Outline

- Introduction to Templates
  - Function Templates
  - Class Templates
  - Default Template Types & Non-type Template Parameters
- Standard Template Library
  - Introduction to STL
  - Sequence Containers (`vector, list, deque`)
  - Associative Containers (`map/multimap, set/multiset`)
  - Function Objects
  - Container Adapters (`stack, queue, priority_queue`)
  - Algorithms
  - Near Containers (`bitset`)

STL Algorithms

- STL algorithms do not depend on the implementation details of the containers on which they operate
  - Algorithms can be added easily to the STL without modifying the container classes
- STL algorithms can work on
  - C-style, pointer-based arrays
  - STL containers
  - User-defined data structures

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

fill and fill_n

- Assigns val to all the elements in the range [first,last)

```
template <class ForwardIterator, class T>
void fill (ForwardIterator first, ForwardIterator last, const T& val)
{
  while (first != last) {
    *first = val;
    ++first;
  }
}
```

- Assigns val to the first n elements of the sequence pointed by first

```
template <class OutputIterator, class Size, class T>
void fill_n (OutputIterator first, Size n, const T& val)
{
  while (n>0) {
    *first = val;
    ++first; --n;
  }
}
```

generate and generate_n

```
template <class ForwardIterator, class Generator>
void generate ( ForwardIterator first, ForwardIterator last,
Generator gen )
{
  while (first != last) {
    *first = gen();
    ++first;
  }
}
```

```
template <class OutputIterator, class Size, class Generator>
void generate_n ( OutputIterator first, Size n, Generator gen )
{
  while (n>0) {
    *first = gen();
    ++first; --n;
  }
}
```

Using fill and fill_n

```
 3  #include <iostream>
 4  #include <algorithm> // algorithm definitions
 5  #include <vector> // vector class-template definition
 6  #include <iterator> // ostream_iterator
 7  using namespace std;
 8
 9  char nextLetter(); // prototype of generator function
10
11  int main()
12  {
13     vector< char > chars( 10 );
14     ostream_iterator< char > output( cout, " " );
15     fill( chars.begin(), chars.end(), '5' ); // fill chars with 5s
16
17     cout << "Vector chars after filling with 5s:\n";
18     copy( chars.begin(), chars.end(), output );
19
20     // fill first five elements of chars with As
21     fill_n( chars.begin(), 5, 'A' );
22
23     cout << "\n\nVector chars after filling five elements with As:\n";
24     copy( chars.begin(), chars.end(), output );
```

```
Vector chars after filling with 5s:
5 5 5 5 5 5 5 5 5 5

Vector chars after filling five elements with As:
A A A A A 5 5 5 5 5
```

Using generate and generate_n

```
26      // generate values for all elements of chars with nextLetter
27      generate( chars.begin(), chars.end(), nextLetter );
28
29      cout << "\n\nVector chars after generating letters A-J:\n";
30      copy( chars.begin(), chars.end(), output );
31
32      // generate values for first five elements of chars with nextLetter
33      generate_n( chars.begin(), 5, nextLetter );
34
35      cout << "\n\nVector chars after generating K-O for the"
36         << " first five elements:\n";
37      copy( chars.begin(), chars.end(), output );
38      cout << endl;
39   } // end main
40
41   // generator function returns next letter (starts with A)
42   char nextLetter()
43   {
44      static char letter = 'A';
45      return letter++;
46   } // end function nextLetter
```

```
Vector chars after generating letters A-J:
A B C D E F G H I J

Vector chars after generating K-O for the first fiv
K L M N O F G H I J
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

equal

- Compares the elements in the range [first1,last1) with those in the range beginning at first2, and returns true if all of the elements in both ranges match

```
template <class InputIterator1, class InputIterator2>
bool equal(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2)
{
  while (first1!=last1) {
    if (!(*first1 == *first2))
      return false;
    ++first1; ++first2;
  }
  return true;
}
```

```
template <class InputIterator1, class InputIterator2>
bool equal(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2,
BinaryPredicate pred)
{
  while (first1!=last1) {
    if (!pred(*first1,*first2))
      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) )
  { ++first1; ++first2; }
  return std::make_pair(first1,first2);
}
```

```
template <class InputIterator1, class InputIterator2>
pair<InputIterator1, InputIterator2> mismatch (InputIterator1 first1,
InputIterator1 last1, InputIterator2 first2 )
{
  while ( pred(*first1,*first2) )
  { ++first1; ++first2; }
  return std::make_pair(first1,first2);
}
```

lexicographical_compare

- Returns true if the range [first1,last1) compares lexicographically less than the range [first2,last2)

```
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);
}
```

Using equal

```
3  #include <iostream>
4  #include <algorithm> // algorithm definitions
5  #include <vector> // vector class-template definition
6  #include <iterator> // ostream_iterator
7  using namespace std;
8
9  int main()
10 {
11    const int SIZE = 10;
12    int a1[ SIZE ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
13    int a2[ SIZE ] = { 1, 2, 3, 4, 1000, 6, 7, 8, 9, 10 };
14    vector< int > v1( a1, a1 + SIZE ); // copy of a1
15    vector< int > v2( a1, a1 + SIZE ); // copy of a1
16    vector< int > v3( a2, a2 + SIZE ); // copy of a2
17    ostream_iterator< int > output( cout, " " );
18
19    cout << "Vector v1 contains: ";
20    copy( v1.begin(), v1.end(), output );
21    cout << "\nVector v2 contains: ";
22    copy( v2.begin(), v2.end(), output );
23    cout << "\nVector v3 contains: ";
24    copy( v3.begin(), v3.end(), output );
25
26    // compare vectors v1 and v2 for equality
27    bool result = equal( v1.begin(), v1.end(), v2.begin() );
28    cout << "\n\nVector v1 " << ( result ? "is" : "is not" )
29       << " equal to vector v2.\n";
30
31    // compare vectors v1 and v3 for equality
32    result = equal( v1.begin(), v1.end(), v3.begin() );
33    cout << "Vector v1 " << ( result ? "is" : "is not" )
34       << " equal to vector v3.\n";
```

