Extending acceptable predicates and functions in standard library algorithms

Document #:	P0XXXR0
Date:	2018-03-13
Project:	Programming Language C++
Audience:	LEWG

Reply-to: Georg Rudoy <georgii.rudoi@phystech.edu>

1 Revision history

- R0 — Initial draft

2 Abstract

This paper proposes a (strictly relaxing) change in the definitions of several algorithms in the standard library: `std::any_of`, `std::find_if` and the likes are currently defined in terms of function call syntax on the predicate/function, forbidding passing pointers to members where they otherwise would be sufficient.

3 Motivation

```
struct Standard
{
    bool isCurrent;
};

std::vector<Standard> stds;

auto hasCurrent = std::any_of(stds.begin(), stds.end(),
        &Standard::isCurrent);                                  // nope
auto hasCurrent = std::any_of(stds.begin(), stds.end(),
        [](auto&& std) { return std.isCurrent; });              // ok

class Employee
{
public:
    bool shouldBeFired() const;
```

```
    void fire();
};

std::vector<Employee> emps;

auto poorFellow = std::find_if(emps.begin(), emps.end(),
        &Employee::shouldBeFired);                              // nope
auto poorFellow = std::find_if(emps.begin(), emps.end(),
        [](auto&& emp) { return emp.shouldBeFired(); });        // ok

std::for_each(emps.begin(), emps.end(), &Employee::fire);       // nope
std::for_each(emps.begin(), emps.end(),
        [](auto&& emp) { emp.fire(); });                        // ok
```

```
class Employee
{
public:
    bool knowsLanguage(const Language& lang) const;
};

std::vector<Employee> emps;
std::vector<Language> langs;

auto translator = std::find_first_of(emps.begin(), emps.end(),
        langs.begin(), langs.end(),
        &Employee::knowsLanguage);                              // nope
auto translator = std::find_first_of(emps.begin(), emps.end(),
        langs.begin(), langs.end(),
        [](auto&& emp, auto&& lang) { return emp.knowsLanguage(lang); });
                                                                // ok
```

4 The fix

Replace `pred(*i)` and the likes with `std::invoke(pred, *i)` and the likes.

Clarify the passed function object should be applied using `std::invoke` where relevant.

5 Proposed wording

In **[alg.all_of]/2**:

> *Returns:* `true` if `[first, last)` is empty or if ~~`pred(*i)`~~ `std::invoke(pred, *i)` is `true` for every iterator `i` in the range `[first, last)`, and `false` otherwise.

In **[alg.any_of]/2**: ditto.

In **[alg.none_of]/2**: ditto.

In **[alg.find]/2**: ditto.

In **[alg.count]/1**: ditto.

In **[alg.foreach]/2**:

> *Effects:* Applies `f` using `std::invoke` to the result of dereferencing every iterator in the range `[first, last)`, starting from `first` and proceeding to `last - 1`. [*Note:* If the type of `first` satisfies the requirements of a mutable iterator, `f` may apply non-constant functions through the dereferenced iterator. — *end note*]

In **[alg.foreach]/7**: ditto.

In **[alg.foreach]/13**: ditto.

In **[alg.foreach]/18**: ditto.

In **[alg.find.end]/2**:

> *Returns*: The last iterator `i` in the range `[first1, last1 - (last2 - first2))` such that for every non-negative integer `n < (last2 - first2)`, the following corresponding conditions hold: `*(i + n) == *(first2 + n)`, ~~`pred(*(i + n), *(first2 + n)) != false`~~ `std::invoke(pred, *(i + n), *(first2 + n)) != false`. Returns `last1` if `[first2, last2)` is empty or if no such iterator is found.

In **[alg.find.first.of]/2**: ditto.

In **[alg.adjacent.find]/1**: ditto.

In **[alg.mismatch]/2**: ditto.

In **[alg.equal]/2**: ditto.

In **[alg.is_permutation]/3**: ditto.

In **[alg.search]/2**: ditto.