简明STL学习笔记

> C++大学教程第22章.

> 算法参考cplusplus

- 标准数组可以作为容器,只要把指针当做迭代器就好

容器类型

- 序列容器

  `vector` `deque` `list`

- 关联容器(重载`<`运算符,有些需要`==`)

  `set``multiset` **允许重复** `map` `multimap`

- 容器适配器

  `stack``queue` `priority_queue`

基本操作(记得加括号)

- 构造,拷贝构造,析构
- `empty`, `insert` `size`
- 运算符重载(大小关系比较)
- 位置: `max_size`, `begin` `end` `rbegin` `rend` `erase` `clear`

迭代器

- 正向 `iterator`
  - 随机访问`p[i]`表示和迭代器位置差p个元素的元素
- 反向 `reverse_iterator` **对应rbegin和rend**
- `istream_iterator<type>`和`ostream_iterator<type>`

  -
    ```
    ostream_iterator<int> output(cout," ");
    copy(vec.begin(),vec.end(),output)
    ```
- `const_iterator`
- 尽量使用前缀++

重要容器

vector

- `insert(pos,num)`,在原来`pos`和`pos-1`之间插入`num`;
- 容器必须是非空的,不然`front`,`end`没有意义
- `front`,`back`是引用,`begin`,`back`是迭代器

list

- 对于中间元素的插入删除(对于首尾的使用`deque`)
- splice

  ```
    // set some initial values:
    for (int i=1; i<=4; ++i)
       mylist1.push_back(i);      // mylist1: 1 2 3 4

    for (int i=1; i<=3; ++i)
       mylist2.push_back(i*10);   // mylist2: 10 20 30

    it = mylist1.begin();
    ++it;                         // points to 2
  //剪切到第一个迭代器位置(全部)
    mylist1.splice (it, mylist2); // mylist1: 1 10 20 30 2 3 4
                                  // mylist2 (empty)
                                  // "it" still points to 2 (the 5th
  element)
  //剪切到第一个迭代器位置(之间)
    mylist2.splice (mylist2.begin(),mylist1, it);
                                  // mylist1: 1 10 20 30 3 4
                                  // mylist2: 2
                                  // "it" is now invalid.
    it = mylist1.begin();
    std::advance(it,3);           // "it" points now to 30

    mylist1.splice ( mylist1.begin(), mylist1, it, mylist1.end());
                                  // mylist1: 30 3 4 1 10 20
  ```

- `.sort()`排序,`.unique()`排重`.merge(ano)`和ano依次合并112233...
- `.assign(beg,end);`把beg,end连个迭代器之间的赋值给本身
- `.remove(val)`删除所有这个值

关联容器

- 查询是否包含这个元素
  - `.count(a) == 0`
  - `.find(a) == sl.end()`
- 插入
  - `insert(make_pair( ) )`
  - `Map[a] = b`

- `set` 相关函数
    - `it=std::set_union (first, first+5, second, second+5, v.begin());`
    - `it=std::set_difference (first, first+5, second, second+5, v.begin());`
    - `it=std::set_intersection (first, first+5, second, second+5, v.begin());`

    -
      ```
      set_union(
         setA.begin(), setA.end(),
         setB.begin(), setB.end(),
         insert_iterator<set<int>>(ans,ans.begin()));
      ```

容器适配器

- 不支持迭代器
- 小顶堆 `priority_queue<int, vector<int>, greater<int> > q;`

-
  ```
  struct cmp {
       bool operator() (const node &a, const node &b)
       {
           return true;
       }
  };

  priority_queue<node, vector<node>, cmp> p;
  ```

STL算法

变序算法

- **copy**

```
//std::copy ( myints, myints+7,
myvector.begin() );
template<class InputIterator, class
OutputIterator>
  OutputIterator copy (InputIterator
first, InputIterator last, OutputIterator
result)
{
  while (first!=last) {
    *result = *first;
    ++result; ++first;
  }
  return result;
}
```

- fill

