Priority Queue in C++ Standard Template Library (STL)

Priority queues are a type of container adapters, specifically designed such that the first element of the queue is the greatest of all elements in the queue and elements are in non increasing order(hence we can see that each element of the queue has a priority{fixed order}).

|  |
| --- |
| // Note that by default C++ creates a max-heap  // for priority queue  #include <iostream>  #include <queue>    using namespace std;    void showpq(priority\_queue <int> gq)  {      priority\_queue <int> g = gq;      while (!g.empty())      {          cout << '\t' << g.top();          g.pop();      }      cout << '\n';  }    int main ()  {      priority\_queue <int> gquiz;      gquiz.push(10);      gquiz.push(30);      gquiz.push(20);      gquiz.push(5);      gquiz.push(1);        cout << "The priority queue gquiz is : ";      showpq(gquiz);        cout << "\ngquiz.size() : " << gquiz.size();      cout << "\ngquiz.top() : " << gquiz.top();          cout << "\ngquiz.pop() : ";      gquiz.pop();      showpq(gquiz);        return 0;  } |

**Output:**

The priority queue gquiz is : 30 20 10 5 1

gquiz.size() : 5

gquiz.top() : 30

gquiz.pop() : 20 10 5 1

**How to create a min heap for priority queue?**  
C++ provides below syntax for the same.

*// Syntax to create a min heap for priority queue  
priority\_queue <int, vector<int>, greater<int>> g = gq;*

|  |
| --- |
| // C++ program to demonstrate min heap  #include <iostream>  #include <queue>    using namespace std;    void showpq(priority\_queue <int, vector<int>, greater<int>> gq)  {      priority\_queue <int, vector<int>, greater<int>> g = gq;      while (!g.empty())      {          cout << '\t' << g.top();          g.pop();      }      cout << '\n';  }    int main ()  {      priority\_queue <int, vector<int>, greater<int>> gquiz;      gquiz.push(10);      gquiz.push(30);      gquiz.push(20);      gquiz.push(5);      gquiz.push(1);        cout << "The priority queue gquiz is : ";      showpq(gquiz);        cout << "\ngquiz.size() : " << gquiz.size();      cout << "\ngquiz.top() : " << gquiz.top();          cout << "\ngquiz.pop() : ";      gquiz.pop();      showpq(gquiz);        return 0;  } |

**Output:**

The priority queue gquiz is : 1 5 10 20 30

gquiz.size() : 5

gquiz.top() : 1

gquiz.pop() : 5 10 20 30

**Note :** The above syntax is difficult to remembers, so in case of numeric values, we can multiply values with -1 and use max heap to get the effect of min heap

**Methods of priority queue are:**

* [priority\_queue::empty() in C++ STL](https://www.geeksforgeeks.org/priority_queueempty-priority_queuesize-c-stl/)– **empty()** function returns whether the queue is empty.
* [priority\_queue::size() in C++ STL](https://www.geeksforgeeks.org/priority_queueempty-priority_queuesize-c-stl/)– **size()** function returns the size of the queue.
* [priority\_queue::top() in C++ STL](https://www.geeksforgeeks.org/priority_queuetop-c-stl/)– Returns a reference to the top most element of the queue
* [priority\_queue::push() in C++ STL](https://www.geeksforgeeks.org/priority_queuepush-priority_queuepop-c-stl/)**– push(g)** function adds the element ‘g’ at the end of the queue.
* [priority\_queue::pop() in C++ STL](https://www.geeksforgeeks.org/priority_queuepush-priority_queuepop-c-stl/)– **pop()** function deletes the first element of the queue.
* [priority\_queue::swap() in C++ STL](https://www.geeksforgeeks.org/priority_queueswap-c-stl/)– This function is used to swap the contents of one priority queue with another priority queue of same type and size.
* [priority\_queue::emplace() in C++ STL](https://www.geeksforgeeks.org/priority_queueemplace-c-stl/)– This function is used to insert a new element into the priority queue container, the new element is added to the top of the priority queue.
* [priority\_queue value\_type in C++ STL](https://www.geeksforgeeks.org/priority_queue-value_type-in-c-stl/)– Represents the type of object stored as an element in a priority\_queue. It acts as a synonym for the template parameter.

// PRIORITY\_QUEUE of PAIRS //

// following program demonstrates the priority\_queue of vectors//

#include<iostream>

#include<queue>

#include<utility>

using namespace std;

int main()

{

priority\_queue<pair<int, int> > pq;

pq.push(make\_pair(10, 200));

pq.push(make\_pair(20, 100));

pq.push(make\_pair(15, 400));

pq.push(make\_pair(20, 400));

pq.push(make\_pair(15, 300));

while(!pq.empty())

{ pair<int, int> top = pq.top();

cout << top.first << " " << top.second;

cout<<endl;

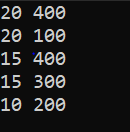
pq.pop() ;

}

return 0;

}

Output :



|  |
| --- |
| // C++ program to create a priority queue of pairs.  // We can create a min heap by passing adding two  // parameters, vector and greater().  #include <bits/stdc++.h>    using namespace std;    **typedef pair<int, int> pi;**    // Driver program to test methods of graph class  int main()  {  **priority\_queue<pi, vector<pi>, greater<pi> > pq;**      pq.push(make\_pair(10, 200));      pq.push(make\_pair(20, 100));      pq.push(make\_pair(15, 400));        pair<int, int> top = pq.top();      cout << top.first << " " << top.second;      return 0;  } |
|  |

Output :

10 200

# Applications of Priority Queue

A [Priority Queue](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/) is different from a normal [queue](https://www.geeksforgeeks.org/queue-set-1introduction-and-array-implementation/), because instead of being a “first-in-first-out”, values come out in order by priority. It is an abstract data type that captures the idea of a container whose elements have “priorities” attached to them. An element of highest priority always appears at the front of the queue. If that element is removed, the next highest priority element advances to the front.

