Тема
Стандартна бібліотека шаблонів. Контейнери та алгоритми.

Мета
Навчитись використовувати контейнери стандартної бібліотеки шаблонів та вбудовані алгоритми.

Теоретичні відомості
Контейнери
Контейнери – це об’єкти, які зберігають інші об’єкти. Вони керують алокацією та деалокацією цих об’єктів через конструктори, деструктори, операції вставки та видалення.

Бібліотека контейнерів містить узагальнену колекцію шаблонів класів та алгоритмів, що дозволяють програмістам легко реалізувати поширені структури даних, такі як черги, списки чи стеки. Починаючи із C++11 існує три категорії контейнерів:

1. послідовні контейнери;
2. асоціативні контейнери;
3. невпорядковані асоціативні контейнери (починаючи із C++11).

Кожен контейнер яких спроектовано для підтримки певних операцій.

Контейнер керує місцем, необхідним для зберігання його елементів, надає функції-члени для доступу до них або безпосередньо, або через ітератори (об’єкти із властивостями схожими до вказівників).

Більшість контейнерів мають хоча б кілька схожих функцій та функціонал. Вибір кращого контейнера залежить від конкретної задачі та від ефективності реалізації функціоналу потрібного для розв’язування цієї задачі.

Послідовні контейнери
Послідовні контейнери реалізують структури даних, що надають можливість послідовного доступу до однотипних елементів.

Нижче наведені шаблони класів послідовних контейнерів

Таблиця 1. Послідовні контейнери

Назва шаблону	**Опис**
std::array (з C++11)	Статичний безперервний масив
std::vector	Динамічний безперервний масив
std::deque	Двобічна черга
std::forward_list (з C++11)	Однобічно зв’язаний список
std::list	Двобічно зв’язаний список

Шаблон std::array надає інтерфейс для роботи із масивами фіксованого розміру, що сумісний з іншими контейнерами та позбавлений деяких недоліків звичайних масивів (наприклад, неявне приведення до вказівника на елемент, втрата інформації про розмір, тощо).

Шаблон std::vector надає інтерфейс для роботи із динамічними масивами. Він дбає про перевиділення пам’яті, коли це необхідно. Елементи в ньому також розташовані послідовно. Це, напевно, найбільш вживаний контейнер із бібліотеки.

Асоціативні контейнери

Асоціативні контейнери реалізують впорядковані структури даних, у яких швидкий пошук за ключем (алгоритмічна складність - O(log n)).

Кожен контейнер параметризується типом ключа та типом об'єкта для порівняння.

Нижче наведені шаблони класів асоціативних контейнерів.

Таблиця 2. Асоціативні контейнери

Назва	**Опис**
std::set	Колекція неповторюваних впорядкованих ключів
std::map	Колекція пар ключ-значення, ключі – неповторювані та впорядковані
std::multiset	Колекція впорядкованих ключів
std::multimap	Колекція пар ключ-значення, ключі – впорядковані

Невпорядковані асоціативні контейнери (починаючи із C++11)

Невпорядковані асоціативні контейнери реалізують невпорядковані (хешовані) структури даних, у яких є швидкий пошук за ключем (O(1) – у середньому, O(n) – в найгіршому випадку).

Нижче наведені шаблони класів невпорядкованих асоціативних контейнерів.

Таблиця 3. Невпорядковані асоціативні контейнери

Назва шаблону	**Опис**
std::unordered_set	Колекція неповторюваних хешованих ключів
std::unordered_map	Колекція пар ключ-значення, ключі – неповторювані та хешовані
std::unordered_multiset	Колекція хешованих ключів
std::unordered_multimap	Колекція пар ключ-значення, ключі – хешованих

Адаптери для контейнерів

Адаптери надають інший інтерфейс для послідовних контейнерів.

Нижче наведені шаблони класів адаптерів.

Таблиця 4. Адаптери

Назва шаблону	**Опис**
std::stack	Адаптує контейнер до стеку (LIFO структури даних)
std::queue	Адаптує контейнер до черги (FIFO структури даних)
std::priority_queue	Адаптує контейнер до черги із пріоритетами

Рядки

Бібліотека для роботи із рядками символів підтримує такі загальні типи рядків:

- std::basic_string – шаблон класу спроектований для маніпуляції із рядками будь-якого символьного типу (char, wchar_t, char16_t, char32_t і т.п.)
- std::basic_string_view (C++17) – легке відображення частини рядка. Лише для читання та без власності
- рядки, що закінчується нулем – масиви символів, що закінчуються спеціальним нульовим значенням. Так звані рядки мови C.