```
std::fill
(myvector.begin(),myvector.begin()+4,5);
  // myvector: 5 5 5 5 0 0 0 0
fill_n(beg,num,val);
template <class ForwardIterator, class T>
  void fill (ForwardIterator first,
ForwardIterator last, const T& val)
{
  while (first != last) {
    *first = val;
    ++first;
  }
}
```

- generate   *Assigns the value returned by successive calls to gen to the elements in the range `[first,last)`.*

```
#include <ctime>         // std::time
#include <cstdlib>       // std::rand,
std::srand
std::srand ( unsigned ( std::time(0) ) );
int RandomNumber () { return
(std::rand()%100); }
std::generate (myvector.begin(),
myvector.end(), RandomNumber);
```

template

```
void generate ( ForwardIterator first,
ForwardIterator last, Generator gen )
```

{

```
while (first != last) {
  *first = gen();
  ++first;
}
```

}

~~~
- partion *用一个函数把容器分成两个部分*

  ```c++
  // partition algorithm example
  #include <iostream>      // std::cout
  #include <algorithm>     // std::partition
  #include <vector>        // std::vector

  bool IsOdd (int i) { return (i%2)==1; }

  int main () {
    std::vector<int> myvector;

    // set some values:
    for (int i=1; i<10; ++i)
~~~

```
myvector.push_back(i); // 1 2 3 4 5 6 7 8 9

    std::vector<int>::iterator bound;
    bound = std::partition
(myvector.begin(), myvector.end(), IsOdd);

    // print out content:
    std::cout << "odd elements:";
    for (std::vector<int>::iterator
it=myvector.begin(); it!=bound; ++it)
      std::cout << ' ' << *it;
    std::cout << '\n';

    std::cout << "even elements:";
    for (std::vector<int>::iterator
it=bound; it!=myvector.end(); ++it)
      std::cout << ' ' << *it;
    std::cout << '\n';

    return 0;
  }

  template <class BidirectionalIterator,
class UnaryPredicate>
    BidirectionalIterator partition
(BidirectionalIterator first,

BidirectionalIterator last, UnaryPredicate
pred)
  {
    while (first!=last) {
      while (pred(*first)) {
        ++first;
        if (first==last) return first;
      }
      do {
        --last;
        if (first==last) return first;
      } while (!pred(*last));
```

```
        swap (*first,*last);
        ++first;
      }
      return first;
    }
```

- random_shuffle

  ```
  // using built-in random generator:
    std::random_shuffle ( myvector.begin(),
  myvector.end() );

  // using myrandom:
  // random generator function:
  int myrandom (int i) { return
  std::rand()%i;}
  std::random_shuffle ( myvector.begin(),
  myvector.end(), myrandom);
  ```

template

```
void random_shuffle (RandomAccessIterator
first, RandomAccessIterator last,
                           RandomNumberGenerator&
gen)
```

{

```
iterator_traits<RandomAccessIterator>::diffe
rence_type i, n;
n = (last-first);
for (i=n-1; i>0; --i) {
  swap (first[i],first[gen(i+1)]);
}
```

}

~~~~
- replace(注意这里和string不太一样)

  ```c++
  int myints[] = { 10, 20, 30, 30, 20, 10,
10, 20 };
  std::vector<int> myvector (myints,
myints+8);            // 10 20 30 30 20 10
10 20
  std::replace (myvector.begin(),
myvector.end(), 20, 99); // 10 99 30 30 99
10 10 99

  template <class ForwardIterator, class T>
    void replace (ForwardIterator first,
ForwardIterator last,
                  const T& old_value, const
T& new_value)
  {
    while (first!=last) {
      if (*first == old_value)
*first=new_value;
      ++first;
    }
  }
  //replace_if
  template < class ForwardIterator, class
UnaryPredicate, class T >
    void replace_if (ForwardIterator first,
ForwardIterator last,
                     UnaryPredicate pred,
const T& new_value)
  {
    while (first!=last) {
      if (pred(*first)) *first=new_value;
      ++first;
    }
  }
~~~~

- reverse

