Practice of for_each()

- Using for_each(), increment each element in an int vector (v) by 3.

Wrong: Does not change value (copy by value)

- void incr(int i) { i +=3;}
- for_each(v.begin(), v.end(), incr);

Solution:

```
template <class T>
void incr(T &i) { i +=3;}

for_each(v.begin(), v.end(), incr);
```

sort

void sort(RandomAccessIterator first, RandomAccessIterator last);

- Sorts the elements in the range [first,last) into ascending order.
- Another version:

void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);

- The elements are compared using operator< for the first version, and *comp* for the second.
- Binary function that accepts two elements in the range as arguments, and returns a value convertible to bool.

binary_search

- bool binary_search(ForwardIterator first, ForwardIterator last, const T& val);
  - The sequence must be sorted.
  - Return true if found.

Searching and sorting

```
using namespace std;
bool greater10(int value) { return (value > 10); }
int main() {
  const int N = 10;
  int a[N] = { 10, 2, 17, 5, 16, 8, 12, 11, 20, 7 };
  vector <int> v(a, a+N); // contents of array a in vector
  auto loc = find(v.begin(), v.end(), 16);
  if(loc != v.end()) cout << "Found 16 at " << (loc - v.begin()) << endl;
  else cout << "16 not found \n" ;
  loc = find_if(v.begin(), v.end(), greater10);
  if(loc != v.end()) cout << "First value > 10 is " << *loc << endl;
  else cout << "Nothing > 10 found \n" ;
  if(binary_search(v.begin(), v.end(), 12)) cout << "12 found\n";
  else cout << "12 not found\n";
  sort(v.begin(), v.end());
  if(binary_search(v.begin(), v.end(), 12)) cout << "12 found\n";
  else cout << "12 not found\n" ;
  return 0; }
```

copy

- iterator copy(first, last, destination);
- The copy function will copy all of the elements in the range [first, last) to destination and returns an iterator indicating the end of the range of copied elements.
- Copy assumes that the destination already has room for the elements being copied. It would be an error to copy into an empty list or vector (buffer overflow). However, this can be overcome by iterator adapters.
- One interesting use for this function is to output the elements of a container (see next slides for explanation)
  - *ostream_iterator<int> screen(cout, ", ");*
  - *copy(vecList.begin(), vecList.end(), screen);*

Iterator adaptor: Stream iterators

- *We visited reverse iteratos and insert iterators before.*
- *Stream iterator is another type of iterator adaptor.*
- *istream_iterator*
  - *Used to input data from an input stream*
- *ostream_iterator*
  - *Used to output data to an output stream*

istream_iterator

- Syntax:
  - *istream_iterator<Type> isIdentifier(istream &);*
- Example on next page:
  - Read integer from cin and copy into v
  - The 2nd argument is a special istream_iterator that works like end()

Example of istream_iterator

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <iterator>
using namespace std;
int main(void) {
  vector <int> v;
  copy(istream_iterator <int>(cin),
   istream_iterator <int>(), // works like end
   back_inserter(v));
   for(auto it = v.begin(); it != v.end(); it++)
        cout << *it << " ";
return 0; }
```

ostream_iterator

- *Syntax:*
  - *ostream_iterator<Type> osIdentifier(ostream &);*
  - *ostream_iterator<Type> osIdentifier(ostream &, char * delimiter);*

Example of ostream_iterator

```
// include <vector> <algorithm> <iostream>
#include <iterator> // std::ostream_iterator
using namespace std;
int main(void) {
vector <int> v;
for(int i=0; i<5; i++)
    v.push_back(i);
ostream_iterator <int> outIter(cout, " ");
copy(v.begin(), v.end(), outIter);
return 0; }
// Output: 0 1 2 3 4
```

Revist copy and the need of back_inserter

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
   int arr[] = { 1,2,3 };
   std::vector <int> v(arr, arr + sizeof(arr) / sizeof(arr[0]));
   std::vector <int> v1(3); // 3 elements, {0,0,0}
   copy(v.begin(), v.end(), v1.begin()+1);
   for(auto it = v1.begin(); it != v1.end(); it++)
         cout << *it << " ";
   return 0; }
// Output: 0 1 2
// Why does it start with 0?
// Where did the 3 go?  →  overflow, need the  iterator adaptor such as back_inserter
```

revisit iterator adaptor

```
#include <iostream>
#include <iterator> // std::back_inserter
#include <vector>
#include <algorithm>
using namespace std;
int main() {
vector <int> v, w;
for(int i = 3; i <= 5; i++) {
   v.push_back(i);
   w.push_back(i * 10); }
copy(w.begin(), w.end(), back_inserter(v));
cout << "v contains: ";
for(auto it = v.begin(); it != v.end(); ++it)
   cout << *it << " ";
return 0; }
// v contains: 3 4 5 30 40 50
```

min; max; swap; random_shuffle

- min: determines the minimum of two values
    - data_type_of_elements min(element1, element2);
    - data_type_of_elements min_element(start_iterator, end_iterator);
- max
- swap:
    - `void swap( object1, object2 );`
- random_shuffle: randomly orders the elements within a given range
    - `void random_shuffle( start_iterator, end_iterator );`