Лабораторная работа №3

STL-совместимый контейнер

Инструментарий и требования к работе

Работа выполняется на C++. На сервере сборка под C++20.

Задание

В программе должен быть реализован шаблонный **класс BucketStorage**, представляющий собой stl-совместимый контейнер. В качестве шаблонного параметра выступает `T` – тип хранимых элементов. Использовать STL или другие стандартные классы C++ с нетривиальными деструкторами (= отличными от деструктора, который вы получаете по умолчанию с классом, который ничего не делает, Destructors – cppreference.com) запрещено.

К контейнеру предоставляются требования типа Container. Реализация без аллокаторов (`std::allocator`).

Требования к контейнеру: DefaultConstructible, CopyConstructible, CopyAssignable, MoveConstructible, MoveAssignable, Destructible. Все конструкторы/операторы из “правила пяти”, ~~реализованные с использованием copy-and-swap idiom~~.

Обработка ошибок (например, неуспешное выделение памяти) должна реализовываться через исключения C++.

Описание контейнера

Контейнер создан для сценариев с частыми созданием и удалением объектов фиксированного размера.

Все элементы в контейнере имеют стабильное расположение в памяти, что означает, что указатели/итераторы на хранимые объекты действительны всё время жизни этих объектов независимо от вставок/удалений в контейнер других объектов.

Общая идея: контейнер хранит набор блоков, каждый из которых может хранить некоторое число объектов, и идентификаторы (числовые значения) для каждого места под объект (элемент блока), указывающие, является ли эта позиция “активной” или “свободной”. Если место помечено как “свободное”, то оно пропускается при переходе по итераторам. Когда весь блок свободен, то он удаляется. Если происходит вставка, когда все блоки заполнены, то выделяется новый блок.

Контейнер хранит следующую информацию:

1. Блоки памяти.
2. Идентификаторы “активности”.
3. Прочие метаданные (размер, вместимость, ...).

В массиве идентификаторов значение 0 указывает на то, что элемент “активен”, а значение большее 0 – расстояние, помогающее найти соседний активный элемент.

Операция	**Сложность**
Вставка элемента	O(1)
Удаление элемента	O(1)
Переход итератора на следующий/предыдущий активный элемент	O(1)

Конструкторы

1. По умолчанию (DefaultConstructible).
2. Все из “правила пяти” (CopyConstructible, MoveConstructible).
3. Устанавливающий вместимость блока (принимает `size_type block_capacity`). Создаёт контейнер и задаёт вместимость блока равным `block_capacity`.
   По умолчанию `block_capacity` равен 64.

Функции/операторы

Функция/оператор	**Описание**
`iterator insert (const value_type &value)`	Вставляет в контейнер объект (copy-constructor)
`iterator insert (value_type&& value)`	Вставляет в контейнер объект (move-constructor)
`iterator erase(const_iterator it)`	Удаляет объект, на который указывает итератор. Возвращает итератор на следующий (активный) элемент (или end(), если таковых нет).
`bool empty()`	Возвращает true, если контейнер пустой.
`size_type size()`	Возвращает число элементов, хранящихся в контейнере.
`size_type capacity()`	Возвращает число элементов, которые могут быть сохранены в контейнере без его расширения.
`void shrink_to_fit()`	Изменяет ёмкость (capacity) контейнера до минимально необходимого для хранения всех текущих элементов. Обратите внимание: этот метод может нарушать свойство стабильности адреса в памяти хранящихся объектов.

Функция/оператор	Описание
`void clear()`	Разрушает все объекты в контейнере и устанавливает размер size() в 0.
`void swap(BucketStorage &other)`	Меняет местами содержимое текущего и переданного контейнеров.
`iterator begin(), const_iterator begin(), const_iterator cbegin()`	Возвращают итераторы на первый элемент контейнера.
`iterator end(), const_iterator end(), const_iterator cend()`	Возвращают итераторы за последний элемент контейнера.
`iterator get_to_distance(iterator it, const difference_type distance)`	Сдвигает итератор на переданное расстояние (в элементах). Возвращает новый итератор.

Описание итератора

Помимо класса контейнера необходимо реализовать класс **итератор**, представляющий собой реализацию двунаправленного итератора (требования: bidirectional). Больше/меньше указывают на то, находится ли итератор дальше / ближе к концу по сравнению с другим итератором в том же контейнере. Асимптотика всех операций O(1).

