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 (С++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 e 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, i-та сума не включає i-ий елемент |
| std::inclusive\_scan (C++17) | Схоже до partial\_sum, i-та сума включає i-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:
   1. для збереження елементів використати **послідовний контейнер** чи клас-рядок який найбільше підходить для вирішення задачі. Обґрунтувати вибір;
   2. змінити реалізацію існуючих функцій та операторів, використовуючи алгоритми із бібліотеки;
   3. додатково реалізувати функції члени, використовуючи алгоритми із бібліотеки.
2. Продемонструвати роботу із **асоціативним** та **невпорядкованим асоціативним контейнером** згідно із варіантом:
   1. вставку значень із вже присутнім та відсутнім ключем;
   2. успішний та неуспішний пошук за ключем;
   3. успішне та неуспішне видалення за ключем;
   4. послідовну ітерацію (проходження) по контейнеру для зчитування значень; 3. Продемонструвати роботу із **адаптером** згідно із варіантом:
   5. вставку значень;
   6. вичитку значень;
3. 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.

1. 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 |