Chapter 21
The STL
(maps and algorithms)

Bjarne Stroustrup
www.stroustrup.com/Programming

Abstract
This talk presents the idea of STL algorithms and introduces map as an example of a container.

Overview
- Common tasks and ideals
- Containers, algorithms, and iterators
- The simplest algorithm: find()
- Parameterization of algorithms
  - find_if() and function objects
- Sequence containers
  - vector and list
- Associative containers
  - map, set
- Standard algorithms
  - copy, sort, ...
  - Input iterators and output iterators
- List of useful facilities
  - Headers, algorithms, containers, function objects

Basic model
- A pair of iterators defines a sequence
  - The beginning (points to the first element – if any)
  - The end (points to the one-beyond-the-last element)

Accumulate (sum the elements of a sequence)

```cpp
template<class In, class T> T accumulate(In first, In last, T init)
{
    while (first!=last) {
        init = init + *first;
        ++first;
    }
    return init;
}
```

```cpp
void f(vector<double>& vd, int* p, int n)
{
    double sum = accumulate(vd.begin(), vd.end(), 0.0); // add the elements of vd
    // note: the type of the 3rd argument, the initializer, determines the precision used
    int si = accumulate(p, p+n, 0); // sum the ints in an int (danger of overflow)
    // p+n means (roughly) &p[n]
    long sl = accumulate(p, p+n, long(0)); // sum the ints in a long
    double s2 = accumulate(p, p+n, 0.0); // sum the ints in a double
    // popular idiom, use the variable you want the result in as the initializer:
    double ss = accumulate(vd.begin(), vd.end(), ss); // do remember the assignment
}
```
Accumulate (generalize: process the elements of a sequence)

// we don't need to use only +, we can use any binary operation (e.g., *)
// any function that "updates the init value" can be used:

```cpp
template<class In, class T, class BinOp>
T accumulate(In first, In last, T init, BinOp op) {
  while (first!=last) {
    init = op(init, *first); // means "init op *first"
    ++first;
  }
  return init;
}
```

Accumulate (what if the data is part of a record?)

```cpp
struct Record {
  int units; // number of units sold
  double unit_price; // ...
};

// let the "update the init value" function extract data from a Record element:
double price(double v, const Record& r) {
  return v + r.unit_price * r.units;
}
```

void f(const vector<Record>& vr, map<string,Record*>& m) {
  double total = accumulate(vr.begin(), vr.end(), 0.0, price);
  // ...
}

Inner product (generalize!)

```cpp
// we can supply our own operations for combining element values with "init":
template<class In, class In2, class T, class BinOp, class BinOp2>
T inner_product(In first, In last, In2 first2, T init, BinOp op, BinOp2 op2) {
  while(first!=last) {
    init = op(init, op2(*first, *first2)); // multiply pairs of elements and sum
    ++first;
    ++first2;
  }
  return first;
}
```

Inner product example

```cpp
// calculate the Dow Jones industrial index:
vector<double> dow_price; // share price for each company
dow_price.push_back(81.86);
dow_price.push_back(54.45); // ... vector<double> dow_weight; // weight in index for each company
dow_weight.push_back(5.8549);
dow_weight.push_back(3.8940); // ...
double dj_index = inner_product(    
  multiply (price,weight) pairs and add
  dow_price.begin(), dow_price.end(),
  dow_weight.begin(),
  0.0);
```
Map (an associative array)

- For a vector, you subscript using an integer
- For a map, you can define the subscript to be (just about) any type

```cpp
int main() {
    typedef map<int, string> my_map;
    my_map m;
    m.insert(make_pair(1, "one"));
    m.insert(make_pair(2, "two"));
    m.insert(make_pair(3, "three"));
    cout << m[1] << endl; // Output: one
}
```

Some implementation-defined type

```cpp
namespace std {
    template<class Key, class Value> class map {
        // ... implementation details...
    };
}
```

Output (word frequencies)

```
Apple   7
Kiwi  2345
Plum  8
Orange 99
Grape 100
Quince 0
```

Map example (build some maps)

```cpp
map<string,double> dow; // Dow Jones industrial index (symbol, price)
map<string,double> dow_weight;
```

An input for the words program (the abstract)

This lecture and the next presents the STL (the containers and algorithms part of the C++ standard library). It is an extensible framework dealing with data in a C++ program. First, I present the general ideal, then the fundamental concepts, and finally examples of containers and algorithms. The key notions of sequence and iterator used to tie containers (data) together with algorithms (processing) are presented. Function objects are used to parameterize algorithms with “policies”.

```
Map

- After vector, map is the most useful standard library container
- Maps (and/or hash tables) are the backbone of scripting languages
- A map is really an ordered balanced binary tree
- By default ordered by < (less than)
- For example, map<string,int> fruits:

```
```cpp
map<string,int> fruits;
fruits.insert(make_pair("Apple", 7));
fruits.insert(make_pair("Kiwi", 2345));
fruits.insert(make_pair("Plum", 8));
```
```cpp
```
Containers and “almost containers”

- Sequence containers
  - vector, list, deque
- Associative containers
  - map, set, multimap, multiset
- “almost containers”
  - array, string, stack, queue, priority_queue
- Soon-to-become standard containers
  - unordered_map (a hash table), unordered_set, ...
- For anything non-trivial, consult documentation
  - Online
    - SGI, RogueWave, Dinkumware
    - Other books
      - Stroustrup: The C++ Programming language (Chapters 16-19, 22.6)
      - Austern: Generic Programming and the STL
      - Josuttis: The C++ Standard Library

