndef 检查是否未定义；#endif 检查末尾

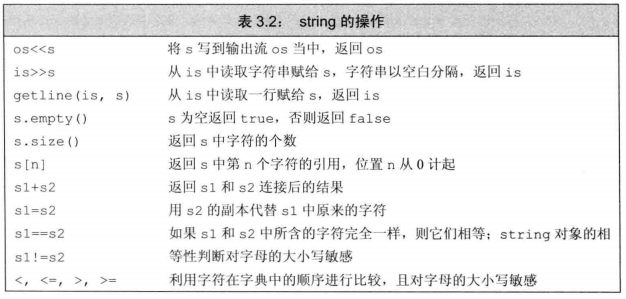
#define 定义预处理变量（往往均大写）

第三章 字符串、数组和向量

头文件不应该用using namespace声明，因为每个使用了该头文件的文件都会使用这个声明，有时会产生命名冲突

标准库类型string：



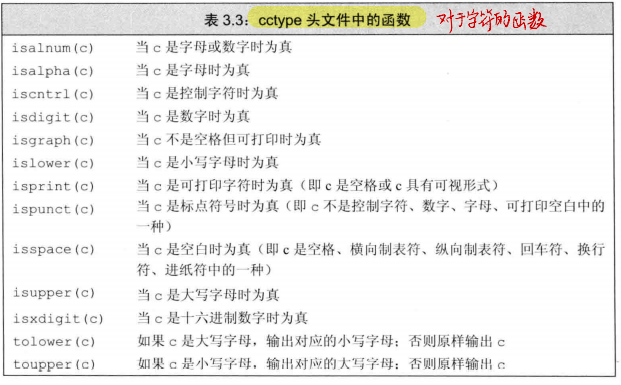


在用cin读取字符串时，string会自动忽略空白，如“ a b c ”会变成”abc”

使用getline读取一整行：string line; getline(cin,line); 返回流参数，不包含换行符

可以将两个string直接相加，一个字符串字面值与string相加，不能直接把两个字符串字面值相加

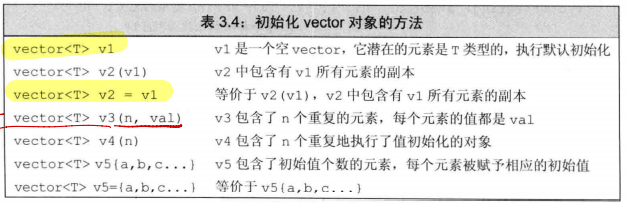
用于处理string中字符的一些函数：

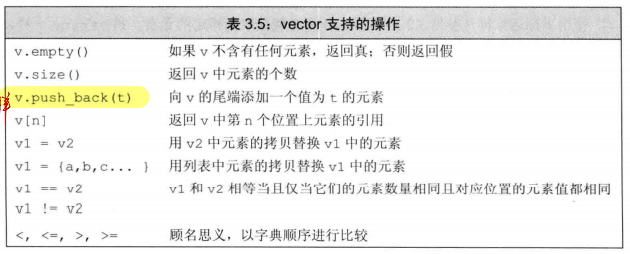


范围for语句：for(auto c:s)只读，for(auto &c:s)引用可改

标准库类型vector:

Vector是一个类模板，使用时需要实例化：vector<int> ivec;



