STL中的容器（比如vector）

- 内存分配：vector中的元素是分配在heap上的，尽管vector本身是局部变量
  - int a[1024][1024];
  - vector<vector<int> > v(1024, vector<int>(1024));
  - 注意函数不要返回对局部变量的引用，包括vector变量
- vector的动态扩展
  - 下标操作符只能用于已经分配好的内存，不能通过赋值的方法动态扩展数组
    - vector<int> v; for (i=0; i<n; i++) v[i] = i; // 下标访问越界！
  - 用push_back来动态扩展数组

vector（续）

- 区分reserve和resize
  - reserve不分配内存，与push_back配伍
    - 用v[i]会导致数组访问越界
  - resize分配并初始化内存，与下标操作符配伍
    - 用push_back会添加到初始内存之后。用数组定义也有同样的问题：vector<int> v(n);
- 关于resize
  - 如果vector的定义可以推迟到知道数组大小之后，那么可以省掉resize
  - 如果resize之前数组的内容无需保留，那么clear之后再resize，而不是resize之后再用fill初始化

内存管理与性能

- 内存管理
  - 系统会自动回收vector中分配的内存，不用delete
  - 除非vector中的元素含有用new显式分配的内存
- 性能考虑
  - 函数的形参是vector类型时，一般用引用类型来避免拷贝
    - 使用加const的常引用来避免误改
  - 返回vector的类型一般不会引起多余的拷贝：现代的编译器一般有优化

访问vector中的元素

- 显示地遍历vector中的各元素，可以就用下标，不需要用iterator
  - for (int i=0; i<v.size(); i++) v[i] = …
  - for (vector<int>::const_iterator it=v.begin(); …) *it=…
- 如果用STL中的标准算法隐式地遍历vector，则需要使用iterator，但是不用前面的繁琐声明
  - 比如reverse(v.begin(), v.end())
  - 注意：C语言中的数组名可以作为iterator，但是没有定义begin和end等成员函数
    - 比如int a[10]; sort(a, a+10); 而不是sort(a.begin(), a.end());

STL中的iterator

- 支持随机访问的容器一般不会需要显式的iterator声明
  - 比如vector、string
- 用auto可以让编译器自动推断类型信息