DATA STRUCTURES LAB EXPERIMENTS

SANSKRUTI SAMANT

USN: 2GI17EC113

DIV: B

# **EXPERIMENT 4:**

# Implementation of Queue Using C.

* **PROBLEM STATEMENT**:

Write a menu driven program in C to perform following operations on the Queue.

1. Insert
2. Delete

* **ALGORITHM:**

Start.

1. Create the Queue by declaring the array of fixed size.
2. Initialize the variables front and rear to -1 which indicate the front end and rear end of the queue respectively.
3. I) For insert operation,

First check if rear has reached to the last of the queue if yes don’t insert the new element, if no, increment the rear variable so that it points to next location of queue and insert the new element.

II) For delete operation,

First check if front variable is -1 or front>rear which indicates all the elements are deleted from the queue,

If yes, don’t delete.

If no, return the element from the front index and increment the front variable so that it points to the next value.

1. Stop.

# **PROGRAM CODE:**

# //C program to implement queue

# #include <stdio.h>

# #include <stdlib.h>

# #define MAX 5

# void insert();

# void delete();

# void display();

# int queue\_arr[MAX];

# int rear = -1;

# int front = -1;

# int main()

# {

# int choice;

# while(1)

# {

# printf("1. Insert an element\t");

# printf("2. Delete an element\t");

# printf("3. Display the queue\t");

# printf("4. Quit\n");

# printf("Enter your choice:");

# scanf("%d", & choice);

# switch(choice)

# {

# case 1 : insert();

# break;

# case 2 : delete();

# break;

# case 3 : display();

# break;

# case 4 : exit(1);

# default : printf("wrong choice\n");

# }//end of switch

# }//end of while

# }//end main

# void insert()

# {

# int add\_item;

# if(rear == MAX-1)

# printf("Queue overflow!! Isertion not possible !!\n");

# else

# {

# if (front == -1)

# /\* if queue is initially empty\*/

# front = 0;

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

# scanf("%d", &add\_item);

# rear = rear +1;

# queue\_arr[rear] = add\_item;

# }

# }//end of insert function

# void delete()

# {

# if (front == -1 || front >rear)

# {

# printf("Queue underflow!! Deletion not possible!\n");

# return;

# }

# else

# {

# printf("Element deleted from queue is: %d \n",queue\_arr[front]);

# front = front +1;

# }

# }//end of delete()

# void display()

# {

# int i;

# if (front == -1)

# printf(" Queue is empty!!\n");

# else

# {

# printf("Queue is :\n");

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

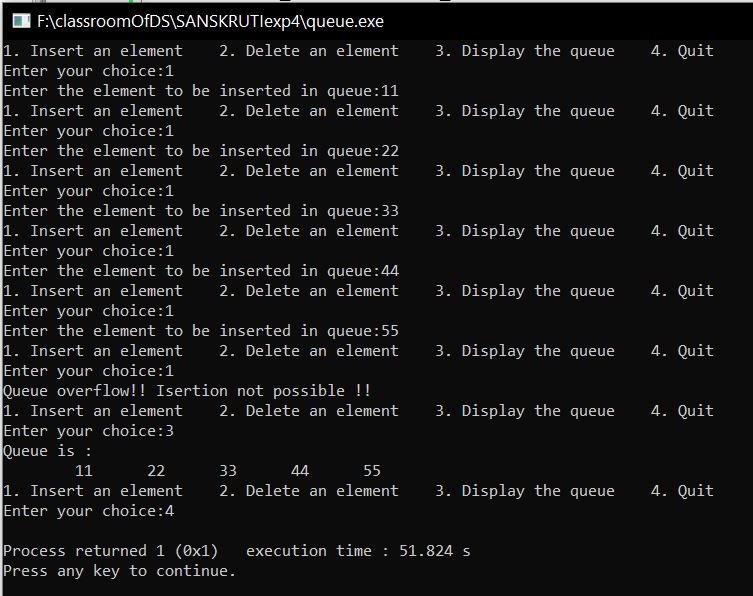
# printf("\t%d", queue\_arr[i]);

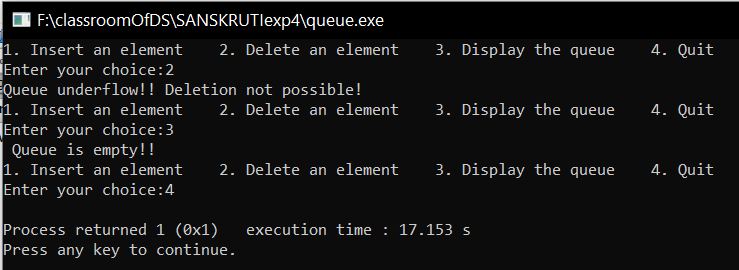
# printf("\n");

# }

# }//end of display()

* **PROGRAM OUTPUT:**

****

****

* **ANALYSIS (LIMITATIONS):**
* More memory is required to store elements in linked list as compared to array, the reason being: in linked list each node contains a pointer and it requires extra memory for itself.
* Elements or nodes traversal is difficult in linked list.