Range-Based for

- It's common to process *all* the elements of an array. The C++11 range based for statement allows you to do this *without using a counter*.
- This avoids the possibility of "stepping outside" the array and eliminates the need to do your own bounds checking.
- When processing all elements of an array, if you don't need access to an array elements's index, use the range-based for statement.

```
int array[] = {1, 2, 3, 4, 5};

cout << "items before modification: ";
for (int item: array)        // access by value
{
    cout << item << " ";
}
cout << endl;

// multiply the elements of items by 2
for (int& itemRef : array)   // access by reference
{
    itemRef *= 2;
}

cout << "items after modififcation: ";
for (int item : array)
{
    cout << item << " ";
}
cout << endl;
```

- Notice when range-based for accesses the elements *by value* in the array, a copy is passed. When modifying values in the array elements need be access *by reference* to modify the original values in the array.
- The range-based for can be used in place of a counter-controlled for when the code looping through the array does not require access to the element's index.
- Range-based for may be used with most of C++ Standard Library's prebuilt data structures.
- C++11 also allows the `auto` keyword in the range-based for, which tells the compiler to infer (determine) a variables's data type based on the initializer value.

```
cout << "items before modification: ";
for (auto item: array)       // access by value
{
    cout << item << " ";
}
cout << endl;
```

- When compiling for C++11, use `-std=c++11` parameter to compile from terminal, e.g.,

```
c++ -std=c++11 main.cpp -o test
```