Concept of Queue

• **Definition**:

The queue can be formally defined as ordered collection of elements that has two ends named as front and rear. From the front end one can delete the elements and from the rear end one can insert the elements.

• **For Example :**

The typical example can be a queue of people who are waiting for a city bus at the bus stop. Any new person is joining at one end of the queue, you can call it as the rear end. When the bus arrives the person at the other end first enters in the bus. You can call it as the front end of the queue.

Following Fig. 3.1.1 represents the queue of few element

Fig. 3.1.1 Queue

Queue ADT

AbstractData Type Queue

{

Instances: The queue is a collection of elements in which the element can be inserted by one end called rear and elements can be deleted by other end called front.

Operations:

1. **Insert** The insertion of the element in the queue is done by the end called rear. Before the insertion of the element in the queue it is checked whether or not the queue is full.

2. **Delete**: The deletion of the element from the queue is done by the end called front. Before performing the delete operation it checked whether the queue is empty or not.

}

Queue Operations

• Queue is nothing but the collection of items. Both the ends of the queue are having their own functionality.

• The Queue is also called as FIFO i.e. a First In First Out data structure.

• All the elements in the queue are stored sequentially.

• Various operations on the queue are

1. Queue overflow.

2. Insertion of the element into the queue.

3. Queue underflow.

4.Deletion of the element from the queue.

5. Display of the queue.

Let us see each operation one by one

'C' representation of queue.

```
struct queue

{

int que [size];

int front;

int rear;

} Q;
```

1. Insertion of element into the queue

The insertion of any element in the queue will always take place from the rear end.

Fig. 3.3.1 Representing the insertion

Before performing insert operation you must check whether the queue is full or not. If the rear pointer is going beyond the maximum size of the queue then the queue overflow Occurs.

Fig. 3.3.2 Representing the queue overflow

2. Deletion of element from the queue

The deletion of any element in the queue takes place by the front end always.

Fig. 3.3.3 Representing the deletion

Before performing any delete operation one must check whether the queue is empty or not. If the queue is empty, you cannot perform the deletion. The result of illegal attempt to delete an element from the empty queue is called the queue underflow condition.

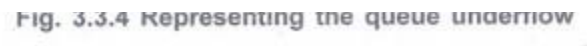


Fig. 3.3.4 Representing the queue underflow

Let us see the `C`mplementation of queue.

Ex. 3.3.1 Implementation of queue using arrays.

/*

Program for implementing the Queue using arrays

```
*****************************************
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define size 5
struct queue
{
int que[size];
int front,rear;
}Q;
/*
The Qfull Function
Input:none
Output:1 or 0 for q full or not
Called By:main
*/
Qfull()
{
if(Q.rear >=size-1)
return 1;
```

Note: If Queue exceeds the maximum size of the array then it returns 1 - means queue full is true otherwise 0 means queue full is false

else

return 0;

}

/*

The insert Function

Input:item which is to be inserted in the Q

Output:rear value

Called By:main

Calls:none

*/

int insert(int item)

{

if(Q.front == -1)

Q.front++;

Q.que[++Q.rear] = item;

Note:This condition will occur initially when queue is empty when single element will be present then both front and rear points to the same

return Q.rear;

```
}
int Qempty()
{
if((Q.front == -1) || (Q.front > Q.rear))
return 1;
else
return 0;
}
/*
The delet Function
Input:none
Output:front value
Called By:main
Calls:none
*/
int delet()
{
int item;
item = Q.que[Q.front];
Q.front++;
```

```
printf("\n The deleted item is %d",item);
return Q.front;
}
/*
The display Function
Input:none
Output:none
Called By:main
Calls:none
*/
void display()
{
int i;
for(i=Q.front;i<=Q.rear;i++)
printf("%d",Q.que[i]);
}
void main(void)
.{
int choice,item;
char ans;
```

```
clrscr();
Q.front = -1;
Q.rear = -1;
do
{
printf("\n Main Menu");
printf("\n1.Insert\n2.Delete\n3.Display");
printf("\n Enter Your Choice");
scanf("%d", &choice);
switch(choice)
{
case 1:if(Qfull()) //checking for Queue overflow
printf("\n Can not insert the element");
else
{
printf("\n Enter The number to be inserted");
scanf("%d",&item);
insert(item);
}
break;
```

```
case 2:if(Qempty())
printf("\n Queue Underflow!!");
else
delet();
break;
case 3:if(Qempty())
printf("\nQueue Is Empty!");
else
display();
break;
default:printf("\n Wrong choice!");
break;
}
printf("\n Do You Want to continue?");
ans = getche();
} while(ans == 'Y' || ans =='y');
}
/********************** End Of Program **********************
```

Output

Main Menu

1. Insert

2. Delete

3. Display

Enter Your Choice 1

Enter The number to be inserted 10

Do You Want to continue?y

Main Menu

1. Insert

2. Delete

3. Display

Enter Your Choice 1

Enter The number to be inserted 20

Do You Want to continue?y

Main Menu

1. Insert

2. Delete

3. Display

Enter Your Choice 3

10 20

Do You Want to continue?y

Main Menu

1. Insert

2. Delete

3. Display

Enter Your Choice 2

The deleted item is 10

Do You Want to continue?y

Main Menu

1. Insert

2. Delete

3. Display

Enter Your Choice 3

20

Do You Want to continue?n

Key Point Always check queue full before insert operation and queue empty before delete operation.

