Министр науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет информационных технологий и программирования

Лабораторная работа №7

Кольцевой буфер

Выполнила студент группы № M3119
Самигуллин Руслан Рустамович
Подпись:

Проверил:
Повышев Владислав Вячеславович

Санкт-Петербург
2024

Задание:

Реализовать кольцевой буфер в виде STL-совместимого контейнера (например, может быть использован с стандартными алгоритмами), обеспеченного итератором произвольного доступа. Реализация не должна использовать ни одни из контейнеров STL. Буфер должен обладать следующими возможностями:

1. Вставка и удаление в конец.
2. Вставка и удаление в начало.
3. Вставка и удаление в произвольное место по итератору.
4. Доступ в конец, начало.
5. Доступ по индексу.
6. Изменение capacity.

Отчет:

1. CircularBuffer – шаблонный класс, реализующий кольцевой буфер. Включает в себя методы для добавления, удаления, доступа к элементам, изменения размера и другие операции с буфером.

2. Методы:

- getSize(): Возвращает текущее количество элементов в буфере.
- getCapacity(): Возвращает емкость буфера.
- isEmpty(): Проверяет, пуст ли буфер.
- isFull(): Проверяет, заполнен ли буфер.
- clear(): Очищает буфер, устанавливая его размер в 0 и обнуляя указатели.
- push_back(const T& value): Добавляет элемент в конец буфера.
- pop_back(): Удаляет последний элемент из буфера.
- push_front(const T& value): Добавляет элемент в начало буфера.
- pop_front(): Удаляет первый элемент из буфера.
- insert(const T& value, size_t index): Вставляет элемент в буфер по указанному индексу.
- erase(size_t index): Удаляет элемент из буфера по указанному индексу.
- resize(size_t new_capacity): Изменяет емкость буфера на новое значение.
- front(): Возвращает ссылку на первый элемент буфера.
- back(): Возвращает ссылку на последний элемент буфера.
- operator[]: Перегруженный оператор доступа к элементам буфера по индексу.

3. Тестирование:

В основной функции main создается экземпляр CircularBuffer<int> и проводится ряд операций с буфером, включая добавление, удаление, доступ к элементам, изменения размера и т.д. Результаты этих операций выводятся на консоль для отображения работы класса CircularBuffer.

4. Контейнеры:

Тестирование производится на контейнере CircularBuffer<int>, но класс CircularBuffer можно использовать с любым типом данных.

Результат работы:
Initial buffer capacity: 5

Capacity: 5 | Buffer after push_back(1), push_back(2), push_back(3): 1 2 3 4 5
Capacity: 5 | Buffer after pop_back(): 1 2 3 4

Capacity: 5 | Buffer after push_front(5): 5 1 2 3 4
Capacity: 5 | Buffer after pop_front(): 1 2 3 4

Capacity: 5 | Buffer after erase(1): 1 3 4
Capacity: 5 | Buffer after insert(10, 0): 10 1 3 4

Front element: 10
Back element: 3

Capacity: 8 | Buffer after resize(8): 10 1 3 4 2 5 6 7
Capacity: 8 | Buffer after clear():

Вопросы и ответы:
1. Что такое STL? Что значит “STL-совместимый контейнер”?
Ответ: это стандартная библиотека шаблонов, которая предоставляет множество шаблонных классов и функций для работы с данными, алгоритмами и контейнерами. Это означает, что такой контейнер предоставляет определенный набор методов и функций для работы с данными, который аналогичен тому, что предоставляют стандартные контейнеры STL.

2. Что такое итератор, зачем они были придуманы?
Ответ: это объект, позволяющий просматривать элементы в контейнере и осуществлять над ними различные операции. Они были придуманы для создания способа обхода и доступа к элементам в различных контейнерах и коллекциях данных.

3. Как реализован итератор произвольного доступа? Какие методы/как работает?
Итератор произвольного доступа позволяет перемещаться по элементам контейнера в любом направлении и выполнять операции типа доступа к элементам, арифметики указателей и т. д. Реализация итератора произвольного доступа включает методы для перемещения вперед, назад, получения значения, а также операторы сравнения.

4. Как вы подошли к задаче реализации кольцевого буфера? Как он устроен “под капотом”? Как работают операции (например, операция A меняет размер стека, может привести к изменению capacity и делает то-то и то-то)?
Ответ: я реализовал кольцевой буфер, которая включает в себя массив фиксированного размера и индексы head и tail, которые указывают на начало и конец буфера.
При добавлении элементов, tail сдвигается вправо, а при удалении - влево. Если tail достигает конца буфера, он переходит в начало (или наоборот).
Операция изменения размера буфера включает выделение нового массива, копирование элементов и обновление индексов head, tail и capacity.

Операции с кольцевым буфером:

push_back: элемент добавляется в конец буфера, при этом индекс tail сдвигается вправо. Если буфер полон, операция не выполняется.
pop_back: последний элемент буфера удаляется, и индекс tail сдвигается влево.
Если буфер пуст, операция не выполняется.
resize: ыыделяется новый массив с новой емкостью, копируются элементы из старого массива, обновляются индексы и емкость буфера.
(operator[], front(), back()): позволяет получить доступ к элементам буфера по индексу или получить первый и последний элементы.

```
Initial buffer capacity: 5

Capacity: 5 | Buffer after push_back(1), push_back(2), push_back(3): 1 2 3 4 5
Capacity: 5 | Buffer after pop_back(): 1 2 3 4

Capacity: 5 | Buffer after push_front(5): 5 1 2 3 4
Capacity: 5 | Buffer after pop_front(): 1 2 3 4

Capacity: 5 | Buffer after erase(1): 1 3 4
Capacity: 5 | Buffer after insert(10, 0): 10 1 3 4

Front element: 10
Back element: 3

Capacity: 8 | Buffer after resize(8): 10 1 3 4 2 5 6 7
Capacity: 8 | Buffer after clear():
```

Пример использования стандартного алгоритма, который находит максимальный и минимальный элемент в контейнере:

```
        #include < algorithm>
        // min + max
    auto min_element_it = std::min_element(&buffer.front(), &buffer.back());
    auto max_element_it = std::max_element(&buffer.front(), &buffer.back());
        // Вывод результатов
    std::cout << "Minimum element from head to tail: " << *min_element_it << std::endl;
    std::cout << "Maximum element from head to tail: " << *max_element_it << std::endl;
```