distance\_type n = **distance**(first, last);  
**advance**(i, n);  
Function result = **for\_each**(first, last, func);  
InputIter loc = **find**(first, last, target);  
InputIter loc = **find\_if**(first, last, pred);  
ForwardIter loc = **adjacent\_find**(first, last[, bin\_pred]);  
ForwardIter loc = **find\_first\_of**(first, last, targets\_first, targets\_last[, bin\_pred]);  
ForwardIter loc = **search**(first, last, subseq\_first, subseq\_last[, bin\_pred]);  
ForwardIter loc = **search\_n**(first, last, count, target[, bin\_pred]);  
ForwardIter loc = **find\_end**(first, last, subseq\_first, subseq\_last[, bin\_pred]);  
difference\_type n = **count**(first, last, value);  
difference\_type n = **count\_if**(first, last, value, pred);  
pair<InputIter1, InputIter2> differ = **mismatch**(first1, last1, first2[, bin\_pred]);  
bool is\_equal = **equal**(first1, last1, first2[, bin\_pred]);  
OutputIter result\_end = **copy**(first, last, result);  
BidirectionalIter result\_begin = **copy\_backward**(first, last, result\_end);  
**swap**(a, b);   
ForwardIter2 new\_last2 = **swap\_ranges**(first1, last1, first2);  
OutputIter result\_end = **transform**(first, last, result, op);  
OutputIter result\_end = **transform**(first1, last2, first2, result, bin\_op);  
**replace** (first, last, old\_value, new\_value);  
**replace\_if**(first, last, pred, new\_value);  
OutputIter result\_end = **replace\_copy**(first, last, result, old\_value, new\_value);  
OutputIter result\_end = **replace\_copy\_if**(first, last, result, pred, new\_value);  
**fill**(first, last, value);  
**fill\_n**(first, count, value);  
**generate**(first, last, generator);  
**generate\_n**(first, count, generator);  
ForwardIter new\_last **remove**(first, last, value);  
ForwardIter new\_last **remove\_if**(first, last, pred);  
OutputIter result\_end **remove\_copy**(first, last, result, value);  
OutputIter result\_end **remove\_copy\_if**(first, last, result, pred);  
ForwardIter new\_last = **unique**(first, last[, bin\_pred]);  
ForwardIter result\_end = **unique\_copy**(first, last[, bin\_pred]);  
**reverse**(first, last);  
OutputIter result\_end = **reverse\_copy**(first, last, result);  
**rotate**(first, middle, last);  
OutputIter result\_end = **rotate\_copy**(first, middle, last, result);  
**random\_shuffle**(first, last[, rand]);  
BidirectionalIter middle = **partition**(first, last, pred);  
BidirectionalIter middle = **stable\_partition**(first, last, pred);  
**sort**(first, last[, comp]);  
**stable\_sort**(first, last[, comp]);  
**partial\_sort**(first, middle, last[, comp]);  
RandomAccessIter result\_end = **partial\_sort\_copy**(first, middle, last, result\_first, result\_last[, comp]);  
**nth\_element**(first, nth, last[, comp]);  
ForwardIter loc = **lower\_bound**(first, last, value[, comp]);  
ForwardIter loc = **upper\_bound**(first, last, value[, comp]);  
pair<ForwardIter, ForwardIter> loc\_range = **equal\_range**(first, last, value[, comp]);  
bool is\_present = **binary\_search**(first, last, value[, comp]);  
OutputIter result\_end = **merge**(first1, last1, first2, last2, result[, comp]);  
**inplace\_merge**(first, middle, last[, comp]);  
bool is\_included = **includes**(first, last, subseq\_first, subseq\_last[, comp]);  
OutputIter result\_end = **set\_union**(first1, last1, first2, last2, result[, comp]);  
OutputIter result\_end = **set\_intersection**(first1, last1, first2, last2, result[, comp]);  
OutputIter result\_end = **set\_difference**(first1, last1, first2, last2, result[, comp]);  
OutputIter result\_end = **set\_symetric\_difference**(first1, last1, first2, last2, result[, comp]);  
minimum = **min**(a, b);   
maximum = **max**(a, b);  
ForwardIter loc = **min\_element**(first, last[, bin\_pred]);  
ForwardIter loc = **max\_element**(first, last[, bin\_pred]);  
bool is\_less = **lexicographical\_compare**(first1, last1, first2, last2[, bin\_pred]);  
bool **next\_permutation**(first, last[, comp]);  
bool **prev\_permutation**(first, last[, comp]);  
T total = **accumulate**(first, last, init\_total[, bin\_op]);  
T prod = **inner\_product**(first1, last1, first2, init[, bin\_op1, bin\_op2]);  
OutputIter result\_end = **partial\_sum**(first, last, result[, bin\_op]);  
OutputIter result\_end = **adjacent\_difference**(first, last, result[, bin\_op]);  
RandomAccessIter result\_end = **random\_sample**(first, last, out\_first, out\_last[, rand])  
RandomAccessIter result\_end = **random\_sample\_n**(first, last, out\_first, count[, rand])  
bool is\_inorder = **is\_sorted**(first, last[, comp]);