```
Vector v1 contains: 1 2 3 4 5 6 7 8 9 10
Vector v2 contains: 1 2 3 4 5 6 7 8 9 10
Vector v3 contains: 1 2 3 4 1000 6 7 8 9 10

Vector v1 is equal to vector v2.
Vector v1 is not equal to vector v3.
```

Using mismatch and lexicographical_compare

```
36      // location represents pair of vector iterators
37      pair< vector< int >::iterator, vector< int >::iterator > location;
38
39      // check for mismatch between v1 and v3
40      location = mismatch( v1.begin(), v1.end(), v3.begin() );
41      cout << "\nThere is a mismatch between v1 and v3 at location "
42         << ( location.first - v1.begin() ) << "\nwhere v1 contains "
43         << *location.first << " and v3 contains " << *location.second
44         << "\n\n";
45
46      char c1[ SIZE ] = "HELLO";
47      char c2[ SIZE ] = "BYE BYE";
48
49      // perform lexicographical comparison of c1 and c2
50      result = lexicographical_compare( c1, c1 + SIZE, c2, c2 + SIZE );
51      cout << c1 << ( result ? " is less than " :
52         " is greater than or equal to " )  << c2 << endl;
53   } // end main
```

```
There is a mismatch between v1 and v3 at location 4
where v1 contains 5 and v3 contains 1000

HELLO is greater than or equal to BYE BYE
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

remove

- Transforms the range [first,last) into a range with all the elements that compare equal to val removed, and returns an iterator to the new end of that range

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

remove_if

- Transforms the range [first,last) into a range with all the elements for which pred returns true removed, and returns an iterator to the new end of that range

```
template <class ForwardIterator, class UnaryPredicate>
ForwardIterator remove_if (ForwardIterator first,
ForwardIterator last, UnaryPredicate pred)
{
  ForwardIterator result = first;
  while (first!=last) {
    if (!pred(*first)) {
      *result = *first;
      ++result;
    }
    ++first;
  }
  return result;
}
```

remove_copy

- Copies the elements in the range [first,last) to the range beginning at result, except those elements that compare equal to val

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

remove_copy_if

- Copies the elements in the range [first,last) to the range beginning at result, except those elements for which pred returns true

```
template <class InputIterator, class OutputIterator, class
UnaryPredicate>
OutputIterator remove_copy_if (InputIterator first,
InputIterator last, OutputIterator result, UnaryPredicate pred)
{
  while (first!=last) {
    if (!pred(*first)) {
      *result = *first;
      ++result;
    }
    ++first;
  }
  return result;
}
```

Using remove

```
 4  #include <iostream>
 5  #include <algorithm> // algorithm definitions
 6  #include <vector> // vector class-template definition
 7  #include <iterator> // ostream_iterator
 8  using namespace std;
 9
10  bool greater9( int ); // prototype
11
12  int main()
13  {
14     const int SIZE = 10;
15     int a[ SIZE ] = { 10, 2, 10, 4, 16, 6, 14, 8, 12, 10 };
16     ostream_iterator< int > output( cout, " " );
17     vector< int > v( a, a + SIZE ); // copy of a
18     vector< int >::iterator newLastElement;
19
20     cout << "Vector v before removing all 10s:\n   ";
21     copy( v.begin(), v.end(), output );
22
23     // remove all 10s from v
24     newLastElement = remove( v.begin(), v.end(), 10 );
25     cout << "\nVector v after removing all 10s:\n   ";
26     copy( v.begin(), newLastElement, output );
27
28     vector< int > v2( a, a + SIZE ); // copy of a
29     vector< int > c( SIZE, 0 ); // instantiate vector c
30     cout << "\n\nVector v2 before removing all 10s and copying:\n   ";
31     copy( v2.begin(), v2.end(), output );
```

```
Vector v before removing all 10s:
   10 2 10 4 16 6 14 8 12 10
Vector v after removing all 10s:
   2 4 16 6 14 8 12

Vector v2 before removing all 10s and copying:
   10 2 10 4 16 6 14 8 12 10
```

Using remove_copy, remove_if, and remove_copy_if

```
33      // copy from v2 to c, removing 10s in the process
34      remove_copy( v2.begin(), v2.end(), c.begin(), 10 );
35      cout << "\nVector c after removing all 10s from v2:\n   ";
36      copy( c.begin(), c.end(), output );
37
38      vector< int > v3( a, a + SIZE ); // copy of a
39      cout << "\n\nVector v3 before removing all elements"
40         << "\ngreater than 9:\n   ";
41      copy( v3.begin(), v3.end(), output );
42
43      // remove elements greater than 9 from v3
44      newLastElement = remove_if( v3.begin(), v3.end(), greater9 );
45      cout << "\nVector v3 after removing all elements"
46         << "\ngreater than 9:\n   ";
47      copy( v3.begin(), newLastElement, output );
48
49      vector< int > v4( a, a + SIZE ); // copy of a
50      vector< int > c2( SIZE, 0 ); // instantiate vector c2
51      cout << "\n\nVector v4 before removing all elements"
52         << "\ngreater than 9 and copying:\n   ";
53      copy( v4.begin(), v4.end(), output );
54
55      // copy elements from v4 to c2, removing elements greater
56      // than 9 in the process
57      remove_copy_if( v4.begin(), v4.end(), c2.begin(), greater9 );
58      cout << "\nVector c2 after removing all elements"
59         << "\ngreater than 9 from v4:\n   ";
60      copy( c2.begin(), c2.end(), output );
61      cout << endl;
62   } // end main
```

```
bool greater9( int x )
{
   return x > 9;
} // end function greater9
```

```
Vector c after removing all 10s from v2:
   2 4 16 6 14 8 12 0 0 0

Vector v3 before removing all elements
greater than 9:
   10 2 10 4 16 6 14 8 12 10
Vector v3 after removing all elements
greater than 9:
   2 4 6 8

Vector v4 before removing all elements
greater than 9 and copying:
   10 2 10 4 16 6 14 8 12 10
Vector c2 after removing all elements
greater than 9 from v4:
   2 4 6 8 0 0 0 0 0 0
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

replace

- Assigns new_value to all the elements in the range [first,last) that compare equal to old_value

```
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

- Assigns new_value to all the elements in the range [first,last) for which pred returns true

```
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;
  }
}
```

replace_copy

- Copies the elements in the range [first,last) to the range beginning at result, replacing the appearances of old_value by new_value

```
template <class InputIterator, class OutputIterator, class T>
OutputIterator replace_copy (InputIterator first, InputIterator
last, OutputIterator result, const T& old_value, const T&
new_value)
{
  while (first!=last) {
    *result = (*first==old_value)? new_value: *first;
    ++first; ++result;
  }
  return result;
}
```

replace_copy_if

- Copies the elements in the range [first,last) to the range beginning at result, replacing those for which pred returns true by new_value

```
template <class InputIterator, class OutputIterator, class
UnaryPredicate, class T>
OutputIterator replace_copy_if (InputIterator first,
InputIterator last, OutputIterator result, UnaryPredicate pred,
const T& new_value)
{
  while (first!=last) {
    *result = (pred(*first))? new_value: *first;
    ++first; ++result;
  }
  return result;
}
```

Using replace and replace_copy

```
4  #include <iostream>
5  #include <algorithm>
6  #include <vector>
7  #include <iterator> // ostream_iterator
8  using namespace std;
9
10 bool greater9( int ); // predicate function pro
11
12 int main()
13 {
14    const int SIZE = 10;
15    int a[ SIZE ] = { 10, 2, 10, 4, 16, 6, 14, 8, 12, 10 };
16    ostream_iterator< int > output( cout, " " );
17
18    vector< int > v1( a, a + SIZE ); // copy of a
19    cout << "Vector v1 before replacing all 10s:\n   ";
20    copy( v1.begin(), v1.end(), output );
21
22    // replace all 10s in v1 with 100
23    replace( v1.begin(), v1.end(), 10, 100 );
24    cout << "\nVector v1 after replacing 10s with 100s:\n   ";
25    copy( v1.begin(), v1.end(), output );
26
27    vector< int > v2( a, a + SIZE ); // copy of a
28    vector< int > c1( SIZE ); // instantiate vector c1
29    cout << "\n\nVector v2 before replacing all 10s and copying:\n   ";
30    copy( v2.begin(), v2.end(), output );
31
32    // copy from v2 to c1, replacing 10s with 100s
33    replace_copy( v2.begin(), v2.end(), c1.begin(), 10, 100 );
34    cout << "\nVector c1 after replacing all 10s in v2:\n   ";
35    copy( c1.begin(), c1.end(), output );
```

```
Vector v1 before replacing all 10s:
   10 2 10 4 16 6 14 8 12 10
Vector v1 after replacing 10s with 100s:
   100 2 100 4 16 6 14 8 12 100

Vector v2 before replacing all 10s and copying:
   10 2 10 4 16 6 14 8 12 10
Vector c1 after replacing all 10s in v2:
   100 2 100 4 16 6 14 8 12 100
```

Using replace_if and replace_copy_if

```
37      vector< int > v3( a, a + SIZE ); // copy of a
38      cout << "\n\nVector v3 before replacing values greater than 9:\n   ";
39      copy( v3.begin(), v3.end(), output );
40
41      // replace values greater than 9 in v3 with 100
42      replace_if( v3.begin(), v3.end(), greater9, 100 );
43      cout << "\nVector v3 after replacing all values greater"
44         << "\nthan 9 with 100s:\n   ";
45      copy( v3.begin(), v3.end(), output );
46
47      vector< int > v4( a, a + SIZE ); // copy of a
48      vector< int > c2( SIZE ); // instantiate vector c2‘
49      cout << "\n\nVector v4 before replacing all values greater "
50         << "than 9 and copying:\n   ";
51      copy( v4.begin(), v4.end(), output );
52
53      // copy v4 to c2, replacing elements greater than 9 with 100
54      replace_copy_if( v4.begin(), v4.end(), c2.begin(), greater9, 100 );
55      cout << "\nVector c2 after replacing all values greater "
56         << "than 9 in v4:\n   ";
57      copy( c2.begin(), c2.end(), output );
58      cout << endl;
59   } // end main
```

```
bool greater9( int x )
{
   return x > 9;
} // end function greater9
```

```
Vector v3 before replacing values greater than 9:
   10 2 10 4 16 6 14 8 12 10
Vector v3 after replacing all values greater
than 9 with 100s:
   100 2 100 4 100 6 100 8 100 100

Vector v4 before replacing all values greater than 9 and copying:
   10 2 10 4 16 6 14 8 12 10
Vector c2 after replacing all values greater than 9 in v4:
   100 2 100 4 100 6 100 8 100 100
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

random_shuffle

- Rearranges the elements in the range [first,last) randomly

```
template <class RandomAccessIterator, class RandomNumberGenerator>
void random_shuffle (RandomAccessIterator first,
RandomAccessIterator last, RandomNumberGenerator& gen)
{
  iterator_traits<RandomAccessIterator>::difference_type i, n;
  n = (last-first);
  for (i=n-1; i>0; --i) {
    swap (first[i],first[gen(i+1)]);
  }
}
```

Using random_shuffle

