Deque in C++ Standard Template Library (STL)

Double ended queues are sequence containers with the feature of expansion and contraction on both the ends.
They are similar to vectors, but are more efficient in case of insertion and deletion of elements. Unlike vectors, contiguous storage allocation may not be guaranteed.
Double Ended Queues are basically an implementation of the data structure double ended queue. A queue data structure allows insertion only at the end and deletion from the front. This is like a queue in real life, wherein people are removed from the front and added at the back. Double ended queues are a special case of queues where insertion and deletion operations are possible at both the ends.

The functions for deque are same as vector, with an addition of push and pop operations for both front and back.

```
#include <iostream>
#include <deque>

using namespace std;

void showdq(deque <int> g)
{
    deque <int> :: iterator it;
    for (it = g.begin(); it != g.end(); ++it)
        cout << '\t' << *it;
    cout << '\n';
}

int main()
{
    deque <int> gquiz;
    gquiz.push_back(10);
    gquiz.push_front(20);
    gquiz.push_back(30);
    gquiz.push_front(15);
    cout << "The deque gquiz is : ";
    showdq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.max_size() : " << gquiz.max_size();

    cout << "\ngquiz.at(2) : " << gquiz.at(2);
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop_front() : ";
    gquiz.pop_front();
    showdq(gquiz);

    cout << "\ngquiz.pop_back() : ";
    gquiz.pop_back();
    showdq(gquiz);

    return 0;
}
```

The output of the above program is :

```
The deque gquiz is :     15     20     10     30

gquiz.size() : 4
gquiz.max_size() : 4611686018427387903
gquiz.at(2) : 10
gquiz.front() : 15
gquiz.back() : 30
gquiz.pop_front() :     20    10    30

gquiz.pop_back() :     20    10
```

Methods of Deque:

- deque insert() function in C++ STL: Inserts an element. And returns an iterator that points to the first of the newly inserted elements.
- deque rbegin() function in C++ STL: Returns a reverse iterator which points to the last element of the deque (i.e., its reverse beginning).
- deque rend() function in C++ STL: Returns a reverse iterator which points to the position before the beginning of the deque (which is considered its reverse end).
- deque cbegin() in C++ STL: Returns a constant iterator pointing to the first element of the container, that is, the iterator cannot be used to modify, only traverse the deque.
- deque max_size() function in C++ STL: Returns the maximum number of elements that a deque container can hold.
- deque assign() function in C++ STL: Assign values to the same or different deque container.
- deque resize() function in C++ STL: Function which changes the size of the deque.
- deque::push_front() in C++ STL: This function is used to push elements into a deque from the front.
- deque::push_back() in C++ STL: This function is used to push elements into a deque from the back.
- deque::pop_front() and deque::pop_back() in C++ STL: **pop_front()** function is used to pop or remove elements from a deque from the front. **pop_back()** function is used to pop or remove elements from a deque from the back.
- deque::front() and deque::back() in C++ STL: **front()** function is used to reference the first element of the deque container. **back()** function is used to reference the last element of the deque container.
- deque::clear() and deque::erase() in C++ STL: **clear()** function is used to remove all the elements of the deque container, thus making its size 0. **erase()** function is used to remove elements from a container from the specified position or range.
- deque::empty() and deque::size() in C++ STL: **empty()** function is used to check if the deque container is empty or not. **size()** function is used to return the size of the deque container or the number of elements in the deque container.
- deque::operator= and deque::operator[] in C++ STL:
  operator= operator is used to assign new contents to the container by replacing the existing contents. **operator[]** operator is used to reference the element present at position given inside the operator.
- deque::at() and deque::swap() in C++ STL: **at()** function is used reference the element present at the position given as the parameter to the function. **swap()** function is used to swap the contents of one deque with another deque of same type and size.
- deque::begin() and deque::end in C++ STL: **begin()** function is used to return an iterator pointing to the first element of the deque container. **end()** function is used to return an iterator pointing to the last element of the deque container.
- deque::emplace_front() and deque::emplace_back() in C++ STL: **emplace_front()** function is used to insert a new element into the deque container. The new element is added to the beginning of the deque. **emplace_back()** function is used to insert a new element into the deque container. The new element is added to the end of the deque.

Recent Articles on Deque

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Map in C++ Standard Template Library (STL)
The C++ Standard Template Library (STL)
Set in C++ Standard Template Library (STL)
Multiset in C++ Standard Template Library (STL)
List in C++ Standard Template Library (STL)

Pair in C++ Standard Template Library (STL)
Sort in C++ Standard Template Library (STL)
Multimap in C++ Standard Template Library (STL)
Queue in Standard Template Library (STL)
Unordered Sets in C++ Standard Template Library
Priority Queue in C++ Standard Template Library (STL)
Binary Search in C++ Standard Template Library (STL)
deque::pop_front() and deque::pop_back() in C++ STL
deque::front() and deque::back() in C++ STL
deque::empty() and deque::size() in C++ STL

Improved By : dkp1903

Article Tags : C++ cpp-containers-library cpp-deque deque STL

Practice Tags : STL CPP

13

1.4

☐ To-do ☐ Done

Based on **40** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments