Implementation of std::rotate algorithm

Aditya Kumar

Aug. 3, 2016

std::rotate(Introduction)

- What is std::rotate
- Uses of std::rotate

What is std::rotate

```
// I models Forward Iterator
template<class I> // (until C++11)
void rotate(I first, I n_first, I last);

template<class I> // (since C++11)
I rotate(I first, I n_first, I last);

Performs a left rotation on a range of elements.

vector<int> s{0, 1, 2, 2, 3, 4, 5, 7, 7, 10};
rotate(s.begin(), s.begin()+1, s.end());
// rotate left : 1, 2, 2, 3, 4, 5, 7, 7, 10, 0
```

Uses: Inserting elements in vector using rotate in O(n)

```
// I models Input Iterator to T
template <typename T, typename I>
void insert(vector<T>& v,
            typename vector<T>::iterator ip,
            I first, I limit) {
  auto n = v.end() - v.begin();
  while (first != limit) {
    v.push_back(*first);
    ++first;
  }
  rotate(ip, v.begin() + n, v.end());
}
```

Uses: Sliding a continuous set of elements in a container to another position

```
// It models Random Access Iterator
template <typename It>
auto slide(It f, It l, It p) -> std::pair<It, It> {
  if (p < f) return { p, std::rotate(p, f, l) };
  if (l < p) return { std::rotate(f, l, p), p };
  return { f, l };
}
```

Implementation using bidirectional iterator

```
// I models Bidirectional Iterator
// f: first
// m: rotation point
// l: limit
// [f, l) is valid and m is in [f, l)
template <typename I>
void rotate(I f, I m, I l) {
  reverse(f, m);
  reverse(m, l);
  reverse(f, l);
}
```

Implementation using forward iterator

```
// I models Forward Iterator
// f: first
// m: rotation point
// l: limit
// [f, l) is valid and m is in [f, l)
template <typename I>
void rotate(I f, I m, I l) {
  pair<I, I> p = swap_ranges(f, m, m, l);
  I u = p.first;
  I v = p.second;
  if (v != l)
    rotate(u, v, l);
  else if (u != m)
    rotate(u, m, l);
}
```

Implementation using random-access iterator

```
// I models Forward Iterator
// f: first
// m: rotation point
// l: limit
// [f, l) is valid and m is in [f, l)
template <typename I>
pair<I, I> rotate(I f, I m, I l) {
  DISTANCE_TYPE(I) n = gcd(m - f, l - m);
  rotate_iterator_action<I> action(f, m, l);
  while (n > 0) {
    --n;
    do_cycle(f + n, action);
  }
  I n_m = f + (l - m);
  return (n_m < m) ? pair<I, I>(n_m, m)
                   : pair<I, I>(m, n_m);
}
```