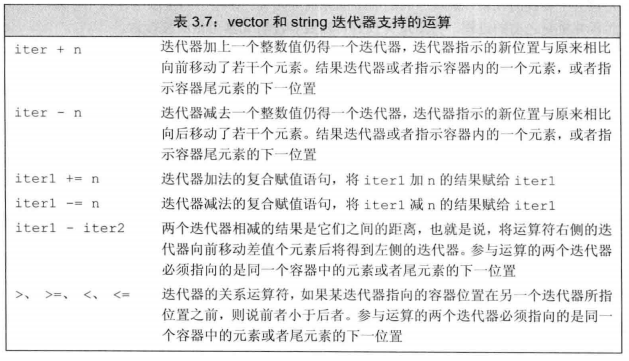


迭代器：

auto b = v.begin(),e=v.end();bedin表示第一个元素，end表示本不存在的尾后，如不需修改，可用cbegin和cend

范围for语句和迭代器都不可以动态改变容器大小，而普通循环语句可以

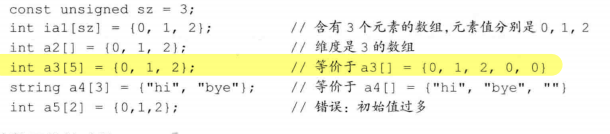




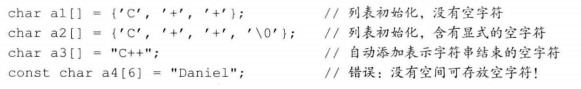
数组：

编译的时候，数组大小应该是已知且固定的，数组的元素应为对象

数组的初始化：



用字符串来给字符数组进行初始化时，大小要加一（因为末尾的空字符）：



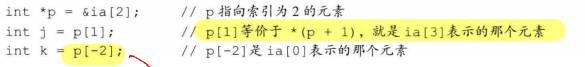
数组不允许拷贝和赋值，而vector可以

编译器会把数组名字转化为指向数组首元素的指针，但是使用decltype、sizeof时不会这样

数组的指针也支持vector的迭代器的操作

可用标准库的函数获取数组指针：int \*beg = std::begin(ia);int \*end = std::end(ia);类似于迭代器

指针和迭代器一样，不支持相加（可理解为地址不能相加，而相减则得到距离）

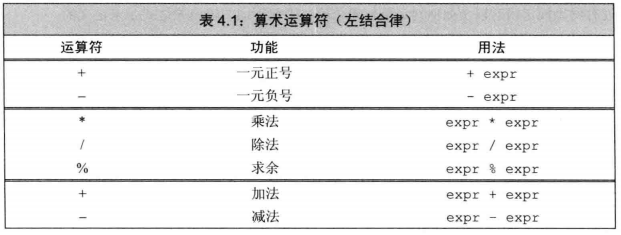
数组的下标相当于指针的下标：且指针下标可以为负：  


使用范围for语句处理多维数组时，除了最内层的循环外，其他循环都应该是引用类型，否则会把外面的数组名当作指针而不是数组

第四章 表达式

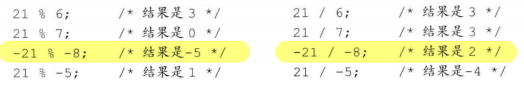
运算符重载：运算符作用于类类型的运算对象时，可以自定义其含义，如<<,+等

算术运算符：



整数除法和取余：

整数除法直接去除小数部分。除法和取余的正负区分：



逻辑关系运算符：

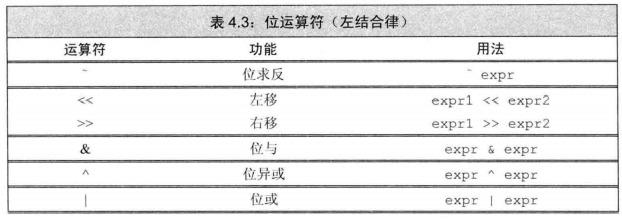


后置递增运算符优先级高于解引用运算符，如\*p++等价于\*(p++)，先解引用再++

成员访问运算符：ptr->men等价于(\*ptr).mem

条件运算符：cond?expr1:expr2

位运算符：（强烈建议只用于无符号类型）



sizeof运算符：sizeof (type) 或者sizeof expr ，返回type的所占字节数或者expr类型的所占字节数。可用此返回数组元素个数：sizeof(ia)/sizeof(\*ia)

隐式类型转换：

在条件中，非布尔转化为布尔；初始化和赋值时，右侧对象转化为左侧的类型；数组名转化为指向数组首元素的指针；

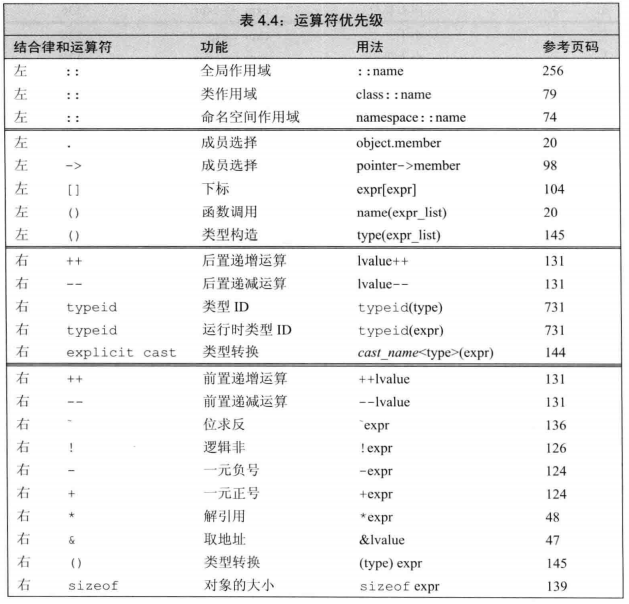
显式类型转换（强制类型转换）：

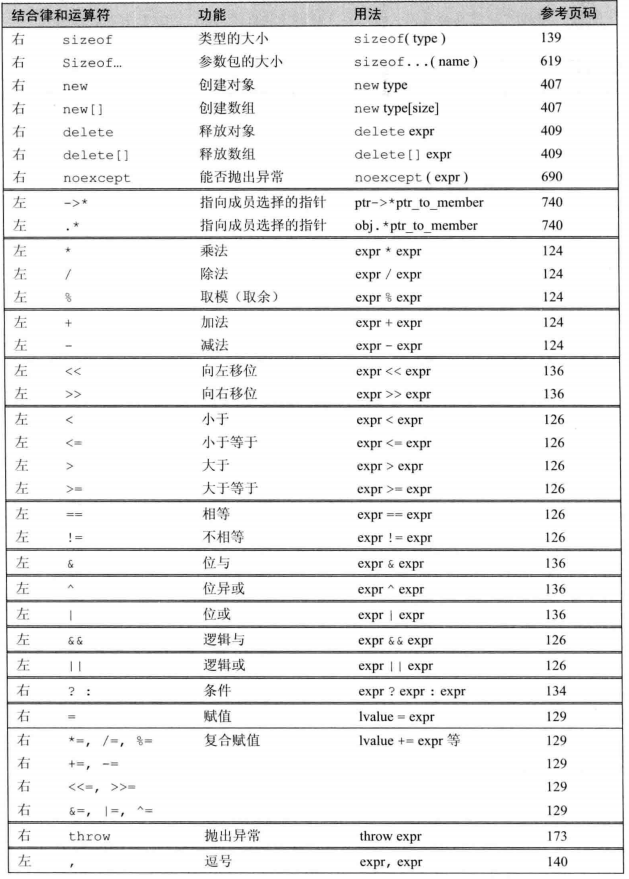
命名强制类型转换：cast-name<type>(expression)