Бібліотека містить декілька спеціалізацій шаблонів std::basic_string та std::basic_string_view для поширених типів:

Таблиця 5. Спеціалізації шаблонів std::basic_string та std::basic_string_view

Тип	**Визначення**
std::string	std::basic_string<char>
std::wstring	std::basic_string<wchar_t>
std::u16string (C++11)	std::basic_string<char16_t>
std::u32string (C++11)	std::basic_string<char32_t>
std::string_view (C++17)	std::basic_string_view<char>
std::wstring_view (C++17)	std::basic_string_view<wchar_t>
std::u16string_view (C++17)	std::basic_string_view<char16_t>
std::u32string_view (C++17)	std::basic_string_view<char32_t>

Ітератори

Ітератори є узагальненням вказівників, вони дозволяють C++ програмі працювати із різними структурами даних однаково. Бібліотека ітераторів надає визначення ітераторів, адаптери та допоміжні функції.

Існують різні категорії ітераторів, залежно від того, які операції вони підтримують.

Таблиця 6. Категорії ітераторів та вимоги до них

Категорія	**Вимоги на операції та зберігання даних**						
	запис	читання	інкремент		декремент	довільний доступ	послідовне зберігання
			разово	багаторазово			
Output	так		так				
Input		так	так				
Forward		так	так	так			
Bidirectional		так	так	так	так		
Random Access		так	так	так	так	так	
Contiguous		так	так	так	так	так	так

Різні контейнери можуть надавати ітератори різних категорій, що вказано у документації. В документації до кожного алгоритму вказано категорії ітераторів, з якими він працює.

Бібліотека алгоритмів

Бібліотека алгоритмів містить функції для різних задач (таких як пошук, сортування, підрахунок, маніпулювання), що працюють із діапазонами елементів.

Діапазони задаються парою ітераторів [first, last), де last – відповідає елементу, що йде після останнього.

Багато алгоритмів дозволяють передати функційні об’єкти (предикати), які дозволяють налаштувати алгоритм під конкретну задачу. Наприклад, можна шукати елемент не просто по значенню, а за якоюсь складною умовою.

Використання алгоритмів із бібліотеки має такі переваги:

1. Алгоритми вже готові та відлагоджені. Це зменшує час написання програми та зменшує кількість потенційних помилок.
2. Алгоритми стандартної бібліотеки мають схожий інтерфейс та працюють із різними структурами даних. Це сприяє перевикористанню та покращує розуміння коду.
3. Алгоритми стандартної бібліотеки розроблені для забезпечення максимальної ефективності операцій. Багато з них мають різні реалізації для різних категорій ітераторів.

Операції, що не змінюють послідовності

Таблиця 7. Операції, що не змінюють послідовності

Назва шаблону функції	**Опис**
std::all_of (C++11) std::any_of (C++11) std::none_of (C++11)	Перевіряють, чи предикат true для всіх, якогось чи жодного елементу із заданого діапазону
std::for_each std::for_each_n (C++17)	Викликає функцію для кожного елементу із діапазону Застосовує функційний об'єкт до перших n елементів послідовності
std::count std::count_if	Повертають кількість елементів, що задовольняють певним критеріям
std::mismatch	Повертають першу позицію, де два діапазони різняться
std::find std::find_if std::find_if_not (C++11)	Знаходить перший елемент, що задовольняє певним критеріям
std::find_end	Знаходить останнє входження послідовності у заданому діапазоні
std::find_first_of	Шукає будь-який перший із елементів у заданому діапазоні
std::adjacent_find	Шукає два послідовних елемента, які однакові, або для яких однакове значення предиката
std::search	Шукає послідовність елементів у заданому діапазоні
std::search_n	Шукає послідовність із n елементів певного значення у заданому діапазоні