```
 3  #include <iostream>
 4  #include <algorithm> // algorithm definitions
 5  #include <numeric> // accumulate is defined here
 6  #include <vector>
 7  #include <iterator>
 8  using namespace std;
 9
10  bool greater9( int ); // predicate function prototype
11  void outputSquare( int ); // output square of a value
12  int calculateCube( int ); // calculate cube of a value
13
14  int main()
15  {
16     const int SIZE = 10;
17     int a1[ SIZE ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
18     vector< int > v( a1, a1 + SIZE ); // copy of a1
19     ostream_iterator< int > output( cout, " " );
20
21     cout << "Vector v before random_shuffle: ";
22     copy( v.begin(), v.end(), output );
23
24     random_shuffle( v.begin(), v.end() ); // shuffle elements of v
25     cout << "\nVector v after random_shuffle: ";
26     copy( v.begin(), v.end(), output );
```

```
Vector v before random_shuffle: 1 2 3 4 5 6 7 8 9 10
Vector v after random_shuffle: 5 4 1 3 7 8 9 10 6 2
```

Using count, count_if, min_element, max_element, and accumulate

```
28      int a2[ SIZE ] = { 100, 2, 8, 1, 50, 3, 8, 8, 9, 10 };
29      vector< int > v2( a2, a2 + SIZE ); // copy of a2
30      cout << "\n\nVector v2 contains: ";
31      copy( v2.begin(), v2.end(), output );
32
33      // count number of elements in v2 with value 8
34      int result = count( v2.begin(), v2.end(), 8 );
35      cout << "\nNumber of elements matching 8: " << result;
36
37      // count number of elements in v2 that are greater than 9
38      result = count_if( v2.begin(), v2.end(), greater9 );
39      cout << "\nNumber of elements greater than 9: " << result;
40
41      // locate minimum element in v2
42      cout << "\n\nMinimum element in Vector v2 is: "
43         << *( min_element( v2.begin(), v2.end() ) );
44
45      // locate maximum element in v2
46      cout << "\nMaximum element in Vector v2 is: "
47         << *( max_element( v2.begin(), v2.end() ) );
48
49      // calculate sum of elements in v
50      cout << "\n\nThe total of the elements in Ve
51         << accumulate( v.begin(), v.end(), 0 );
```

```
Vector v2 contains: 100 2 8 1 50 3 8 8 9 10
Number of elements matching 8: 3
Number of elements greater than 9: 3

Minimum element in Vector v2 is: 1
Maximum element in Vector v2 is: 100

The total of the elements in Vector v is: 55
```

Using for_each and transform

```
53      // output square of every element in v
54      cout << "\n\nThe square of every integer in Vector v is:\n";
55      for_each( v.begin(), v.end(), outputSquare );
56
57      vector< int > cubes( SIZE ); // instantiate vector cubes
58
59      // calculate cube of each element in v; place results in cubes
60      transform( v.begin(), v.end(), cubes.begin(), calculateCube );
61      cout << "\n\nThe cube of every integer in Vector v is:\n";
62      copy( cubes.begin(), cubes.end(), output );
63      cout << endl;
64   } // end main
65
66   // determine whether argument is greater than 9
67   bool greater9( int value )
68   {
69      return value > 9;
70   } // end function greater9
71
72   // output square of argument
73   void outputSquare( int value )
74   {
75      cout << value * value << ' ';
76   } // end function outputSquare
77
78   // return cube of argument
79   int calculateCube( int value )
80   {
81      return value * value * value;
82   } // end function calculateCube
```

```
The square of every integer in Vector v is:
25 16 1 9 49 64 81 100 36 4

The cube of every integer in Vector v is:
125 64 1 27 343 512 729 1000 216 8
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

Using find

```
3  #include <iostream>
4  #include <algorithm> // algorithm definitions
5  #include <vector> // vector class-template definition
6  #include <iterator>
7  using namespace std;
8
9  bool greater10( int value ); // predicate function prototype
10
11 int main()
12 {
13    const int SIZE = 10;
14    int a[ SIZE ] = { 10, 2, 17, 5, 16, 8, 13, 11, 20, 7 };
15    vector< int > v( a, a + SIZE ); // copy of a
16    ostream_iterator< int > output( cout, " " );
17
18    cout << "Vector v contains: ";
19    copy( v.begin(), v.end(), output ); // display output vector
20
21    // locate first occurrence of 16 in v
22    vector< int >::iterator location;
23    location = find( v.begin(), v.end(), 16 );
24
25    if ( location != v.end() ) // found 16
26       cout << "\n\nFound 16 at location " << ( location - v.begin() );
27    else // 16 not found
28       cout << "\n\n16 not found";
29
30    // locate first occurrence of 100 in v
31    location = find( v.begin(), v.end(), 100 );
32
33    if ( location != v.end() ) // found 100
34       cout << "\nFound 100 at location " << ( location - v.begin() );
35    else // 100 not found
36       cout << "\n100 not found";
```

```
Vector v contains: 10 2 17 5 16 8 13 11 20 7

Found 16 at location 4
100 not found
```

Using find_if, sort, and binary_search

```
38      // locate first occurrence of value greater than 10 in v
39      location = find_if( v.begin(), v.end(), greater10 );
40
41      if ( location != v.end() ) // found value greater than 10
42         cout << "\n\nThe first value greater than 10 is " << *location
43            << "\nfound at location " << ( location - v.begin() );
44      else // value greater than 10 not found
45         cout << "\n\nNo values greater than 10 were found";
46
47      // sort elements of v
48      sort( v.begin(), v.end() );
49      cout << "\n\nVector v after sort: ";
50      copy( v.begin(), v.end(), output );
51
52      // use binary_search to locate 13 in v
53      if ( binary_search( v.begin(), v.end(), 13 ) )
54         cout << "\n\n13 was found in v";
55      else
56         cout << "\n\n13 was not found in v";
57
58      // use binary_search to locate 100 in v
59      if ( binary_search( v.begin(), v.end(), 100 ) )
60         cout << "\n100 was found in v";
61      else
62         cout << "\n100 was not found in v";
63
64      cout << endl;
65   } // end main
66
```

```
bool greater10( int value )
{
   return value > 10;
} // end function greater10
```

```
The first value greater than 10 is 17
found at location 2

Vector v after sort: 2 5 7 8 10 11 13 16 17 20

13 was found in v
100 was not found in v
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

swap_ranges

- Exchanges the values of each of the elements in the range [first1,last1) with those of their respective elements in the range beginning at first2