Cast-name有以下四种：static\_cast（除了底层const都可以转换）\dynamic\_cast（支持运行时类型识别）\const\_cast（只能转换底层const ）\reinterpret\_cast

旧式的强制类型转换：type (expr)或者(type) expr

运算符优先级表：





第五章 语句

空语句： ；语法上需要但是逻辑上不需要

悬垂else：else与离它最近的尚未匹配的if匹配

Switch语句：

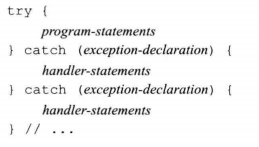
Case标签必须时整型常量表达式；每个case里面都应该有break，不加的话，进入一个case之后会无视后面case的条件，直接执行；

Do while语句与while语句的区别：do while无论如何都会至少执行一次，且while后有分号

Break语句：打破最近的循环或者switch

Continue：终止当前迭代

Throw表达式：



第六章 函数

函数的返回类型：不能时数组类型或者函数类型，但可以是指向函数或者数组的指针

函数内的局部对象：会随着函数的调用而创建，随着函数离开而销毁；

函数内的局部静态对象（static修饰）：不会随着函数离开而销毁

函数应该在头文件里面被声明，在源文件里被定义

函数的传参：

传值调用：实参的值拷贝给形参，f(int a);f(a)；指针形参，拷贝实参指针，但是指向同一对象，所以可以改变函数外部的值，f(int \*a);f(&a)

传引用调用：避免了拷贝，也可以改变外部的值，f(int &a);f(a)

给函数传数组的三种方法：

如char数组末尾有一个标记，可用作数组结束的标记；

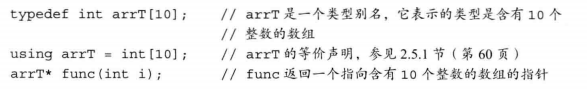
使用标准库函数，传入begin和end指针；

传入数组名（实际是头指针）和数组的长度；

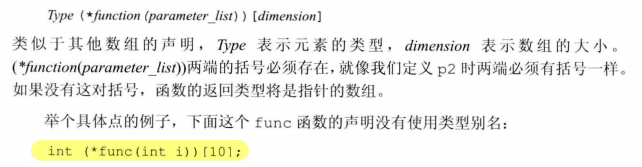
函数的返回值：只有void函数才能不包含return；返回vector的可以return{a1,a2,a2}列表初始化；

返回数组指针四种方法：

使用类型别名：、



不适用类型别名：



使用尾置返回类型：



使用decltype：



函数重载：

重载：同一作用域内几个函数名字相同但是形参列表不同

不允许两个函数除了返回类型外其他要素都相同

函数特殊的语言特性：

默认实参：一旦某个形参被赋予了默认值，其后面的都必须有默认值。默认实参只能在声明和使用时使用，不应该在定义中使用

内联函数：使用inline修饰的函数，在编译时直接用函数体替换函数调用

Constexpr函数：返回类型和形参类型都得是字面值类型（不一定是常量表达式），有且只有一条return语句；被隐式内联

assert预处理宏：assert(expr) 若表达式为假，则输出信息并终止程序

打印一些信息：\_ \_func\_ \_当前调试的函数名，其他诸如：\_ \_FILE\_ \_, \_ \_LINE\_ \_, \_ \_TIME\_ \_, \_ \_DATE\_ \_

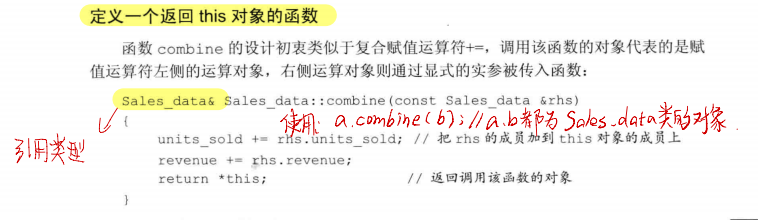
第七章 类

类的实现包括：类的数据成员、负责接口实现的函数体、各种私有函数

成员函数：声明在内部，定义可内可外，调用 类.函数名（）。定义在内部隐式内联

非成员函数：声明在外，定义在外，调用：函数名（实参）

定义一个返回this对象的函数：如+=



构造函数：

构造函数名与类名一致，用于初始化类对象的数据成员，没有返回类型，通常使用列表初始化：Sales\_data(string &s,int n,double p):bookNo(s),sold(n),price(p) { } 可允许一步类类型转换

