Batch: D2	**Roll No.:16010122323**
Experiment / assignment / tutorial No.	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of Basic operations on queue for the assigned application using Array and Linked List- Create, Insert, Delete, Destroy

Objective: To implement Basic Operations on Queue i.e. Create, Push, Pop, Destroy for the given application

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4.

Abstract:

(Define Queue, enlist queue operations).

Queue- A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Queue Operations:-

- **enqueue** - adds an item to the end of the queue
- **dequeue** - remove an item from front of the queue
- **initialize** - create an empty queue
- **isEmpty** - tests for whether or not queue is empty
- **isFull** - tests to see if queue is full (not needed if data structure grows automatically)
- **front** - looks at value of the first item but do not remove it

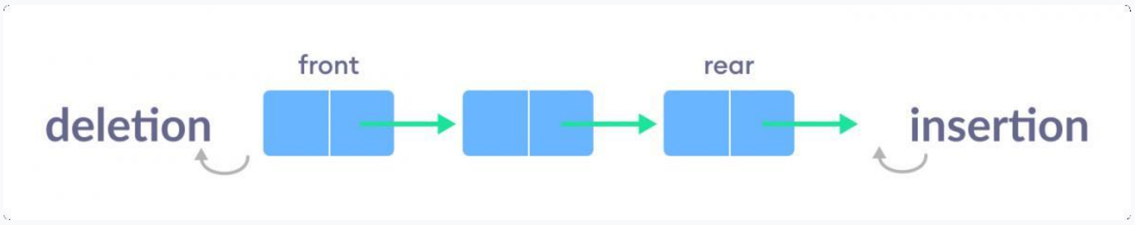
List 5 Real Life applications of Queue:

- A queue of people at ticket-window.
- Vehicles on toll-tax bridge.
- Phone answering system.
- Luggage checking machine.
- Patients waiting outside the doctor's clinic.

Define and explain various types of queue with suitable diagram and their application(s):

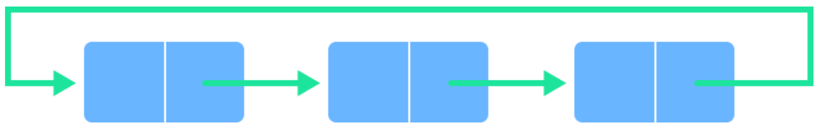
1.Simple Queue

In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.



2.Circular Queue

In a circular queue, the last element points to the first element making a circular link.

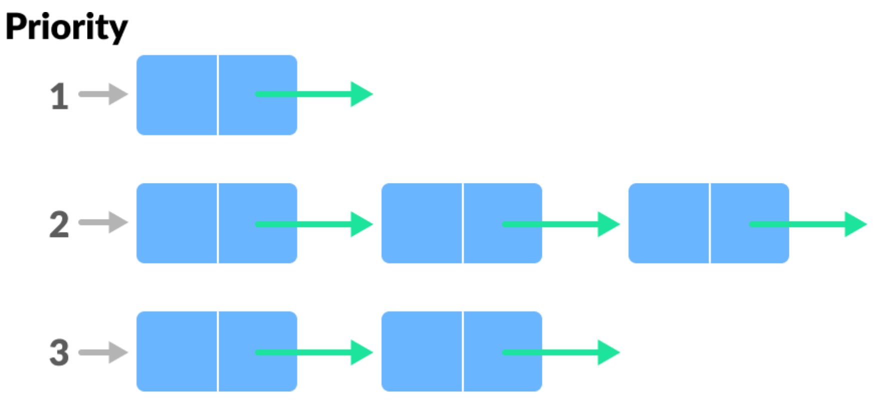


Circular Queue Representation

The main advantage of a circular queue over a simple queue is better memory utilization. If the last position is full and the first position is empty, we can insert an element in the first position. This action is not possible in a simple queue.

3.Priority Queue

A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue.

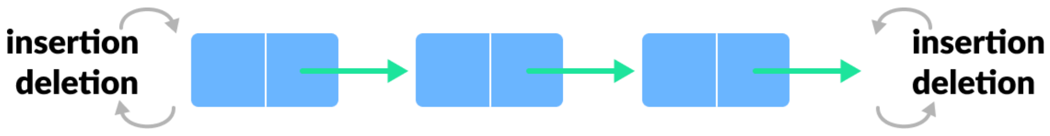


Priority Queue Representation

Insertion occurs based on the arrival of the values and removal occurs based on priority.