Ex. 3.3.2: Write a C program to implement queue functions using Arrays and Macros.

Solution :

#include<stdio.h>

```
#include<stdlib.h>
#include<conio.h>
/* Macro definitions */
#define size 5
#define Isfull Q.rear >=size-1
#define Isempty (Q.front == -1) || (Q.front > Q.rear)
struct queue
{
int que[size]; //Queue using Arrays
int front,rear;
}Q;
Qfull()
{
if(Isfull)
return 1;
else
return 0;
}
int insert(int item)
{
```

```
if(Q.front==-1)
Q.front++;
Q.que[++Q.rear] = item;
return Q.rear;
}
int Qempty()
{
if(Isempty)
return 1;
else
return 0;
}
int delet()
{
int item;
item = Q.que[Q.front];
Q.front++;
printf("\n The deleted item is %d",item);
return Q.front;
}
```

```
void display()
{
int i;
for(i=Q.front;i<=Q.rear;i++)
printf("%d",Q.que[i]);
}
void main(void)
{
int choice,item;
char ans;
Q.front = -1;
Q.rear = -1;
do
{
printf("\n Main Menu");
printf("\n1.Insert\n2.Delete\n3.Display");
printf("\n Enter Your Choice");
scanf("%d", &choice);
switch(choice)
{
```

```
case 1:if(Qfull()) //checking for Queue overflow
printf("\n Can not insert the element");
else
{
printf("\n Enter The number to be inserted");
scanf("%d", &item);
insert(item);
}
break;
case 2:if(Qempty())
printf("\n Queue Underflow!!");
else
delet();
break;
case 3:if(Qempty())
printf("\nQueue Is Empty!");
else
display();
break;
default:printf("\n Wrong choice!");
```

```
break;
}
printf("\n Do You Want to continue?");
ans = getche();
} while(ans =='Y' || ans =='y');
}
```

Review Question

1. Develop an algorithm to implement Queue ADT. Give relevant examples and diagrammatic representations.

Implementation of Queue using Linked List

• The main advantage of using linked representation of queues is that there is no limit on the size of the queue. We can insert as many elements as we want in the queue by creating the required number of nodes. If the elements are no longer needed then those the nodes can be removed from the queue.

• The linked representation of the queue is as shown below -

Fig. 3.4.1

The typical node structure will be

```
typedef struct node
{
int data;
struct node next;
```

}Q;

Various operations that can be performed on queue are,

1. Insertion of a node in queue.

2. Deletion of node from the queue.

3. Checking whether queue is empty or not.

4. Display of a queue.

All these operations are implemented in following C program.

'C' program

```
/*****************************************************************

Program For implementing the Queue using the linked list. The queue full condition will never occur in this program.

*****************************************************************/

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

/*Declaration of Linked Queue data structure*/

typedef struct node

{

int data;

struct node *next;
```

```
}Q;
Q *front, *rear;
void main(void)
{
char ans;
int choice;
void insert();
Q *delet();
void display(Q *);
front=NULL;
rear= NULL;
do
{
printf("\n\tProgram For Queue Using Linked List\n");
printf("\n Main Menu")
printf("\n1.Insert\n2.Delete \n3.Display");
printf("\n Enter Your Choice");
scanf("%d", &choice);
switch(choice)
{
```

```
case 1: insert();
break;
case 2:front = delet();
break;
case 3 display(front);
break;
default: printf("\nYou have entered Wrong Choice\n");
break;
}
printf("\nDo you want to continue?\n");
flushall();
ans = getch();
} while(ans == 'y' || ans == 'Y');
getch();
clrscr();
}
Q *get_node(Q *temp)
{
temp =(Q*)malloc(sizeof(Q));
temp->next=NULL;
```

```
return temp;
}
void insert()
{
char ch;
Q *temp;
clrscr();
temp = get_node(temp); /* allocates memory for temp node*/
printf("\n\n\n\tInsert the element in the Queue\n");
scanf("%d", &temp->data);
if(front == NULL)/*creating first node*/
{
front= temp;
rear=temp;
else /* attaching other nodes*/
{
rear->next=temp; .
rear = rear->next;
}
}
```

```
int Qempty(Q *front)
{
if(front== NULL)/*front is NULL means memory is not allocated to queue*
return 1;
else
return 0;
}
Q *delet()
{
Q *temp;
temp=front;
if(Qempty(front))
{
```

