Обобщенные алгоритмы

Во всех заданиях данной группы заготовки решения уже содержат операторы, обеспечивающие заполнение исходных контейнеров. Если в задании требуется преобразовать содержимое исходного контейнера, то заготовка содержит отладочную печать преобразованного контейнера. Кроме того, если в задании требуется вывести преобразованный контейнер, заготовка содержит закомментированный оператор вывода требуемого контейнера (для вывода используется алгоритм copy). Например, если в задании требуется преобразовать вектор целых чисел, то заготовка решения имеет следующий вид:

```
typedef ptin_iterator<int> ptin;
typedef ptout_iterator<int> ptout;
vector<int> V(ptin(0), ptin());

ShowLine(V.begin(), V.end(), "V: ");
//copy(V.begin(), V.end(), ptout());
```

Решение следует ввести перед оператором отладочной печати ShowLine, после чего раскомментировать вызов copy для вывода результатов и их автоматической проверки.

Если тип элементов контейнера не указан, то предполагается, что элементами являются целые числа.

Алгоритмы поиска

STL3Alg1. Дан вектор *V*. Удалить второй из элементов вектора, равных нулю. Если нулевых элементов меньше двух, то вектор не изменять. Использовать два вызова алгоритма find и функцию-член erase.

STL3Alg2. Дан дек *D*. Удалить последний нулевой элемент дека. Если нулевых элементов нет, то дек не изменять. Использовать алгоритм find с обратными итераторами и функцию-член erase.

Указание. Алгоритм find может возвращать обратный итератор, однако функция-член erase не позволяет его использовать для удаления элемента. Необходимо перейти от обратного итератора `r` к связанному с ним обычному итератору, используя функцию-член обратного итератора `r.base()`. При этом следует учитывать, что функция `r.base()` возвращает итератор, связанный с элементом, следующим за тем, с которым связан обратный итератор `r`. Поэтому в функции-члене erase следует указать одно из следующих выражений (предполагается, что `r` — это обратный итератор, который вернул алгоритм find, и этот итератор отличен от rend): `--r.base()` или `(++r).base()`.

STL3Alg3. Дан список *L*. Удалить первый и последний нулевой элемент списка. Если нулевых элементов нет, то список не изменять, если нулевой элемент всего один, то удалить только его. Использовать два вызова алгоритма find и два вызова функции-члена erase.

Указание. Имеющуюся в классе list функцию-член remove в данном случае использовать нельзя, так как она удаляет все элементы с определенным значением. См. также указание к задаче STL3Alg2.

STL3Alg4. Дан список *L*. Удалить все элементы списка, расположенные между первым и вторым отрицательным элементом (не включая сами отрицательные элементы). Если список не содержит отрицательных элементов, то не изменять его, если отрицательный элемент всего один, то удалить все элементы, следующие за этим отрицательным элементом. Использовать два вызова алгоритма find_if и один вызов функции-члена erase.

STL3Alg5. Дан список *L*, содержащий как отрицательные, так и положительные элементы. Вставить нулевой элемент после первого отрицательного элемента и перед последним положительным элементом. Использовать два вызова алгоритма find_if и два вызова функции-члена insert.

Указание. По поводу использования алгоритма find_if с обратным итератором см. указание к задаче STL3Alg2.

STL3Alg6. Даны вектор *V* и список *L*; вектор *V* имеет четное количество элементов. Продублировать последний элемент списка, совпадающий с каким-либо элементом из первой половины исходного вектора. Если список не содержит требуемых элементов, то не изменять его. Использовать алгоритм find_first_of и функцию-член insert для списка.

STL3Alg7. Дан вектор *V* с четным количеством элементов. Добавить нулевой элемент перед последним элементом в первой половине вектора, совпадающим с каким-либо элементом из второй половины этого же вектора. Если вектор не содержит требуемых элементов, то не изменять его. Использовать алгоритм find_first_of и функцию-член insert.

STL3Alg8. Дано целое число *K* (> 0) и вектор *V*, содержащий только нули и единицы. Удалить в векторе *V* последний набор из *K* подряд расположенных нулей (если в этом наборе имеется больше *K* нулей, то требуется удалить только последние *K* из них). Если вектор не содержит требуемого набора нулей, то не изменять его. Использовать алгоритм search_n и функцию-член erase, а также обратные итераторы.