Public与private：public：成员在整个程序可被访问，private：只能在类内被访问

Class与struct的唯一区别：默认访问权限：struct：public而class：private

友元：允许其他类或者函数访问它的非公有成员，则加一条类或者函数的声明，并在前面加friend

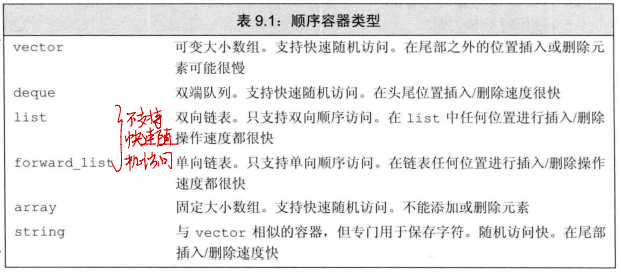
用explicit声明构造函数：可阻止隐式类型转换，且只支持直接初始化，不支持赋值初始化

聚合类满足四个条件：所有成员公有；没定义构造函数；没有类内初始值；没有基类也没有虚函数

类的静态成员static：成员与类关联而不与类的对象关联，static只能出现在类内，但是必须在类外定义和初始化每个静态成员

第八章 IO库

第九章 顺序容器



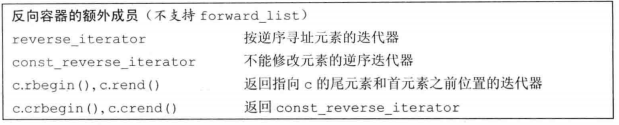
如何选择顺序容器：

通常选择vector是最好的选择，如果要求在容器中间插入或删除元素，则使用链表

容器都定义在同名头文件中，均为模板类

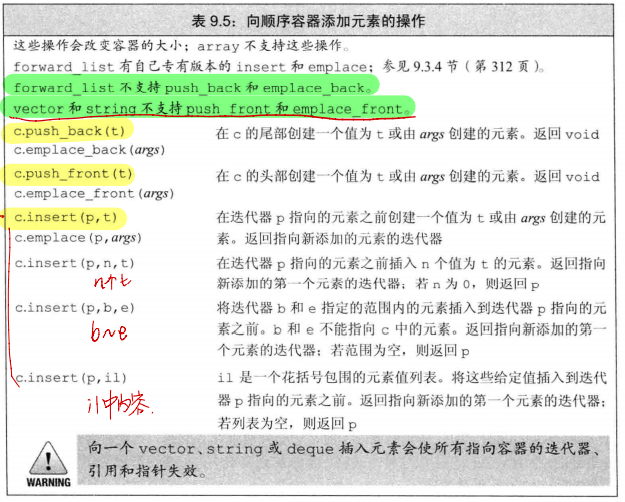
常见容器操作：



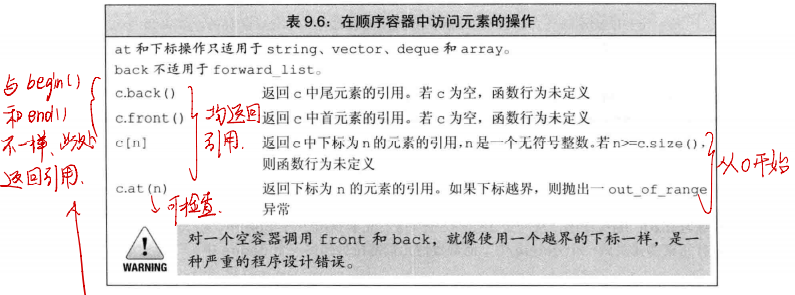


可将容器两个迭代器这个范围，拷贝给同样容器类型和元素类型的容器对象

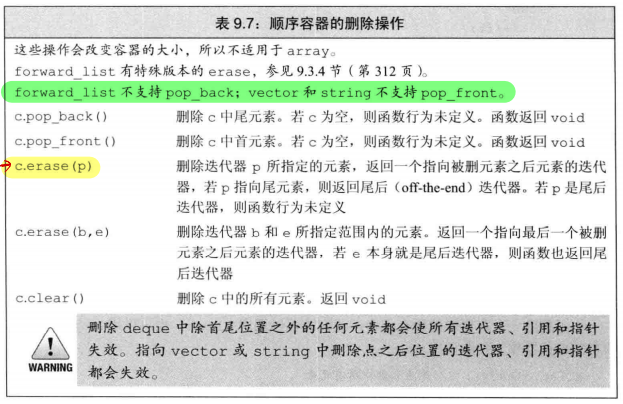
向顺序容器中添加元素：