```
template<class ForwardIterator1, class ForwardIterator2>
ForwardIterator2 swap_ranges (ForwardIterator1 first1,
ForwardIterator1 last1, ForwardIterator2 first2)
{
  while (first1!=last1) {
    swap (*first1, *first2);
    ++first1; ++first2;
  }
  return first2;
}
```

Using swap, iter_swap, and swap_ranges

```
Array a contains:
   1 2 3 4 5 6 7 8 9 10
Array a after swapping a[0] and a[1] using swap:
   2 1 3 4 5 6 7 8 9 10
Array a after swapping a[0] and a[1] using iter_swap:
   1 2 3 4 5 6 7 8 9 10
Array a after swapping the first five elements
with the last five elements:
   6 7 8 9 10 1 2 3 4 5
```

```
 3  #include <iostream>
 4  #include <algorithm> // algorithm definitions
 5  #include <iterator>
 6  using namespace std;
 7
 8  int main()
 9  {
10     const int SIZE = 10;
11     int a[ SIZE ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
12     ostream_iterator< int > output( cout, " " );
13
14     cout << "Array a contains:\n   ";
15     copy( a, a + SIZE, output ); // display array a
16
17     // swap elements at locations 0 and 1 of array a
18     swap( a[ 0 ], a[ 1 ] );
19
20     cout << "\nArray a after swapping a[0] and a[1] using swap:\n   ";
21     copy( a, a + SIZE, output ); // display array a
22
23     // use iterators to swap elements at locations 0 and 1 of array a
24     iter_swap( &a[ 0 ], &a[ 1 ] ); // swap with iterators
25     cout << "\nArray a after swapping a[0] and a[1] using iter_swap:\n   ";
26     copy( a, a + SIZE, output );
27
28     // swap elements in first five elements of array a with
29     // elements in last five elements of array a
30     swap_ranges( a, a + 5, a + 5 );
31
32     cout << "\nArray a after swapping the first five elements\n"
33          << "with the last five elements:\n   ";
34     copy( a, a + SIZE, output );
35     cout << endl;
36  } // end main
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

Using copy_backward

```
 3  #include <iostream>
 4  #include <algorithm> // algorithm definitions
 5  #include <vector> // vector class-template definition
 6  #include <iterator> // ostream_iterator
 7  using namespace std;
 8
 9  int main()
10  {
11     const int SIZE = 5;
12     int a1[ SIZE ] = { 1, 3, 5, 7, 9 };
13     int a2[ SIZE ] = { 2, 4, 5, 7, 9 };
14     vector< int > v1( a1, a1 + SIZE ); // copy of a1
15     vector< int > v2( a2, a2 + SIZE ); // copy of a2
16     ostream_iterator< int > output( cout, " " );
17
18     cout << "Vector v1 contains: ";
19     copy( v1.begin(), v1.end(), output ); // display vector output
20     cout << "\nVector v2 contains: ";
21     copy( v2.begin(), v2.end(), output ); // display vector output
22
23     vector< int > results( v1.size() );
24
25     // place elements of v1 into results in reverse order
26     copy_backward( v1.begin(), v1.end(), results.end() );
27     cout << "\n\nAfter copy_backward, results contains: ";
28     copy( results.begin(), results.end(), output );
```

```
Vector v1 contains: 1 3 5 7 9
Vector v2 contains: 2 4 5 7 9

After copy_backward, results contains: 1 3 5 7 9
```

Using merge, unique, and reverse

```
30      vector< int > results2( v1.size() + v2.size() );
31
32      // merge elements of v1 and v2 into results2 in sorted order
33      merge( v1.begin(), v1.end(), v2.begin(), v2.end(), results2.begin() );
34
35      cout << "\n\nAfter merge of v1 and v2 results2 contains:\n";
36      copy( results2.begin(), results2.end(), output );
37
38      // eliminate duplicate values from results2
39      vector< int >::iterator endLocation;
40      endLocation = unique( results2.begin(), results2.end() );
41
42      cout << "\n\nAfter unique results2 contains:\n";
43      copy( results2.begin(), endLocation, output );
44
45      cout << "\n\nVector v1 after reverse: ";
46      reverse( v1.begin(), v1.end() ); // reverse elements of v1
47      copy( v1.begin(), v1.end(), output );
48      cout << endl;
49   } // end main
```

```
After merge of v1 and v2 results2 contains:
1 2 3 4 5 5 7 7 9 9

After unique results2 contains:
1 2 3 4 5 7 9

Vector v1 after reverse: 9 7 5 3 1
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

inplace_merge

- Merges two consecutive sorted ranges: [first,middle) and [middle,last), putting the result into the combined sorted range [first,last)

```
template <class BidirectionalIterator>
void inplace_merge (BidirectionalIterator first,
BidirectionalIterator middle, BidirectionalIterator last);
```

```
template <class BidirectionalIterator, class Compare>
void inplace_merge (BidirectionalIterator first,
BidirectionalIterator middle, BidirectionalIterator last, Compare
comp);
```

Using inplace_merge

```
 4  #include <iostream>
 5  #include <algorithm> // algorithm definitions
 6  #include <vector> // vector class-template definition
 7  #include <iterator> // back_inserter definition
 8  using namespace std;
 9
10  int main()
11  {
12     const int SIZE = 10;
13     int a1[ SIZE ] = { 1, 3, 5, 7, 9, 1, 3, 5, 7, 9 };
14     vector< int > v1( a1, a1 + SIZE ); // copy of a
15     ostream_iterator< int > output( cout, " " );
16
17     cout << "Vector v1 contains: ";
18     copy( v1.begin(), v1.end(), output );
19
20     // merge first half of v1 with second half of v1 such that
21     // v1 contains sorted set of elements after merge
22     inplace_merge( v1.begin(), v1.begin() + 5, v1.end() );
23
24     cout << "\nAfter inplace_merge, v1 contains: ";
25     copy( v1.begin(), v1.end(), output );
```