Операторы	`++`	`==`	`<`	`<=`	`*` `(унарный)`
`=`	`--`	`!=`	`>`	`>=`	`->`

Правила инвалидации итераторов:

Случай	**Инвалидация**
Все операции, не модифицирующие содержимое (read-only)	Не происходит
clear, operator=	Всегда
shrink_to_fit	если capacity() != size()
erase	Только на удаляемый элемент

Примечания по тестам и проверке

1. После взятия репозитория прочитайте README.md.
2. Код с тестами должен компилироваться. Некомпилирующийся код на проверку не принимается.
3. Код не принимается на проверку, если этап `build` не пройден успешно.
4. Проверка на форматирование кода является блокирующей. Следите за тем, что ваш код отформатирован в соответствии с приведённым `.clang-format`.
5. Если какой-то метод не реализован, то в коде всё-равно должна быть “заглушка”, чтобы код скомпилировался.
6. Оценка критерия “Асимптотика” будет проводится за день до защиты или на самой защите. Баллы по этому критерию появятся не позже баллов за защиту.
7. [Known Issue] Использование `std::format` приведёт к ошибке компиляции на GitHub. Не используйте `std::format`.
8. При любых неполадках / проблем с тестами в репозиториях следует обращаться Виктории в лс, не забывая представиться кто вы, из какой группы и с какой дисциплины.

В случае, когда после взятия репозитория не было ни одного успешного запуска `Init` (есть только один неуспешный), при первом `BuildTest` запуске произойдет вызов этого самого `Init` workflow и всё заработает (должно). Как это выглядит показано на рисунке 1. Когда хоть один `Init` отработал успешно, то повторно он отрабатывать не будет и все шаги с `Init` будут помечены как `skipped` (рисунок 2).

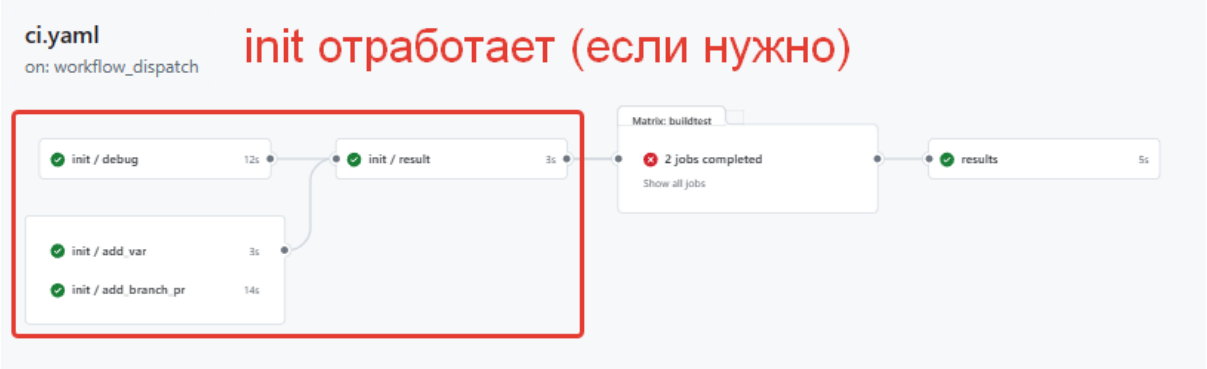

Рисунок 1 – Автоматический запуск `Init` из `BuildTest` при необходимости

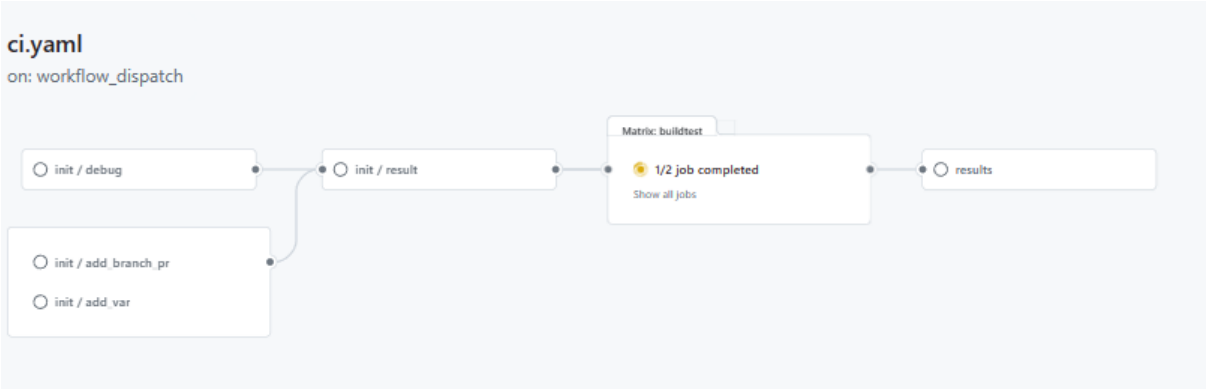

Рисунок 2 – Init из `BuildTest` не запускается, если хоть один `Init` отработал успешно