4.Deque (Double Ended Queue)

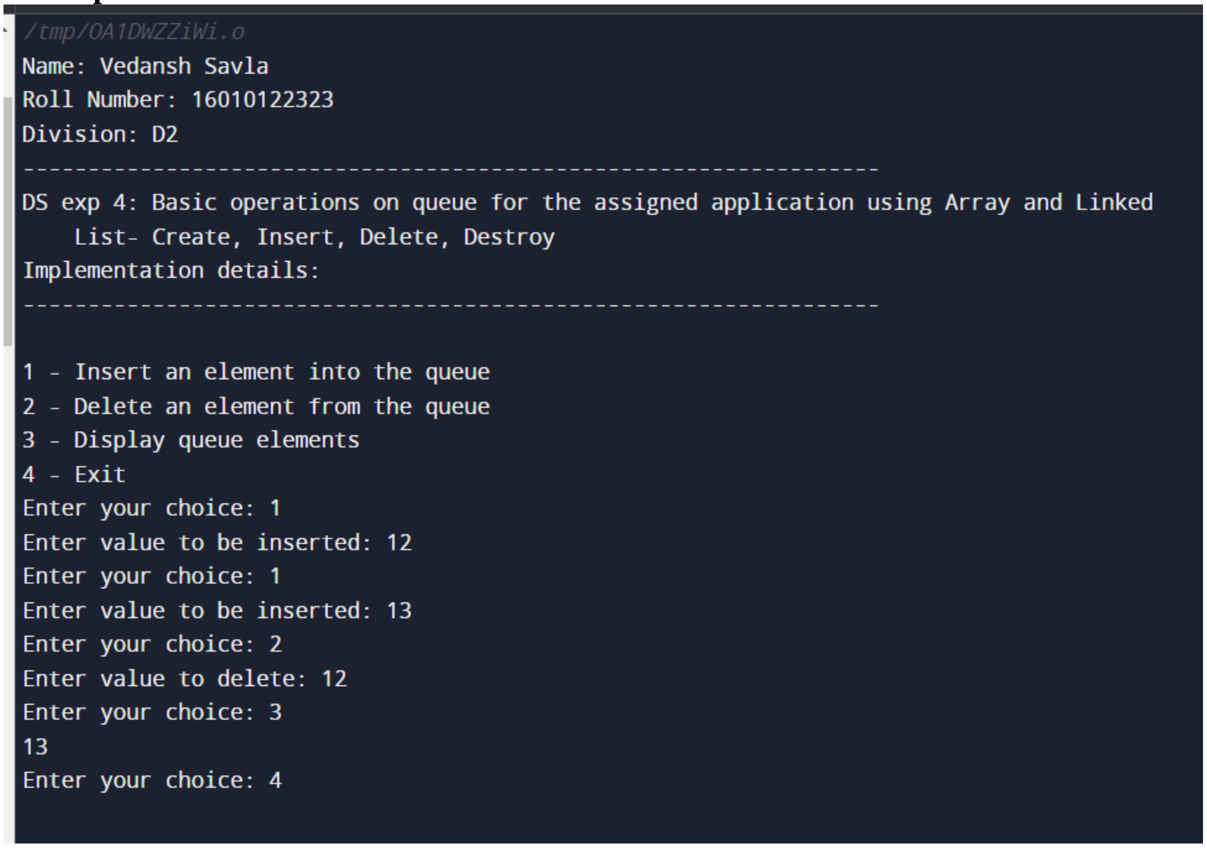
In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow the FIFO (First In First Out) rule.



Deque Representation

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX 5

void insert_by_priority(int);
void delete_by_priority(int);
void create();
void check(int);
void display_pqueue();

int pri_que[MAX];
int front = -1, rear = -1;

int main() {
    int n, ch;

    printf("Name: Vedansh Savla\n");
    printf("Roll Number: 16010122323\nDivision: D2\n");
    printf("-----------------------------------------------------------------
-\n");
    printf("DS exp 4: Basic operations on queue for the assigned application
using Array and Linked List- Create, Insert, Delete, Destroy\n");
    printf("Implementation details:\n");
    printf("-----------------------------------------------------------------
-\n");


    printf("\n1 - Insert an element into the queue");
    printf("\n2 - Delete an element from the queue");
    printf("\n3 - Display queue elements");
    printf("\n4 - Exit");

    create();

    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("\nEnter value to be inserted: ");
                scanf("%d", &n);
                insert_by_priority(n);
                break;
            case 2:
                printf("\nEnter value to delete: ");
                scanf("%d", &n);
                delete_by_priority(n);
                break;
            case 3:
                display_pqueue();
                break;
            case 4:
                exit(0);
            default:
                printf("\nChoice is incorrect, Enter a correct choice");
        }
    }
}

/* Function to create an empty priority queue */
void create() {
    front = rear = -1;
}

