159.234 Lecture 24

- A closer look at vector
- **STL Algorithms**

Vector member functions

- **`v.push_back(value)`**
  - Add element to end (found in all sequence containers).
- **`v.size()`**
  - Current size of vector
- **`v.capacity()`**
  - How much vector can hold before reallocating memory
  - Reallocation doubles size
- **`vector<`*`type`*`> v(a, a + SIZE)`**
  - Creates **`vector v`** with elements from array **`a`** up to (not including) **`a + SIZE`**

Vector member functions

- **`v.insert(`***iterator*`,`** ***value*** **`)`**
  - Inserts *value* before location of *iterator*
- **`v.insert(`***iterator*`,`** ***array*** **`,`** ***array*** + ***SIZE*****`)`**
  - Inserts array elements (up to, but not including *array* + *SIZE)* into vector
- **`v.erase( iterator )`**
  - Remove element from container
- **`v.erase( iter1, iter2 )`**
  - Remove elements starting from **`iter1`** and up to (not including) **`iter2`**
- **`v.clear()`**
  - Erases entire container

```
int main(){
  std::vector<char> v; // create zero-length vector
  int i;

  // put values into a vector
  for(i=0; i<10; ++i)  { v.push_back('A' + i); }

  // can access vector contents using subscripting
  for(i=0; i<10; ++i) { cout << v[i] << " ";}
  cout << endl;

  // access via iterator
 std:: vector<char>::iterator p = v.begin();
  while(p != v.end()) {
    cout << *p << " ";
    p++;
  }
}
```

Algorithms

Algorithms act on the contents of containers.

They include capabilities for:

- initializing
- sorting
- searching and
- transforming the contents of containers.

All algorithms are template functions.

To access them: `#include <algorithms>`

Algorithms

```
string words[5] = { "my", "hop", "mop", "hope",
                            "cope"};
string*   where;
where = find(words, words + 5, "hop");

cout << *++where << endl;                    //mop
```

Algorithms

```
vector<int> v;
int i;

for(i=0; i<10; ++i) v.push_back(i);

cout << "Initial: ";
for(i=0; i<v.size(); ++i)
   cout << v[i] << " ";
cout << endl;

reverse(v.begin(), v.end());
```

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

double reciprocal( double d){  return 1.0 / d; }

template <class T>
void printVector( vector<T> v ){
  for(int j=0;j<v.size();j++){
    cout << v[j] << " ";
  }
  cout << endl;
}

int main(){

  vector<double> vals;
  for(int i=1;i<10;i++) vals.push_back( (double)i );

  printVector( vals );

  transform( vals.begin(), vals.end(), vals.begin(), reciprocal );

  printVector( vals );

  reverse( vals.begin(), vals.end() );

  printVector( vals );

  return 0;
}
```

- transform & reverse algorithms
- Example output:

```
1 2 3 4 5 6 7 8 9
1 0.5 0.333333 0.25 0.2 0.166667 0.142857 0.125 0.111111
0.111111 0.125 0.142857 0.166667 0.2 0.25 0.333333 0.5 1
```

- Note use of function name identifier as an argument to transform
- transform takes arguments: start-iter, end-iter, result-iter, func
- Note use of template mechanism to write a generalised printVector function

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

double reciprocal( double d){  return 1.0 / d; }

template <class T>
void printVector( vector<T> v ){
  for(int j=0;j<v.size();j++){
    cout << v[j] << " ";
  }
  cout << endl;
}

int main(){
  vector<int> v1;
  for(int i=1;i<20;i+=2) v1.push_back( i );   // odd

  vector<int> v2;
  for(int i=0;i<20;i+=2) v2.push_back( i );  // even


  printVector( v1 );
  printVector( v2 );

  vector<int> v3( v1.size() + v2.size() );  // must have enough space

  printVector( v3 );

  merge( v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin() );

  printVector( v3 );

  return 0;
}
```

- merge algorithm

- Example output:

```
1 3 5 7 9 11 13 15 17 19
0 2 4 6 8 10 12 14 16 18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

- Note arguments:
  - start1, end1, start2, end2, result
- Returns end-iter of result (not used in this example)

```
merge( v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin() );
printVector( v3 );

random_shuffle( v3.begin(), v3.end() );
printVector( v3 );

cout << "end  - begin = " << v3.end() - v3.begin() << endl;

cout << "Max is " << * max_element( v3.begin(), v3.end() ) << endl;
cout << "Min is " << * min_element( v3.begin(), v3.end() ) << endl;

sort( v3.begin(), v3.end() );
printVector( v3 );

for_each( v3.begin(), v3.end(), sum );
cout << "Sum  was " << total << endl;

vector<int> v4;
v4.push_back(11);   v4.push_back(12);  v4.push_back(13);

cout << "subsequence  included is " << includes( v3.begin(), v3.end(),

                        v4.begin(), v4.end() ) << endl;
v4[1] = 10;
cout << "subsequence  included is " << includes( v3.begin(), v3.end(),

                        v4.begin(), v4.end() ) << endl;


 return 0;
}
```

- Algorithms:
  - merge
  - random_shuffle
  - sort
  - for_each
  - Includes
- Example Output:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
13 19 8 10 14 16 17 6 7 12 15 3 9 1 11 4 2 0 5 18
end  - begin = 20
Max is 19
Min is 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Sum  was 190
subsequence  included is 1
subsequence  included is 0
```

Algorithms in the STL

- adjacent_find
- binary_search
- copy
- copy_backward
- count
- count_if
- equal
- equal_range
- fill and fill_n
- find
- find_end
- find_first_of
- find_if
- for_each
- generate and generate_n
- includes
- inplace_merge
- iter_swap
- lexicographical_compare
- lower_bound
- make_heap
- max
- max_element
- merge
- min
- min_element
- mismatch
- next_permutation

(More) Algorithms in the STL

- nth_element
- partial_sort
- partial_sort_copy
- partition
- pop_heap
- prev_permutation
- push_heap
- random_shuffle
- remove and remove_if…
- replace and replace_if…
- reverse and reverse_copy
- rotate and rotate_copy
- search
- search_n
- set_difference
- set_intersection
- set_symmetric_difference
- set_union
- sort
- sort_heap
- stable_partition
- stable_sort
- swap
- swap_ranges
- transform
- unique and unique_copy
- upper_bound

Summary

- Algorithms in the STL are just template functions
- There are some useful ones that may save you reinventing the wheel.
- Library functions have the great advantage - someone else has tested them!
- (RTFM) Read the Fine Manual :-)