```
for (int i=1; i<10; ++i)
myvector.push_back(i);   // 1 2 3 4 5 6 7
8 9
std::reverse(myvector.begin(),myvector.en
d());    // 9 8 7 6 5 4 3 2 1

template <class BidirectionalIterator>
  void reverse (BidirectionalIterator
first, BidirectionalIterator last)
{
  while ((first!=last)&&(first!=--last))
{
    std::iter_swap (first,last);
    ++first;
  }
}
```

- rotate按照第二个参数位置旋转

```
for (int i=1; i<10; ++i)
myvector.push_back(i); // 1 2 3 4 5 6 7 8
9
std::rotate(myvector.begin(),myvector.beg
in()+3,myvector.end());

         // 4 5 6 7 8 9 1 2 3
template <class ForwardIterator>
  void rotate (ForwardIterator first,
ForwardIterator middle,
               ForwardIterator last)
{
  ForwardIterator next = middle;
  while (first!=next)
  {
    swap (*first++,*next++);
    if (next==last) next=middle;
    else if (first==middle) middle=next;
  }
}
```

- swap_ranges

```
std::vector<int> foo (5,10);        //
foo: 10 10 10 10 10
std::vector<int> bar (5,33);        //
bar: 33 33 33 33 33
std::swap_ranges(foo.begin()+1,
foo.end()-1, bar.begin());

foo contains: 10 33 33 33 10
bar contains: 10 10 10 33 33

template<class ForwardIterator1, class
ForwardIterator2>
  ForwardIterator2 swap_ranges
(ForwardIterator1 first1,
ForwardIterator1 last1,

ForwardIterator2 first2)
{
  while (first1!=last1) {
    swap (*first1, *first2);
    ++first1; ++first2;
  }
  return first2;
}
```

- transform 转换到另一个数组内

```
for (int i=1; i<6; i++)
foo.push_back (i*10);
    // foo: 10 20 30 40 50

bar.resize(foo.size());
      // allocate space

std::transform (foo.begin(), foo.end(),
bar.begin(), op_increase);

         // bar: 11 21 31 41 51
  // std::plus adds together its two
arguments:
std::transform (foo.begin(), foo.end(),
bar.begin(), foo.begin(), std::plus<int>
());

         // foo: 21 41 61 81 101
```

- **unique**

```
bool myfunction (int i, int j) {
  return (i==j);
}

int myints[] =
{10,20,20,20,30,30,20,20,10};
// 10 20 20 20 30 30 20 20 10
std::vector<int> myvector
(myints,myints+9);

  // using default comparison:
  std::vector<int>::iterator it;
  it = std::unique (myvector.begin(),
myvector.end());   // 10 20 30 20 10 ?  ?
?  ?
                    //

myvector.resize(
std::distance(myvector.begin(),it) ); //
10 20 30 20 10

// using predicate comparison:
std::unique (myvector.begin(),
myvector.end(), myfunction);   // (no
changes)
```

非变序算法

- count

```
  int myints[] =
{10,20,30,30,20,10,10,20};   // 8
elements
  int mycount = std::count (myints,
myints+8, 10);
  std::cout << "10 appears " << mycount
<< " times.\n";//3
```

- find

```
int myints[] = { 10, 20, 30, 40 };
int * p;

p = std::find (myints, myints+4, 30);
if (p != myints+4)//no find


template<class InputIterator, class T>
  InputIterator find (InputIterator
first, InputIterator last, const T& val)
{
  while (first!=last) {
    if (*first==val) return first;
    ++first;
  }
  return last;
}
```

- search

> Searches the range `[first1,last1)` for the first occurrence of the sequence defined by `[first2,last2)`, and returns an iterator to its first element, or last1 if no occurrences are found.

