- Solution

Let's have a look at the solution to the last exercise in this lesson.

WE'LL COVER THE FOLLOWING

Solution Review

```
// templatesTagDispatchingImplementation.cpp

#include <iterator>
#include <forward_list>
#include <list>
#include <vector>
#include <iostream>

template <typename InputIterator, typename Distance>
void advance_impl(InputIterator& i, Distance n, std::input_iterator_tag) {
        std::cout << "InputIterator used" << std::endl;
    while (n--) ++i;
}

template <typename BidirectionalIterator, typename Distance>
void advance_impl(BidirectionalIterator& i, Distance n, std::bidirectional_iterator_tag) {
        std::cout << "BidirectionalIterator used" << std::endl;
    if (n >= 0)
        while (n--) ++i;
    else
        while (n++) --i;
}

template <typename RandomAccessIterator, typename Distance>
void advance_impl(RandomAccessIterator& i, Distance n, std::random_access_iterator_tag) {
        std::cout << "RandomAccessIterator used" << std::endl;
    i += n;
}

template <typename InputIterator, typename Distance>
void advance_(InputIterator& i, Distance n) {
    typename std::iterator_traits<InputIterator>::iterator_category category;
    advance_impl(i, n, category);
}

int main(){


    std::cout << std::endl;

    std::vector<int> myVec{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    auto myVecIt = myVec.begin();
    std::cout << "*myVecIt: " << *myVecIt << std::endl;
    advance_(myVecIt, 5);
    std::cout << "*myVecIt: " << *myVecIt << std::endl;

    std::cout << std::endl;

    std::list<int> myList{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    auto myListIt = myList.begin();
    std::cout << "*myListIt: " << *myListIt << std::endl;
    advance_(myListIt, 5);
    std::cout << "*myListIt: " << *myListIt << std::endl;

    std::cout << std::endl;

    std::forward_list<int> myForwardList{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    auto myForwardListIt = myForwardList.begin();
    std::cout << "*myForwardListIt: " << *myForwardListIt << std::endl;
    advance_(myForwardListIt, 5);
    std::cout << "*myForwardListIt: " << *myForwardListIt << std::endl;

    std::cout << std::endl;

}
```

Explanation

The expression `std::iterator_traits::iterator_category` category in line 32 determines the iterator category at compile-time. Based on the iterator category, the most specific variant of the function template `advance_impl(i, n, category)` is used in line 33. Each container returns an iterator of the iterator category which corresponds to its structure. Therefore, line 41 gives a random-access iterator, line 49 gives a bidirectional iterator, and line 57 gives a forward iterator, which is also an input iterator.

Performance

From the performance point of view, this distinction makes a lot of sense because a random-access iterator can be more quickly incremented than a bidirectional iterator, and a bidirectional iterator can be more quickly incremented than an input iterator. From the users perspective, we invoked `std::advance(it, 5)` in lines 43, 51, and 59 to get the fastest version of the function template which our container satisfies.

Let’s move on to type erasure idioms and patterns in the next lesson.