STL3Alg9. Дано целое число *K* (> 0) и вектор *V*. Продублировать в векторе *V* первый набор из *K* подряд расположенных положительных чисел, вставив после этого набора его копию (если в наборе имеется больше *K* положительных чисел, то лишние числа во внимание не принимаются, и дублируется только начальная часть набора, содержащая *K* чисел). Если вектор не содержит требуемого набора чисел, то не изменять его. Использовать алгоритм search_n с параметром — функциональным объектом.

STL3Alg10. Даны список *L* и дек *D*; дек *D* содержит четное количество элементов. Продублировать в деке *D* первый набор элементов, расположенный в его второй половине и совпадающий с элементами списка *L*, взятыми в обратном порядке. Например, для списка 1, 2 и дека 2, 1, 1, 2, 3, 2, 1, 1, 2, 1 дек должен быть преобразован следующим образом: 2, 1, 1, 2, 3, 2, 1, (2, 1,) 1, 2, 1 (вставленные элементы заключены в скобки). Если дек не содержит требуемого набора чисел, то не изменять его. Использовать алгоритм search и функцию-член insert.

STL3Alg11. Даны вектор *V* и дек *D*; дек *D* содержит четное количество элементов. Удалить в векторе *V* последний набор элементов, в котором четные и нечетные числа располагаются в том же порядке, что и в первой половине дека *D*. Например, для дека 1, 2, 3, 4, 5, 6 и вектора 11, 14, (15, 16, 17,) 17, 18, 10 в векторе должны быть удалены элементы, заключенные в скобки. Если вектор не содержит требуемого набора чисел, то не изменять его. Использовать алгоритм search с параметром — функциональным объектом и функцию-член erase.

STL3Alg12. Решить задачу STL3Alg11, используя алгоритм find_end с параметром — функциональным объектом и функцию-член erase.

STL3Alg13. Дан список *L*. Удвоить значения в последней паре соседних совпадающих элементов. Например, список 1, 2, 2, 3, 3, 1, 2, 2, 2, 4 должен быть преобразован следующим образом: 1, 2, 2, 3, 3, 1, 2, 4, 4, 4. Если список не содержит соседних совпадающих элементов, то не изменять его. Использовать алгоритм adjacent_find и обратные итераторы.

STL3Alg14. Дан вектор *V*. Обнулить первую пару соседних элементов, имеющих одинаковую четность. Например, список 1, 2, 3, 4, 6, 8, 3, 1 должен быть преобразован следующим образом: 1, 2, 3, 0, 0, 8, 3, 1. Если список не содержит сосед-

них элементов с одинаковой четностью, то не изменять его. Использовать алгоритм adjacent_find с параметром — функциональным объектом.

STL3Alg15. Дан дек *D*. Удвоить значения всех пар соседних элементов, исходные значения которых отличаются только знаком (преобразованные элементы не должны повторно анализироваться при последующих вызовах алгоритма). Например, дек 1, −1, 2, −3, 3, 3, −3, 6 должен быть преобразован следующим образом: 2, −2, 2, −6, 6, 6, −6, 6. Использовать алгоритм adjacent_find с параметром — функциональным объектом в цикле по итератору дека.

Базовые модифицирующие алгоритмы. Итераторы вставки

STL3Alg16. Даны числа *A* и *B* и векторы V_1 и V_2, каждый из которых содержит не менее 10 элементов. Заполнить первые 5 элементов каждого вектора значениями *A*, а последние 5 — значениями *B*. При преобразовании вектора V_1 использовать два вызова алгоритма fill, при преобразовании вектора V_2 использовать два вызова алгоритма fill_n.

STL3Alg17. Даны числа *A* и *B* и векторы V_1 и V_2. Добавить в начало каждого вектора 5 элементов со значениями *A*, а в конец — 5 элементов со значениями *B*. При преобразовании вектора V_1 использовать два вызова алгоритма fill_n с функциями inserter и back_inserter (данные функции возвращают итераторы вставки), при преобразовании вектора V_2 использовать два вызова функции-члена insert.

Замечание. Второй способ является более эффективным.

