**Experiment No. : 4**

**Title: Implementation of Priority Queue**

**Batch: A1 Roll No.: 16010420026 Experiment No.: 4**

**Aim:** Write a menu driven program to implement a static priority queue using supporting following operations.

1. Create empty queue,

2. Insert an element on the queue,

3. Delete an element from the queue,

4. Display front element

5. Display all elements of the queue.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Resources Used:** Turbo C/ C++/JAVA editor and compiler.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Theory:**

**Queue -**A queue is an ordered list in which insertion and deletion happens two different ends.The insertion happens from the *rear* and the deletion takes place at the *front*. It works with the FIFO concept i.e. first in first out. Basic operations of Queue are enqueue, dequeue, isempty, etc.

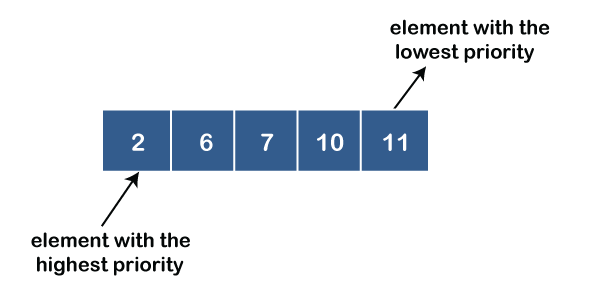
**Priority Queue** : A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue. The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue. The priority queue supports only comparable elements, which means that the elements are either arranged in an ascending or descending order.

* Every element in a priority queue has some priority associated with it.
* An element with the higher priority will be deleted before the deletion of the lesser priority.
* If two elements in a priority queue have the same priority, they will be arranged using the FIFO principle.

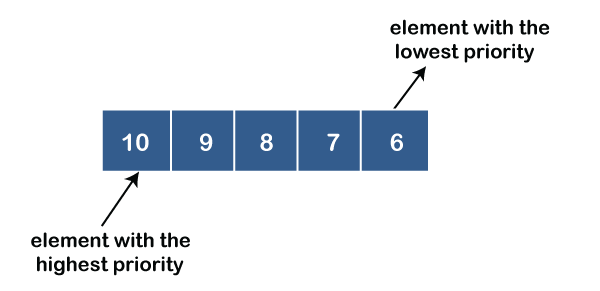
**Types of Priority Queue -**

There are two types of priority queue:

Ascending order priority queue: In ascending order priority queue, a lower priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in an ascending order like 1,2,3,4,5; therefore, the smallest number, i.e., 1 is given as the highest priority in a priority queue.



Descending order priority queue: In descending order priority queue, a higher priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in descending order like 5, 4, 3, 2, 1; therefore, the largest number, i.e., 5 is given as the highest priority in a priority queue.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Algorithm :**

1. **typedef createQueue():** This method creates and empty queue by setting initial values of front=0, rear=-1.
2. **void enqueue(typedef val) :**

This operation adds an item at the rear of the queue. Before insert operation, ensure that there is a room for the new item. If there is not enough room, then the queue is in an ‘Overflow’ state.

1. **typedef dequeueMax() :** This operation removes the item at the *front* of the *Queue* which has maximum priority and returns it to the user. Because the front item has been removed, the next item becomes the front. When the last item in the queue is deleted, it must be set to its empty state. If *dequeue* is called when the queue is empty, then it’s in an ‘Underflow’ state.
2. **typedef getFront() :** This operation will return maximum priority item of the queue at that instance.
3. **void displayAll():** This operation will display all item present in the queue at that instance.
4. **boolean isEmpty() :** This operation will check whether *Queue* is empty or not at a given instance. The function will return

0 if *Queue* is not empty.

1 if *Queue* is empty.

**8. boolean isFull() :** This operation will check whether *Queue* is full or not at a given instance. The function will return

0 if *Queue* is not full.

1 if *Queue* is full.

**9. int size() :** This operation will count the total number of elements present in the *Queue* at a given instance and return the count.

0 if Queue is empty.

n if Queue is having n no. of elements.

**Activity:** students are expected to implement priority queue using **array.**

**NOTE : All functions should be able to handle boundary(exceptional) conditions.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Results:** A program implementing solution depicting the correct behaviour of priority queue and capable of handling all possible exceptional conditions and the same is reflecting clearly in the output.