/* Function to insert value into the priority queue */
void insert_by_priority(int data) {
    if (rear >= MAX - 1) {
        printf("\nQueue overflow, no more elements can be inserted\n");
        return;
    }
    if (front == -1 && rear == -1) {
        front++;
        rear++;
        pri_que[rear] = data;
        return;
    } else {
        check(data);
        rear++;
    }
}

/* Function to check priority and place element */
void check(int data) {
    int i, j;
    for (i = 0; i <= rear; i++) {
        if (data >= pri_que[i]) {
            for (j = rear + 1; j > i; j--) {
                pri_que[j] = pri_que[j - 1];
            }
            pri_que[i] = data;
            return;
        }
    }
    pri_que[i] = data;
}

/* Function to delete an element from the queue */
void delete_by_priority(int data) {
    int i;
    if (front == -1 && rear == -1) {
        printf("\nQueue is empty, no elements to delete\n");
        return;
    }
    for (i = 0; i <= rear; i++) {
        if (data == pri_que[i]) {
            for (; i < rear; i++) {
                pri_que[i] = pri_que[i + 1];
            }
            pri_que[i] = -99;
            rear--;
            if (rear == -1) {
                front = -1;
            }
            return;
        }
    }
    printf("\n%d not found in the queue to delete\n", data);
}

/* Function to display queue elements */
void display_pqueue() {
    if (front == -1 && rear == -1) {
        printf("\nQueue is empty\n");
        return;
    }
    for (; front <= rear; front++) {
        printf(" %d ", pri_que[front]);
    }
    front = 0;
}
```

Output :



```
/tmp/OA1DWZZiWi.o
Name: Vedansh Savla
Roll Number: 16010122323
Division: D2
------------------------------------------------------------------
DS exp 4: Basic operations on queue for the assigned application using Array and Linked
    List- Create, Insert, Delete, Destroy
Implementation details:
------------------------------------------------------------------

1 - Insert an element into the queue
2 - Delete an element from the queue
3 - Display queue elements
4 - Exit
Enter your choice: 1
Enter value to be inserted: 12
Enter your choice: 1
Enter value to be inserted: 13
Enter your choice: 2
Enter value to delete: 12
Enter your choice: 3
13
Enter your choice: 4
```

Algorithm for Queue operations using array/Linked list : (Write only the algorithm for assigned type)

```
○ Step 1: Allocate the space for the new node PTR
○ Step 2: SET PTR -> DATA = VAL
○ Step 3: IF FRONT = NULL
      SET FRONT = REAR = PTR
      SET FRONT -> NEXT = REAR -> NEXT = NULL
  ELSE
      SET REAR -> NEXT = PTR
      SET REAR = PTR
      SET REAR -> NEXT = NULL
  [END OF IF]
○ Step 4: END
```

Implementation Details:

1) Mention the application assigned to you and explain how you implemented the solution using the assigned type of Queue.

Application – JOSEPHUS PROBLEM.

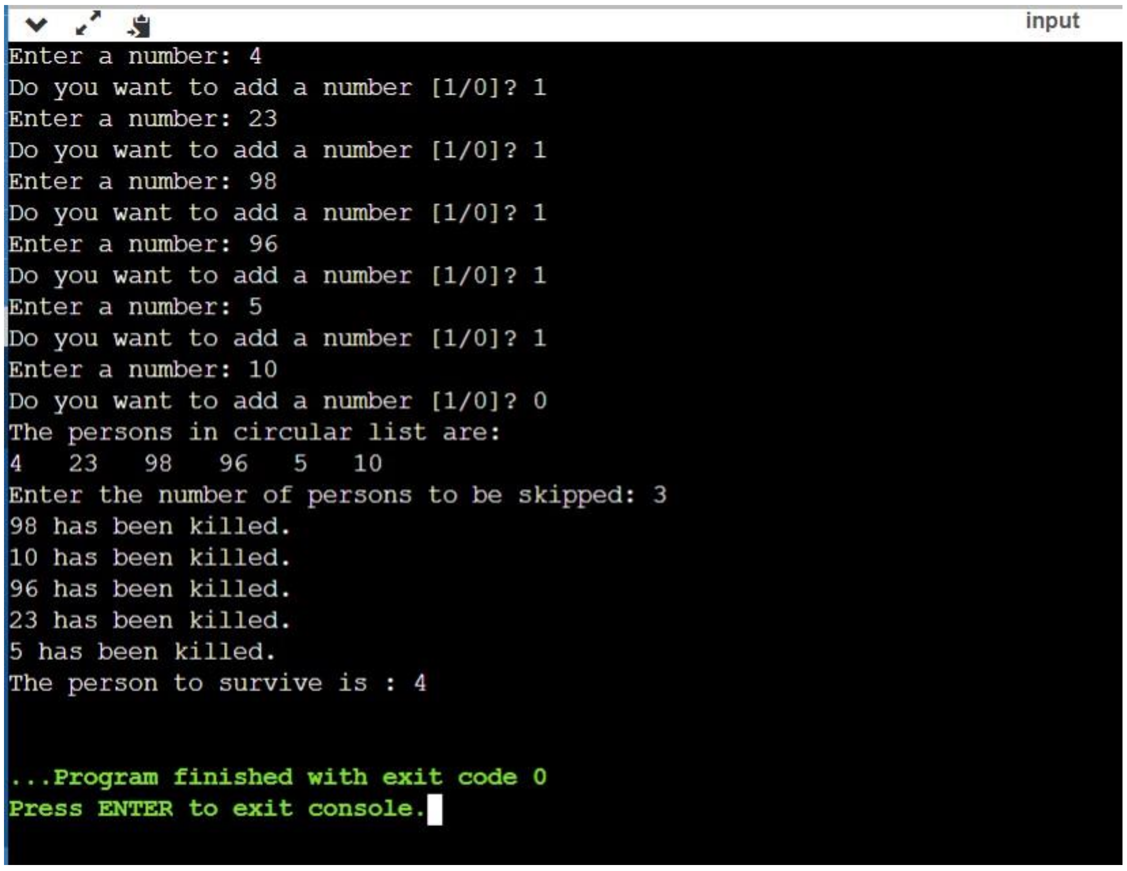
Program source code:

