4.5 Priority Queue

We have seen earlier that queue is a list of elements in which we can add the element only at one end called rear of the queue and delete the element only at the other end called front of the queue. In priority queue every element of queue has some priority and it is processed based on this priority. An element with higher priority will be processed before the element which has less priority. If two elements have same priority then in this case FIFO rule will follow, i.e. the element which comes first in the queue will be processed first. An example of priority queue is in CPU scheduling algorithm, in which CPU needs to process those jobs first which have higher priority.

There are two ways of implementing a priority queue-

(i) Through queue : In this case insertion is simple because the element is simply added at the rear end as usual. For performing deletion, first the element with highest priority is searched and then deleted.

(ii) Through sorted list : In this case insertion is costly because the element is inserted at the proper place in the list based on its priority. Here deletion is easy since the element with highest priority will always be in the beginning of the list.

In first case insertion is rapid but deletion is O(n) and in second case deletion is rapid but insertion is O(n). If the priority queue is implemented using arrays then in both the above cases, shifting of elements will be required. So it is advantageous to use a linked list, because insertion or deletion in between the linked list is more efficient. We will implement priority queue as a sorted linked list. The structure of the node would be-

```
struct node{
        int priority;
        int info;
        struct node *link;
};
```

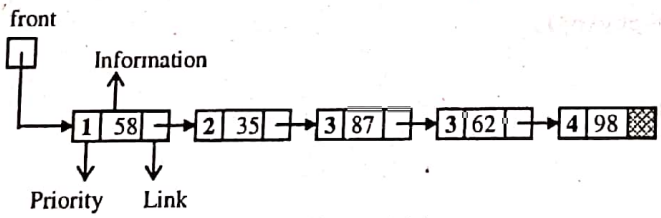
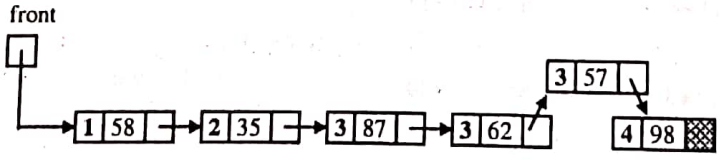


Figure 4.13

Here priority number 1 means the highest priority. If priority number of an element is 2 then it means it has priority more than the element which has priority number 3. Insertion of an element would be performed in a similar way as in sorted linked list. Here we insert the new element on the basis of priority of element.



Delete operation will be the deletion of first element of list because it has more priority than other elements of queue.

```
/*P4.8 Program of priority queue using linked list*/
#include<stdio.h>
#include<stdlib.h>
```

```
struct node
{
        int priority;
        int info;
        struct node *link;
}*front = NULL;
void insert(int item, int item_priority);
int del();
void display();
int isEmpty();

main()
{
       int choice,item,item_priority;
       while(1)
       {
              printf("1.Insert\n");
              printf("2.Delete\n");
              printf("3.Display\n");
              printf("4.Quit\n");
              printf("Enter your choice : ");
              scanf("%d", &choice);
              switch(choice)
              {
               case 1:
                     printf("Input the item to be added in the queue : ");
                     scanf("%d",&item);
                     printf("Enter its priority : ");
                     scanf("%d",&item_priority);
                     insert(item,item_priority);
                     break;
               case 2:
                     printf("Deleted item is %d\n",del());
                     break;
               case 3:
                     display();
                     break;
               case 4:
                     exit(1);
               default :
                     printf("Wrong choice\n");
              }/*End of switch*/
       }/*End of while*/
}/*End of main()*/
void insert(int item,int item_priority)
{
      struct node *tmp,*p;
      tmp = (struct node *)malloc(sizeof(struct node));
      if(tmp == NULL)
      {
            printf("Memory not available\n");
            return;
      }
      tmp->info = item;
      tmp->priority = item_priority;
      /*Queue is empty or item to be added has priority more than first
      element*/
      if(isEmpty() || item_priority < front->priority)
      {
            tmp->link = front;
            front = tmp;
      }
      else
      {
            p = front;
```

```
                while(p->link!=NULL && p->link->priority<=item_priority)
                        p = p->link;
                tmp->link = p->link;
                p->link = tmp;
        }
}/*End of insert()*/
int del()
{
        struct node *tmp;
        int item;
        if(isEmpty())
        {
                printf("Queue Underflow\n");
                exit(1);
        }
        else
        {
                tmp = front;
                item = tmp->info;
                front = front->link;
                free(tmp);
        }
        return item;
}/*End of del()*/
int isEmpty()
{
        if(front==NULL)
                return 1;
        else
                return 0;

}/*End of isEmpty()*/
void display()
{
        struct node *ptr;
        ptr = front;
        if(isEmpty())
                printf("Queue is empty\n");
        else
        {       printf("Queue is :\n");
                printf("Priority        Item\n");
                while(ptr!=NULL)
                {
                        printf("%5d        %5d\n",ptr->priority,ptr->info);
                        ptr = ptr->link;
                }
        }
}/*End of display() */
```

The priority queue that we have used is max-priority queue or descending priority queue. The other priority queue is min-priority or ascending priority queue in which the element with lowest priority is processed first. The best implementation of priority queue is through heap tree which we will study in the next chapter.