Algorithms

- An STL-style algorithm
  - Takes one or more sequences
    - Usually as pairs of iterators
  - Takes one or more operations
    - Usually as function objects
    - Ordinary functions also work
    - Usually reports “failure” by returning the end of a sequence

Some useful standard algorithms

- r=find(b,e,v)  r points to the first occurrence of v in [b,e]
- r=find_if(b,e,p) r points to the first element x in [b,e) so that p(x)
- v=count(b,e,v) x is the number of occurrences of v in [b,e]
- sort(b,e) sort [b,e) using <
- copy(b,e,b2) copy [b,e) to [b2,b2+(e-b))
  - there had better be enough space after b2
- unique_copy(b,e,b2) copy [b,e) to [b2,b2+(e-b)) but
  - don’t copy adjacent duplicates
- merge(b1,e1,b2,e2,r) merge two sorted sequences [b1,e1) and [b2,e2)
  - into [r+(e1-b1)+(e2-b2),r)
- equal(b1,e1,b2,e2) is the subsequence of [b1,e1) with the value v
  - (basically a binary search for v)
- equal_range(b1,e1,v) r is the subsequence of [b1,e1) that contains v
  - copy[v,b1+(x-e1)]
- copy[0,v,b1+(x-e1)]
  - copy(v,b1+(x-e1)]
Input and output iterators

// we can provide iterators for output streams

```cpp
ostream_iterator<string> oo(cout);  // assigning to *oo is to write to cout
*oo = "Hello, ";  // meaning cout << "Hello, "
++oo;  // "get ready for next output operation"
*oo = "world!"
```

// we can provide iterators for input streams:

```cpp
istream_iterator<string> ii(cin);  // reading *ii is to read a string from cin
```

string s1 = *ii;  // meaning cin>>s1
++ii;  // "get ready for next input operation"
string s2 = *ii;  // meaning cin>>s2

Make a quick dictionary (using a vector)

```cpp
int main()
{
    string from, to;
    cin >> from >> to;  // get source and target file names
    ifstream is(from.c_str());  // open input stream
    ofstream os(to.c_str());  // open output stream
    istream_iterator<string> ii(is);  // make input iterator for stream
    ostream_iterator<string> oo(os,"\n");  // make output iterator for stream
    // append "\n" each time
    vector<string> b(ii,eos);  // b is a vector initialized from input
    unique_copy(b.begin(), b.end(), oo);  // copy buffer to output,  
    // discard replicated values
}
```

Make a quick dictionary (using a set)

```cpp
int main()
{
    string from, to;
    cin >> from >> to;  // get source and target file names
    ifstream is(from.c_str());  // open input stream
    ofstream os(to.c_str());  // open output stream
    istream_iterator<string> ii(is);  // make input iterator for stream
    istringstream eos;  // input sentinel (defaults to EOF)
    ostream_iterator<string> oo(os,"\n");  // make output iterator for stream
    // append "\n" each time
    vector<string> b(ii,eos);  // b is a vector initialized from input
    unique_copy(b.begin(), b.end(), oo);  // copy buffer to output,
    // discard replicated values
}
```

An input file (the abstract)

This lecture and the next presents the STL (the containers and algorithms part of the C++ standard library). It is an extensible framework dealing with data in a C++ program. First, I present the general ideal, then the fundamental concepts, and finally examples of containers and algorithms. The key notions of sequence and iterator used to tie containers (data) together with algorithms (processing) are presented. Function objects are used to parameterize algorithms with “policies”.

Make a quick dictionary (using a set)

```cpp
int main()
{
    string from, to;
    cin >> from >> to;  // get source and target file names
    ifstream is(from.c_str());  // make input stream
    ofstream os(to.c_str());  // make output stream
    istream_iterator<string> ii(is);  // make input iterator for stream
    istream_iterator<string> eos;  // input sentinel (defaults to EOF)
    ostream_iterator<string> oo(os,"\n");  // make output iterator for stream
    // append "\n" each time
    vector<string> b(ii,eos);  // b is a vector initialized from input
    unique_copy(b.begin(), b.end(), oo);  // copy buffer to output,
    // discard replicated values
}
```

• We are doing a lot of work that we don’t really need
  – Why store all the duplicates? (in the vector)
  – Why sort?
  – Why suppress all the duplicates on output?
• Why not just
  – Put each word in the right place in a dictionary as we read it?
  – In other words: use a set

Part of the output

- Simple definition: a set is a map with no values, just keys
Set

- A set is really an ordered balanced binary tree
- By default ordered by <
- For example, set<string> fruits;

![Set Diagram]

copy_if()

// a very useful algorithm (missing from the standard library):

template<class In, class Out, class Pred>
Out copy_if(In first, In last, Out res, Pred p)
// copy elements that fulfill the predicate
{
    while (first != last)
    {
        if (p(*first)) *res++ = *first;
        ++first;
    }
    return res;
}

Some standard function objects

- From <functional>
  - Binary
    - plus, minus, multiplies, divides, modulus
    - equal_to, not_equal_to, greater, less, greater_equal, less_equal,
      logical_and, logical_or
  - Unary
    - negate
    - logical_not
  - Unary (missing, write them yourself)
    - less_than, greater_than, less_than_or_equal, greater_than_or_equal