```
#include <stdio.h>

   #include <stdlib.h>

   struct node

   {

       int number;

       struct node *next;

   };

   void create(struct node **);
   void display(struct node *);

   int survivor(struct node **, int);

   int main()

   {

       struct node *head = NULL;

       int survive, skip;

       create(&head);

       printf("The persons in circular list are:\n");

       display(head);

       printf("Enter the number of persons to be skipped: ");

       scanf("%d", &skip);

       survive = survivor(&head, skip);

       printf("The person to survive is : %d\n", survive);

       free(head);

       return 0;

   }
```

```
int survivor(struct node **head, int k)

{

    struct node *p, *q;

    int i;

    q = p = *head;

    while (p->next != p)

    {

        for (i = 0; i < k - 1; i++)

        {

            q = p;

            p = p->next;

        }

        q->next = p->next;

        printf("%d has been killed.\n", p->number);

        free(p);
        p = q->next;
    }
    *head = p;
    return (p->number);
}
void create (struct node **head)
{
    struct node *temp, *rear;
    int a, ch;
    do
    {
        printf("Enter a number: ");
        scanf("%d", &a);
        temp = (struct node *)malloc(sizeof(struct node));
        temp->number = a;
        temp->next = NULL;
        if (*head == NULL)

        {

            *head = temp;

        }

        else

        {

            rear->next = temp;

        }

        rear = temp;

        printf("Do you want to add a number [1/0]? ");

        scanf("%d", &ch);

    } while (ch != 0);

    rear->next = *head;

}

void display(struct node *head)
```

```
{

    struct node *temp;

    temp = head;

    printf("%d  ", temp->number);

    temp = temp->next;

    while (head != temp)

    {

        printf("%d  ", temp->number);

        temp = temp->next;

    }

    printf("\n");

}
```

Output Screenshots:



```
Enter a number: 4
Do you want to add a number [1/0]? 1
Enter a number: 23
Do you want to add a number [1/0]? 1
Enter a number: 98
Do you want to add a number [1/0]? 1
Enter a number: 96
Do you want to add a number [1/0]? 1
Enter a number: 5
Do you want to add a number [1/0]? 1
Enter a number: 10
Do you want to add a number [1/0]? 0
The persons in circular list are:
4   23   98   96   5   10
Enter the number of persons to be skipped: 3
98 has been killed.
10 has been killed.
96 has been killed.
23 has been killed.
5 has been killed.
The person to survive is : 4


...Program finished with exit code 0
Press ENTER to exit console.
```

Applications of Queue in computer science:

- Queue is useful in CPU scheduling, Disk Scheduling. When multiple processes require CPU at the same time, various CPU scheduling algorithms are used which are implemented using Queue data structure.
- When data is transferred asynchronously between two processes.Queue is used for synchronization. Examples : IO Buffers, pipes, file IO, etc.
- Print Spooling.In print spooling, documents are loaded into a buffer and then the printer pulls them off the buffer at its own rate. Spooling also lets you place a number of print jobs on a queue instead of waiting for each one to finish before specifying the next one.
- Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, First come first served.
- In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free

Conclusion:-
Priority Queue was successfully implemented in C .