

```

//: C05:Comparison.cpp
// The STL range comparison algorithms
#include "PrintSequence.h"
#include <vector>
#include <algorithm>
#include <functional>
#include <string>
using namespace std;

int main() {
    // strings provide a convenient way to create
    // ranges of characters, but you should
    // normally look for native string
operations:
    string s1("This is a test");
    string s2("This is a Test");
    cout << "s1: " << s1 << endl
        << "s2: " << s2 << endl;
    cout << "compare s1 & s1: "
        << equal(s1.begin(), s1.end(), s1.begin())
        << endl;
    cout << "compare s1 & s2: "
        << equal(s1.begin(), s1.end(), s2.begin())
        << endl;
    cout << "lexicographical_compare s1 & s1: "
<<
    lexicographical_compare(s1.begin(),
s1.end(),
    s1.begin(), s1.end()) << endl;
    cout << "lexicographical_compare s1 & s2: "
<<
    lexicographical_compare(s1.begin(),
s1.end(),
    s2.begin(), s2.end()) << endl;
    cout << "lexicographical_compare s2 & s1: "
<<
    lexicographical_compare(s2.begin(),
s2.end(),
    s1.begin(), s1.end()) << endl;
    cout << "lexicographical_compare shortened "
        "s1 & full-length s2: " << endl;
    string s3(s1);
    while(s3.length() != 0) {
        bool result = lexicographical_compare(
            s3.begin(), s3.end(),
s2.begin(), s2.end());
        cout << s3 << endl << s2 << ", result = "
            << result << endl;
        if(result == true) break;
        s3 = s3.substr(0, s3.length() - 1);
    }
    pair<string::iterator, string::iterator> p =
        mismatch(s1.begin(), s1.end(), s2.begin());
    print(p.first, s1.end(), "p.first", "");
    print(p.second, s2.end(), "p.second", "");
} //:~

```

```

s1: This is a test
s2: This is a Test
compare s1 & s1: 1
compare s1 & s2: 0
lexicographical_compare s1 & s1: 0
lexicographical_compare s1 & s2: 0
lexicographical_compare s2 & s1: 1

```

```

lexicographical_compare shortened s1 & full-
length s2:
This is a test
This is a Test, result = 0
This is a tes
This is a Test, result = 0
This is a te
This is a Test, result = 0
This is a t
This is a Test, result = 0
This is a
This is a Test, result = 1
p.first: test
p.second: Test
-----
```

```

//: C05:MergeTest.cpp
// From Thinking in C++, 2nd Edition
// Test merging in sorted ranges
#include <algorithm>
#include "PrintSequence.h"
#include "Generators.h"
using namespace std;

int main() {
    const int sz = 15;
    int a[sz*2] = {0};
    // Both ranges go in the same array:
    generate(a, a + sz, SkipGen(0, 2));
    generate(a + sz, a + sz*2, SkipGen(1, 3));
    print(a, a + sz, "range1", " ");
    print(a + sz, a + sz*2, "range2", " ");
    int b[sz*2] = {0}; // Initialize all to zero
    merge(a, a + sz, a + sz, a + sz*2, b);
    print(b, b + sz*2, "merge", " ");
    // set_union is a merge that removes
duplicates
    set_union(a, a + sz, a + sz, a + sz*2, b);
    print(b, b + sz*2, "set_union", " ");
    inplace_merge(a, a + sz, a + sz*2);
    print(a, a + sz*2, "inplace_merge", " ");
} //:~

```

```

range1: 0 2 4 6 8 10 12 14 16 18 20 22 24 26
28
range2: 1 4 7 10 13 16 19 22 25 28 31 34 37 40
43
merge: 0 1 2 4 4 6 7 8 10 10 12 13 14 16 16 18
19 20 22 22 24 25 26 28 28 31 34 37 40 43
set_union: 0 1 2 4 6 7 8 10 12 13 14 16 18 19
20 22 24 25 26 28 31 34 37 40 43 31 34 37 40 43
inplace_merge: 0 1 2 4 4 6 7 8 10 10 12 13 14
16 16 18 19 20 22 22 24 25 26 28 28 31 34 37 40
43

