Document Number: P0253R0
Date: 2016-02-12
Audience: Library Evolution Working Group
Reply to: Marshall Clow <marshall@idio.com>

Fixing a design mistake in the searchers interface in Library Fundamentals

Rationale

In N3703 (and previous papers), I introduced the concept of "searchers", objects that implement a search algorithm with a common interface. They each take a pattern to search for in their constructor, and a corpus to search in their operator() method. They return an iterator to the start of the pattern in the corpus, or end if the pattern is not found. This is the behavior of std::search as well.

I have come to the conclusion that this is the wrong thing to return. When you are doing multiple searches in the same corpus, frequently you want to start the next search after the thing that you just found. You can calculate that information yourself, by advancing the iterator returned to you the appropriate number of times. But that's (potentially) an O(N) operation, and I can imagine (regex) a matcher where the pattern length may not be fixed.

I believe that the correct behavior here is to have the searchers return a pair of iterators, denoting the position of the pattern in the corpus. I came to this conclusion while attempting to write a split algorithm, which separates a sequence into a series, delimited by a separator. Recovering the end of the matched pattern was a necessity for this algorithm. See https://cplusplusmusings.wordpress.com/2016/02/01/sometimes-you-get-things-wrong/ for a description of the algorithm.

I propose changing the searchers to return the beginning and end of the pattern found. They already have this information, since they have to traverse the entire pattern in the corpus to verify that it is a match. I am not proposing that std::experimental::search return this information, because that would not be compatible with std::search.

Wording changes (relative to N4564)

Change section 4.3.1 [func.searchers.default]:

```
template<class ForwardIterator1, class BinaryPredicate = equal_to<>>
class default_searcher {
public:
  default_searcher(ForwardIterator1 pat_first, ForwardIterator1
pat_last,
                   BinaryPredicate pred = BinaryPredicate());

  template<class ForwardIterator2>
  ForwardIterator2
  pair<ForwardIterator2, ForwardIterator2>
  operator()(ForwardIterator2 first, ForwardIterator2 last) const;

private:
  ForwardIterator1 pat_first_; // exposition only
  ForwardIterator1 pat_last_;  // exposition only
  BinaryPredicate pred_;       // exposition only
};

. . .
```

5 *Effects:* Equivalent to `return` Returns a pair of iterators `i` and `j` such that `i == std::search(first, last, pat_first_, pat_last_, pred_);` and `j == next(i, distance(pat_first_, pat_last_))`

Change section 4.3.2 [func.searchers.boyer_moore]:

```
template<class RandomAccessIterator1,
         class Hash = hash<typename
iterator_traits<RandomAccessIterator1>::value_type>,
         class BinaryPredicate = equal_to<>>
class boyer_moore_searcher {
public:
  boyer_moore_searcher(RandomAccessIterator1 pat_first,
RandomAccessIterator1 pat_last,
                       Hash hf = Hash(), BinaryPredicate pred =
BinaryPredicate());

  template<class RandomAccessIterator2>
  RandomAccessIterator2
  pair<RandomAccessIterator2,RandomAccessIterator2>
  operator()(RandomAccessIterator2 first, RandomAccessIterator2 last)
const;

private:
  RandomAccessIterator1 pat_first_; // exposition only
  RandomAccessIterator1 pat_last_;  // exposition only
  Hash hash_;                       // exposition only
  BinaryPredicate pred_;            // exposition only
};

. . .
```

9 *Returns:* Returns a pair of iterators `i` and `j` such that `i` is the first iterator `i` in the range [`first`, `last - (pat_last_ - pat_first_)`) such that for every non- negative integer `n` less than `pat_last_ - pat_first_` the following condition holds: `pred(*(i + n), *(pat_first_ + n)) != false`, and `j == next(i, distance(pat_first_, pat_last_))`. Returns `first` `make_pair(first,first)` if [`pat_first_`, `pat_last_`) is empty, otherwise returns `last` `make_pair(last,last)` if no such iterator is found.

Change section 4.3.3 [func.searchers.boyer_moore_horspool]:

```
template<class RandomAccessIterator1,
         class Hash = hash<typename
iterator_traits<RandomAccessIterator1>::value_type>,
         class BinaryPredicate = equal_to<>>
class boyer_moore_horspool_searcher {
public:
  boyer_moore_horspool_searcher(RandomAccessIterator1 pat_first,
RandomAccessIterator1 pat_last,
                                Hash hf = Hash(), BinaryPredicate pred =
BinaryPredicate());

  template<class RandomAccessIterator2>
  RandomAccessIterator2
  pair<RandomAccessIterator2, RandomAccessIterator2>
  operator()(RandomAccessIterator2 first, RandomAccessIterator2 last) const;

private:
  RandomAccessIterator1 pat_first_; // exposition only
  RandomAccessIterator1 pat_last_;  // exposition only
  Hash                  hash_;      // exposition only
  BinaryPredicate       pred_;      // exposition only
};
```

. . .

9 *Returns:* Returns a pair of iterators `i` and `j` such that `i` is the first iterator `i` in the range [`first`, `last - (pat_last_ - pat_first_)`) such that for every non- negative integer `n` less than `pat_last_ - pat_first_` the following condition holds: `pred(*(i + n), *(pat_first_ + n)) != false`, and `j == next(i, distance(pat_first_, pat_last_))`. Returns `first` `make_pair(first,first)` if [`pat_first_`, `pat_last_`) is empty, otherwise returns `last` `make_pair(last,last)` if no such iterator is found.

Change section 12.2 [alg.search]:

2 *Effects:* Equivalent to `return searcher(first, last).first;`

Example

Here’s the implementation of split that I ended up with.

```
      template <typename Iter, typename Searcher, typename OutIter>
      OutIter split(Iter first, Iter last, const Searcher &s, OutIter
out)
      {
          while (first != last)
          {
              std::pair<Iter, Iter> found = s(first, last);
              *out++ = std::make_pair(first, found.first);
          //  if the pattern is found at the end of the input,
          //    output an empty chunk.
              if (found.second == last && found.first != found.second)
                  *out++ = std::make_pair(last, last);
              first = found.second; // start the next search here
          }
          return out;
      }
```