6 Counting Strings of Equal lengths in a Vector

Write three wrapper functions with the following prototypes:

```
int testCountStringsLambda (const std::vector<std::string>& vec, int n);
int testCountStringsFreeFun(const std::vector<std::string>& vec, int n);
int testCountStringsFunctor(const std::vector<std::string>& vec, int n);
```

Each function must return the number of strings of length `n` in the `vec`.

For example, suppose

```
std::vector<std::string> vec { "C", "BB", "A", "CC", "A", "B",
                               "BB", "A", "D", "CC", "DDD", "AAA" };
```

Then, for example, the call to any of your wrapper functions with the arguments `(vec, 1)`, `(vec, 2)`, `(vec, 3)`, and `(vec, 4)`, should return 6, 4, 2, and 0, respectively.

Your wrapper functions must each use the `count_if` algorithm from the `<algorithm>` header file.

```
// Returns the number of elements in the range [first,last) for which pred is true.
template <class InputIterator, class UnaryPredicate>          // template header
  typename iterator_traits<InputIterator>::difference_type  // return type
    count_if (InputIterator first,                          // vec.begin()
              InputIterator last,                           // vec.end()
              UnaryPredicate pred);                         // a unary predicate
```

You should implement three versions of the unary predicate, with each version taking a `std::string` as their only parameter and returning whether or not that `std::string` is of length `n`. Limited to implementing unary predicates, we need to figure out how to introduce `n` into these functions. Here are some hints:

A. A lambda expression named `Lambda` — Capture `n` in the lambda introducer

B. A free function named `FreeFun` — Write a `bool FreeFun(std::string, int)` and then turn it into a "unary" function by fixing its 2nd argument to `n` using std::bind.

C. A functor (function object) named `Functor` — Give your functor a data member to store `n` at construction.

Recall that an object or expression is callable if the call operator can be applied to it.

Have your three wrapper functions demonstrate these three implementations of the unary predicate argument, respectively.