Note; Qempty (front) function returns 1 (true). Hence queue is empty.

```
printf("\n\n\t\tSorry!The Queue Is Empty\n");
printf("\n Can not delete the element");
}
else
{
printf("\n\tThe deleted Element Is %d ",temp->data);
```

```
front front->next;/*deleting the front node*/

temp->next = NULL;

free(temp);

}
```

Note; After delete operation, front is changed. Hence updated front pointer has to be returned from delete function.

```
return front;

}

void display(Q *front)

{

if(Qempty(front))

printf("\n The Queue Is Empty\n");

else

{

printf("\n\t The Display Of Queue Is \n"); /*queue is displayed from front to rear*/

for(;front !=rear->next;front=front->next)

printf("\t%d ",front->data);

}

getch();

}
```

Output

Program For Queue Using Linked List

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice1

Insert the element in the Queue

10

Do you want to continue?

Program For Queue Using Linked List

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice1

Insert the element in the Queue 20

Do you want to continue?

Program For Queue Using Linked List

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice1

Insert the element in the Queue 30

Do you want to continue?

Program For Queue Using Linked List

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice3

The Display Of Queue Is

10 20 30

Do you want to continue?

Program For Queue Using Linked List

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice2

The deleted Element Is 10

Do you want to continue?

Program For Queue Using Linked List

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice3

The Display Of Queue Is

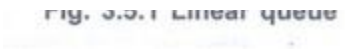
20 30

Do you want to continue?

Circular Queue

Definition Circular queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle.

As we have seen, in case of linear queue the elements get deleted logically. This can be shown by following Fig. 3.5.1.



We have deleted the elements 10, 20 and 30 means simply the front pointer is shifted ahead. We will consider a queue from front to rear always. And now if we try to insert any more element then it won't be possible as it is going to give "queue full !" message. Although there is a space of elements 10, 20 and 30 (these are deleted

elements), we can not utilize them because queue is nothing but a linear array ! Hence there is a concept called circular queue. The main advantage of circular queue is we can utilize the space of the queue fully. The circular queue is shown by following Fig. 3.5.2.

Considering that the elements deleted are 10, 20 and 30.

Fig. 3.5.2 Circular queue

There is a formula which has to be applied for setting the front and rear pointers, for a circular queue.

Rear =(rear + 1)% size

Front = (front + 1) % size

rear = (rear + 1)% size = (4 + 1) % 5

rear =0

So we can store the element 60 at 0th location similarly while deleting the element.

front = (front + 1)% size = (3 + 1) % 5

front= 4

So delete the element at 4" location i.e. element 50

Let us see the 'C' program now,

Ex. 3.5.1: Implementation of circular queue using arrays.

Sol. :

/**********

Program For circular queue using arrays. It performs the basic operations such as insertion, deletion of the elements by taking care of queue Full and queue Empty conditions. The appropriate display is for displaying the queue

**********/

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define size 5

int queue[size];

int front=-1; /* queue initialization by front = -1 and rear = 0*/

int rear =0;

/*

The qFull Function

Input:none

Output:returns 1 or 0 for q.full or not

Called By:add

Calls:none

*/

int qFull()

{
```

```
if(front==(rear+1)%size)
return 1;
else
return 0;
}
/*
The qEmpty Function
Input:none
Output:returns 1 or 0 for q.empty or not
Called By:delete and display
Calls:none
*/
int qEmpty()
{
if(front ==-1)
return 1;
else
return 0;
}
/*
```

The add Function

Input:the element to be inserted

Output:none

Called By:main

Calls:qFull

*/

```
void add(int Item)
{
if(qFull()) /* qFull() returns true value */
printf("\n The circular Queue is full!");
 else
{
if(front ===-1)/* only one element in the queue */
front=rear=0;
else
rear = (rear+1)%size;
queue [rear] = Item;
}
}
```

/*

```
the delet Function

Input:none

Output:returns the deleted element

Called By:main

Calls:qEmpty

*/

void delet()

{

int Item;

if(qEmpty())

printf("\n Queue Is Empty!");

else

{

Item-queue[front];

if(front == rear)

{

front=rear=-1;

}

else

front (front+1)%size;
```

```
printf("\n The deleted item is %d",Item);
}
}
/*
The display Function
Input: none
Output: prints the contents of the queue
Called By:main
Calls:qEmpty
*/
void display()
{
int i;
if(qEmpty())
{
printf("\nThe Queue Is Empty");
return;
}
i=front;
while(i!=rear)
```

```
{
printf("%d", queue[i]);
i=(i+1)%size;
}
printf("%d", queue[i]);
}
/*
The main Function
Input:none
Output:none
Called By:O.S.
Calls:add, delete, display
*/
void main(void)
{
int choice,Item;
char ans;
clrscr();
do
{
```

