STL Algorithms

- Standalone, not member functions of container classes
- Do not access container directly
- Operate on data by means of iterators
  - They can work with regular c-style arrays or containers
- Header file:
  - #include <algorithm>

4 types of algorithms

- Nonmodifying algorithms (find, count)
- Modifying algorithms (unique, replace, copy)
- Numeric algorithms (inner_product)
- Heap algorithm (sort,set, heap operations etc.)

- Common ones:
  - count
  - sort
  - find
  - …

A sample of them

- find(begin, end, value); //return an iterator of position of value in the unsorted sequence; if not present, return end; (Note: different from string::find(…) syntax.)
- binary_search(beg,end,value); //return true if value is in the sorted sequence; false otherwise
- search(beg1,end1,beg2,end2); //find 2nd sequence in the 1st sequence; if not present, return end1
- copy(beg, end, con); //copy a sequence into a container con
- count(begin, end, value); //return how many times value occurs in the sequence.
- equal(beg1, end1, beg2);//return true if two sequences are identical, false otherwise
- fill(begin, end, value); //assign value to every element in the sequence
- for_each(begin, end, F); //apply function F to every element in the sequence
- lower_bound(begin, end, value);//return an iterator to the 1st position at which value can inserted and the sequence remain sorted.
- upper_bound(begin, end, value); //return an iterator to the last position at which value can be inserted and the sequence remain sorted.

- max_element(begin, end); //return an iterator to the max value in the sequence.
- min_element(begin, end); //return an iterator to the max value in the sequence.
- next_permutation(begin, end); //shuffle the sequence to its next permutation
- prev_permutation(begin, end); //shuffle the sequence to its previous permutation, and return true (If there is none, return false);
- random_shuffle(beg, end); //shuffle the values in the sequence randomly
- replace(begin, end, old, new);//in the sequence, replace each values old with new
- reverse(begin, end); //reverse the order of the values in the sequence
- sort(begin, end); //sort the sequence into ascending order
- unique(begin, end); //in the sequence, replace any consecutive occurences of the same value with one instance of that value

find (and find_if, more later)

- find() searches for a specified value within a range specified by 2 iterators. If the value is found, an iterator that refers to the found value is returned. If the value is not found, an iterator that refers to the *end* of the range is returned (end is beyond the last element).
- iterator find(start_iterator, end_iterator, search_value);
- find_if() searches within a specified range for the first element that meets a specified criteria. If a value is found, an iterator that refers to the found value is returned. If the value is not found, an iterator to the "end" of the range is returned.
- iterator find_if(start_iterator, end_iterator, boolean_function_that_takes_one_argument);

Exercise:

If only given find(), how would you implement count? (Call it Count())

```
template<class C,class T>
int Count(const C& v,T val)
{
    int n = 0;

    // what goes here?


    return n;
}
```

```
template<class C,class T>
int Count(const C& v,T val)
{
    int n = 0;
    class C::const_iterator  it = find(v.begin(),v.end(),val);   //or use auto
    while (it != v.end()) {
        n++;
        it = find(it +1,v.end(),val);   //Note: it is an iterator; not an int
    }
    return n;
}
```

Example cont.

```
int main()
{
    string m = "Mary had a little lamb";
    int arr[] = { 3, 5, 2, 5, -1, -1, 7, 5, 2, 9 };

    int count_a = Count(m,'a');
    cout << "count_a = " << count_a << endl;

    count_a = count(m.begin(),m.end(),'a');
    cout << "count_a = " << count_a << endl;

    int sz = sizeof(arr) / sizeof(int);  //size of the int array
    int count_5 = count(arr,arr+sz, 5);
    cout << "count_5 = " << count_5 << endl;
    count_5 = count(arr,arr+sz/2, 5);
    cout << "count_5 = " << count_5 << endl;
    return 0;
}
//answer: count_a = 4
count_a = 4
count_5 = 3
half count_5 = 2
```

High Order Function and Function Object

- High order function is a function that takes another function as a parameter.
- Many STL algorithms use this concept, such as find_if; count_if; for_each.

```
int square(int n) { return n * n; }     // returns square of an int
int cube(int n) { return n * n * n; }   // returns cube of an int

// This function is used to compute sum of ints returned
//  by a function, which is pointed by argument p.
int sum(int (*p)(int), int N)
{
        int s = 0;
        for (int i = 1; i <= N; i++) s += p(i);
                return s;
}

int main()
{
  //Compute: 1**2 + 2**2 + ... + 5**2
  cout << "sum(square, 5) = " << sum(square, 5) << endl;

  // Compute: 1**3 + 2**3 + ... + 5**3
  cout << "sum(cube, 5)   = " << sum(cube, 5) << endl;
  return 0;
}
```

count_if; find_if;

- iterator find_if(start_iterator, end_iterator, boolean_function_that_takes_one_argument); //searches within a specified range for the first element that meets a specified criteria. If a value is found, an iterator that refers to the found value is returned. If the value is not found, an iterator to the "end" of the range is returned.
- int count_if(start_iterator, end_iterator, boolean_function_that_takes_one_argument);

Example of find_if

```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;
bool IsOdd(int i) // true if odd, false if even
{ return ((i % 2) == 1); }
int main() {
list <int> li;
for(int nCount = 0; nCount < 6; nCount++)
   li.push_back(nCount);
// What is the * for? what if li were empty?
cout << *(find_if(li.begin(), li.end(), IsOdd));
return 0; }
// Output: 1
```

for_each

- The **for_each** algorithm will access and process each element within a given range. The process that is performed is specified by function.
- for_each(start_iterator, end_iterator, function);

Example of for_each

```
map<string,int> hist;
void record(const string&);  //increase the value for the key in the hist that records frequency (histogram)
void print(const pair<const string,int>&); //print a pair
bool gt_1(const pair<const string,int>&);  //return true if the value in a key/value pair is greater than 1

int main()
{
    string s;

    while(cin >> s) record(s);
    for_each(hist.begin(),hist.end(),print);

    //find the 1st position where value > 1
    typedef map<string, int>::const_iterator MI;  //or auto
    MI i = find_if(hist.begin(),hist.end(),gt_1);
    const pair<string,int>& r = *i;
    cout << setw(8) << setiosflags(ios::left)
        << endl << r.first << ' ' << r.second << endl;

    //find number of entries with values > 1
    int n_gt_1 = count_if(hist.begin(),hist.end(),gt_1);
    cout << "n_gt_1 = " << n_gt_1 << endl;
    return 0;
}
```

Example cont.

```
void record(const string& s) { hist[s]++; }

void print(const pair<const string,int>& r)
{
    cout << setw(8) << setiosflags(ios::left)
         << r.first << ' ' << r.second << endl;
}

bool gt_1(const pair<const string,int>& r)
{ return r.second > 1; }
```

Input and output

```
david
john
mary
david
wayne
jane
bob
jane
wesley
neil
john
jennifer
jane
david
michael
nick
jane
sally
john
david
```

```
bob      1
david    4
jane     4
jennifer 1
john     3
mary     1
michael  1
neil     1
nick     1
sally    1
wayne    1
wesley   1

david    4
n_gt_1 = 3
```