11.1. Permutation Generation

Section authors: Jeff Czarnowski, Andy Schechter, Tom Smith.

Refinement of: Comparison Based (§2.2), Permuting (§2.5), therefore of Sequence Algorithm (§2.1).
Prototype:

```cpp
template <class BidirectionalIterator>
bool next_permutation(
    BidirectionalIterator first,
    BidirectionalIterator last);

template <class BidirectionalIterator, 
    class StrictWeakOrdering>
bool next_permutation(
    BidirectionalIterator first,
    BidirectionalIterator last,
    StrictWeakOrdering comp);

template <class BidirectionalIterator>
bool prev_permutation(
    BidirectionalIterator first,
    BidirectionalIterator last);

template <class BidirectionalIterator, 
    class StrictWeakOrdering>
bool prev_permutation(
    BidirectionalIterator first,
    BidirectionalIterator last,
    StrictWeakOrdering comp);
```
Input: A sequence of comparable elements. See Sequence Algorithm (§2.1).

Output: Next_permutation transforms the range of elements into the next lexicographically greater permutation of the elements. If this permutation exists, the function returns true. If not, then it transforms the given range into the lexicographically smallest permutation and returns false. This is a mutable algorithm, so the container is altered to reflect the changes caused by the algorithm. Prev_permutation similarly generates the next lexicographically lesser permutation of the elements.

Effects: Standard effects of a Sequence Permuting Algorithm (§2.5).

This algorithm will change the container between the two given iterators to assign the values of the next greatest permutation. For example, if a range initially contained \{1, 2, 3, 4\}, this would transform it to \{1, 2, 4, 3\}. Called once again on the new data, it would yield \{1, 3, 2, 4\}. If the next greatest permutation does not exist, then the range would equal the lowest permutation. For example, \{4, 3, 2, 1\} would be changed to
\{1, 2, 3, 4\}. This is the only case where the function would return false. Thus, this can be used for a very slow sorting routine that would call the next_permutation function repeatedly until it returns false. Of course, since there are \(N!\) different permutations, the running time of this sort will be of the order \(O(N!)\).

**Asymptotic complexity:** Let \(N = \text{last} - \text{first}\).

- Average case (random data): \(O(N)\)
- Worst case: \(O(N)\)
Complexity in terms of operation counts:

- **Average case:**
  - **next_permutation:**
    - total: 0.0564
    - iterator assign.: 0.0198
    - iterator comp.: 0.0048
    - value assign.: 0.0060
    - value comp.: 0.0034
  - **prev_permutation:**
    - total: 0.0500
    - iterator assign.: 0.0192
    - iterator comp.: 0.0042
    - value assign.: 0.0060
    - value comp.: 0.0022
  - **reverse:**
    - total: $7.5N + 0.0055$
    - iterator assign.: $2.5N + 0.0042$
    - iterator comp.: $0.5N + 0.0010$
    - value assign.: $1.5N$
    - value comp.: 0
\begin{itemize}
  \item \textbf{Worst case:} \\
  \quad \text{permutation generation:} \\
  \quad \quad \text{total:} \quad 13.5N + 0.0079 \\
  \quad \quad \text{iterator assign.:} \quad 3.5N + 0.0068 \\
  \quad \quad \text{iterator comp.:} \quad 1.5N \\
  \quad \quad \text{value assign.:} \quad 1.5N \\
  \quad \quad \text{value comp.:} \quad N - 0.0011 \\
  \quad \text{prev\_permutation:} \\
  \quad \quad \text{total:} \quad 13.5N + 0.0079 \\
  \quad \quad \text{iterator assign.:} \quad 3.5N + 0.0068 \\
  \quad \quad \text{iterator comp.:} \quad 1.5N \\
  \quad \quad \text{value assign.:} \quad 1.5N \\
  \quad \quad \text{value comp.:} \quad N - 0.0011 \\
  \quad \text{reverse:} \\
  \quad \quad \text{Same as average} \\
  \quad \quad \text{case.} \\
  \item \textbf{Note:} reverse is included for comparison because both next\_permutation and prev\_permutation \\
  \quad rely on it. Results are scaled by 1000 and based upon the Operation Count tables below.
\end{itemize}
Table 1: Performance of Permutation Generation Algorithms on Random Sequences (Sizes and Operations Counts in Multiples of 1,000)

