A priority queue is an extension of a normal queue.

A priority queue has the following properties:

- Every element that is present in the queue has a **priority** associated with it.
- An element with **high priority is dequeued** before an element with low priority, no matter where it is present in the queue.
- If two elements have the **same priority**, the element that came first into the queue is dequeued first. (i.e We follow **FIFO principle**)

The queue is created so that the highest priority element is always at the beginning of the queue.
The queue is arranged in descending order of elements based on their priority.
This is done so that we can remove the highest priority element in O(1) time.
When inserting an element, the queue is traversed to find a proper position and then the element is inserted so that the overall order of the priority queue is maintained.

```
//Created By Ritwik Chandra Pandey on 3rd April 2021
//183215
//Priority Queue Using Linked Lists


#include <stdlib.h>
#include <stdio.h>

struct queue {
    int data;
    // Lower values indicate higher priority
    int priority;
    struct queue* next;
};
typedef struct queue* PQueue;
PQueue head=NULL;
PQueue newNode(int d, int p) {
    PQueue temp=(PQueue)malloc(sizeof(struct queue));
    temp->data = d;
    temp->priority = p;
    temp->next=NULL;
```

```c
    return temp;
}
void dequeue() {
    if(head==NULL) {
        printf("Priority queue is underflow.\n");
        return;
    }else{
        PQueue temp = head;
        head = head->next;
        printf("Deleted value = %d\n", temp->data);
        free(temp);
    }
}
void enqueue(int data, int priority) {
    PQueue start = head;
    PQueue temp = newNode(data, priority);
    if(temp==NULL) {
        printf("Priority queue is overflow. \n");
        return;
    }
    if(head==NULL) {
        temp->next=head;
        head=temp;
    }
    else if(start->next==NULL) {
        if(start -> priority <=temp->priority){
            start->next=temp;
        }else{
            temp->next= start;
            head=temp;
        }
    }
    else{
        if(priority<head->priority){
            temp->next=head;
            head=temp;
            return;
    }
        while(start->next!=NULL && start->next->priority<=priority){
            start=start->next;
        }
        temp->next=start->next;
        start->next=temp;
    }
}
void isEmpty(){
    if(head==NULL) {
        printf("Priority queue is empty. \n");
    }else{
        printf("Priority queue is not empty. \n");
    }
}
void display() {
    if(head==NULL) {
        printf("Priority queue is empty. \n");
    }else{
        PQueue temp=head;
        printf("Elements in the priority queue : ");
        while(temp!=NULL) {
            printf("%d (%d) ", temp->data, temp->priority);
            temp=temp->next;
        }
        printf("\n");
    }
}
void size() {
    int count=0;
    if(head==NULL) {
        printf("Priority queue size : 0\n");
        return;
    }else{
        PQueue temp=head;
        while(temp!=NULL) {
            count++;
            temp=temp->next;
        }
        printf("Priority queue size : %d\n", count);
    }
}
    int main() {
        int op, p, x;
        while(1) {
            printf("1. Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
            printf("Enter your option : ");
            scanf("%d",&op);
            switch(op) {
                case 1:
                    printf("Enter element : ");
                    scanf("%d",&x);
                    printf("Enter priority : ");
                    scanf("%d",&p);
                    enqueue(x,p);
                    break;
                case 2:
                    dequeue();
                    break;
                case 3:
                    display();
                    break;
                case 4:
                    isEmpty();
                    break;
                case 5:
                    size();
                    break;
                case 6: exit(0);
            }
        }
    }
```