Priority Queue - In a queue, elements are inserted from rear position & removed from front position. Thus FIFO ordering is strictly maintained during insertion & deletion operations. However sometimes, we need to violate (break) the FIFO principle & use intrinsic ordering among the elements to determine which element should be removed first. For e.g. rescheduling of jobs by Operating System. Here all the jobs are arranged in a queue & priority is assigned to each one of them. The job with maximum priority is executed first.

Definition - Priority Queue is a data structure in which some intrinsic ordering among the elements decides the result of it's basic operations.

There are 2 types of Priority Queues

1. Descending Priority Queue - Here the elements are added arbitrarily (randomly) but the largest element is removed first.

2. Ascending Priority Queue - Here the smallest element is removed first.

Implementation of Descending Priority Queue of int numbers

struct el

{

int num;

int flag;

};

int front, rear;

// member flag is used to determine whether a particular element is deleted or not. Initially value of flag can be 0 i.e. it is undeleted element. Here we are going to delete the element logically & not physically bcoz this queue is implemented using a static structure i.e. array. During deletion of a particular element, we will change value of flag of that element from 0 to 1.

#define SIZE 5

int main()

{

struct el pq[SIZE]; // array of structure

front = rear = -1; // queue is initialised

……

}

front is at 1st position & rear is at last position. Here after deletion, front doesn’t move at all like normal queue. To know that the element is deleted, change the value of it’s flag to 1.

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 45  ~~0~~ 1 | 20  0 | 90  ~~0~~ 1 | 35  0 | 10  0 |

num

flag

f r

In display( ) we will write a code which will display only those elements of pq[ ] in which the flag is 0.

Code to add elements

void addq(struct queue pq[ ],int no)

{

rear++;

pq[rear].num = no;

pq[rear].flag = 0; // undeleted element

// check whether it is 1st element

if (front == -1)

front = 0;

}

void displayq(struct queue pq[ ])

{

// display only undeleted elements

// i.e. elements whose flag is 0

int i;

for (i=front;i<=rear;i++)

{

if (pq[i].flag == 0)

printf(“%d “,pq[i].num);

}  
 }

i=0 i=1 i=2 i=3 i=4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 45  ~~0~~ 1 | 20  0 | 90  ~~0~~ 1 | 35  0 | 10  0 |

num

flag

f r

// To know the largest undeleted element. Before calling this fn, we are sure that the pq is

// not empty

int delq(struct queue pq[ ])

{

int i, large, pos;

// search begins with undeleted element only

for (i=front;i<=rear;i++)

{

if (pq[i].flag == 0)

{

large = pq[i].num; // 20

pos = i; // 1

break;

}

}

// continue the search further & check whether there is a larger element

for (i=pos+1;i<=rear;i++)

{

if (large < pq[i].num && pq[i].flag == 0)

{

large = pq[i].num; // 35

pos = i; // 3

}

}

// we have the largest undeleted element

pq[pos].flag = 1;

printf(“Element %d deleted successfully”,large);

return large;

}