Операції, що змінюють послідовності

Таблиця 8. Операції, що змінюють послідовності

Назва шаблону функції	**Опис**
std::copy std::copy_if (C++11)	Копіює діапазон елементів у нове розташування
std::copy_n (C++11)	Копіює число елементів у нове розташування
std::copy_backward	Копіює діапазон елементів у зворотному порядку
std::move (C++11)	Переміщає діапазон елементів у нове розташування
std::move_backward (C++11)	Переміщає діапазон елементів у нове розташування у зворотному порядку
std::fill	Присвоює передане значення кожному елементу із діапазону
std::fill_n	Присвоює передане значення N елементам із діапазону
std::transform	Застосовує функцію до діапазону елементів та зберігає результат у іншому діапазоні
std::generate	Присвоює результат послідовного виклику функції до діапазону елементів
std::generate_n	Присвоює результат послідовного виклику функції до N елементів із діапазону
std::remove std::remove_if	Видаляє елементи, що задовольняють певним умовам
std::remove_copy std::remove_copy_if	Копіює діапазон елементів, окрім тих, що задовольняють певним умовам
std::replace std::replace_if	Замінює всі значення, що задовольняють певним умовам, на передане значення
std::replace_copy std::replace_copy_if	Копіює діапазон, замінюючи значення елементів, що задовольняють певним умовам, на передане значення
std::swap	Обмінює значення двох об'єктів
std::swap_ranges	Обмінює два діапазони елементів
std::iter_swap	Обмінює значення двох об'єктів на які вказують передані ітератори
std::reverse	Інвертує порядок елементів у діапазоні

std::reverse_copy	Створює копію діапазону в оберненому порядку
std::rotate	Обертає порядок елементів у діапазоні
std::rotate_copy	Копіює діапазон обертаючи порядок елементів
std::shuffle	Довільним чином змінює порядок елементів у діапазоні
std::sample (C++17)	Вибирає n випадкових елементів із послідовності
std::unique	Видаляє послідовні дублікати елементів у діапазоні
std::unique_copy	Створює копію деяких елементів із діапазону, що не містить послідовні дублікати

Операції розбиття

Таблиця 9. Операції розбиття

Назва шаблону функції	**Опис**
std::is_partitioned (C++11)	Визначає чи діапазон є розбитим по заданому предикату – усі елементи діапазону, що задовольняють предикату, розміщені перед усіма іншими
std::partition	Розбиває діапазон елементів на дві групи
std::partition_copy (C++11)	Копіює діапазон розбиваючи його на дві групи
std::stable_partition	Розбиває діапазон елементів на дві групи зберігаючи відносний порядок елементів
std::partition_point (C++11)	Знаходить точку поділу розбитого діапазону по заданому предикату

Операції сортування

Таблиця 10. Операції сортування

Назва шаблону функції	**Опис**
std::is_sorted (C++11)	Перевіряє, чи діапазон є відсортовано за зростанням
std::is_sorted_until (C++11)	Повертає найбільший відсортовану піддіапазон
std::sort	Сортує послідовність за зростанням
std::partial_sort	Сортує перші N елементів діапазону
std::partial_sort_copy	Сортує перші N елементів діапазону та результат копіює
std::stable_sort	Сортує діапазон елементів зберігаючи порядок між однаковими елементами
std::nth_element	Частково сортує діапазон розбиваючи його за заданим елементом

Операції бінарного пошуку (на відсортованих діапазонах)

Таблиця 11. Операції бінарного пошуку

Назва шаблону функції	**Опис**
std::lower_bound	Повертає ітератор на перший елемент, що не менший від заданого
std::upper_bound	Повертає ітератор на перший елемент, більший від заданого
std::binary_search	Перевіряє чи елемент існує у частково-впорядкованому діапазоні
std::equal_range	Повертає діапазон елементів, у яких співпадає ключ

Інші операції на відсортованих діапазонах

Таблиця 12. Операції бінарного пошуку

Назва шаблону функції	**Опис**
std::lower_bound	Повертає ітератор на перший елемент, що не менший від заданого
std::upper_bound	Повертає ітератор на перший елемент, більший від заданого
std::binary_search	Перевіряє чи елемент існує у частково-впорядкованому діапазоні
std::equal_range	Повертає діапазон елементів, у яких співпадає ключ

Операції з множинами (на відсортованих діапазонах)

Таблиця 13. Операції із множинами

Назва шаблону функції	**Опис**
std::includes	Повертає true, якщо одна послідовність є частиною іншої
std::set_difference	Обчислює різницю двох множин
std::set_intersection	Обчислює перетин двох множин
std::set_symmetric_difference	Обчислює симетричну різницю двох множин
std::set_union	Обчислює об'єднання двох множин

Операції над купами (структура даних)