```

```

//: C05:SortedSearchTest.cpp
// Test searching in sorted ranges
#include "StreamTokenizer.h"
#include "PrintSequence.h"
#include "NSString.h"
#include <algorithm>
```

```

#include <fstream>
#include <queue>
#include <list>
#include <vector>
using namespace std;

int main() {
    ifstream in("SortedSearchTest.cpp");
    StreamTokenizer words(in);
    deque<NString> dstr;

    string word;

    while((word = words.next()).size() != 0)
        dstr.push_back(NString(word));

    vector<NString> v;
    copy(dstr.begin(),
          dstr.end(), back_inserter(v));

    sort(v.begin(), v.end());
    print(v.begin(), v.end(), "sorted");
    typedef vector<NString>::iterator sit;
    sit it, it2;
    string f("include");
    cout << "binary search: "
        << binary_search(v.begin(), v.end(), f)
        << endl;
    it = lower_bound(v.begin(), v.end(), f);
    it2 = upper_bound(v.begin(), v.end(), f);
    print(it, it2, "found range");
    pair<sit, sit> ip =
        equal_range(v.begin(), v.end(), f);
    print(ip.first, ip.second,
          "equal_range");
} //:~

-----
[2] vector [1] vector [0] while [0] word [1]
word [2] word [0] words [0] words [1]
binary search: 1
found range: include [4] include [0] include
[8] include [1] include [5] include [7] include
[6] include [3] include [2]
equal_range: include [4] include [0] include
[8] include [1] include [5] include [7] include
[6] include [3] include [2]
-----

//: C04:BitSet.cpp
#include <bitset>
#include <cstdlib>
#include <ctime>
#include <climits>
#include <iostream>
using namespace std;
const int sz = 32;
typedef bitset<sz> BS;

template<int bits>
bitset<bits> randBitset() {
    bitset<bits> r(rand());
    for(int i = 0; i < bits/16 - 1; i++) {
        r <<= 16;
        // "OR" together with a new lower 16 bits:
        r |= bitset<bits>(rand());
    }
    return r;
}

const char* cbits = "111011010110111";

int main() {
    srand(time(0));
    cout << "sizeof(bitset<16>) = "
        << sizeof(bitset<16>) << endl;
    cout << "sizeof(bitset<32>) = "
        << sizeof(bitset<32>) << endl;
    cout << "sizeof(bitset<48>) = "
        << sizeof(bitset<48>) << endl;
    cout << "sizeof(bitset<64>) = "
        << sizeof(bitset<64>) << endl;
    cout << "sizeof(bitset<65>) = "
        << sizeof(bitset<65>) << endl;
    BS a(randBitset<sz>()), b(randBitset<sz>());
    // Converting from a bitset:
    unsigned long ul = a.to_ulong();
    string s = b.to_string();
    // Converting a string to a bitset:
    cout << "char* cbits = " << cbits << endl;
    cout << BS(cbits) << " [BS(cbits)]" << endl;
    cout << BS(cbits, 2)
        << " [BS(cbits, 2)]" << endl;
    // cout << BS(cbits, 2, 11)
    // << " [BS(cbits, 2, 11)]" << endl;
    cout << a << " [a]" << endl;
    cout << b << " [b]" << endl;
    // Bitwise AND:
    cout << (a & b) << " [a & b]" << endl;
    cout << (BS(a) &= b) << " [a &= b]" << endl;
    // Bitwise OR:
    cout << (a | b) << " [a | b]" << endl;
    cout << (BS(a) |= b) << " [a |= b]" << endl;
}