Program:

#include<stdlib.h>

#include<stdbool.h>

#include<stdio.h>

#define MAX 5

struct Queue

{

int front;

int rear;

int data[MAX];

};

void createQueue(struct Queue \*q)

{

q->rear = -1;

q->front = 0;

}

bool isFull(struct Queue \*q)

{

int full=0;

if( q->rear - q->front == MAX -1 )

full = 1;

return full;

}

bool isEmpty(struct Queue \*q)

{

int empty=0;

if( q->rear < q->front )

empty = 1;

return empty;

}

void enqueue(struct Queue \*q,int item)

{

if( isFull(q) )

{

printf("\nQueue Overflow");

return ;

}

q->rear++;

q->data[q->rear] = item;

printf("\nInserted element : %d.\n",item);

}

int dequeue\_max(struct Queue \*q)

{

int s=q->front;

int n=q->data[q->front];

int index=q->front;

while(s<=q->rear)

{

if(n < q->data[s])

{

n=q->data[s];

index=s;

}

s++;

}

int temp=q->data[index];

q->data[index]=q->data[q->front];

q->data[q->front]=temp;

q->front++;

return n;

}

int dequeue(struct Queue \*q)

{

if( isEmpty(q) )

{

printf("\nQueue Underflow");

return -1;

}

return dequeue\_max(q);

}

int getFront(struct Queue \*q)

{

if(!isEmpty(q))

return q->data[q->front];

printf("No element present in the Queue.\n");

return -1;

}

int getsize(struct Queue \*q)

{

if(isEmpty(q))

return 0;

return q->rear-q->front+1;

}

void display(struct Queue \*q)

{

int size=q->front;

if(getsize(q)==0)

printf("No element prsent in Queue.\n");

while(size<=q->rear)

{

printf("%d ",q->data[size]);

size++;

}

}

int main()

{

int item = 0;

struct Queue q;

createQueue(&q);

int choice,data;

do

{

printf("\n1. Insert\n2. Delete\n3. Display\n4. Get Front \n5. Get Size \n6. Exit\n\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

printf("Enter the datament to be inserted in the queue: ");

scanf("%d",&data);

enqueue(&q,data);

break;

}

case 2:

{

printf("Deleted element = %d.\n",dequeue(&q));

break;

}

case 3:

{

display(&q);

break;

}

case 4:

{

printf("The front most element in queue is = %d\n",getFront(&q));

break;

}

case 5:

{

printf("The size of the queue is = %d\n",getsize(&q));

break;

}

case 6:

{

exit(0);

break;

}

default:

{

printf("Sorry, invalid choice!\n");

break;