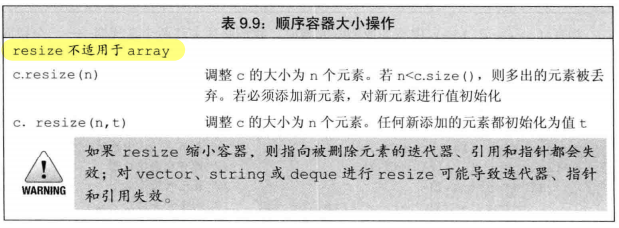
顺序容器的访问操作：



顺序容器的删除操作：

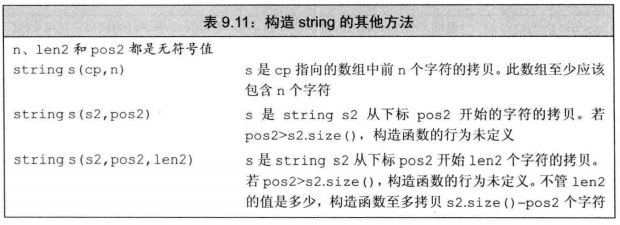


顺序容器的大小操作：

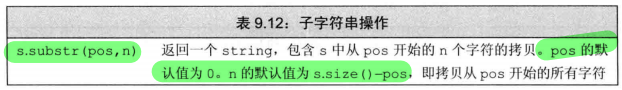


额外的string操作：

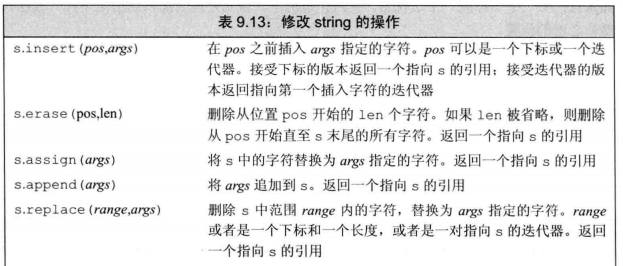
构造string



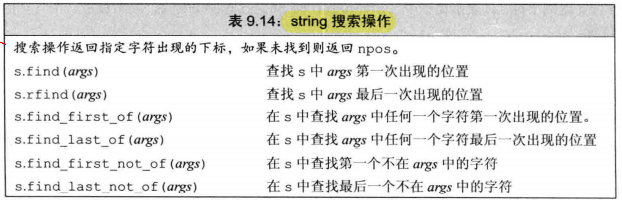
子字符串substr



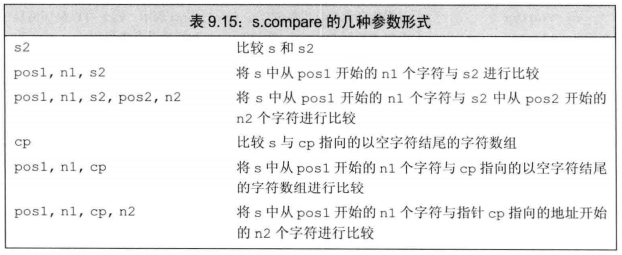
修改string的操作：



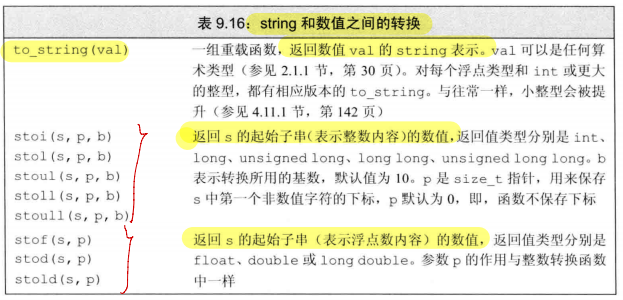
String的搜索操作：



String的compare函数：

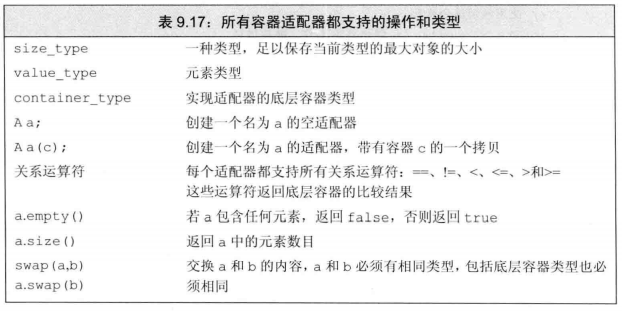


String和数值之间的转换：



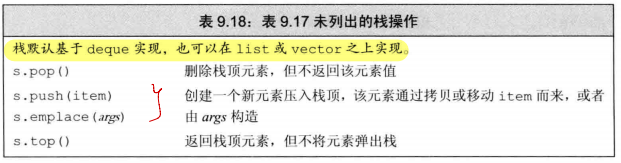
容器适配器：stack, queue, priority\_queue

容器适配器的操作：

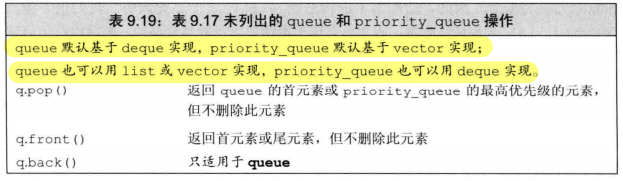


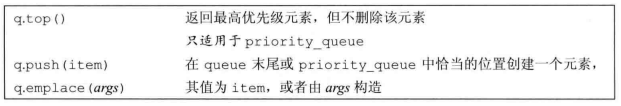
可用deq来初始化stack和queue，可用vector来初始化priority­\_queue