```
printf("\n Main Menu");
printf("\n 1.Insert\n2.Delete\n3.Display");
printf("\n Enter Your Choice");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("\nEnter The element");
scanf("%d", &Item);
add(Item);
break;
case 2: delet();
break;
case 3: display();
break;
default:exit(0);
}
printf("\n Do u want to continue?");
ans = getch();
} while(ans == 'Y' || ans == 'y');
getch();
```

}

********** End Of Main **********

Advantages of Circular Queue over Linear Queue

(1) Efficient utilization of memory: There is no wastage of memory in circular queue as it uses the unoccupies space and memory is used properly in an effective manner. Thus, no space is left over.

(**2) Flexible insertion and deletion operations**: In the circular queue, if the queue is not fully occupied, then the elements can be inserted easily on the vacant locations. But in the case of a linear queue, insertion is not possible once the rear pointer reaches the last index even if there are empty locations present in the queue.

Difference between Linear Queue and Circular Queue

4.	It requires more memory space.	It requires less memory space.

Review Questions

1. Write a routine to implement the circular queue using array.

2. Write an algorithm to convert the infix expression to postfix expression.

3. What are the circular queues. Write the procedure to insert an element to circular queue and delete an element from a circular queue using array implementation.

Priority Queue

• **Definition**: The priority queue is a data structure having a collection of elements which are associated with specific ordering.

• There are two types of priority queues -

1. Ascending priority queue

2. Descending priority queue.

Application of Priority Queue

1. The typical example of priority queue is scheduling the jobs in operating system. Typically operating system allocates priority to jobs. The jobs are placed in the queue and position of the job in priority queue determines their priority. In operating system there are three kinds of jobs. These are real time jobs, foreground jobs and background jobs. The operating system always schedules the real time jobs first. If there is no real time job pending then it schedules foreground jobs. Lastly if no real time or foreground jobs are pending then operating system schedules the background jobs.

2. In network communication, to manage limited bandwidth for transmission the priority queue is used.

3.In simulation modelling, to manage the discrete events the priority queue is used.

Types of Priority Queue

The elements in the priority queue have specific ordering. There are two types of priority queues -

1. Ascending Priority Queue: An ascending order priority queue gives the highest priority to the lower number in that queue.

Fig. 3.6.1 Ascending priority queue

2. Descending Priority Queue: A descending order priority queue gives the highest priority to the highest number in that queue.

Fig. 3.6.2 Descending priority queue

In priority queue, the elements are arranged in any order and out of which only the smallest or largest element allowed to delete each time.

The implementation of priority queue can be done using arrays or linked list. The data structure heap is used to implement the priority queue effectively.

ADT for Priority Queue

Various operations that can be performed on priority queue are

1. Insertion

2. Deletion

3. Display

Hence the ADT for priority queue is given as below

Instances:

P_que[MAX] is a finite collection of elements associated with some priority.

Precondition :

The front and rear should be within the maximum size MAX.

Before insertion operation, whether the queue is full or not is checked.

Before any deletion operation, whether the queue is empty or not is checked.

Operations :

1. create() - The queue is created by declaring the data structure for it.

2. insert() - The element can be inserted in the queue.

3. delet() - If the priority queue is ascending priority queue then only smallest element is deleted each time. And if the priority queue is descending priority queue then only largest element is deleted each time.

4. display() - The elements of queue are displayed from front to rear.

The implementation of priority queue using arrays is as given below -

1. Insertion operation

While implementing the priority queue we will apply a simple logic. That is while inserting the element we will insert the element in the array at the proper position. For example, if the elements are placed in the queue as

The C function for this operation is as given below -

```
int insert(int que[SIZE], int rear,int front)

int item,j;

printf("\nEnter the element: ");

scanf("%d",&item);

if(front ==-1)

front++;

j=rear;

while(j>=0 && item<que[j])

{

quelj+1]=que[j];
```

```
j--;
}
}
quelj+1]=item;
rear rear+1;
return rear;
}
```

2. Deletion operation

In the deletion operation we are simply removing the element at the front.

and then new front will be que[1].

The deletion operation in C is as given below -

```
int delet(int que[SIZE], int front)
{
int item;
item=que[front];
printf("\n The item deleted is %d",item);
front++;
return front;
}
```

The complete implementation of priority queue is as given below -

Ex. 3.6.1 Implementation of Priority Queue

Sol. :

```
/*************************************************

Program for implementing the ascending priority Queue

*************************************************/

/*Header Files*/

#include<stdio.h>

#include<conio.h>

#define SIZE 5

void main(void)

{

int rear,front,que [SIZE], choice;

int Qfull(int rear), Qempty(int rear,int front);

int insert(int que[SIZE], int rear,int front);

int delet(int que [SIZE], int front);

void display(int que[SIZE], int rear,int front);

char ans;

clrscr();

front=0;
```

```
rear=-1;
do
{
clrscr();
printf("\n\t\t Priority Queue\n");
printf("\n Main Menu");
printf("\n1.Insert\n2.Delete\n3.Display");
printf("\n Enter Your Choice: ");
scanf("%d", &choice);
switch(choice)
{
case 1:if(Qfull(rear))
printf("\n Queue IS full");
else
rear=\insert(que,rear, front);
break;
case 2:if(Qempty(rear,front))
printf("\n Cannot delete element");
else
front=delet(que, front);
```

