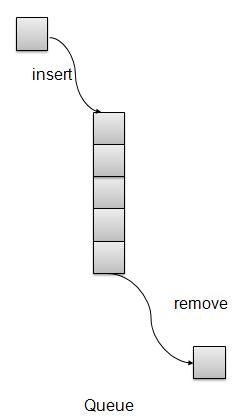
Priority Queue is more specialized data structure than Queue. Like ordinary queue, priority queue has same method but with a major difference. In Priority queue items are ordered by key value so that item with the lowest value of key is at front and item with the highest value of key is at rear or vice versa. So we're assigned priority to item based on its key value. Lower the value, higher the priority. Following are the principal methods of a Priority Queue.

**Basic Operations**

* **insert / enqueue** − add an item to the rear of the queue.
* **remove / dequeue** − remove an item from the front of the queue.

**Priority Queue Representation**

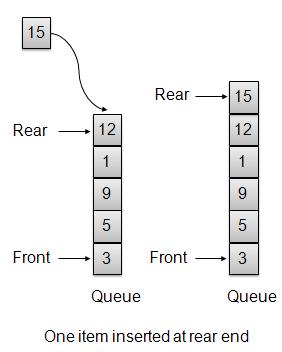


We're going to implement Queue using array in this article. There is few more operations supported by queue which are following.

* **Peek** − get the element at front of the queue.
* **isFull** − check if queue is full.
* **isEmpty** − check if queue is empty.

**Insert / Enqueue Operation**

Whenever an element is inserted into queue, priority queue inserts the item according to its order. Here we're assuming that data with high value has low priority.



void insert(int data){

int i = 0;

if(!isFull()){

// if queue is empty, insert the data

if(itemCount == 0){

intArray[itemCount++] = data;

}else{

// start from the right end of the queue

for(i = itemCount - 1; i >= 0; i-- ){

// if data is larger, shift existing item to right end

if(data > intArray[i]){

intArray[i+1] = intArray[i];

}else{

break;

}

}

// insert the data

intArray[i+1] = data;

itemCount++;

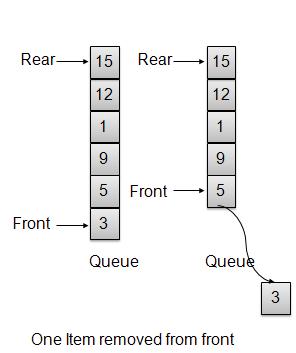
}

}

}

**Remove / Dequeue Operation**

Whenever an element is to be removed from queue, queue get the element using item count. Once element is removed. Item count is reduced by one.



int removeData(){

return intArray[--itemCount];

}

**Demo Program**

*PriorityQueueDemo.c*

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 6

int intArray[MAX];

int itemCount = 0;

int peek(){

return intArray[itemCount - 1];

}

bool isEmpty(){

return itemCount == 0;

}

bool isFull(){

return itemCount == MAX;

}

int size(){

return itemCount;

}

void insert(int data){

int i = 0;

if(!isFull()){

// if queue is empty, insert the data

if(itemCount == 0){

intArray[itemCount++] = data;

}else{

// start from the right end of the queue

for(i = itemCount - 1; i >= 0; i-- ){

// if data is larger, shift existing item to right end

if(data > intArray[i]){

intArray[i+1] = intArray[i];

}else{

break;

}

}

// insert the data

intArray[i+1] = data;

itemCount++;

}

}

}

int removeData(){

return intArray[--itemCount];

}

int main() {

/\* insert 5 items \*/

insert(3);

insert(5);

insert(9);

insert(1);

insert(12);

// ------------------

// index : 0 1 2 3 4

// ------------------

// queue : 12 9 5 3 1

insert(15);

// ---------------------

// index : 0 1 2 3 4 5

// ---------------------

// queue : 15 12 9 5 3 1

if(isFull()){

printf("Queue is full!\n");

}

// remove one item

int num = removeData();

printf("Element removed: %d\n",num);

// ---------------------

// index : 0 1 2 3 4

// ---------------------

// queue : 15 12 9 5 3

// insert more items

insert(16);

// ----------------------

// index : 0 1 2 3 4 5

// ----------------------

// queue : 16 15 12 9 5 3

// As queue is full, elements will not be inserted.

insert(17);

insert(18);

// ----------------------

// index : 0 1 2 3 4 5

// ----------------------

// queue : 16 15 12 9 5 3

printf("Element at front: %d\n",peek());

printf("----------------------\n");

printf("index : 5 4 3 2 1 0\n");

printf("----------------------\n");

printf("Queue: ");

while(!isEmpty()){

int n = removeData();

printf("%d ",n);

}

}

If we compile and run the above program then it would produce following result −

Queue is full!

Element removed: 1

Element at front: 3

----------------------

index : 5 4 3 2 1 0

----------------------

Queue: 3 5 9 12 15 16