

```

//: C06:FunctionObjects.cpp {-bor}
// From "Thinking in C++, Volume 2", by Bruce
Eckel & Chuck Allison.
// templates from the Standard C++ library.
//{L} Generators
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <functional>
#include <iostream>
#include <iterator>
#include <vector>
#include "Generators.h"
#include "PrintSequence.h"
using namespace std;
using namespace std::placeholders;

template<typename Contain, typename UnaryFunc>
void testUnary(Contain& source, Contain& dest,
    UnaryFunc f) {
    transform(source.begin(), source.end(),
dest.begin(), f);
}

template<typename Contain1, typename Contain2,
    typename BinaryFunc>
void testBinary(Contain1& src1, Contain1& src2,
    Contain2& dest, BinaryFunc f) {
    transform(src1.begin(), src1.end(),
        src2.begin(), dest.begin(), f);
}

// Executes the expression, then stringizes the
// expression into the print statement:
#define T(EXPR) EXPR; print(r.begin(), r.end(),
\
    "After " #EXPR);
// For Boolean tests:
#define B(EXPR) EXPR; print(br.begin(),
br.end(), \
    "After " #EXPR);

// Boolean random generator:
struct BRand {
    bool operator()() { return rand() % 2 == 0; }
};

int main() {
    const int SZ = 10;
    const int MAX = 50;
    vector<int> x(SZ), y(SZ), r(SZ);
    // An integer random number generator:
    URandGen urg(MAX);
    srand(time(0)); // Randomize
    generate_n(x.begin(), SZ, urg);
    generate_n(y.begin(), SZ, urg);
    // Add one to each to guarantee nonzero
divide:
    transform(y.begin(), y.end(), y.begin(),
        bind(plus<int>(), _1, 1));
    // Guarantee one pair of elements is ==:
    x[0] = y[0];
    print(x.begin(), x.end(), "x");
    print(y.begin(), y.end(), "y");
    // Operate on each element pair of x & y,
    // putting the result into r:

```

```

T(testBinary(x, y, r, plus<int>()));
T(testBinary(x, y, r, minus<int>()));
T(testBinary(x, y, r, multiplies<int>()));
T(testBinary(x, y, r, divides<int>()));
T(testBinary(x, y, r, modulus<int>()));
T(testUnary(x, r, negate<int>()));
vector<bool> br(SZ); // For Boolean results
B(testBinary(x, y, br, equal_to<int>()));
B(testBinary(x, y, br, not_equal_to<int>()));
B(testBinary(x, y, br, greater<int>()));
B(testBinary(x, y, br, less<int>()));
B(testBinary(x, y, br,
greater_equal<int>()));
B(testBinary(x, y, br, less_equal<int>()));
B(testBinary(x, y, br,
not2(greater_equal<int>())));

B(testBinary(x,y,br,not2(less_equal<int>())));
vector<bool> b1(SZ), b2(SZ);
generate_n(b1.begin(), SZ, BRand());
generate_n(b2.begin(), SZ, BRand());
print(b1.begin(), b1.end(), "b1");
print(b2.begin(), b2.end(), "b2");
B(testBinary(b1, b2, br,
logical_and<int>()));
B(testBinary(b1, b2, br, logical_or<int>()));
B(testUnary(b1, br, logical_not<int>()));
B(testUnary(b1, br,
not1(logical_not<int>())));
} ///:~

-----

x: 30 1 3 22 16 38 30 23 14 40
y: 30 8 50 6 9 25 19 43 33 26
After testBinary(x, y, r, plus<int>()): 60 9
53 28 25 63 49 66 47 66
After testBinary(x, y, r, minus<int>()): 0 -7
-47 16 7 13 11 -20 -19 14
After testBinary(x, y, r, multiplies<int>()):
900 8 150 132 144 950 570 989 462 1040
After testBinary(x, y, r, divides<int>()): 1 0
0 3 1 1 1 0 0 1
After testBinary(x, y, r, modulus<int>()): 0 1
3 4 7 13 11 23 14 14
After testUnary(x, r, negate<int>()): -30 -1
-3 -22 -16 -38 -30 -23 -14 -40
After testBinary(x, y, br, equal_to<int>()): 1
0 0 0 0 0 0 0 0 0
After testBinary(x, y, br,
not_equal_to<int>()): 0 1 1 1 1 1 1 1 1
After testBinary(x, y, br, greater<int>()): 0
0 0 1 1 1 1 0 0 1
After testBinary(x, y, br, less<int>()): 0 1 1
0 0 0 0 1 1 0
After testBinary(x, y, br,
greater_equal<int>()): 1 0 0 1 1 1 0 0 1
After testBinary(x, y, br, less_equal<int>()):
1 1 1 0 0 0 0 1 1 0
After testBinary(x, y, br,
not2(greater_equal<int>())): 0 1 1 0 0 0 0 1 1
0
After
testBinary(x,y,br,not2(less_equal<int>())): 0
0 0 1 1 1 1 0 0 1
b1: 1 1 1 0 0 1 0 1 0 1