Таблиця 13. Операції над купами

Назва шаблону функції	**Опис**
std::is_heap (C++11)	Перевіряє, чи заданий діапазон є купою (найбільший елемент – корінь)
std::is_heap_until (C++11)	Знаходить найбільший піддіапазон у діапазоні, що є купою
std::make_heap	Створює купу із діапазону елементів (найбільший елемент – корінь)
std::push_heap	Додає елемент до купи (найбільший елемент – корінь)
std::pop_heap	Видаляє максимальний елемент (корінь) з купи
std::sort_heap	Перетворює купу у діапазон відсортований за зростанням

Операції пов'язані із мінімальним/максимальним елементом

Таблиця 14. Операції пов'язані із мінімальним/максимальним елементом

Назва шаблону функції	**Опис**
std::max	Повертає максимальний елемент із двох заданих, або із списку ініціалізації
std::max_element	Повертає максимальний елемент із заданого діапазону
std::min	Повертає мінімальний елемент із двох заданих, або із списку ініціалізації
std::min_element	Повертає мінімальний елемент із заданого діапазону
std::minmax	Повертає пару із мінімальним та максимальним елементом із двох заданих, або із списку ініціалізації
std::minmax_element	Повертає пару із мінімальним та максимальним елементом із заданого діапазону
std::clamp	Повертає значення елемента, якщо він потрапляє у діапазон між двома іншими заданими елементами. Якщо значення елементу є за межами діапазону, то повертається найближче значення із діапазону

Операції порівняння

Таблиця 15. Операції порівняння

Назва шаблону функції	**Опис**
std::equal	Визначає, чи два діапазони містять однакові елементи
std::lexicographical_compare	Повертає true, якщо один діапазон елементів лексикографічно менший за інший

Алгоритм std::equal не слід використовувати для порівняння діапазонів, утворених ітераторами з std::unordered_set, std::unordered_multiset, std::unordered_map або std::unordered_multimap, оскільки порядок, у якому елементи зберігаються е цих контейнерах, може відрізнятися, навіть якщо два контейнери зберігають однакові елементи.

Операції перестановок

Таблиця 16. Операції перестановок

Назва шаблону функції	**Опис**
std::is_permutation	Перевіряє чи послідовність є перестановкою іншої послідовності
std::next_permutation	Ґенерує лексикографічно наступну перестановку діапазону елементів
std::prev_permutation	Ґенерує лексикографічно попередню перестановку діапазону елементів

Числові операції

Таблиця 17. Числові операції

Назва шаблону функції	**Опис**
std::iota	Заповнює діапазон послідовними інкрементами початкового значення
std::accumulate	Обчислює суму елементів
std::inner_product	Обчислює скалярний добуток
std::adjacent_difference	Обчислює різницю між сусідніми елементами
std::partial_sum	Обчислює часткові суми елементів
std::reduce (C++17)	Те саме, що accumulate, але в довільному порядку
std::exclusive_scan (C++17)	Схоже до partial_sum, і-та сума не включає і-ий елемент
std::inclusive_scan (C++17)	Схоже до partial_sum, і-та сума включає і-ий елемент
std::transform_reduce (C++17)	Застосовує transform, а потім reduce
std::ransform_exclusive_scan (C++17)	Застосовує transform, а потім exclusive_scan
std::transform_inclusive_scan (C++17)	Застосовує transform, а потім inclusive_scan

Завдання

1. Змінити реалізацію класу із лабораторної роботи №9 відповідно до варіанту таким чином:
    a. для збереження елементів використати **послідовний контейнер** чи клас-рядок який найбільше підходить для вирішення задачі. Обґрунтувати вибір;
    b. змінити реалізацію існуючих функцій та операторів, використовуючи алгоритми із бібліотеки;
    c. додатково реалізувати функції члени, використовуючи алгоритми із бібліотеки.
2. Продемонструвати роботу із **асоціативним** та **невпорядкованим асоціативним контейнером** згідно із варіантом:
    a. вставку значень із вже присутнім та відсутнім ключем;
    b. успішний та неуспішний пошук за ключем;
    c. успішне та неуспішне видалення за ключем;
    d. послідовну ітерацію (проходження) по контейнеру для зчитування значень;
3. Продемонструвати роботу із **адаптером** згідно із варіантом:
    a. вставку значень;
    b. вичитку значень;
4. Продемонструвати роботу шаблону класу, контейнерів та адаптерів таким чином (на вибір):
    a. За допомогою тестів Google Test.
    b. За допомогою інтерактивної програми із демонстрацією її роботи.
5. Оформити звіт до лабораторної роботи. В звіті, окрім стандартних пунктів, сформувати таблицю використаних в програмі методів для кожного залученого std-контейнера. Таблиця має містити назву контейнера, назву метода, пояснення дії метода.