```

```

// Exclusive OR:
cout << (a ^ b) << " [a ^ b]" << endl;
cout << (BS(a) ^= b) << " [a ^= b]" << endl;
cout << a << " [a]" << endl; // For reference
// Logical left shift (fill with zeros):
cout << (BS(a) <= sz/2)
    << " [a <= (sz/2)]" << endl;
cout << (a << sz/2) << endl;
cout << a << " [a]" << endl; // For reference
// Logical right shift (fill with zeros):
cout << (BS(a) >= sz/2)
    << " [a >= (sz/2)]" << endl;
cout << (a >> sz/2) << endl;
cout << a << " [a]" << endl; // For reference
cout << BS(a).set() << " [a.set()]" << endl;
for(int i = 0; i < sz; i++)
    if(!a.test(i)) {
        cout << BS(a).set(i)
            << " [a.set(" << i << ")]" << endl;
        break; // Just do one example of this
    }
cout << BS(a).reset() << " [a.reset()]" <<
endl;
for(int j = 0; j < sz; j++)
    if(a.test(j)) {
        cout << BS(a).reset(j)
            << " [a.reset(" << j << ")]" << endl;
        break; // Just do one example of this
    }
cout << BS(a).flip() << " [a.flip()]" <<
endl;
cout << ~a << " [~a]" << endl;
cout << a << " [a]" << endl; // For reference
cout << BS(a).flip(1) << " [a.flip(1)]" <<
endl;
BS c;
cout << c << " [c]" << endl;
cout << "c.count() = " << c.count() << endl;
cout << "c.any() = "
    << (c.any() ? "true" : "false") << endl;
cout << "c.none() = "
    << (c.none() ? "true" : "false") << endl;
c[1].flip(); c[2].flip();
cout << c << " [c]" << endl;
cout << "c.count() = " << c.count() << endl;
cout << "c.any() = "
    << (c.any() ? "true" : "false") << endl;
cout << "c.none() = "
    << (c.none() ? "true" : "false") << endl;
// Array indexing operations:
c.reset();
for(int k = 0; k < c.size(); k++)
    if(k % 2 == 0)
        c[k].flip();
cout << c << " [c]" << endl;
c.reset();
// Assignment to bool:
for(int ii = 0; ii < c.size(); ii++)
    c[ii] = (rand() % 100) < 25;
cout << c << " [c]" << endl;
if(c[1] == true) // bool test:
    cout << "c[1] == true";
else
    cout << "c[1] == false" << endl;
} ///:~

```

```

-----
sizeof(bitset<16>) = 8
sizeof(bitset<32>) = 8
sizeof(bitset<48>) = 8
sizeof(bitset<64>) = 8
sizeof(bitset<65>) = 16
char* cbits = 111011010110111
0000000000000000111011010110111 [BS(cbits)]
0000000000000000000000000000000011 [BS(cbits, 2)]
0111111100110010100010110000110 [a]
1101111011111111101101111010001 [b]
01011110100110010100000110000000 [a & b]
01011110100110010100000110000000 [a &= b]
1111111111111111101111111010111 [a | b]
1111111111111111101111111010111 [a |= b]
10100001011001101001111001010111 [a ^ b]
10100001011001101001111001010111 [a ^= b]
0111111100110010100010110000110 [a]
01000101100001100000000000000000 [a <= (sz/2)]
01000101100001100000000000000000
0111111100110010100010110000110 [a]
000000000000000011111110011001 [a >= (sz/2)]
000000000000000011111110011001
0111111100110010100010110000110 [a]
1111111111111111111111111111111 [a.set()]
0111111100110010100010110000111 [a.set(0)]
00000000000000000000000000000000 [a.reset()]
0111111100110010100010110000100 [a.reset(1)]
10000000011001101011101001111001 [a.flip()]
10000000011001101011101001111001 [~a]
0111111100110010100010110000110 [a]
0111111100110010100010110000100 [a.flip(1)]
00000000000000000000000000000000 [c]
c.count() = 0
c.any() = false
c.none() = true
00000000000000000000000000000000110 [c]
c.count() = 2
c.any() = true
c.none() = false
01010101010101010101010101010101 [c]
11111000000000010000000010011100 [c]
c[1] == false
-----
```

```

// valarray_eckel.cpp : Defines the entry point
for the console application.
//: C07:Valarray1.cpp {-bor}
#include <iostream>
#include <valarray>
using namespace std;

template<typename T> void print(const char*
lbl, const valarray<T>& a) {
    cout << lbl << ":" ;
    for(size_t i = 0; i < a.size(); ++i)
        cout << a[i] << ' ';
    cout << endl;
}

double f(double x) { return 2.0 * x - 1.0; }

int main() {
{

```

```

double n[] = {1.0, 2.0, 3.0, 4.0};
valarray<double> v(n, sizeof n / sizeof
n[0]);
print("v", v);
valarray<double> sh(v.shift(1));
print("shift 1", sh);
valarray<double> acc(v + sh);
print("sum", acc);
valarray<double> trig(sin(v) + cos(acc));
print("trig", trig);
valarray<double> p(pow(v, 3.0));
print("3rd power", p);
valarray<double> app(v.apply(f));
print("f(v)", app);
valarray<bool> eq(v == app);
print("v == app?", eq);
double x = v.min();
double y = v.max();
double z = v.sum();
cout << "x = " << x << ", y = " << y
    << ", z = " << z << endl;
}

{
int data[] = {1,2,3,4,5,6,7,8,9,10,11,12};
valarray<int> v(data, 12);
valarray<int> r1(v[slice(0, 4, 3)]);
print("slice(0,4,3)", r1);
// Выделение элементов по условию
valarray<int> r2(v[v > 6]);
print("elements > 6", r2);
// Возведение в квадрат первого столбца
v[slice(0, 4, 3)] *=
valarray<int>(v[slice(0, 4, 3)]);
print("after squaring first column", v);
// Восстановление исходных значений
int idx[] = {1,4,7,10};
valarray<int> save(idx, 4);
v[slice(0, 4, 3)] = save;
print("v restored", v);
// Выделение двухмерного подмножества: {{1,
3, 5}, {7, 9, 11}}
valarray<size_t> siz(2);
siz[0] = 2;
siz[1] = 3;
valarray<size_t> gap(2);
gap[0] = 6;
gap[1] = 2;
valarray<int> r3(v[gslice(0, siz, gap)]);
print("2-d slice", r3);
// Выделение подмножества по логической
маске (элементы bool)
valarray<bool> mask(false, 5);
mask[1] = mask[2] = mask[4] = true;
valarray<int> r4(v[mask]);
print("v[mask]", r4);
// Выделение подмножества по индексной
маске (элементы size_t)
size_t idx2[] = {2,2,3,6};
valarray<size_t> mask2(idx2, 4);
valarray<int> r5(v[mask2]);
print("v[mask2]", r5);
// Использование индексной маски при
присваивании
valarray<char> text("now is the time", 15);
valarray<char> caps("NITT", 4);
valarray<size_t> idx3(4);

```

```

idx3[0] = 0; idx3[1] = 4; idx3[2] = 7;
idx3[3] = 11; text[idx3] = caps;
print("capitalized", text);
}
}

v: 1 2 3 4
shift 1: 2 3 4 0
sum: 3 5 7 4
trig: -0.148522 1.19296 0.895022 -1.41045
3rd power: 1 8 27 64
f(v): 1 3 5 7
v == app?: 1 0 0 0
x = 1, y = 4, z = 10
slice(0,4,3): 1 4 7 10
elements > 6: 7 8 9 10 11 12
after squaring first column: 1 2 3 16 5 6 49 8
9 100 11 12
v restored: 1 2 3 4 5 6 7 8 9 10 11 12
2-d slice: 1 3 5 7 9 11
v[mask]: 2 3 5
v[mask2]: 3 3 4 7
capitalized: N o w   I s   T h e   T i m e

```