```
Vector v1 contains: 1 3 5 7 9 1 3 5 7 9
After inplace_merge, v1 contains: 1 1 3 3 5 5 7 7 9 9
```

Using unique_copy and reverse_copy

```
27      vector< int > results1;
28
29      // copy only unique elements of v1 into results1
30      unique_copy( v1.begin(), v1.end(), back_inserter( results1 ) );
31      cout << "\nAfter unique_copy results1 contains: ";
32      copy( results1.begin(), results1.end(), output );
33
34      vector< int > results2;
35
36      // copy elements of v1 into results2 in reverse order
37      reverse_copy( v1.begin(), v1.end(), back_inserter( results2 ) );
38      cout << "\nAfter reverse_copy, results2 contains: ";
39      copy( results2.begin(), results2.end(), output );
40      cout << endl;
41   } // end main
```

```
After unique_copy results1 contains: 1 3 5 7 9
After reverse_copy, results2 contains: 9 9 7 7 5 5 3 3 1 1
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

Using includes

```
 4  #include <iostream>
 5  #include <algorithm> // algorithm definitions
 6  #include <iterator> // ostream_iterator
 7  using namespace std;
 8
 9  int main()
10  {
11     const int SIZE1 = 10, SIZE2 = 5, SIZE3 = 20;
12     int a1[ SIZE1 ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
13     int a2[ SIZE2 ] = { 4, 5, 6, 7, 8 };
14     int a3[ SIZE2 ] = { 4, 5, 6, 11, 15 };
15     ostream_iterator< int > output( cout, " " );
16
17     cout << "a1 contains: ";
18     copy( a1, a1 + SIZE1, output ); // display array a1
19     cout << "\na2 contains: ";
20     copy( a2, a2 + SIZE2, output ); // display array a2
21     cout << "\na3 contains: ";
22     copy( a3, a3 + SIZE2, output ); // display array a3
23
24     // determine whether set a2 is completely contained in a1
25     if ( includes( a1, a1 + SIZE1, a2, a2 + SIZE2 ) )
26        cout << "\n\na1 includes a2";
27     else
28        cout << "\n\na1 does not include a2";
29
30     // determine whether set a3 is completely contained in a1
31     if ( includes( a1, a1 + SIZE1, a3, a3 + SIZE2 ) )
32        cout << "\na1 includes a3";
33     else
34        cout << "\na1 does not include a3";
```

```
a1 contains: 1 2 3 4 5 6 7 8 9 10
a2 contains: 4 5 6 7 8
a3 contains: 4 5 6 11 15

a1 includes a2
a1 does not include a3
```

Using set_difference, set_intersection, set_symmetric_difference, and set_union

```
36      int difference[ SIZE1 ];
37
38      // determine elements of a1 not in a2
39      int *ptr = set_difference( a1, a1 + SIZE1,
40         a2, a2 + SIZE2, difference );
41      cout << "\n\nset_difference of a1 and a2 is: ";
42      copy( difference, ptr, output );
43
44      int intersection[ SIZE1 ];
45
46      // determine elements in both a1 and a2
47      ptr = set_intersection( a1, a1 + SIZE1,
48         a2, a2 + SIZE2, intersection );
49      cout << "\n\nset_intersection of a1 and a2 is: ";
50      copy( intersection, ptr, output );
51
52      int symmetric_difference[ SIZE1 + SIZE2 ];
53
54      // determine elements of a1 that are not in a2 and
55      // elements of a2 that are not in a1
56      ptr = set_symmetric_difference( a1, a1 + SIZE1,
57         a3, a3 + SIZE2, symmetric_difference );
58      cout << "\n\nset_symmetric_difference of a1 and a3 is: ";
59      copy( symmetric_difference, ptr, output );
60
61      int unionSet[ SIZE3 ];
62
63      // determine elements that are in either or both sets
64      ptr = set_union( a1, a1 + SIZE1, a3, a3 + SIZE2, unionSet );
65      cout << "\n\nset_union of a1 and a3 is: ";
66      copy( unionSet, ptr, output );
67      cout << endl;
68   } // end main
```

```
set_difference of a1 and a2 is: 1 2 3 9 10

set_intersection of a1 and a2 is: 4 5 6 7 8

set_symmetric_difference of a1 and a3 is: 1 2 3 7 8 9 10 11 15

set_union of a1 and a3 is: 1 2 3 4 5 6 7 8 9 10 11 15
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

Using lower_bound and upper_bound

```
4  #include <iostream>
5  #include <algorithm> // algorithm definitions
6  #include <vector> // vector class-template definition
7  #include <iterator> // ostream_iterat
8  using namespace std;
9
10 int main()
11 {
12    const int SIZE = 10;
13    int a1[ SIZE ] = { 2, 2, 4, 4, 4, 6, 6, 6, 6, 8 };
14    vector< int > v( a1, a1 + SIZE ); // copy of a1
15    ostream_iterator< int > output( cout, " " );
16
17    cout << "Vector v contains:\n";
18    copy( v.begin(), v.end(), output );
19
20    // determine lower-bound insertion point for 6 in v
21    vector< int >::iterator lower;
22    lower = lower_bound( v.begin(), v.end(), 6 );
23    cout << "\n\nLower bound of 6 is element "
24       << ( lower - v.begin() ) << " of vector v";
25
26    // determine upper-bound insertion point for 6 in v
27    vector< int >::iterator upper;
28    upper = upper_bound( v.begin(), v.end(), 6 );
29    cout << "\nUpper bound of 6 is element "
30       << ( upper - v.begin() ) << " of vector v";
```

```
Vector v contains:
2 2 4 4 4 6 6 6 6 8

Lower bound of 6 is element 5 of vector v
Upper bound of 6 is element 9 of vector v
```

Using equal_range