Варіанти завдань до пункту 1

Варіант	Завдання
1	Шаблон класу CMatrix<typename T> – двовимірна матриця значень типу T. Кількість колонок та рядків задавати динамічно. Елементи зберігати у вибраному контейнері. Використати такі алгоритми: • std::min_element / std::max_element – для пошуку найменшого та найбільшого значення • std::transform – для додавання та віднімання матриць. • std::for_each – для оператора виведення у потік Реалізувати такі функції: • сортування елементів у рядках – використати алгоритм std::sort • сума всіх елементів – використати алгоритм std::accumulate
2	Шаблон класу CArray<typename T> – одновимірний масив значень типу T. Розмір масиву має змінюватись динамічно. Використати такі алгоритми: • std::min_element / std::max_element – для пошуку найменшого та найбільшого значення • std::accumulate – для середнього арифметичного • std::sort – для сортування за зростанням чи спаданням. • std::transform – для додавання, віднімання, множення на скаляр. Реалізувати такі функції: • перевірка, чи масив відсортований – використати std::is_sorted • кількість елементів певного значення – використати std::count
3	Шаблон класу CSet<typename T> – множина значень типу T. Елементи у множину можна додавати ти видаляти. Клас має підтримувати елементи у визначеному порядку (відсортовано). Використати такі алгоритми: • std::binary_search – перевірка на належність елемента множині • std::set_union – для оператора додавання (об'єднання множин) • std::set_difference – для оператора ділення (різниця множин) • std::set_intersection – для оператора віднімання (перетин множин) • std::sort – для сортування елементів у множині Реалізувати такі функції: • заповнення множини значеннями – використати функцію std::iota • повернення std::set<T> із копією елементів множини
4	Шаблон класу CMultiSet<typename T> – множина значень типу T, які можуть повторюватись. Елементи у множину можна додавати та видаляти. Клас має підтримувати елементи у визначеному порядку (відсортовано). Використати такі алгоритми: • std::equal_range – для визначення кількості елементів певного значення • std::set_union – для оператора додавання (об'єднання множин) • std::set_difference – для оператора ділення (різниця множин) • std::set_intersection – для оператора віднімання (перетин множин) • std::sort – для сортування елементів у множині Реалізувати такі функції: • заповнення множини значеннями – використати функцію std::generate • повернення std::multiset<T> із копією елементів множини
5	Шаблон класу CDictionary<typename TKey, typename TValue> – асоціативний масив із ключем типу TKey та значенням типу TValue. Клас має підтримувати елементи у визначеному порядку (відсортовано за ключем).

	Використати такі алгоритми: • std::sort – для сортування за ключем (для швидкого пошуку) • std::lower_bound – для швидкого пошуку по елемента за ключем • std::set_union – для оператора додавання (об’єднання множин) • std::set_difference – для оператора ділення (різниця множин) • std::set_intersection – для оператора віднімання (перетин множин) Реалізувати такі функції: • повернення std::map<TKey, TValue> із копіями значень
6	Шаблон класу CString<typename T> – стрічка символів (масив елементів символьного типу T). Розмір рядка змінюється динамічно Використати такі алгоритми: • std::sort – для сортування у порядку зростання та спадання • std::search – для пошуку підстрічки у стрічці • std::for_each – для оператора виведення у потік Реалізувати такі функції: • перевірка чи рядок починається із заданого рядка – використати std::equal • перевірка чи рядок закінчується заданим рядком – використати std::equal • повернення std::string<T> із копіями значень • оператор менше – використати std::lexicographical_compare
7	Шаблон класу CSingleLinkedList<typename T> – однозв'язний список об’єктів типу T. Елементи у список додаються динамічно. Використати такі алгоритми: • std::min_element – для знаходження мінімального елемента • std::max_element – для знаходження максимального елемента • std::transform – для додавання, віднімання, множення на скаляр. • std::reduce – для пошуку середнього арифметичного Реалізувати такі функції: • повернення std::vector<T> із копіями значень • інверсія порядку елементів
8	Шаблон класу CDoubleLinkedList<typename T> – двозв'язний список об’єктів типу T. Елементи у список додаються динамічно. Використати такі алгоритми: • std::min_element – для знаходження мінімального елемента • std::max_element – для знаходження максимального елемента • std::transform – для додавання, віднімання, множення на скаляр. • std::reduce – для пошуку середнього арифметичного Реалізувати такі функції: • повернення std::vector<T> із копіями значень • інверсія порядку елементів • видалення дублікатів значень – використати std::unique
9	Шаблон класу CStack<typename T> – стек об’єктів типу T. Елементи у стек додаються динамічно. Використати такі алгоритми: • std::transform – для додавання, віднімання, множення на скаляр. • std::for_each – для оператора виведення у потік Реалізувати такі функції: • інверсія порядку елементів • повернення std::vector<T> із копіями значень • повернення std::vector<T> із унікальними копіями значень