STL3Alg18. Дано число N (> 0) и дек *D*, содержащий не менее $2N$ элементов. Заполнить первые N элементов дека последовательностью чисел 1, 2, …, N, а последние N элементов дека — последовательностью N, $N-1$, …, 2, 1. Использовать два вызова алгоритма generate с одинаковым параметром — функциональным объектом, а также обратные итераторы.

STL3Alg19. Дано число N (> 0) и дек *D*. Добавить в начало дека последовательность чисел 1, 2, …, N, а в его конец — последовательность N, $N-1$, …, 2, 1. Использовать два вызова алгоритма generate_n с одинаковым параметром — функциональным объектом, а также итераторы вставки, возвращаемые функциями front_inserter и back_inserter.

STL3Alg20. Даны списки L_1 и L_2, каждый из которых содержит четное количество элементов. Поменять местами первую и вторую половину каждого списка (например, список 1, 2, 3, 4 должен быть преобразован следующим образом: 3, 4, 1, 2). Для первого списка использовать алгоритм swap_ranges, для второго — алгоритм rotate. Использовать также функцию advance.

STL3Alg21. Дано число K ($0 < K < 10$) и списки L_1 и L_2, каждый из которых содержит не менее 10 элементов. Выполнить для списка L_1 циклический сдвиг элементов вправо на K позиций, а для списка L_2 — циклический сдвиг влево на K позиций. Использовать алгоритм rotate и функцию advance.

STL3Alg22. Дано число K ($0 < K < 5$) и список *L*, содержащий не менее 10 элементов. Набор из первых 5 элементов списка скопировать в конец списка, выполнив для этой копии циклический сдвиг на K позиций вправо, а набор из последних 5 элементов исходного списка скопировать в начало списка, выполнив для этой копии циклический сдвиг на K позиций влево. Использовать два вызова алгоритма rotate_copy и итераторы вставки, а также функцию advance.

STL3Alg23. Даны списки L_1 и L_2, у каждого из которых количество элементов делится на 4. В первом списке инвертировать (расположить в обратном порядке) первую половину элементов, во втором списке — вторую половину. Для первого списка использовать алгоритм swap_ranges и обратные итераторы, для второго — алгоритм reverse. Использовать также функцию advance.

STL3Alg24. Даны списки L_1 и L_2, каждый из которых содержит четное количество элементов. В каждом списке продублировать первую половину, добавив ее элементы в конец списка в обратном порядке. Для первого списка использовать алгоритм reverse_copy и итератор вставки, для второго — функцию-член insert и обратные итераторы. Использовать также функцию advance.

Замечание. Второй способ является более эффективным.

STL3Alg25. Дан вектор *V* с четным количеством элементов. В первой половине исходного вектора заменить все отрицательные числа на −1, а во второй — все положительные числа на 1. Использовать два вызова алгоритма replace_if с различными параметрами — функциональными объектами.

STL3Alg26. Дан список *L* с четным количеством элементов. Скопировать в конец списка все элементы, расположенные в его первой половине, заменив при этом отрицательные элементы на нули и расположив скопированные элементы в обратном порядке. Использовать алгоритм replace_copy_if, итератор вставки и обратные итераторы, а также функцию advance.

STL3Alg27. Дан дек *D* с четным количеством элементов. Решить для него задачу STL3Alg26. Поскольку операция вставки делает все итераторы дека недействительными, для решения задачи использовать вспомогательный дек D_0. Инициализировать дек D_0 первой половиной элементов дека *D*, после чего применить алгоритм replace_copy_if, используя дек D_0 как источник, а дек *D* как приемник данных.

STL3Alg28. Дан список *L* с четным количеством элементов. Скопировать в начало списка все положительные элементы, расположенные в его второй половине, сохранив для них исходный порядок следования. Использовать алгоритм remove_copy_if и итератор вставки, а также функцию advance.

STL3Alg29. Дан вектор *V* с четным количеством элементов. Решить для него задачу STL3Alg28. Поскольку операция вставки в начало вектора делает все его итераторы недействительными, для решения задачи использовать вспомогательный вектор V_0. Инициализировать вектор V_0 второй половиной элементов вектора *V*, после чего применить алгоритм remove_copy_if, используя вектор V_0 как источник, а вектор *V* как приемник данных.

STL3Alg30. Дан вектор *V* с четным количеством элементов. Удалить все нулевые элементы, расположенные во второй половине исходного вектора. Использовать алгоритм remove и функцию-член erase.