```
32      // use equal_range to determine both the lower- and
33      // upper-bound insertion points for 6
34      pair< vector< int >::iterator, vector< int >::iterator > eq;
35      eq = equal_range( v.begin(), v.end(), 6 );
36      cout << "\nUsing equal_range:\n   Lower bound of 6 is element "
37           << ( eq.first - v.begin() ) << " of vector v";
38      cout << "\n   Upper bound of 6 is element "
39         << ( eq.second - v.begin() ) << " of vector v";
40      cout << "\n\nUse lower_bound to locate the first point\n"
41         << "at which 5 can be inserted in order";
42
43      // determine lower-bound insertion point for 5 in v
44      lower = lower_bound( v.begin(), v.end(), 5 );
45      cout << "\n   Lower bound of 5 is element "
46         << ( lower - v.begin() ) << " of vector v";
47      cout << "\n\nUse upper_bound to locate the last point\n"
48         << "at which 7 can be inserted in order";
49
50      // determine upper-bound insertion point for 7 in v
51      upper = upper_bound( v.begin(), v.end(), 7 );
52      cout << "\n   Upper bound of 7 is element "
53         << ( upper - v.begin() ) << " of vector v";
54      cout << "\n\nUse equal_range to locate the first and\n"
55         << "last point at which 5 can be inserted in order";
56
57      // use equal_range to determine both the lower- and
58      // upper-bound insertion points for 5
59      eq = equal_range( v.begin(), v.end(), 5 );
60      cout << "\n   Lower bound of 5 is element "
61         << ( eq.first - v.begin() ) << " of vector v";
62      cout << "\n   Upper bound of 5 is element "
63         << ( eq.second - v.begin() ) << " of vector v" << endl;
64   } // end main
```

```
Using equal_range:
   Lower bound of 6 is element 5 of vector v
   Upper bound of 6 is element 9 of vector v

Use lower_bound to locate the first point
at which 5 can be inserted in order
   Lower bound of 5 is element 5 of vector v

Use upper_bound to locate the last point
at which 7 can be inserted in order
   Upper bound of 7 is element 9 of vector v

Use equal_range to locate the first and
last point at which 5 can be inserted in order
   Lower bound of 5 is element 5 of vector v
   Upper bound of 5 is element 5 of vector v
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

make_heap

- Rearranges the elements in the range [first,last) in such a way that they form a heap

```
template <class RandomAccessIterator>
void make_heap (RandomAccessIterator first, RandomAccessIterator last);
```

```
template <class RandomAccessIterator, class Compare>
void make_heap (RandomAccessIterator first, RandomAccessIterator last,
Compare comp );
```

- The element with the highest value is always pointed by first
- The order of the other elements depends on the particular implementation, but it is consistent throughout all heap-related functions of this header

Using make_heap and sort_heap

```
 4  #include <iostream>
 5  #include <algorithm>
 6  #include <vector>
 7  #include <iterator>
 8  using namespace std;
 9
10  int main()
11  {
12     const int SIZE = 10;
13     int a[ SIZE ] = { 3, 100, 52, 77, 22, 31, 1, 98, 13, 40 };
14     vector< int > v( a, a + SIZE ); // copy of a
15     vector< int > v2;
16     ostream_iterator< int > output( cout, " " );
17
18     cout << "Vector v before make_heap:\n";
19     copy( v.begin(), v.end(), output );
20
21     make_heap( v.begin(), v.end() ); // create heap from vector v
22     cout << "\nVector v after make_heap:\n";
23     copy( v.begin(), v.end(), output );
24
25     sort_heap( v.begin(), v.end() ); // sort elements with sort_heap
26     cout << "\nVector v after sort_heap:\n";
27     copy( v.begin(), v.end(), output );
```

```
Vector v before make_heap:
3 100 52 77 22 31 1 98 13 40
Vector v after make_heap:
100 98 52 77 40 31 1 3 13 22
Vector v after sort_heap:
1 3 13 22 31 40 52 77 98 100
```

Using push_heap and pop_heap (1/2)

```
29      // perform the heapsort with push_heap and pop_heap
30      cout << "\n\nArray a contains: ";
31      copy( a, a + SIZE, output ); // display array a
32      cout << endl;
33
34      // place elements of array a into v2 and
35      // maintain elements of v2 in heap
36      for ( int i = 0; i < SIZE; i++ )
37      {
38         v2.push_back( a[ i ] );
39         push_heap( v2.begin(), v2.end() );
40         cout << "\nv2 after push_heap(a[" << i << "]): ";
41         copy( v2.begin(), v2.end(), output );
42      } // end for
43
44      cout << endl;
45
46      // remove elements from heap in sorted order
47      for ( unsigned int j = 0; j < v2.size(); j++ )
48      {
49         cout << "\nv2 after " << v2[ 0 ] << " popped from heap\n";
50         pop_heap( v2.begin(), v2.end() - j );
51         copy( v2.begin(), v2.end(), output );
52      } // end for
53
54      cout << endl;
55   } // end main
```

Using push_heap and pop_heap (2/2)

```
Array a contains: 3 100 52 77 22 31 1 98 13 40

v2 after push_heap(a[0]): 3
v2 after push_heap(a[1]): 100 3
v2 after push_heap(a[2]): 100 3 52
v2 after push_heap(a[3]): 100 77 52 3
v2 after push_heap(a[4]): 100 77 52 3 22
v2 after push_heap(a[5]): 100 77 52 3 22 31
v2 after push_heap(a[6]): 100 77 52 3 22 31 1
v2 after push_heap(a[7]): 100 98 52 77 22 31 1 3
v2 after push_heap(a[8]): 100 98 52 77 22 31 1 3 13
v2 after push_heap(a[9]): 100 98 52 77 40 31 1 3 13 22

v2 after 100 popped from heap
98 77 52 22 40 31 1 3 13 100
v2 after 98 popped from heap
77 40 52 22 13 31 1 3 98 100
v2 after 77 popped from heap
52 40 31 22 13 3 1 77 98 100
v2 after 52 popped from heap
40 22 31 1 13 3 52 77 98 100
v2 after 40 popped from heap
31 22 3 1 13 40 52 77 98 100
v2 after 31 popped from heap
22 13 3 1 31 40 52 77 98 100
v2 after 22 popped from heap
13 1 3 22 31 40 52 77 98 100
v2 after 13 popped from heap
3 1 13 22 31 40 52 77 98 100
v2 after 3 popped from heap
1 3 13 22 31 40 52 77 98 100
v2 after 1 popped from heap
1 3 13 22 31 40 52 77 98 100
```