A priority queue is typically implemented using [Heap data structure](https://www.geeksforgeeks.org/heap-data-structure/).

**Applications:**

[Dijkstra’s Shortest Path Algorithm using priority queue](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/): When the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra’s algorithm.

[Prim’s algorithm](https://www.geeksforgeeks.org/prims-algorithm-using-priority_queue-stl/): It is used to implement Prim’s Algorithm to store keys of nodes and extract minimum key node at every step.

[Data compression](https://en.wikipedia.org/wiki/Data_compression): It is used in [Huffman codes](https://www.geeksforgeeks.org/tag/huffman-coding/) which is used to compresses data.

**Artificial Intelligence**: [A\* Search Algorithm](https://www.geeksforgeeks.org/a-search-algorithm/) : The A\* search algorithm finds the shortest path between two vertices of a weighted graph, trying out the most promising routes first. The priority queue (also known as the fringe) is used to keep track of unexplored routes, the one for which a lower bound on the total path length is smallest is given highest priority.

[Heap Sort](https://www.geeksforgeeks.org/heap-sort/) : Heap sort is typically implemented using Heap which is an implementation of Priority Queue.

[Operating systems](https://en.wikipedia.org/wiki/Operating_system): It is also use in Operating System for [load balancing](https://en.wikipedia.org/wiki/Load_balancing_(computing)) ([load balancing on server](https://www.geeksforgeeks.org/load-balancing-on-servers-random-algorithm/)), [interrupt handling](https://practice.geeksforgeeks.org/problems/interrupt-handlers).

# STL Priority Queue for Structure or Class

[STL priority\_queue](https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/)is the implementation of [Heap Data-structure.](https://www.geeksforgeeks.org/heap-data-structure/). By default, it’s a max heap and we can easily use it for primitive datatypes. There are some important applications of it which can be found [here](https://www.geeksforgeeks.org/applications-priority-queue/)

**Prerequisite:** [Prioirty\_queue Basics](https://www.geeksforgeeks.org/priority_queuepush-priority_queuepop-c-stl/)

In this article, we will see how can we use priority\_queue for custom datatypes like class or structure.  
suppose we have a structure name Person which consist of two variables **Age** and **height**  
and we want to store that that in priority\_queue then a simple method won’t work here.

Given below is a example of declaration of struct Person:

|  |
| --- |
| struct Person{  int Age;  float Height;  } |

On defining the Priority Queue as shown below, it’ll give us error since priority\_queue doesn’t know on what order(min or max) we need to arrange the objects.

|  |
| --- |
| priority\_queue<Person> pq; |

To rectify the error above, we will use [operator overloading](https://www.geeksforgeeks.org/operator-overloading-c/) to define the priority. So that priority\_queue can decide how to store the structure object.

Given below is the priority\_queue implementation with structure below:

|  |
| --- |
| // program in c++ to use priority\_queue with structure    #include <iostream>  #include <queue>  using namespace std;  #define ROW 5  #define COL 2    struct Person {        int age;        float height;        // this will used to initialize the variables      // of the structure      Person(int age, float height)          : age(age), height(height)      {      }  };    // this is an strucure which implements the  // operator overlading  struct CompareHeight {      bool operator()(Person const& p1, Person const& p2)      {          // return "true" if "p1" is ordered          // before "p2", for example:          return p1.height < p2.height;      }  };    int main()  {      priority\_queue<Person, vector<Person>, CompareHeight> Q;        // When we use priority\_queue with  structure      // then we need this kind of syntax where      // CompareAge is the functor or comparision function      float arr[ROW][COL] = { { 30, 5.5 }, { 25, 5 },                      { 20, 6 }, { 33, 6.1 }, { 23, 5.6 } };        for (int i = 0; i < ROW; ++i) {            Q.push(Person(arr[i][0], arr[i][1]));            // insert an object in priority\_queue by using          // the Person strucure constructor      }        while (!Q.empty()) {          Person p = Q.top();          Q.pop();          cout << p.age << " " << p.height << "\n";      }      return 0;  } |

Output :

33 6.1

20 6

23 5.6

30 5.5

25 5

Given below is the implementation of priority\_queue using Class

|  |
| --- |
| // program in c++ to use priority\_queue with class  #include <iostream>  #include <queue>  using namespace std;    #define ROW 5  #define COL 2    class Person {    public:      int age;        float height;        // this is used to initialize the variables of the class      Person(int age, float height)          : age(age), height(height)      {      }  };    // we are doing operator overloading through this  bool operator<(const Person& p1, const Person& p2)  {        // this will return true when second person      // has greater height. Suppose we have p1.height=5      // and p2.height=5.5 then the object which      // have max height will be at the top(or      // max priority)  **return p1.height < p2.height;**  }    int main()  {        priority\_queue<Person> Q;  float arr[ROW][COL] = { { 30, 5.5 }, { 25, 5 },                 { 20, 6 }, { 33, 6.1 }, { 23, 5.6 } };        for (int i = 0; i < ROW; ++i) {     Q.push(Person(arr[i][0], arr[i][1]));            // insert an object in priority\_queue by using          // the Person class constructor      }        while (!Q.empty()) {     Person p = Q.top();    Q.pop();    cout << p.age << " " << p.height << "\n";      }      return 0;  } |
|  |

Output :

33 6.1

20 6

23 5.6

30 5.5

25 5