Указание. Алгоритм remove и другие алгоритмы, связанные с удалением элементов, не удаляют требуемые элементы, а лишь перемещают их в конец указанного диапазона и возвращают итератор на их начало. Для фактического удаления элементов после выполнения алгоритма необходимо вызвать функцию-член erase.

STL3Alg31. Дан вектор *V* с четным количеством элементов. Удалить все четные элементы, расположенные в первой половине исходного вектора. Использовать алгоритм remove_if и функцию-член erase.

Указание. См. указание к задаче STL3Alg30.

STL3Alg32. Дан вектор *V*. В каждой группе подряд расположенных элементов с одинаковой четностью удалить все элементы, кроме начального. Использовать алгоритм unique с параметром — функциональным объектом и функцию-член erase.

Указание. См. указание к задаче STL3Alg30.

STL3Alg33. Дан дек *D*. Из каждой группы подряд расположенных положительных, отрицательных или нулевых элементов выбрать последний и скопировать выбранные элементы в том же порядке в начало дека. Например, дек 1, 2, −3, 4, 0, 0, −5, −6, 7, 8, 9 должен быть преобразован следующим образом: (2, −3, 4, 0, −6, 9,) 1, 2, −3, 4, 0, 0, −5, −6, 7, 8, 9 (добавленные элементы заключены в скобки). Использовать вспомогательный дек D_0 (инициализировав его с помощью дека *D*), алгоритм unique_copy с функциональным объектом и обратными итераторами, а также подходящий итератор вставки (в алгоритме дек D_0 должен быть источником, а дек *D* — приемником данных).

STL3Alg34. Дан дек *D* с четным количеством элементов. Заменить в нем вторую половину элементов на удвоенные значения элементов первой половины, расположив эти удвоенные значения в обратном порядке. Использовать алгоритм transform с обратным итератором.

STL3Alg35. Дан список *L* с четным количеством элементов *N*. Добавить в начало списка *N*/2 новых элементов со следующими значениями: $A_1 + A_N$, $A_2 + A_{N-1}$, …, $A_{N/2} + A_{N/2+1}$, где A_1, A_2, …, A_N обозначают исходные элементы списка. Использовать алгоритм transform с обратным итератором и итератором вставки.

Сортировка и слияние

STL3Alg36. Дан вектор *V* с нечетным количеством элементов N (≥ 3). Определить значения трех средних элементов вектора после того, как вектор будет отсортирован (по возрастанию), и вывести их в порядке возрастания. Использовать алгоритмы nth_element (для нахождения центрального элемента), max_element (для нахождения элемента, предшествующего центральному) и min_element (для нахождения элемента, следующего за центральным).

STL3Alg37. Дан вектор *V*, содержащий не менее 3 элементов. Определить значения трех начальных элементов вектора после того, как вектор будет отсортирован (по возрастанию), и вывести их в порядке возрастания. Использовать один вызов алгоритма partial_sort и алгоритм copy для вывода требуемых элементов.

STL3Alg38. Дан вектор *V*, содержащий не менее 3 элементов. Определить значения трех конечных элементов вектора после того, как вектор будет отсортирован (по возрастанию) и вывести их в порядке убывания. Использовать один вызов алгоритма partial_sort с параметром — функциональным объектом greater и алгоритм copy для вывода требуемых элементов.

STL3Alg39. Дан вектор *V* с четным количеством элементов. Добавить в его конец первую половину элементов, которые содержал бы отсортированный по возрастанию вариант исходного вектора. Использовать функцию-член insert для увеличения количества элементов вектора на требуемую величину и алгоритм partial_sort_copy.

STL3Alg40. Дан список *L*. Определить количество четных и нечетных чисел в исходном списке (вначале вывести количество четных, затем количество нечетных чисел). Использовать алгоритм partition и два вызова функции distance для итераторов.

STL3Alg41. Дан список *L*. Перегруппировать элементы списка, расположив в нем вначале положительные, а затем неположительные элементы (порядок расположения элементов в каждой группе должен совпадать с исходным). Использовать алгоритм stable_partition.

STL3Alg42. Дан список *L*. Перегруппировать элементы списка, расположив в нам вначале отрицательные, затем нулевые, а затем положительные элементы (порядок расположения элементов в каждой группе должен совпадать с исходным). Использовать два вызова алгоритма stable_partition.