```
break;
case 3:if(Qempty(rear,front))
printf("\n Queue is empty");
else
display(que,rear, front);
break;
default:printf("\n Wrong choice");
break;
}
printf("\n Do You Want To continue?");
ans =getche();
} while(ans == 'Y' || ans =='y');
getch();
}
int insert(int que[SIZE], int rear,int front)
{
int item,j;
printf("\nEnter the element: ");
scanf("%d", &item);
if(front ==-1)
```

```
front++;
j=rear;
while(j>=0 && item<que [j])
{
que[j+1]=que[j];
j- -;
}
que[j+1]=item;
rear = rear+ 1;
return rear;
}
int Qfull(int rear)
{
if(rear==SIZE-1)
return 1;
else
return 0;
}
int delet(int que[SIZE], int front)
{
```

```
int item;
item=que[front];
printf("\n The item deleted is %d",item);
front++;
return front;
}
Qempty(int rear,int front)
{
}
if((front==-1)||(front>rear))
return 1;
else
return 0;
}
void display(int que[SIZE], int rear,int front)
{
int i;
printf("\n The queue is:");
for(i=front;i<=rear;i++)
printf("%d",que[i]);
```

}

Output

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 1

Enter the element: 30

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 1

Enter the element: 10

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 1

Enter the element: 20

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 1

Enter the element: 50

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 1

Enter the element: 40

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 3

The queue is: 10 20 30 40 50

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 2

The item deleted is 10

Do You Want TO continue?

Main Menu

1.Insert

2.Delete

3.Display

Priority Queue

Enter Your Choice: 2

The item deleted is 20

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 2

The item deleted is 30

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 2

The item deleted is 40

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 2

The item deleted is 50

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 2

Cannot delete element

Do You Want TO continue?

Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice: 3

Queue is empty

Do You Want TO continue?n

Ex. 3.6.2: Write program for implementing the linked priority queue.

Solution :

```
#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

/*Declaration of Linked Queue data structure*/

typedef struct node

{

int data;

struct node *next;

}Q;

/*

Q *front,*rear;

The main Function
```

```
Calls:insert, delet, display

Called By:O.S.

*/

void main(void)

{

char ans;

int choice;

void insert();

Q *delet();

void display(Q *);

front= NULL;

rear= NULL;

do

{

printf("\n\tProgram For Linked priority Queue\n");

printf("\n Main Menu");

printf("\n1.Insert\n2.Delete \n3.Display");

printf("\n Enter Your Choice");

scanf("%d", &choice);

switch(choice)
```

```
{
case 1 : insert();
break;
case 2 : front = delet();
break;
case 3 :display(front);
break;
default : printf("\nYou have entered Wrong Choice\n");
break;
}
printf("\nDo you want to continue?\n");
flushall();
ans = getch();
} while(ans == 'y' || ans == 'Y');
getch();
clrscr();
}
/*
The get_node function
Input:temp node
```

```
Output:memory allocated temp node

Called By:insert

Calls:none

*/

Q *get_node(Q *temp)

{

temp =(Q*) malloc(sizeof(Q));

temp->next = NULL;

return temp;

}

/*

The insert Function

Called By:main

Calls:get_node

*/

void insert()

{

char ch;

Q *temp, *current,*prev;

clrscr();
```

```
temp =get_node(temp); /*allocates memory for temp node*/
printf("\n\n\n\tInsert the element in the Queue\n");
scanf("%d",&temp->data);
/*The new node which is to be inserted is temp */
if(front == NULL)/*creating first node*/
{
front= temp;
rear=temp;
}
else/*first node is created and next node is to be attached*/
{
current=front;
/* current node is to be compared with every node from
beginnning*/
prev=current;
/*if temp node is attached as a first node to remaining list*/
if(current->data > temp->data)
{
temp->next=current;
front=temp;/*update front*/
```

```
}
/*if temp node is to be inserted in between*/
else
{
while((current->data<temp->data)&&(current!= NULL))
{
/*prev is previous to current node*/
prev=current;
current=current->next;
}/*end of while*/
/*if temp->data>current->data and current node!= NULL*/
if(current)
{
prev->next=temp;
temp->next = current;
}
/*attaching temp as a last node when temp has greaest among all the nodes in the list and we have reached at the end of the list*/
else
{
```

```
prev->next=temp;
rear=temp;/*as temp is appended, rear is
changed,temp becomes rear*/
}
}
}/*closing the outermost else*/
}
/*
The delet Function
Called By:main
Calls:Qempty
/*
Q *delet()
{
Q *temp;
int Qempty(Q *front);
temp=front;
if(Qempty(front))
{
printf("\n\n\t\tSorry!The Queue Is Empty\n");
```

