**Priority Queue**

A priority queue in C++ STL is a **container adapter** that provides constant time lookup for the highest or lowest element, depending on the priority set. It is implemented using a heap data structure.

Here are some important features of a priority queue in C++ STL:

1. Default priority queue is a max-heap i.e. the largest element is always on the top. We can create a min-heap by passing a comparator function as a template argument.

2. The priority queue can be initialized using a range of elements, an array, or an initializer list.

3. The priority queue provides constant time access to the top element using the **top()** function.

4. The elements in a priority queue are automatically sorted according to the priority set.

5. The **push()** function is used to insert an element into the priority queue.

6. The **pop()** function is used to remove the top element from the priority queue.

7. The **size()** function is used to return the number of elements in the priority queue.

8. The **empty()** function is used to check if the priority queue is empty.

Here is an example of how to use priority queue in C++ STL:

int main() {

priority\_queue<int> pq;

pq.push(10);

pq.push(20);

pq.push(30);

while (!pq.empty()) {

cout << pq.top() << " ";

pq.pop();

}

return 0;

}

Priority Queue can be implemented in many Different ways:

* priority\_queue<int> pq;
* priority\_queue<pair<int, vector<pair<int, int>>>> cont;

**Max-Heap**

A heap is a **specialized tree-based data structure** that satisfies the heap property. In a heap, the **parent node is either greater than or equal** to (for max heap) or less than or equal to (for min heap) its children.

Heaps are a powerful data structure that can provide efficient access to the maximum or minimum element of a collection.

A max heap is a binary tree where the value of each parent node is greater than or equal to the values of its children nodes. In a max heap, the node with the highest value is always the root node.

The heap property in a max heap means that every parent node is greater than or equal to its children. The largest value is always at the root of the heap.

Some important operations in a max heap include:

Insertion: Inserting a new node into the heap while maintaining the heap property.

Deletion: Removing the root node while maintaining the heap property.

Peek: Returning the value of the root node without removing it from the heap.

A heap is commonly implemented as an array, where the root is stored at index 0, and the left and right children of a node at index i are stored at indices 2i+1 and 2i+2, respectively. This makes it easy to maintain the heap property and to perform heap operations efficiently.

Heap operations have a time complexity of O(log n) for both insertion and deletion, where n is the number of nodes in the heap. The peek operation has a time complexity of O(1).

**NOTE**

Default Implementation of Priority Queue builds Max Heap

* priority\_queue<int> pq

**Min Heap**

A min heap is a binary tree where **the parent node is always smaller than or equal to its child nodes**. This means that the minimum element in the heap is always stored at the root of the tree. Min heaps are commonly used in algorithms that require finding the minimum element in a set of elements, such as Dijkstra's shortest path algorithm.

In C++, you can implement a min heap using the std::priority\_queue container adapter by specifying a **comparison function** that returns true when its first argument should appear after its second argument in the heap ordering. To create a min heap using a priority queue, you can use the following code:

* priority\_queue<int, vector<int>, greater<int>> pq;

**NOTE**

* Min heaps do not remove duplicates by default. Means we have 2 same values at the top and another element after top. That’s why we need to pop 1 extra element.

**Custom Comparator for Min Heap**

// Custom comparator for std::priority\_queue

struct PriorityComparator {

bool operator()(const int& a, const int& b) {

return a > b; // Creates a min heap

}

};

// Using std::priority\_queue with custom comparator

std::priority\_queue<int, std::vector<int>, PriorityComparator> pq;

**NOTE**

* In the context of using operator() within a class, it's often referred to as "function call operator overloading
* operator() is overloaded for the CustomComparator struct, and it defines how two int values should be compared when they are passed to an object of type CustomComparator.
* When you create an instance of CustomComparator and use it as a comparator in a priority queue, the operator() function gets called to determine the ordering of elements.

In the context of creating a min heap using a std::priority\_queue in C++, the vector<int> template parameter refers to the underlying container used to implement the priority queue.

A priority queue is a container adapter that provides constant time access to the largest (or smallest) element in the container, based on a comparison function. The underlying container stores the elements in the priority queue and provides the basic operations of insertion, removal, and access to elements.

The default underlying container for std::priority\_queue is std::vector. When you specify the template parameters for std::priority\_queue, the second parameter specifies the container type to use, and the third parameter specifies the comparison function type.

For example, std::priority\_queue<int, vector<int>, greater<int>> specifies a priority queue of integers, implemented using std::vector<int> as the underlying container, and using std::greater<int> as the comparison function to create a min heap.

By default, std::priority\_queue uses std::less<T> as the comparison function, which creates a max heap.