STL3Alg43. Дан список *L* с четным количеством элементов. Перегруппировать элементы списка, расположив в нем вначале все четные элементы из первой половины исходного списка, затем все нечетные элементы, а затем — все четные элементы из второй половины списка (порядок расположения элементов в каждой группе должен совпадать с исходным). Использовать два вызова алгоритма stable_partition.

STL3Alg44. Дан дек *D*. Решить для него задачу STL3Alg42, используя один вызов алгоритма stable_sort с параметром — функциональным объектом.

STL3Alg45. Дан дек *D*, элементами которого являются английские слова. Отсортировать его элементы по возрастанию их длин, а элементы одинаковой длины — в алфавитном порядке. Использовать алгоритм sort для сортировки по алфавиту и затем алгоритм stable_sort с параметром — функциональным объектом для сортировки по длине. Выводить новое содержимое дека после применения каждого алгоритма.

STL3Alg46. Дан дек *D*, элементами которого являются английские слова. Отсортировать его элементы по убыванию их длин, а элементы одинаковой длины — в алфавитном порядке. Использовать единственный вызов алгоритма sort с параметром — функциональным объектом, включающим как сравнение строк, так и сравнение их длин.

STL3Alg47. Дан вектор *V* с четным количеством элементов. Известно, что первая половина вектора уже отсортирована по возрастанию. Отсортировать все элементы вектора, по возрастанию, выполнив вначале сортировку его второй половины алгоритмом sort, а затем слияние обеих половин алгоритмом inplace_merge. Выводить новое содержимое вектора *V* после применения каждого алгоритма.

STL3Alg48. Дан вектор *V*, количество элементов которого делится на 3. Известно, что каждая треть вектора уже упорядочена по возрастанию. Решить для исходного вектора задачу STL3Alg42, используя два вызова алгоритма inplace_merge с одним и тем же параметром — функциональным объектом.

Перестановки и работа с кучей

STL3Alg49. Дан вектор *V*, элементы которого не упорядочены по убыванию. Получить все перестановки исходных элементов, которые больше исходного вектора в лексикографическом порядке и вывести полученные перестановки по возрастанию их лексикографического порядка (при этом последняя выведенная перестановка должна содержать элементы, упорядоченные по убыванию). Использовать в цикле алгоритм next_permutation.

STL3Alg50. Дан вектор *V*, элементы которого не упорядочены по убыванию. Решить задачу STL3Alg49, используя в цикле алгоритм prev_permutation с параметром — функциональным объектом greater.

STL3Alg51. Дан вектор *V*, содержащий не менее 3 элементов. Решить для него задачу STL3Alg38, используя один вызов алгоритма make_heap, цикл из трех итераций, в котором вызывается алгоритм pop_heap, и алгоритм copy (с обратными итераторами), вызываемый после цикла для вывода требуемых элементов.

STL3Alg52. Дан вектор *V*, содержащий не менее 3 элементов. Решить для него задачу STL3Alg37, используя один вызов алгоритма make_heap, цикл из трех итераций, в котором вызывается алгоритм pop_heap, и алгоритм copy (с обратными итераторами), вызываемый после цикла для вывода требуемых элементов.

Указание. Используйте варианты алгоритмов с параметром — функциональным объектом.

STL3Alg53. Дан дек D_1, содержащий не менее 3 элементов, и дек D_2, содержащий ровно 3 элемента. Определить три максимальных элемента среди всех элементов набора, полученного из дека D_1 в результате удаления трех его максимальных элементов и добавления к нему всех элементов из дека D_2. Вывести требуемые элементы в порядке убывания. Использовать алгоритмы make_heap, pop_heap и push_heap, а также алгоритм copy (с обратными итераторами) для вывода требуемых элементов. Размеры исходных деков не изменять, алгоритм make_heap использовать один раз.

Указание. Алгоритм make_heap используется для преобразования элементов первого дека в кучу. Затем выполняется цикл с тремя вызовами алгоритма pop_heap для удаления из кучи трех максимальных элементов, цикл с тремя вызовами алгоритма push_heap для добавления в кучу элементов из второго дека и еще один цикл с тремя вызовами алгоритма pop_heap для определения требуемых элементов. Алгоритм copy вызывается после цикла.