```
printf("\n Can not delete the element");
}
else
{
printf("\n\tThe deleted Element Is %d ",temp->data);
front front->next;/*deleting the front node */
temp->next = NULL;
free(temp);
}
return front;
}
/*
The QEmpty Function
Called By:delet, display
Input:front pointer
Output:If Q empty then returns 1
otherwise returns 0
Calls:none
*/
int Qempty(Q *front)
```

```
{
if(front== NULL)
return 1;
else
return 0;
}
/*
The display Function
Input: front pointer
Called By:main
Calls:QEmpty
*/
void display(Q *front)
{
if(Qempty(front))
printf("\n The Queue Is Empty\n");
else
{
printf("\n\t The Display Of Queue Is \n");
/*queue is displayed from front to rear*/
```

```
for(;front !=rear->next;front=front->next)

printf("\t%d ",front->data);

}

getch();

}
```

Output

```
Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice 1

Insert the element in the Queue

9

Do you want to continue?

Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display
```

Enter Your Choice 1

Insert the element in the Queue

11

Do you want to continue?

Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice 1

Insert the element in the Queue

15

Do you want to continue?

Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice 1

Insert the element in the Queue

7

Do you want to continue?

Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice 1

Insert the element in the Queue

12

Do you want to continue?

Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice 3

The Display Of Queue Is

7 9 11 12 15

Do you want to continue?

Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice 2

The deleted Element Is 7

Do you want to continue?

Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice 2

The deleted Element Is 9

Do you want to continue?

Program For Linked Priority Queue

Main Menu

1.Insert

2.Delete

3.Display

Enter Your Choice 3

The Display Of Queue Is

11 12 15

Do you want to continue?

Logic Explanation of Linked Priority Queue

Refer above program while understanding this section.

1. Insertion of a node

Initially queue will be empty and if we want to insert the first node. Then first memory will get allocated for temp node as :

Then we will start scanning the queue from beginning because we want to insert node with '11' at proper place.

Now if we want to insert temp node with value '7' in the linked queue then we will find the position for temp node in linked queue.

Fig. 3.6.4

Continuing in this fashion we create a linked list in an ascending order.

2. Deletion of a node

The deletion operation is simple. We always delete a node which is at front.

Consider that the priority queue is

Fig. 3.6.5

Review **Question**

1. Implement a priority queue using linked list.

Dequeue

• **Definition**: Doubly ended queue is a queue in which insertion and deletion both can be done by both the ends.

Fig. 3.7.1 Doubly ended queue

• As we know, normally we insert the elements by rear end and delete the elements from front end. Let us say we have inserted the elements 10, 20, 30 by rear end.

• Now if we wish to insert any element from front end then first we have to shift all the elements to the right.

For example if we want to insert 40 by front end then, the deque will be

Fig. 3.7.2 Operations on deque

We can place -1 for the element which has to be deleted.

Let us see the implementation of deque using arrays

Ex. 3.7.1: Implementation of deque using arrays.

/**********

Program To implement Doubly ended queue using arrays

**********/

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 5
/*Data structure for deque*/
struct queue {
int que[size];
int front,rear;
}Q;
int Qfull()
{
if(Q.rear==size-1)
return 1;
else
return 0;
}
int Qempty()
{
```

```
if((Q.front>Q.rear) || (Q.front ==-1&&Q.rear==-1))
return 1;
else
return 0;
}
int insert_rear(int item)
{
if(Q.front=-1&&Q.rear==-1)
Q.front++;
Q.que[++Q.rear]=item;
return Q.rear;
}
int delete_front()
{
int item;
if(Q.front==-1)
Q.front++;
item=Q.que[Q.front];
Q.que [Q.front]=-1;
Q.front++;
```

```
return item;
}
int insert_front(int item)
{
int i,j;
if(Q.front==-1)
Q.front++;
j=Q.rear;
while(j>=Q.front)
{
Q.quelj+1]=Q.que[j];
j- -;
}
Q.rear++;
Q.que[Q.front]=item;
return Q.front;
}
int delete_rear()
{
int item;
```

```
item=Q.que [Q.rear];
Q.que[Q.rear] =-1;/*logical deletion*/
Q.rear--;
return item;
}
void display()
{
int i;
printf("\n Straight Queue is:");
for(i=Q.front;i<=Q.rear;i++)
printf("%d",Q.que[i]);
}
void main()
{
int choice,i,item;
char ans;
ans='y';
Q.front=-1;
Q.rear=-1;
for(i=0;i<size;i++)
```

```
Q.que[i]=-1;
clrscr();
 printf("\n\t\t Program For doubly ended queue using arrays");
do
{
printf("\n1.insert by rear\n2.delete by front\n3.insert by front\n4.delete by rear");
printf("\n5.display\n6.exit");
printf("\n Enter Your choice");
scanf("%d", &choice);
switch(choice)
{
case 1:if(Qfull())
printf("\n Doubly ended Queue is full");
else
{
printf("\n Enter The item to be inserted");
scanf("%d",&item);
Q.rear-insert_rear(item);
}
break;
```

```
case 2:if(Qempty())
printf("\n Doubly ended Queue is Empty");
else
{
item=delete_front();
printf("\n The item deleted from queue is %d",item);
}
break;
case 3:if(Qfull())
printf("\n Doubly ended Queue is full");
else
{
printf("\n Enter The item to be inserted");
scanf("%d", &item);
Q.front insert_front(item);
}
break;
case 4:if(Qempty())
printf("\n Doubly ended Queue is Empty");
else
```

```
{
item=delete_rear();
printf("\n The item deleted from queue is %d",item);
}
break;
case 5:display();
break;
case 6:exit(0);
}
printf("\n Do You Want To Continue?");
ans=getch();
}while(ans ==`y`|| ans == `Y`);
getch();
}
```