```

```

b2: 1 0 0 0 0 1 0 0 1 0
After testBinary(b1, b2, br,
logical_and<int>()): 1 0 0 0 0 1 0 0 0 0
After testBinary(b1, b2, br,
logical_or<int>()): 1 1 1 0 0 1 0 1 1 1
After testUnary(b1, br, logical_not<int>()): 0
0 0 1 1 0 1 0 1 0
After testUnary(b1, br,
not1(logical_not<int>())): 1 1 1 0 0 1 0 1 0 1

```

```

-----

//: C06:ForEach.cpp {-mwc}
// From "Thinking in C++, Volume 2", by Bruce
Eckel & Chuck Allison.
// Use of STL for_each() algorithm.
//{L} Counted
#include <algorithm>
#include <iostream>
#include <vector>
#include <iostream>
#include <string>
#include <memory>
#include <iomanip>
#include <cmath>
using namespace std;

class Counted {
    static int count;
    shared_ptr<string> ident;
public:
    Counted(shared_ptr<string> id) : ident(id)
    { ++count; }
    ~Counted() {
        std::cout << *ident << " count = "
        << --count << std::endl;
    }
    const string& get_ident() const {return
*ident;}
};

class CountedVector : public
std::vector<Counted*> {
public:
    CountedVector(shared_ptr<string> id) {
        for(int i = 0; i < 5; i++)
            push_back(new Counted(id));
    }
};

// Function object:
template<class T> class DeleteT {
public:
    void operator()(T* x) { delete x; }
};

// Template function:
template<class T> void wipe(T* x) { delete x; }

int Counted::count = 0;

int main() {
    CountedVector B(make_shared<string>("two"));
    for_each(B.begin(), B.end(),
DeleteT<Counted>());

```

```

CountedVector
C(make_shared<string>("three"));
//lambda function with void return type
auto fun_1 = [](Counted* x){std::cout << x ->
get_ident() << " ";};
for_each(C.begin(), C.end(), fun_1);
int j = 10;
//lambda function that catches j
std::cout << std::endl;
std::cout << std::setprecision(10) << [j]
(double x) {return sqrt(x + j);}(3.5) <<
std::endl;
std::cout << std::setprecision(10) << [j]
(double x) -> float {return sqrt(x + j);}(3.5)
<< std::endl;
//non-mutable lambda with two arguments
int epic_counter = 0;
auto fun_3 = [&epic_counter](Counted* x,
string v) {
    epic_counter++;
    std::cout << x -> get_ident() + " " + v +
": " << epic_counter << " ";
};
using namespace std::placeholders;
for_each(C.begin(), C.end(), std::bind(fun_3,
_1, "hooray!"));
std::cout << std::endl << "Epic counter = "
<< epic_counter << std::endl;
//mutable lambda with two arguments
epic_counter = 0;
auto fun_4 = [epic_counter](Counted* x,
string v) mutable {
    epic_counter++;
    std::cout << x -> get_ident() + " " + v +
": " << epic_counter << " ";
};
for_each(C.begin(), C.end(), std::bind(fun_4,
_1, "yarooh!"));
std::cout << std::endl << "Epic counter = "
<< epic_counter << std::endl;

for_each(C.begin(), C.end(), wipe<Counted>);
} ///:~

```

```

-----

two count = 4
two count = 3
two count = 2
two count = 1
two count = 0
three three three three three
3.674234614
3.674234629
three hooray!: 1; three hooray!: 2; three
hooray!: 3; three hooray!: 4; three hooray!: 5;
Epic counter = 5
three yarooh!: 1; three yarooh!: 2; three
yarooh!: 3; three yarooh!: 4; three yarooh!: 5;
Epic counter = 0
three count = 4
three count = 3
three count = 2
three count = 1
three count = 0

```