```
std::vector<int> haystack;

  // set some values:        haystack: 10
20 30 40 50 60 70 80 90
  for (int i=1; i<10; i++)
haystack.push_back(i*10);

  // using default comparison:
  int needle1[] = {40,50,60,70};
  std::vector<int>::iterator it;
  it = std::search (haystack.begin(),
haystack.end(), needle1, needle1+4);

  if (it!=haystack.end())
    std::cout << "needle1 found at
position " << (it-haystack.begin()) <<
'\n';
  else
    std::cout << "needle1 not found\n";
```

template

```
ForwardIterator1 search ( ForwardIterator1
first1, ForwardIterator1 last1,
                          ForwardIterator2
first2, ForwardIterator2 last2)
```

{

```
if (first2==last2) return first1;  //
specified in C++11
```

```
while (first1!=last1)
{
  ForwardIterator1 it1 = first1;
  ForwardIterator2 it2 = first2;
  while (*it1==*it2) {    // or: while
(pred(*it1,*it2)) for version 2
      ++it1; ++it2;
      if (it2==last2) return first1;
      if (it1==last1) return last1;
  }
  ++first1;
}
return last1;
```

}

- equal

  ```c++
  template <class InputIterator1, class
InputIterator2>
    bool equal ( InputIterator1 first1,
InputIterator1 last1, InputIterator2 first2
)
  {
    while (first1!=last1) {
      if (!(*first1 == *first2))   // or: if
(!pred(*first1,*first2)), for version 2
        return false;
      ++first1; ++first2;
    }
    return true;
  }
```

- **mismatch**

> Compares the elements in the range [first1,last1) with those in the range beginning at first2, and returns the first element of both sequences that does not match.

```
template <class InputIterator1, class
InputIterator2>
  pair<InputIterator1, InputIterator2>
    mismatch (InputIterator1 first1,
InputIterator1 last1, InputIterator2
first2 )
{
  while ( (first1!=last1) &&
(*first1==*first2) )  // or:
pred(*first1,*first2), for version 2
  { ++first1; ++first2; }
  return std::make_pair(first1,first2);
}
```

- **lexicographical_compare**

```
std::cout <<
std::lexicographical_compare(foo,foo+5,ba
r,bar+9);

bool mycomp (char c1, char c2)
{ return std::tolower(c1)
<std::tolower(c2); }
std::cout <<
std::lexicographical_compare(foo,foo+5,ba
r,bar+9,mycomp);

template <class InputIterator1, class
InputIterator2>
  bool lexicographical_compare
(InputIterator1 first1, InputIterator1
last1,

InputIterator2 first2, InputIterator2
last2)
{
  while (first1!=last1)
  {
    if (first2==last2 || *first2<*first1)
return false;
    else if (*first1<*first2) return
true;
    ++first1; ++first2;
  }
  return (first2!=last2);
}
```

- `lower_bound`找到一个已排序序列中第一个可能插入不变序位置

```
// lower_bound/upper_bound example
#include <iostream>     // std::cout
#include <algorithm>    //
std::lower_bound, std::upper_bound,
std::sort
#include <vector>       // std::vector

int main () {
  int myints[] =
{10,20,30,30,20,10,10,20};
  std::vector<int> v(myints,myints+8);
          // 10 20 30 30 20 10 10 20

  std::sort (v.begin(), v.end());
         // 10 10 10 20 20 20 30 30

  std::vector<int>::iterator low,up;
  low=std::lower_bound (v.begin(),
v.end(), 20); // 3          ^
  up= std::upper_bound (v.begin(),
v.end(), 20); // 6                  ^

  std::cout << "lower_bound at position "
<< (low- v.begin()) << '\n';
  std::cout << "upper_bound at position "
<< (up - v.begin()) << '\n';

  return 0;
}
```

- `equal_range`

```
 std::pair<std::vector<int>::iterator,std
::vector<int>::iterator> bounds;

  // using default comparison:
  std::sort (v.begin(), v.end());
                     // 10 10 10 20 20 20
30 30
  bounds=std::equal_range (v.begin(),
v.end(), 20);            //          ^
     ^

  // using "mygreater" as comp:
  std::sort (v.begin(), v.end(),
mygreater);                   // 30 30 20
20 20 10 10 10
  bounds=std::equal_range (v.begin(),
v.end(), 20, mygreater); //        ^
^
```

- `for_each` `sort` `min` `max`