Output

Program For doubly ended queue using arrays

1. insert by rear

2. delete by front

3. insert by front

4. delete by rear

5. display

6. exit

Enter Your choice1

Enter The item to be inserted10

Do You Want To Continue?

1. insert by rear

2. delete by front

3. insert by front

4. delete by rear

5. display

6. exit

Enter Your choice 3

Enter The item to be inserted 20.

Do You Want To Continue?

1. insert by rear

2. delete by front

3. insert by front

4. delete by rear

5. display

6. exit

Enter Your choice 5

Straight Queue is: 20 10

Do You Want To Continue?

1. insert by rear

2. delete by front

3. insert by front

4. delete by rear

5. display

6. exit

Enter Your choice4

The item deleted from queue is 10

Do You Want To Continue?

Difference between Doubly End Queue and Circular Queues

Review **Questions**

1. What is a circular queue and double-ended queue? Give suitable examples to differentiate them.

2. Differentiate between double ended queue and Circular queue.

3. What is a DeQueue? Explain its operation with example.

Applications of Queue

There are various applications of queues such as,

1. Job Scheduling

• In the operating system various programs are getting executed.

• We will call these programs as jobs. In this process, some programs are in executing state.

• The state of these programs is called as 'running' state.

• Some programs which are not executing but they are in a position to get executed at any time such programs are in the 'ready' state.

• And there are certain programs which are neither in running state nor in ready state. Such programs are in a state called as 'blocked' state.

• The operating system maintains a queue of all such running state, ready state, blocked state programs. Thus use of queues help the operating system to schedule the jobs.

• The jobs which are in running state are removed after complete execution of each job, then the jobs which are in ready state change their state from ready to running and get entered in the queue for running state.

• Similarly the jobs which are in blocked state can change their state from blocked to ready state.

• These jobs then can be entered in the queue for ready state jobs.

• Thus every job changes its state and finally get executed. Refer Fig. 3.8.1.

• Thus queues are effectively used in the operating system for scheduling the jobs.

Fig. 3.8.1 Job scheduling in operating system

2. Categorizing Data

• The queue can be used to categorize the data.

• The typical example for categorizing the data is our college library.

• The college can have several departments such as Computer Engineering, Information Technology, Mechanical Engineering, Electronics Engineering and so on. The library has various sections in which the books from each stream are arranged.

• These sections are like multiple queues in which appropriate data (books) are stored. Thus categorization of data can be possible using multiple queues.

Ex. 3.8.1: There are 'N' numbers of balls in the box. The colors of the balls are red and blue. You are requested to stack the balls in the bottom sealed basket one by one. The order of placing the balls is two consecutive red balls followed by two consecutive blue balls. Later create two empty queues Q1 and Q2. Remove the last inserted balls from the basket and place it in Q1. Similarly remove the next inserted balls from the basket and insert it in Q2. Develop a program to repeat this process until the basket is empty and also print the color of the balls in both queues.

Solution :

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define size 25
```

```
/* stack structure*/
struct stack
{
char s[size];
int top;
}st;
struct queue
{
char que[size];
int front,rear;
}Q1,02;
int stfull()
{
if(st.top>=size - 1)
return 1;
else
return 0;
}
void push()
{
```

```
for(int i=0;i<2;i++)
{
st.top++;
st.s[st.top] ='r';
}
for(int i=0;i<2;i++)
{
st.top++;
st.s[st.top] ='b';
}
printf("\n Ball is pushed");
}
int stempty()
{
If(st.top== -1)
return 1;
else
return 0;
}
char pop()
```

```
{
char item;
item=st.s[st.top];
st.top--;
printf("\n Ball is popped");
return(item);
}
void insert_Queue1(char item)
{
printf("\n Inserting ball in Q1");
if(Q1.front == - 1).
Q1.front++;
Q1.que[++Q1.rear] = item;
}
void insert_Queue2(char item)
{
printf("\n Inserting ball in Q2");
if(Q2.front == -1)
Q2.front++;
Q2.que[++Q2.rear] = item;
```

```
}
void display_queue(struct queue Q)
{
int i;
for(i=Q.front;i<=Q.rear;i++)
printf("%c",Q.que[i]);
}
void display_stack()
{
int i;
if(stempty())
printf("\n Stack Is Empty!");
else
{
for(i=st.top;i>=0;i--)
printf("\n%c", st.s[i]);
}
}
int main(void)
{
```

