```
#include <algorithm>  // std::sort

bool nameLt(const Employee& e1, const Employee& e2)
    // returns true if e1.name is less than e2.name
{
    return e1.name < e2.name;
}

void sortByName(std::vector<Employee>& employees)
{
    std::sort(employees.begin(), employees.end(), &nameLt);
}
```

The `sortBySalary` function can similarly delegate to `std::sort`. For illustrative purposes, we will use a **function object** (i.e., functor) rather than a function pointer as the callback to compare the salaries of two `Employee` objects. Every **functor class** must provide a **call operator** (i.e., **`operator()`**), which, in this case, compares the salary fields of its arguments:

```
struct SalaryLt
{
    // functor whose call operator compares two Employee objects and returns
    // true if the first has a lower salary than the second, false otherwise

    bool operator()(const Employee& e1, const Employee& e2) const
    {
        return e1.salary < e2.salary;
    }
};

void sortBySalary(std::vector<Employee>& employees)
{
    std::sort(employees.begin(), employees.end(), SalaryLt());
}
```

Although it is a bit more verbose, a call through the function object ~~is easier for the compiler to analyze and automatically inline within `std::sort` than is a call through the function pointer~~. Function objects are also more flexible because they can carry state, as we'll see shortly. Furthermore, if a function object is stateless, it is cheaper to pass around than a function pointer because nothing needs to be copied. The sorting example illustrates how small bits of a function's logic must be factored out into special-purpose auxiliary functions or functor classes that are often not reusable. It is possible, for example, that the `nameLt` function and `SalaryLt` class are not used anywhere else in the program.

When callbacks are tuned to the specific context in which they are used, they become both more complicated and less reusable. Let's say, for example, that we want to count the number of employees whose salary is above the average for the collection. Using Standard