Outline

- Standard Template Library
  - Algorithms
    - `fill`, `fill_n`, `generate`, and `generate_n`
    - `equal`, `mismatch`, and `lexicographical_compare`
    - `remove`, `remove_if`, `remove_copy`, and `remove_copy_if`
    - `replace`, `replace_if`, `replace_copy`, and `replace_copy_if`
    - Mathematical Algorithms
    - Basic Searching and Sorting Algorithms
    - `swap`, `iter_swap`, and `swap_ranges`
    - `copy_backward`, `merge`, `unique`, and `reverse`
    - `inplace_merge`, `unique_copy`, and `reverse_copy`
    - Set Operations
    - `lower_bound`, `upper_bound`, and `equal_range`
    - Heapsort
    - `min` and `max`

Using min and max

```
 3  #include <iostream>
 4  #include <algorithm>
 5  using namespace std;
 6
 7  int main()
 8  {
 9     cout << "The minimum of 12 and 7 is: " << min( 12, 7 );
10     cout << "\nThe maximum of 12 and 7 is: " << max( 12, 7 );
11     cout << "\nThe minimum of 'G' and 'Z' is: " << min( 'G', 'Z' );
12     cout << "\nThe maximum of 'G' and 'Z' is: " << max( 'G', 'Z' );
13     cout << endl;
14  } // end main
```

```
The minimum of 12 and 7 is: 7
The maximum of 12 and 7 is: 12
The minimum of 'G' and 'Z' is: G
The maximum of 'G' and 'Z' is: Z
```

Outline

- Introduction to Templates
  - Function Templates
  - Class Templates
  - Default Template Types & Non-type Template Parameters
- Standard Template Library
  - Introduction to STL
  - Sequence Containers (`vector, list, deque`)
  - Associative Containers (`map/multimap, set/multiset`)
  - Function Objects
  - Container Adapters (`stack, queue, priority_queue`)
  - Algorithms
  - Near Containers (`bitset`)

Class `bitset`

- Class `bitset` makes it easy to create and manipulate bit sets

Using bitset (1/3)

```
 3  #include <iostream>
 4  #include <iomanip>
 5  #include <cmath>
 6  #include <bitset> // bitset class definition
 7  using namespace std;
 8
 9  int main()
10  {
11     const int SIZE = 1024;
12     int value;
13     bitset< SIZE > sieve; // create bitset of 1024 bits
14     sieve.flip(); // flip all bits in bitset sieve
15     sieve.reset( 0 ); // reset first bit (number 0)
16     sieve.reset( 1 ); // reset second bit (number 1)
17
18     // perform Sieve of Eratosthenes
19     int finalBit = sqrt( static_cast< double >( sieve.size() ) ) + 1;
20
21     // determine all prime numbers from 2 to 1024
22     for ( int i = 2; i < finalBit; i++ )
23     {
24        if ( sieve.test( i ) ) // bit i is on
25        {
26           for ( int j = 2 * i; j < SIZE; j += i )
27              sieve.reset( j ); // set bit j off
28        } // end if
29     } // end for
```

Using bitset (2/3)

```
31       cout << "The prime numbers in the range 2 to 1023 are:\n";
32
33       // display prime numbers in range 2-1023
34       for ( int k = 2, counter = 1; k < SIZE; k++ )
35       {
36          if ( sieve.test( k ) ) // bit k is on
37          {
38             cout << setw( 5 ) << k;
39
40             if ( counter++ % 12 == 0 ) // counter is a multiple of 12
41                cout << '\n';
42          } // end if
43       } // end for
44
45       cout << endl;
46
47       // get value from user to determine whether value is prime
48       cout << "\nEnter a value from 2 to 1023 (-1 to end): ";
49       cin >> value;
50
51       // determine whether user input is prime
52       while ( value != -1 )
53       {
54          if ( sieve[ value ] ) // prime number
55             cout << value << " is a prime number\n";
56          else // not a prime number
57             cout << value << " is not a prime number\n";
58
59          cout << "\nEnter a value from 2 to 1023 (-1 to end): ";
60          cin >> value;
61       } // end while
62    } // end main
```

Using bitset (3/3)

```
The prime numbers in the range 2 to 1023 are:
    2    3    5    7   11   13   17   19   23   29   31   37
   41   43   47   53   59   61   67   71   73   79   83   89
   97  101  103  107  109  113  127  131  137  139  149  151
  157  163  167  173  179  181  191  193  197  199  211  223
  227  229  233  239  241  251  257  263  269  271  277  281
  283  293  307  311  313  317  331  337  347  349  353  359
  367  373  379  383  389  397  401  409  419  421  431  433
  439  443  449  457  461  463  467  479  487  491  499  503
  509  521  523  541  547  557  563  569  571  577  587  593
  599  601  607  613  617  619  631  641  643  647  653  659
  661  673  677  683  691  701  709  719  727  733  739  743
  751  757  761  769  773  787  797  809  811  821  823  827
  829  839  853  857  859  863  877  881  883  887  907  911
  919  929  937  941  947  953  967  971  977  983  991  997
 1009 1013 1019 1021

Enter a value from 2 to 1023 (-1 to end): 389
389 is a prime number


Enter a value from 2 to 1023 (-1 to end): 88
88 is not a prime number

Enter a value from 2 to 1023 (-1 to end): -1
```