一些stack栈操作：



一些queue队列和priority\_queue优先队列操作：





第十章 泛型算法

大多数算法都在头文件algorithm中，以及numeric里，这些算法不直接操作容器，不会改变容器大小，而是遍历由两个迭代器指定的一个范围

find函数：auto result = find(vec.cbegin(), vec.cend(), val);若成功，则返回对应位置迭代器，若失败，则返回第二个参数

迭代器令算法不依赖容器，但算法依赖于元素类型的操作

Equal函数：bool isEqual = equal(roster1.cbegin(), roster1.cend(), roster2.cbegin());像这种只接受第二个序列单一迭代器的算法，都假定第二个序列至少和第一个序列一样长

accumulate 求和函数：int sum = accumulate(vec.cbegin(), vec.cend(), 0); 和的初始值为0

fill函数：fill(vec.begin(), vec.end(), 0); 将这个范围填充为0

copy函数：auto ret = copy(begin(a1), end(a1), a2); 第三个参数表示目的序列的起始位置，目的序列长度至少和已有序列一样长，返回拷贝结束后的迭代器

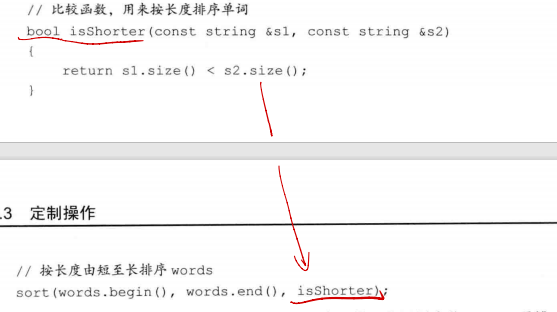
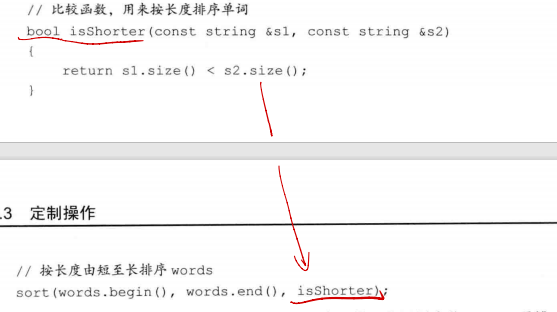
replace函数：replace(ilst.begin(), ilst.end(), 0, 1); 将迭代器范围内值为0的元素改为1

sort函数：sort(vec.begin(), vec.end())；默认升序排序

unique函数：auto end\_unique = unique(vec.begin(), vec.end()); 输出已排序序列vec，输出不重复元素的之后一个位置的迭代器

可向算法传递函数（谓词）：谓词是一个可调用的表达式，返回结果是一个能用作条件的值

如：



stable\_sort：传入谓词，可让单词长度按长短排序的同时，相同长度的单词按字典序排序

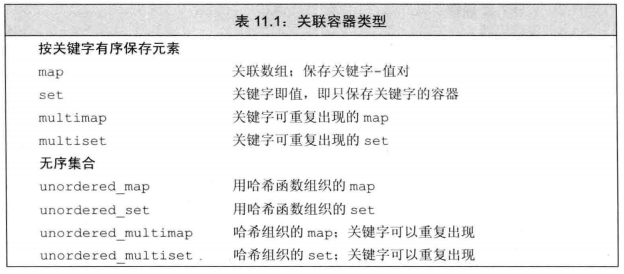
lambda表达式：

auto f = [capture list] (parameter list) -> return type { function body}

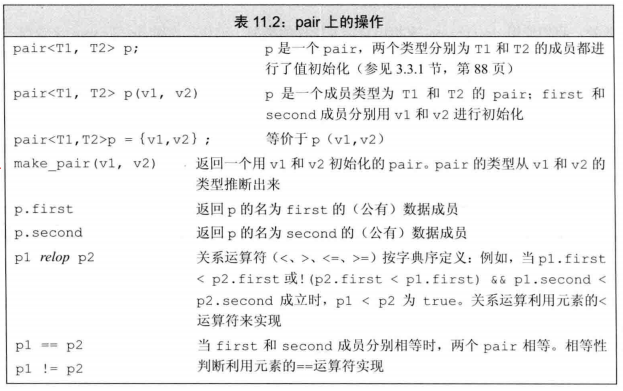
其他可以忽略，但必须保留[capture list]和{ function body}，调用和普通函数一样

