顺序容器

概览以及一些知识点

1. 顺序容器为程序员提供了控制元素存储和访问顺序的能力，这种顺序不依赖于元素的值，而是与元素加入容器时的位置相对应，**提供了快速顺序访问元素的能力**。
2. 标准库中的顺序容器：
   - vector：可变大小的数组
   - deque：双端队列，支持快速随机访问
   - list：双向链表，只支持双向顺序访问
   - forward_list：单向链表
   - array：固定大小的数组，不能添加或删除元素
   - string：与vector相似的容器但专门用于保存字符
3. 顺序容器array在定义时就要确定大小，两个array类型的值可以用赋值的方式进行拷贝
4. 调用swap函数可以交换两个容器的内容，无论是顺序容器还是关联容器。**除array外，swap不对任何元素进行拷贝、删除或插入操作，只是交换了两个容器的内部数据结构，可以保证在常数时间内完成**。这意味着容器内的元素不会移动，故指向容器的迭代器、引用和指针在swap操作后都不会失效，仍然指向swap操作之前所指向的那些元素。

```
vector<int> v1{1,2,3,4},v2{5,6,7};
vector<int>::iterator v1it=v1.begin();
cout<<*v1it<<endl;  //输出1
swap(v1,v2);
cout<<*v1it<<endl;  //仍然输出1
cout<<*(v1it+1);    //输出2
```

顺序容器操作

1. 向一个vector、string或者deque插入元素后会使所有指向容器的迭代器、引用和指针失效。

```
vector<int> v1{1,2,3,4};
vector<int>::iterator v1it=v1.begin();
v1.insert(v1.begin(),{5,6});
cout<<*v1it;    //此处输出的值是一个不确定的值 而不是1或者5
```

2. insert操作的四种模式：其返回值是新插入的第一个元素的迭代器
   - insert(it,val)： 在迭代器it之前插入一个值为val的元素
   - insert(it,n,val)：在迭代器it之前插入n个值为val的元素

- insert(it,begin,end)：在迭代器it之前插入迭代器begin和end指定的范围内的元素
- insert(it,list)：在迭代器it之前插入list花括号内包围的元素

3. emplace函数在容器中直接构造元素（调用构造函数），传递给emplace函数的参数必须与容器的元素类型的构造函数相匹配。而普通的插入元素的函数如push_back则是拷贝元素的副本然后插入到容器中。
4. 删除deque中除首尾之外的任何元素都会使所有迭代器、引用和指针失效。执行vector或string中删除点之后位置的迭代器、引用和指针都会失效。

vector对象和string对象

1. 管理容量的成员函数：
   - capacity：返回不重新分配内存空间的话，vector可以保存多少元素
   - reserve(n)：分配至少能容纳n个元素的内存空间。该函数不改变容器中元素的数量，仅影响vector预先分配多大的内存空间。当n小于当前容量时，reverse什么都不做，vector不会退回内存空间。
   - resize(n)：该函数只改变容器元素的数目，不改变容器的容量
   - shrink_to_fit()：将capacity()减少为与当前的size()一样大。此函数指出不再需要任何多余的内存空间。
2. vector只有当迫不得已时才分配新的内存空间。
3. string对象的搜索函数返回的是string::size_type，是一个unsigned类型，最好不要用int型来保存
4. string与数值之间的转换：
   - to_string()：可以将int型或者double型转为string，但在转double型时会保留小数点后6位。
   - stoi————string转为int；stod————string转为double

```
string str="pi = 3.14";
//str.find_first_of(string)：找出string中任一字符第一次出现在str中的位置
double d=stod(str.substr(str.find_first_of("+-.123456789")));
```

容器适配器

1. 本质上一个适配器是一种机制，能够使某种事物的行为看起来像另外一种事物。一个容器适配器接受一种已有的容器类型，使其行为看起来像一种不同的类型。标准库定义了三个顺序容器适配器：stack、queue和priority_queue。
2. 默认情况下，**stack和queue都是基于deque实现的，priority_queue是在vector之上实现的**。所以这三个适配器除了有默认构造函数可以创建一个空对象之外，还接受一个容器的构造函数拷贝该容器来初始化适配器。此外我们也可以在创建一个适配器时指定第二个参数来指定该适配器是基于什么顺序容器实现的。

```
vector<string> v;
stack<string,vector<string>> st_baseon_vec(v);
```