10	Шаблон класу CQueue<typename T> – однобічна черга об’єктів типу T. Елементи у чергу додаються динамічно. Використати такі алгоритми: • std::min_element – для знаходження мінімального елемента • std::max_element – для знаходження максимального елемента • std::reduce – для пошуку середнього арифметичного • std::for_each – для оператора виведення у потік Реалізувати такі функції: • інверсія порядку елементів • оператор порівняння – використати алгоритм std::equal
11	Шаблон класу CDeque<typename T> – двобічна черга об’єктів типу T. Елементи у чергу додаються динамічно. Використати такі алгоритми: • std::min_element – для знаходження мінімального елемента • std::max_element – для знаходження максимального елемента • std::reduce – для пошуку середнього арифметичного • std::for_each – для оператора виведення у потік Реалізувати такі функції: • інверсія порядку елементів • оператор порівняння – використати алгоритм std::equal
12	Клас CBitArray – одновимірний масив бітів. Кількість збережених бітів може змінюватись динамічно. Використати такі алгоритми: • std::count – для знаходження кількості бітів із вказаним значенням • std::transform – для побітових операцій • std::for_each – для оператора виведення у потік Реалізувати такі функції: • заповнення всіх бітів певним значенням – використати std::fill • знаходження наймолодшого ненульового біта - використати std::find_if
13	Шаблон класу CTable<typename T> – таблиця із колонками та рядками. Містить опис колонок та рядки із даними типу T. Колонки та рядки додаються динамічно. Використати такі алгоритми: • std::copy – для об’єднання рядків та декартового добутку • std::for_each – для оператора виведення у потік Реалізувати такі функції: • розвертання порядку колонок – використати std:: reverse • повернення std::vector<std::vector<std::string>> із значеннями всіх рядків
14	Клас CEncryptor – клас для шифрування/дешифрування тексту зсувом символів по алфавіту на ShiftCount позицій та зсувом вправо елементів на ту ж кількість позицій. Текст для шифрування задається динамічно. Використати такі алгоритми: • std::transform – для шифрування/розшифрування тексту • std::for_each – для оператора виведення у потік • std::equal – для оператора порівняння Реалізувати такі функції: • додатково зсунути елементи вправо на певну кількість символів – використати std::rotate

15	Шаблон класу CString<typename T> – стрічка символів (масив елементів символьного типу T). Розмір рядка змінюється динамічно Використати такі алгоритми: • std::remove – для видалення символу певного значення • std::replace – для заміни всіх входжень одного символу в стрічці заданим • std::sort – для сортування у порядку зростання • std::for_each – для оператора виведення у потік • std::equal – для оператора порівняння • std::lexicographical_compare – для операторів менше/більше Реалізувати такі функції: • перевірка чи рядок починається із заданого рядка – використати std::equal • перевірка чи рядок закінчується заданим рядком – використати std::equal • повернення std::string<T> із копіями значень

Варіанти завдань для пунктів 2 та 3

Варіант	**Асоціативний контейнер**	**Невпорядкований асоціативний контейнер**	**Адаптер**
1	std::set	std::unordered_map	std::stack
2	std::map	std::unordered_set	std::queue
3	std::multiset	std::unordered_multimap	std::priority_queue
4	std::multimap	std::unordered_multiset	std::stack
5	std::set	std::unordered_multimap	std::queue
6	std::map	std::unordered_multiset	std::priority_queue
7	std::multiset	std::unordered_map	std::stack
8	std::multimap	std::unordered_set	std::queue
9	std::set	std::unordered_map	std::priority_queue
10	std::map	std::unordered_set	std::stack
11	std::multiset	std::unordered_multimap	std::queue
12	std::multimap	std::unordered_multiset	std::priority_queue
13	std::set	std::unordered_multimap	std::stack
14	std::map	std::unordered_multiset	std::queue
15	std::multiset	std::unordered_map	std::priority_queue