```
int choice;
char ans,item;
st.top = -1;
Q1.front = - 1;
Q1.rear = -1;
Q2.front= - 1;
Q2.rear = -1;
do
{
printf("\n Main Menu");
printf("\n1.Push the balls\n2.Pop\n3.exit");
printf("\n Enter Your Choice: ");
scanf(“%d”,&choice)
switch(choice)
{
case 1:if(stfull())
printf("\n Stack is Full");
else
{
push();
```

```
display_stack();
}
break;
case 2:if(stempty())
printf("\n Empty stack!Underflow !!");
else
{
while(!stempty())
{
item=pop();
insert_Queue1(item);
item=pop();
insert_Queue2(item);
printf("\n Displaying Queue1...\n");
display_queue(Q1);
printf("\n Displaying Queue2...\n");
display_queue(Q2);
printf("\n---------------------------\n”);
}
}
```

```
display_stack();
break;
case 3:exit(0);
}
printf("\n Do You want To Continue? ");
ans=getche();
} while(ans =='Y' || ans =='y');
getch();
return 0;
}
```

Output

```
Main Menu
1.Push the balls
2.Pop
3.exit
Enter Your Choice: 1
Balls are Pushed !!!
b
b
r
```

r

Do You Want To Continue? y

Main Menu

1.Push the balls

2.Pop

3.exit

Enter Your Choice: 2

Ball is popped

Inserting ball in Q1

Ball is popped

Inserting ball in Q2

Displaying Queue1...

b

Displaying Queue2...

Ball is popped

Inserting ball in Q1

Ball is popped

dried od Inserting ball in Q2

Displaying Queue1...

b r

Displaying Queue2...

b r

Stack Is Empty!

Two Marks Questions with Answers

Q.1 What do the terms LIFO and FIFO means? Explain.

Ans. : The LIFO stands for Last In First Out. This property belongs to stack. That means the element inserted lastly will be the element to be popped off first from the stack.

The FIFO stands for First In First Out. This property belongs to queue. That means the element inserted first in the queue will be the first to get deleted.

Q.2 What are the front and rear pointers of queue.

Ans. : The front end of the queue from which the element gets deleted is called front and the end from which the element is inserted in the queue is called rear.

For example: Refer Fig.

Fig. 3.3.3 Representing the deletion

Q.3 What is a circular queue?

Ans. : Circular queue is a queue in which the front end is just adjacent to the rear end. The elements get arranged in circular manner. For example - Refer Fig.

Fig. 3.5.2 Circular queue

Q.4 Write any four applications of queues.

Ans;

1. In operating system for scheduling jobs, priority queues are used.

2. In Local Area Network (LAN) multiple computers share few printers. Then these printers accumulate jobs in a queue. Thus printers can process multiple print commands with the help of queues.

3. For categorizing data, queues are used.

4. In simulation and modeling queues are used.

5. In computer networks, while broadcasting message, the message packets are accumulated in queue. And then these packets are forwarded.

Q.5 What is a dequeue?

Ans. : The dequeue is a data structure in which the element can be inserted from both the ends i.e. front and rear. Similarly, the element can be deleted from both the front and rear end.

Q.6 How do you test for an empty queue?

Ans. Following is a C code which is used to test an empty queue

```
if((Q.front==-1) | | (Q.front>Q.rear))

printf("\n The Queue is Empty!!");

else

printf("\n The Queue is not Empty!!")
```

Q.7 What is the use of queue in operating system?

Ans. : The queue is used for scheduling the jobs in operating system.

Q.8What is the need for circular queue ?

Ans.

The circular queue is an useful data structure in which all the cells of the queue can be utilized. Secondly due to circular queue one can traverse to the first node from the last node very efficiently.

Q.9 What is stack and Queue ?

Ans. : Stack is linear data structure in which insertion and deletion of element is from one end called top.

Queue is a linear data structure in which insertion of element is from one end called rear and deletion of element is from other end called front.

Q.10 What are priority queues? What are the ways to implement priority queue ?

Ans.

Definition: The priority queue is a data structure having a collection of elements which are associated with specific ordering.

•Ways to implement Priority Queue:

There are two ways to implement priority queue.

1. Ascending Priority Queue - It is a collection of items in which the items can be inserted arbitarily but only smallest element can be removed.

2. Descending Priority Queue - It is a collection of items in which insertion of items can be in any order but only largest element can be removed.

Q.11 A priority queue is impletemented as Max heap. Initially it has 5 elements. The level order traversal of heap is: 10, 8, 5, 3, 2. Two new elements 11 and 7 are inserted into the heap in that order. Give level order traversal of the heap after the insertion of elements.

Ans. Step 1: The heap is as shown below.

Step 2: The level order after insertion of given elements is 11, 8, 10, 3, 2, 7.