第十一章 关联容器

两个主要的关联容器set（支持高效关键字查询）和map（元素是一些键值对）

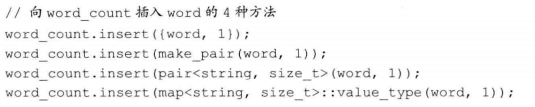


Pair类型，在头文件utility中，pair的操作：

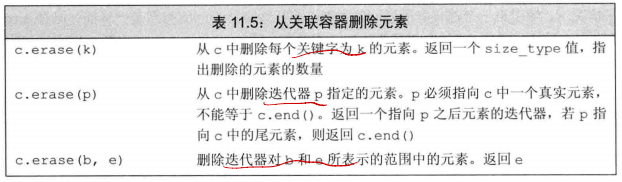


关联容器的关键字是只读的

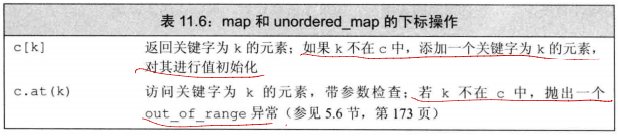
向map添加元素的四个方法：



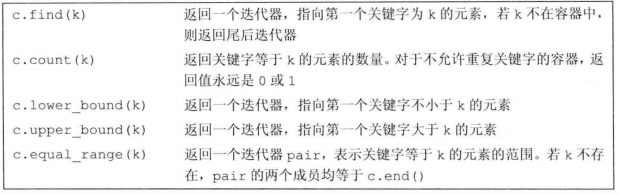
关联容器的删除操作：



关联容器的下标操作：



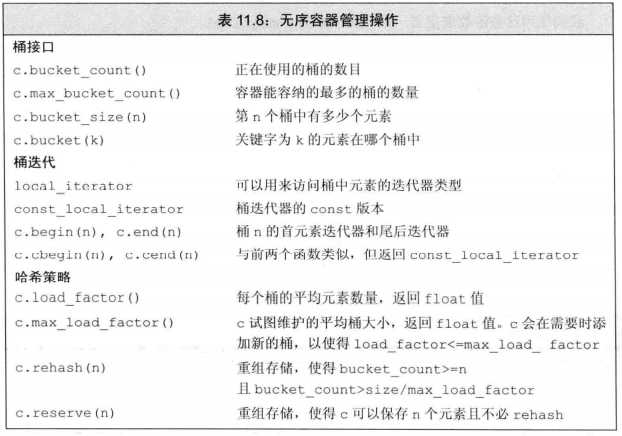
在关联容器中查找元素：



在multimap和multiset中有相同关键字的元素时，它们往往是连续存储的

无序关联容器：使用哈希函数和==运算符来组织元素，前面带有unordered\_

无序容器通过桶来进行管理：



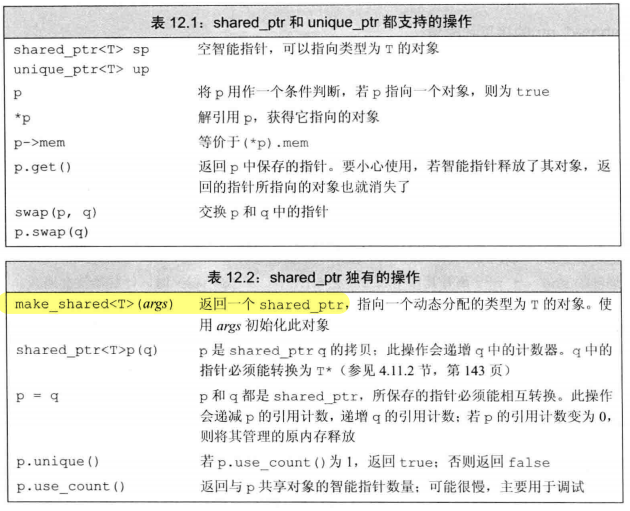
第十二章 动态内存

三种内存：静态内存：保存局部static、类static和函数体之外的变量，在程序结束时销毁；栈内存：保存函数体内非static对象，仅在程序块中存在；自由空间（内存池、堆）：用于动态分配，必须显式分配和销毁

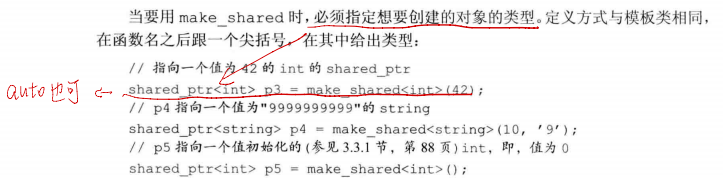
动态内存的管理：用new分配空间并返回指针，用delete接受一个动态对象的指针，销毁对象，并释放内存

两种智能指针：shared\_ptr：允许多个指针指向同一对象；unique\_ptr：指针独占指向的对象。此外还有弱指针weak\_ptr。智能指针也是模板，需要实例化

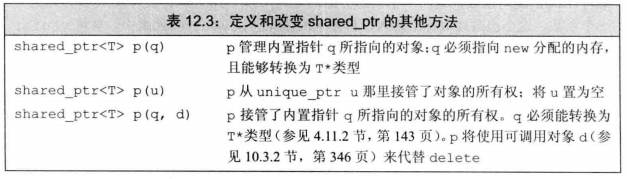
智能指针的一些操作：

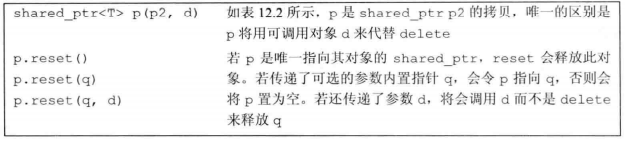


标准库函数make\_shared的使用：



Shared\_ptr的操作：





每个shared\_ptr都有一个关联的计数器，称之为引用计数。当拷贝一个shared\_ptr时，计数器都会递增，当shared\_ptr转移到新的对象，或者它被销毁（如局部指针离开作用域），计数器都会递减，一旦一个对象的shared\_ptr的计数器变为0，就会自动释放这个对象，也会自动释放关联的内存

