unique 和 unique_copy

一.unique函数

类属性算法unique的作用是从输入序列中**“删除”所有相邻的重复元素**。

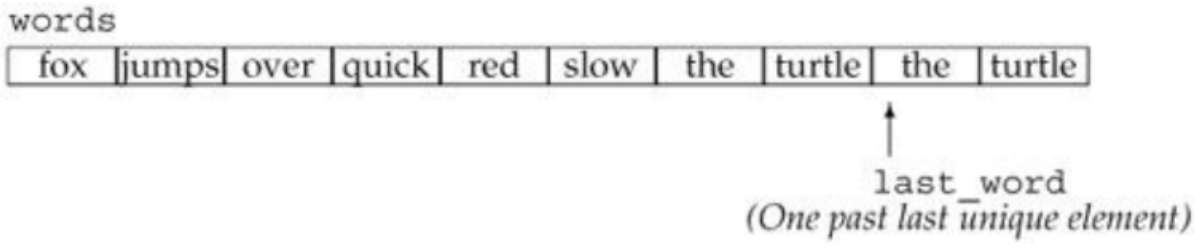
该算法删除相邻的重复元素，然后重新排列输入范围内的元素，并且返回一个迭代器（容器的长度没变，只是元素顺序改变了），表示无重复的值范围得结束。

```
// sort words alphabetically so we can find the duplicates
sort(words.begin(), words.end());
     /* eliminate duplicate words:
      * unique reorders words so that each word appears once in the
      *    front portion of words and returns an iterator one past the
unique range;
      * erase uses a vector operation to remove the nonunique elements
      */
 vector<string>::iterator end_unique =  unique(words.begin(),
words.end());
 words.erase(end_unique, words.end());
```

若调用sort后，vector的对象的元素按次序排列如下：

```
sort  jumps  over quick  red  red  slow  the  the turtle
```

则调用unique后，vector中存储的内容是：



注意，words的大小并没有改变，依然保存着10个元素；只是这些元素的顺序改变了。调用unique“删除”了相邻的重复值。给“删除”加上引号是因为unique实际上并没有删除任何元素，而是将无重复的元素复制到序列的前段，从而覆盖相邻的重复元素。unique返回的迭代器指向超出无重复的元素范围末端的下一个位置。

注意：算法不直接修改容器的大小。如果需要添加或删除元素，则必须使用容器操作。

example:

```
#include <iostream>
#include <cassert>
#include <algorithm>
#include <vector>
#include <string>
#include <iterator>
 using namespace std;

 int main()
{
    //cout<<"Illustrating the generic unique algorithm."<<endl;
    const int N=11;
    int array1[N]={1,2,0,3,3,0,7,7,7,0,8};
    vector<int> vector1;
    for (int i=0;i<N;++i)
        vector1.push_back(array1[i]);

    vector<int>::iterator new_end;
    new_end=unique(vector1.begin(),vector1.end());    //"删除"相邻的重复元素
    assert(vector1.size()==N);

    vector1.erase(new_end,vector1.end());  //删除（真正的删除）重复的元素
    copy(vector1.begin(),vector1.end(),ostream_iterator<int>(cout," "));
    cout<<endl;

    return 0;
}
```

运行结果为：

二、unique_copy函数

算法标准库定义了一个名为unique_copy的函数，其操作类似于unique。

唯一的区别在于：**前者接受第三个迭代器实参，用于指定复制不重复元素的目标序列。**

unique_copy根据字面意思就是去除重复元素再执行copy运算。

编写程序使用unique_copy将一个list对象中不重复的元素赋值到一个空的vector对象中。

```
//使用unique_copy算法
//将一个list对象中不重复的元素赋值到一个空的vector对象中
#include<iostream>
#include<list>
#include<vector>
#include<algorithm>
using namespace std;

int main()
{
    int ia[7] = {5 , 2 , 2 , 2 , 100 , 5 , 2};
    list<int> ilst(ia , ia + 7);
    vector<int> ivec;

    //将list对象ilst中不重复的元素复制到空的vector对象ivec中
    //sort(ilst.begin() , ilst.end());  //不能用此种排序，会报错
    ilst.sort();  //在进行复制之前要先排序，切记
    unique_copy(ilst.begin() , ilst.end() , back_inserter(ivec));

    //输出vector容器
    cout<<"vector: "<<endl;
    for(vector<int>::iterator iter = ivec.begin() ; iter != ivec.end() ; ++iter)
        cout<<*iter<<" ";
    cout<<endl;

    return 0;
}
```

假如

```
list<int> ilst(ia , ia + 7);
改为：vector<int> ilst(ia , ia + 7);
```

则排序时可用：

```
sort(ilst.begin() , ilst.end());
```

这里要注意list和vector的排序用什么方法。

《Effective STL》里这些话可能有用处：
item 31

“我们总结一下你的排序选择：

- 如果你需要在vector、string、deque或数组上进行完全排序，你可以使用sort或stable_sort。
- 如果你有一个vector、string、deque或数组，你只需要排序前n个元素，应该用partial_sort。
- 如果你有一个vector、string、deque或数组，你需要鉴别出第n个元素或你需要鉴别出最前的n个元素，而不用知道它们的顺序，nth_element是你应该注意和调用的。
- 如果你需要把标准序列容器的元素或数组分隔为满足和不满足某个标准，你大概就要找partition或stable_partition。
- 如果你的数据是在list中，你可以直接使用partition和stable_partition，你可以使用list的sort来代替sort和stable_sort。如果你需要partial_sort或nth_element提供的效果，你就必须间接完成这个任务，但正如我在上面勾画的，会有很多选择。

另外，你可以通过把数据放在标准关联容器中的方法以保持在任何时候东西都有序。你也可能会考虑标准非STL容器priority_queue，它也可以总是保持它的元素有序。