STL3Alg54. Дан дек D_1, содержащий не менее 3 элементов, и дек D_2, содержащий ровно 3 элемента. Определить три минимальных элемента среди всех элементов набора, полученного из дека D_1 в результате удаления трех его минимальных элементов и добавления к нему всех элементов из дека D_2. Вывести требуемые элементы в порядке возрастания. Использовать алгоритмы make_heap, pop_heap и push_heap, а также алгоритм copy (с обратными итераторами) для вывода требуемых элементов. Размеры исходных деков не изменять, алгоритм make_heap использовать один раз.

Указание. См. указания к задачам STL3Alg52 и STL3Alg53.

Численные алгоритмы

STL3Alg55. Дан вектор V с четным количеством элементов. Найти сумму элементов из первой и из второй половины вектора. Использовать два вызова алгоритма accumulate.

STL3Alg56. Дан вектор V. Найти сумму отрицательных и сумму положительных элементов вектора. Использовать два вызова алгоритма accumulate с параметрами — функциональными объектами.

STL3Alg57. Дан вектор V, элементами которого являются английские слова. Получить строку, содержащую начальные символы всех элементов вектора, расположенные в обратном порядке. Использовать алгоритм accumulate с параметром — функциональным объектом.

STL3Alg58. Дан вектор V, элементами которого являются английские слова. Получить строку, содержащую конечные символы всех элементов вектора, расположенные в исходном порядке. Использовать алгоритм accumulate с параметром — функциональным объектом.

STL3Alg59. Дан вектор V, элементами которого являются английские слова. Получить строку, являющуюся суммой строк, описанных в задачах STL3Alg57 и STL3Alg58 (в указанном порядке). Использовать один вызов алгоритма accumulate с параметром — функциональным объектом.

STL3Alg60. Дан список L. Получить вектор V вещественных чисел, содержащий значения среднего арифметического для всех пар соседних элементов исходного списка (количество элементов вектора V должно быть на 1 меньше количества элементов списка L). Например, для исходного списка 1, 3, 4, 6 полученный вектор должен содержать значения 2.0, 3.5, 5.0. Использовать алгоритм adjacent_difference с итератором вставки и функциональным объектом, а также функцию-член erase для вектора V.

STL3Alg61. Дан список L, элементами которого являются английские слова. Получить дек D со строковыми элементами, каждый из которых строится по паре соседних элементов исходного списка L следующим образом: последняя буква правого элемента пары приписывается справа к первой букве левого элемента пары. Количество элементов дека D должно быть на 1 меньше количества элементов списка L. Например, для исходного списка ABC, DEF, KLM, XYZ полученный дек должен содержать строки AF, DM, KZ. Использовать алгоритм adjacent_difference с итератором вставки и функциональным объектом, а также функцию-член erase для дека D.

STL3Alg62. Даны векторы V_1 и V_2 одинакового размера N, каждый из которых содержит координаты точки в N-мерном пространстве с евклидовой метрикой $r(x,y) = (\Sigma(x_i - y_i)^2)^{1/2}$ (сумма берется по всем i от 0 до $N-1$). Найти значение квадрата метрики для двух данных точек. Использовать алгоритм inner_product с двумя функциональными объектами (в качестве первого из них можно указать стандартный объект plus).

STL3Alg63. Даны векторы V_1 и V_2 одинакового размера N, каждый из которых содержит координаты точки в N-мерном пространстве с метрикой $r(x,y) = \max\{|x_i - y_i|\}$ (максимум берется по всем i от 0 до $N-1$). Найти значение метрики для двух данных точек. Использовать алгоритм inner_product с двумя функциональными объектами.

STL3Alg64. Дано целое число N ($1 \le N \le 16$). Получить вектор V вещественных чисел, равных факториалам всех чисел от 1 до N (вещественные числа используются для того, чтобы избежать целочисленного переполнения). Для этого последовательно вызвать конструктор вектора V, заполняющий его единицами, алгоритм partial_sum (с параметром — функциональным объектом) для получения в векторе всех чисел от 1 до N и алгоритм partial_sum (с параметром — функциональным объектом, в качестве которого достаточно использовать стандартный объект multiplies) для получения в векторе требуемых факториалов.