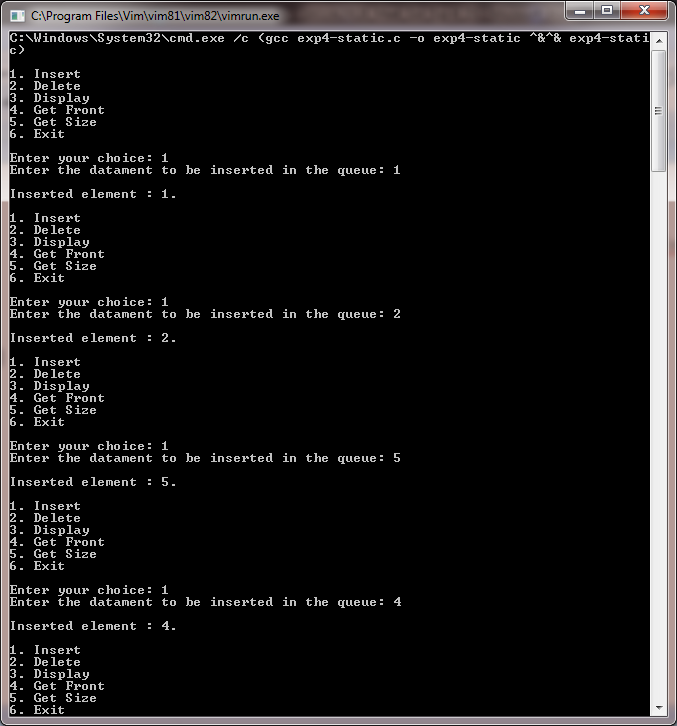
}

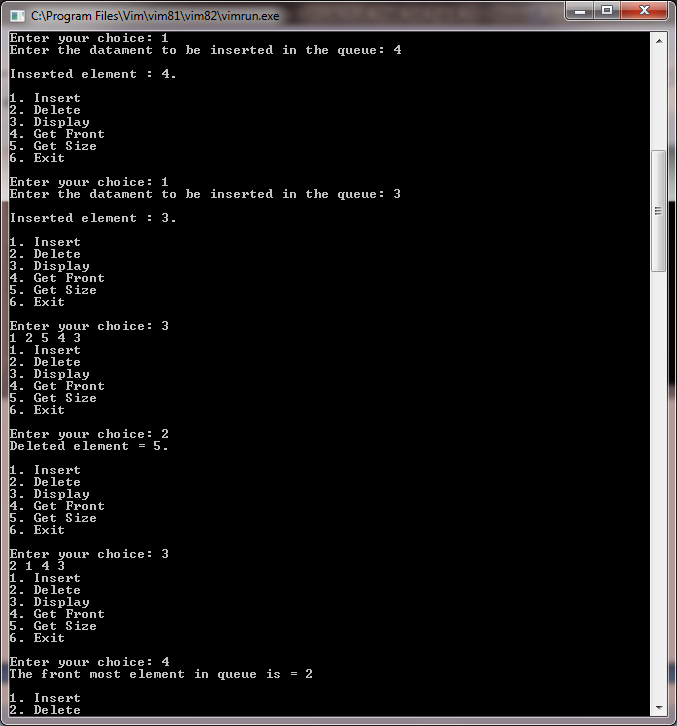
}

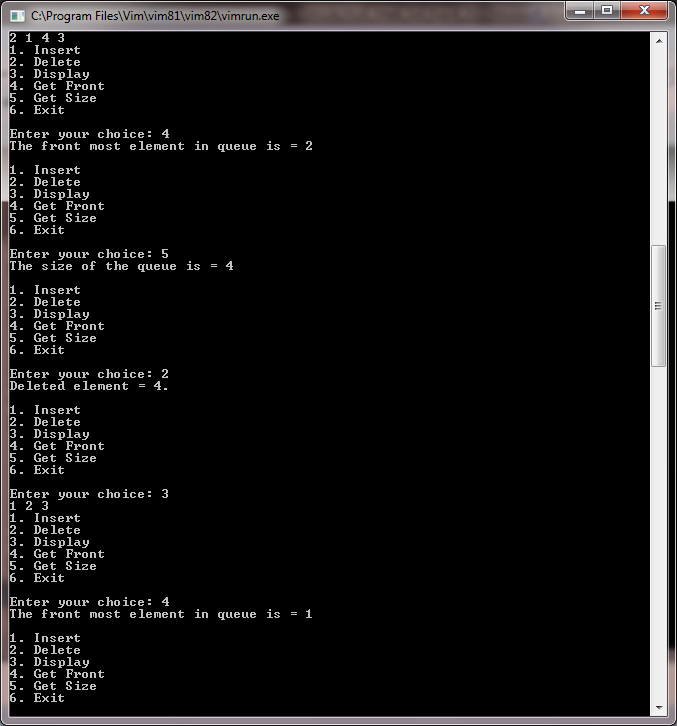
} while(choice!=6);

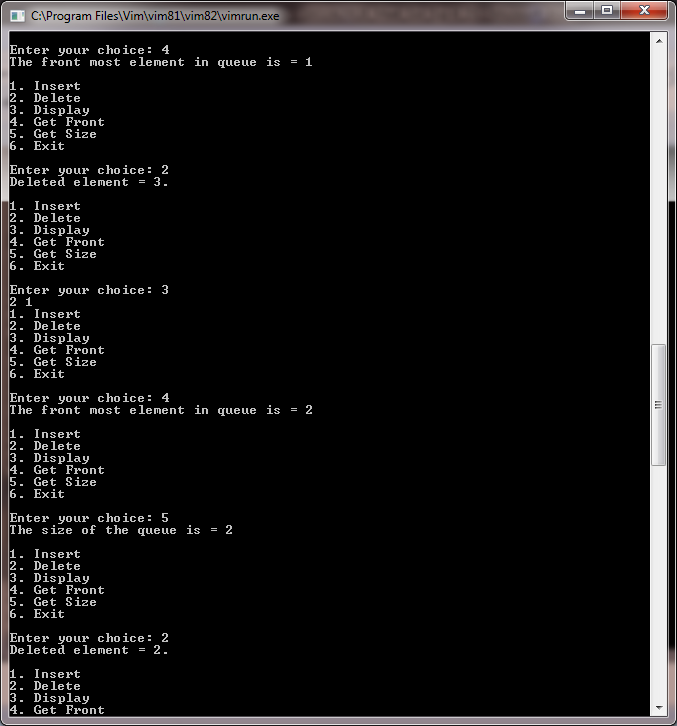
return 0;