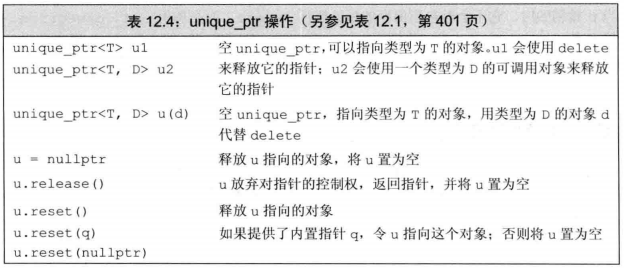
程序使用动态内存的三种原因：程序不知道自己需要使用多少对象，如容器；程序不知道对象的准确类型，如模板类；程序需要在多个对象间共享数据，允许多个对象共享相同的状态

New与delete的使用：

Int \*p = new int{1}; delete p; 可用花括号初始化，也可省去以默认初始化

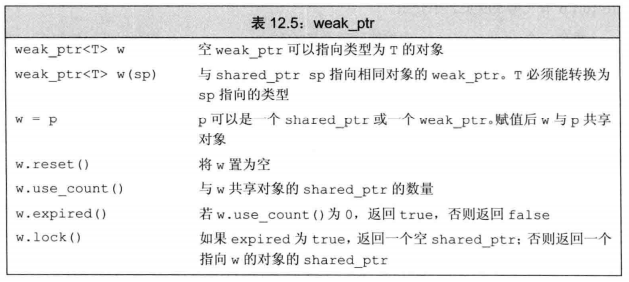
Shared\_ptr与new结合使用：Shared\_ptr<int> p {new int(1024)};

Unique\_ptr的操作：



Unique\_ptr的初始化：unique\_ptr<int> p1(new int(42)); 不支持普通的拷贝或者赋值操作，而可以使用release或者reset来转移所有权：unique\_ptr<int> p2(p1.release); p1释放对象的所有权并转移给p2, unique\_ptr<int> p3(new int(24)); p2.reset(p3.release); p2释放自己的对象内存，并使用p3释放的对象

Weak\_ptr是弱指针，必须用shared\_ptr来初始化，它指向一个shared\_ptr管理的对象，但不会改变计数，当最后一个shared\_ptr被销毁时，即使weak\_ptr指向对象，还是会被释放。常见操作：



New的delete动态数组：

Int \*p = new int[10]; delete [ ] p;

第十五章 面向对象程序设计

对于某些函数，基类希望它的派生类各自定义合适的自身的版本，就把这些函数声明成虚函数

类派生列表：class c1 : public c { };此处的public、private、protected说明符，限制派生使用基类的权限，若与基类本身的成员说明符冲突，则以派生的说明符为准

派生类必须对所有重新定义的虚函数进行声明，在末尾加一个override以标记覆盖。其隐式继续为虚函数，前面的virtual可加可不加，若加final则说明其派生类不能再覆盖 ，派生类如果没有重写覆盖虚函数，则直接继承基类中的版本

当使用基类的引用（指针）调用一个虚函数时，将发生动态绑定

基类通常都应该定义一个虚析构函数

派生类可以访问基类的public（大家都可以访问）和protected（只能派生类和友元访问）

如果基类中定义了一个静态成员，则整个继承体系中只存在该成员的唯一定义

派生类的声明与一般类差别不大，不能出现派生列表

若想防止一个类被继承，则在类名后面跟一个final

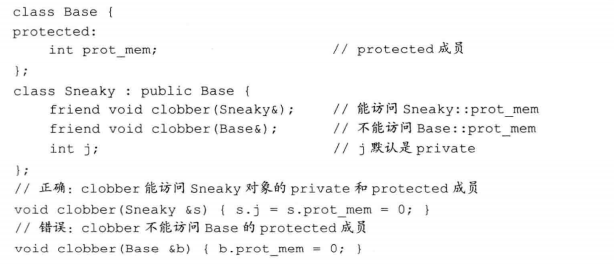
静态类型与动态类型：静态类型是在编译时就已知了，动态类型直到运行时才可知

我们能且只能把基类的指针或者引用绑定到派生类对象上，反之不行。称之为动态绑定，这是通过指针或者引用调用虚函数发生的

若想回避虚函数的机制，则用作用域运算符实现这一目的，这通常发生在成员函数（或友元）的代码中

我们可以定义一个抽象基类，需要声明一个纯虚函数（在分号之前写=0），不能直接创建抽象基类的对象

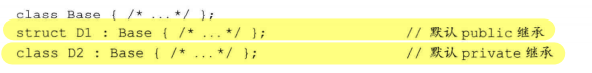
Protected的一条重要性质：派生类或者友元只能通过派生类对象来访问基类的protected，而不能通过基类对象来访问，如：



友元关系不能传递也不能继承

使用using声明可以改变派生类继承某个名字的访问级别，但是只能对派生类能访问的成员使用

若在派生类别前面没有访问级别的说明符，则class默认为private，struct默认为public：



通过基类调用虚函数的例子：（其中D1里面没有重写fcn()，其中Base里面没有f2()）

