STL Queue in C++

The `queue` is a container adapter in the C++ STL that provides **FIFO (First-In-First-Out)** functionality. Elements are inserted at the back and removed from the front.

---

1. Include Header

```
#include <queue>
#include <iostream>
using namespace std;
```

---

2. Creating a Queue

```
queue<int> q1;            // Empty queue of integers
queue<string> q2;         // Empty queue of strings
```

---

3. Common Queue Operations with Examples

3.1. Push Elements

```
queue<int> q;
q.push(10);
q.push(20);
q.push(30);
```

Time Complexity: O(1) (Amortized)

3.2. Pop Element

```
q.pop(); // Removes the front element
```

Note: `pop()` does not return the popped value. Use `front()` to access before popping.

3.3. Front Element

```
cout << q.front() << endl; // Prints the element at the front
```

3.4. Back Element

```
cout << q.back() << endl; // Prints the element at the back
```

3.5. Check Size

```
cout << q.size(); // Number of elements in queue
```

3.6. Check If Empty

```
if (q.empty())
    cout << "Queue is empty";
```

---

4. Print and Empty the Queue

```
while (!q.empty())
{
    cout << q.front() << " ";
    q.pop();
}
```

---

5. Queue Limitations

- No random access
- No iterators
- Cannot be traversed using range-based for loop

STL Deque in C++

The `deque` **(double-ended queue)** is a **sequence container** in C++ STL that allows fast insertion and deletion at both the **front and back**.

---

1. Include Header

```
#include <deque>
#include <iostream>
using namespace std;
```

---

2. Creating a Deque

```
deque<int> d1;            // Empty deque of integers
deque<string> d2;         // Empty deque of strings
```

3. Common Deque Operations with Examples

3.1. Push Elements

```
deque<int> d;
d.push_back(10);
d.push_front(20);
d.push_back(30);
```

Time Complexity: O(1) for both ends (amortized)

3.2. Pop Elements

```
d.pop_back();    // Removes 30
d.pop_front();   // Removes 20
```

3.3. Access Elements

```
cout << d.front() << endl; // First element
cout << d.back() << endl;  // Last element
cout << d[0] << endl;      // Random access
```

3.4. Check Size

```
cout << d.size(); // Number of elements
```

3.5. Check If Empty

```
if (d.empty())
    cout << "Deque is empty";
```

3.6. Insert

```
d.insert(d.begin() + 1, 100);
```

3.7. Erase

```
d.erase(d.begin() + 1, d.begin() + 3);
```

3.8. Sort

```
sort(d.begin(), d.end());
```

3.9. Unique

```
sort(d.begin(), d.end());
d.erase(unique(d.begin(), d.end()), d.end());
```

3.10. Find

```
auto it = find(d.begin(), d.end(), 20);
cout << it - d.begin() << endl;
```

---

4. Traverse the Deque

4.1. Using Loop

```
for (int i = 0; i < d.size(); i++)
    cout << d[i] << " ";
```

4.2. Using Range-Based For Loop

```
for (auto x : d)
    cout << x << " ";
```

4.3. Using Iterator

```
for (auto it = d.begin(); it != d.end(); ++it)
    cout << *it << " ";
```

---

5. Deque Advantages

- Allows random access (like vector)
- Supports insertion/deletion from both ends
- Provides iterators and can be used in range-based loops

Practice Problems

1. Reverse First K Elements of a Queue

Link: [Reversing the first K elements of a Queue - GeeksforGeeks]

Problem: Given a queue and an integer k, reverse the first k elements of the queue, leaving the rest in order.

Sample Input:
Queue: 10 20 30 40 50
K: 3

Output: 30 20 10 40 50

💡 **Idea:** Use a stack to reverse first k elements, then enqueue them back followed by the remaining elements.

```
#include <bits/stdc++.h>
using namespace std;
void reverseK(queue<int> &q, int k)
{
    stack<int> st;
    for (int i = 0; i < k; i++)
    {
        st.push(q.front());
        q.pop();
    }
    while (!st.empty())
    {
        q.push(st.top());
        st.pop();
    }
    int size = q.size();
    for (int i = 0; i < size - k; i++)
    {
        q.push(q.front());
        q.pop();
    }
}

int main()
{
    queue<int> q;
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        q.push(x);
    }
```

```
    int k;
    cin >> k;
    reverseK(q, k);
    while (!q.empty())
    {
        cout << q.front() << " ";
        q.pop();
    }
}
```

---

2. Generate Binary Numbers from 1 to N

Link: [Generate Binary Numbers from 1 to n - GeeksforGeeks]

Problem: Given N, print binary representations of numbers from 1 to N.

Sample Input: N = 5
Output: 1 10 11 100 101

💡 **Idea:** Use a queue to perform level-wise binary number generation like BFS.

```
#include <bits/stdc++.h>
using namespace std;
void generateBinary(int N)
{
    queue<string> q;
    q.push("1");
    for (int i = 0; i < N; i++)
    {
        string curr = q.front();
        q.pop();
        cout << curr << " ";
        q.push(curr + "0");
        q.push(curr + "1");
    }
}
int main()
{
    generateBinary(10);
}
```

---

3. Interleave First and Second Half of Queue

Link: [Interleave the first half of the queue with second half - GeeksforGeeks]

Problem: Given a queue with even number of integers, interleave the first and second half.

Sample Input: Queue: 1 2 3 4 5 6 (size will be even)
Output: 1 4 2 5 3 6

💡 **Idea:** Split into two halves using another queue, then merge alternatively.

```
#include <bits/stdc++.h>
using namespace std;
void interleaveQueue(queue<int> &q)
{
    int n = q.size();
    queue<int> firstHalf;
    for (int i = 0; i < n / 2; i++)
    {
        firstHalf.push(q.front());
        q.pop();
    }
    while (!firstHalf.empty())
    {
        q.push(firstHalf.front());
        firstHalf.pop();
        q.push(q.front());
        q.pop();
    }
}

int main()
{
    queue<int> q;
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        q.push(x);
    }
    interleaveQueue(q);
    while (!q.empty())
    {
        cout << q.front() << " ";
        q.pop();
    }
}
```