<table>
<thead>
<tr>
<th>Size</th>
<th>Alg</th>
<th>Iterator Comp</th>
<th>Iterator Assign</th>
<th>Value Comp</th>
<th>Value Assign</th>
<th>Other Ops</th>
<th>Total Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>next_p</td>
<td>0.005</td>
<td>0.02</td>
<td>0.003</td>
<td>0.006</td>
<td>0.021</td>
<td>0.055</td>
</tr>
<tr>
<td></td>
<td>prev_p</td>
<td>0.004</td>
<td>0.019</td>
<td>0.002</td>
<td>0.006</td>
<td>0.018</td>
<td>0.049</td>
</tr>
<tr>
<td></td>
<td>reverse</td>
<td>0.501</td>
<td>2.504</td>
<td>0</td>
<td>1.5</td>
<td>3</td>
<td>7.505</td>
</tr>
<tr>
<td>4</td>
<td>next_p</td>
<td>0.005</td>
<td>0.02</td>
<td>0.004</td>
<td>0.006</td>
<td>0.024</td>
<td>0.059</td>
</tr>
<tr>
<td></td>
<td>prev_p</td>
<td>0.004</td>
<td>0.019</td>
<td>0.002</td>
<td>0.006</td>
<td>0.018</td>
<td>0.049</td>
</tr>
<tr>
<td></td>
<td>reverse</td>
<td>2.001</td>
<td>10.004</td>
<td>0</td>
<td>6</td>
<td>12</td>
<td>30.005</td>
</tr>
<tr>
<td>16</td>
<td>next_p</td>
<td>0.004</td>
<td>0.019</td>
<td>0.002</td>
<td>0.006</td>
<td>0.018</td>
<td>0.049</td>
</tr>
<tr>
<td></td>
<td>prev_p</td>
<td>0.005</td>
<td>0.02</td>
<td>0.003</td>
<td>0.006</td>
<td>0.021</td>
<td>0.055</td>
</tr>
<tr>
<td></td>
<td>reverse</td>
<td>8.001</td>
<td>40.004</td>
<td>0</td>
<td>24</td>
<td>48</td>
<td>120.005</td>
</tr>
<tr>
<td>64</td>
<td>next_p</td>
<td>0.005</td>
<td>0.02</td>
<td>0.004</td>
<td>0.006</td>
<td>0.018</td>
<td>0.049</td>
</tr>
<tr>
<td></td>
<td>prev_p</td>
<td>0.004</td>
<td>0.019</td>
<td>0.002</td>
<td>0.006</td>
<td>0.024</td>
<td>0.059</td>
</tr>
<tr>
<td></td>
<td>reverse</td>
<td>32.001</td>
<td>160.004</td>
<td>0</td>
<td>96</td>
<td>192</td>
<td>480.005</td>
</tr>
<tr>
<td>256</td>
<td>next_p</td>
<td>0.005</td>
<td>0.02</td>
<td>0.004</td>
<td>0.006</td>
<td>0.018</td>
<td>0.049</td>
</tr>
<tr>
<td></td>
<td>prev_p</td>
<td>0.004</td>
<td>0.019</td>
<td>0.002</td>
<td>0.006</td>
<td>0.18</td>
<td>0.049</td>
</tr>
<tr>
<td></td>
<td>reverse</td>
<td>128.001</td>
<td>640.004</td>
<td>0</td>
<td>384</td>
<td>768</td>
<td>1920.01</td>
</tr>
<tr>
<td>1024</td>
<td>next_p</td>
<td>0.004</td>
<td>0.019</td>
<td>0.002</td>
<td>0.006</td>
<td>0.018</td>
<td>0.049</td>
</tr>
<tr>
<td></td>
<td>prev_p</td>
<td>0.005</td>
<td>0.02</td>
<td>0.003</td>
<td>0.006</td>
<td>0.021</td>
<td>0.055</td>
</tr>
<tr>
<td></td>
<td>reverse</td>
<td>512.001</td>
<td>2560</td>
<td>0</td>
<td>1536</td>
<td>3072</td>
<td>7680.01</td>
</tr>
</tbody>
</table>
Table 2: Performance of Permutation Generation Algorithms on Worst-Case Sequences (Sizes and Operations Counts in Multiples of 1,000)

| Size | Alg     | Iterator Comp | | Iterator Assign | | Value Comp | | Value Assign | | Other Ops | | Total Ops |
|------|---------|---------------||----------------||----------------||---------------||---------------||-------------||-------------|
| 1    | next_p | 1.502         | | 3.507         | | 0.999        | | 1.5         | | 5.999      | | 13.507     |
|      | prev_p | 1.502         | | 3.507         | | 0.999        | | 1.5         | | 5.999      | | 13.507     |
| 4    | next_p | 6.002         | | 14.007        | | 3.999        | | 6           | | 23.999     | | 54.007     |
|      | prev_p | 6.002         | | 14.007        | | 3.999        | | 6           | | 23.999     | | 54.007     |
| 16   | next_p | 24.002        | | 56.007        | | 15.999       | | 24          | | 95.999     | | 216.01     |
|      | prev_p | 24.002        | | 56.007        | | 15.999       | | 24          | | 95.999     | | 216.01     |
| 64   | next_p | 96.002        | | 224.01        | | 63.999       | | 96          | | 384        | | 864.01     |
|      | prev_p | 96.002        | | 224.01        | | 63.999       | | 96          | | 384        | | 864.01     |
| 256  | next_p | 384.00        | | 896.01        | | 256          | | 384         | | 1536       | | 3456.01    |
|      | prev_p | 384.00        | | 896.01        | | 256          | | 384         | | 1536       | | 3456.01    |
| 1024 | next_p | 1536          | | 3584.01       | | 1024         | | 1536        | | 6144       | | 13824      |
|      | prev_p | 1536          | | 3584.01       | | 1024         | | 1536        | | 6144       | | 13824      |

Note: A worst case sequence for next_permutation is a sequence sorted in descending order. A worst case sequence for prev_permutation is a sequence sorted in ascending order.
`NEXT-PERMUTATION(first, last)`

1. if `first = last` then
2. return FALSE
3. end if
4. `iteratorA ← first`
5. `iteratorA ← iteratorA + 1`
6. if `iteratorA = last` then
7. return FALSE
8. end if
9. `iteratorA ← last`
10. `iteratorA ← iteratorA - 1`
11. while TRUE do
12. `iteratorB ← iteratorA`
13. `iteratorA ← iteratorA - 1`
14. if `value[iteratorA] < value[iteratorB]` then
15. `iteratorC ← last`
16. while `value[iteratorA] > value[iteratorC]` do
17. `iteratorC ← iteratorC - 1`
18. end while
19. `ITER-SWAP(iteratorA, iteratorC)`
20. `REVERSE(iteratorB, last)`
21. return TRUE
22. end if
if iteratorA = first then
    REVERSE(first, last)
    return FALSE
end if
end while
PREV-PERMUTATION(first, last)

1: if first = last then
2: return FALSE
3: end if
4: iteratorA ← first
5: iteratorA ← iteratorA + 1
6: if iteratorA = last then
7: return FALSE
8: end if
9: iteratorA ← last
10: iteratorA ← iteratorA - 1
11: while TRUE do
12: iteratorB ← iteratorA
13: iteratorA ← iteratorA - 1
14: if value[iteratorB] < value[iteratorB] then
15: iteratorC ← last
16: while value[iteratorC] > value[iteratorA] do
17: iteratorC ← iteratorC - 1
18: end while
19: ITER-SWAP(iteratorA, iteratorC)
20: REVERSE(iteratorB, last)
21: return TRUE
22: end if
23: if iteratorA = first then
24:       REVERSE(first, last)
25:       return FALSE
26:     end if
27: end while
REVERSE(first, last)
1: while TRUE do
2:   temp ← last − 1
3:   if first = last or first = temp then
4:     last ← last − 1 RETURN
5:   else
6:     first ← first + 1
7:     ITER-SWAP(first, last)
8:   end if
9: end while

ITER-SWAP(iterator1, iterator2)
1: temp ← value[iterator1]
2: value[iterator1] ← value[iterator2]
3: value[iterator2] ← temp