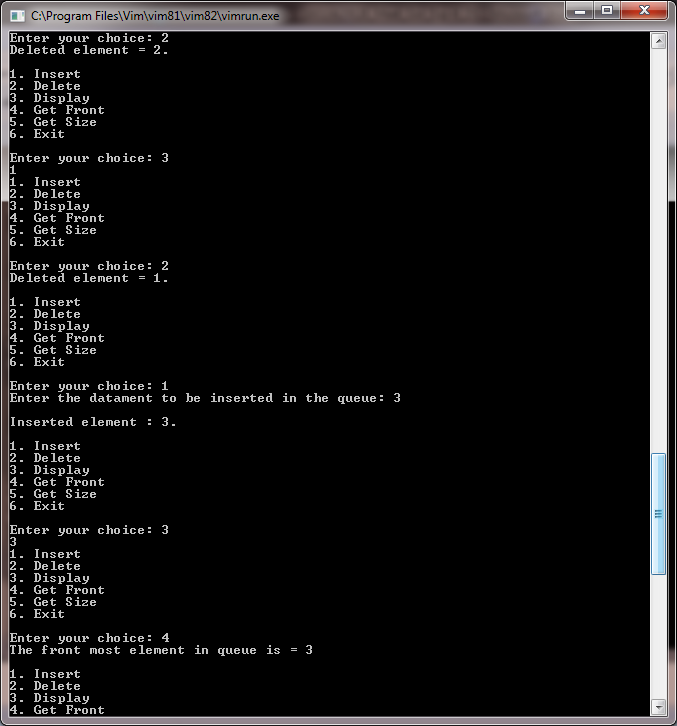
}

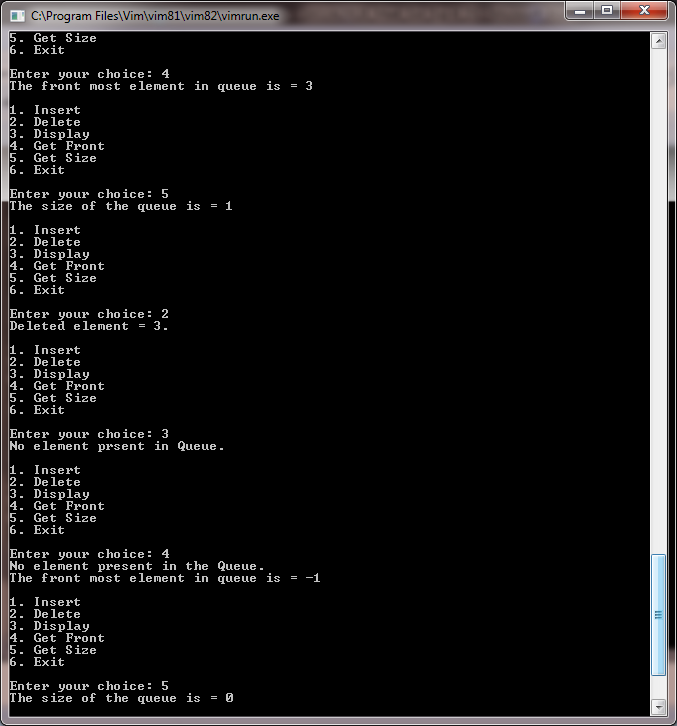












\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Outcome:**

Explain the different data structures used in problem solving

Apply linear and non-linear data structure in application development

**Conclusion:**

.From the following experiment, we were able to understand the concept of Queue Data structure. From that level of understanding, we were able to implement a linear-static priority Queue in the C-programming language.

**Grade: AA / AB / BB / BC / CC / CD /DD:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Signature of faculty in-charge with date :**

**References:**

**Books/ Journals/ Websites:**

* Y. Langsam, M. Augenstin and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002.