JADAVPUR UNIVERSITY

Department of Electronics and
TeleCommunication Engineering

Data Structures and Algorithms Lab

Assignment - 3 : Static and Dynamic Queue

Programming Language : C (ANSI)

==========

Sandip Dutta
2nd Year
Roll - 001910701017

==========

1. **Implement a fixed memory queue. You should implement the two primitive operations, namely, insert and delete with proper guards.You should experiment with a set of insert and delete operations. Show the contents of your queue after each operation.**

```
================================================================================
=
/* Queue Implementation with fixed memory */
# include <stdio.h>
# include <stdlib.h>

// Max size of the Static Queue
# define MAX_SIZE 3

// Queue Declaration
int front = -1, rear = -1;
int queue[MAX_SIZE];

//   Functions to Handle the Queue
//   enqueue : Inputs the value to Queue
void enqueue();
//   dequeue : Removes value from queue, returns it
void dequeue();
//   Displays the Queue
void displayQueue();
//   displays menu
void displayMenu();

int main(){
     int choice;
     do{
        displayMenu();
          scanf("%d", &choice);
          switch(choice){
               case 1:
                    enqueue();
                    displayQueue();
                    break;
               case 2:
                     dequeue();
                     displayQueue();
                     break;
               case 3:
                    displayQueue();
                    break;
               case 4:
                    printf("\nEND OF PROGRAM\n");
          }
     } while(choice != 4);

    return 0;
```

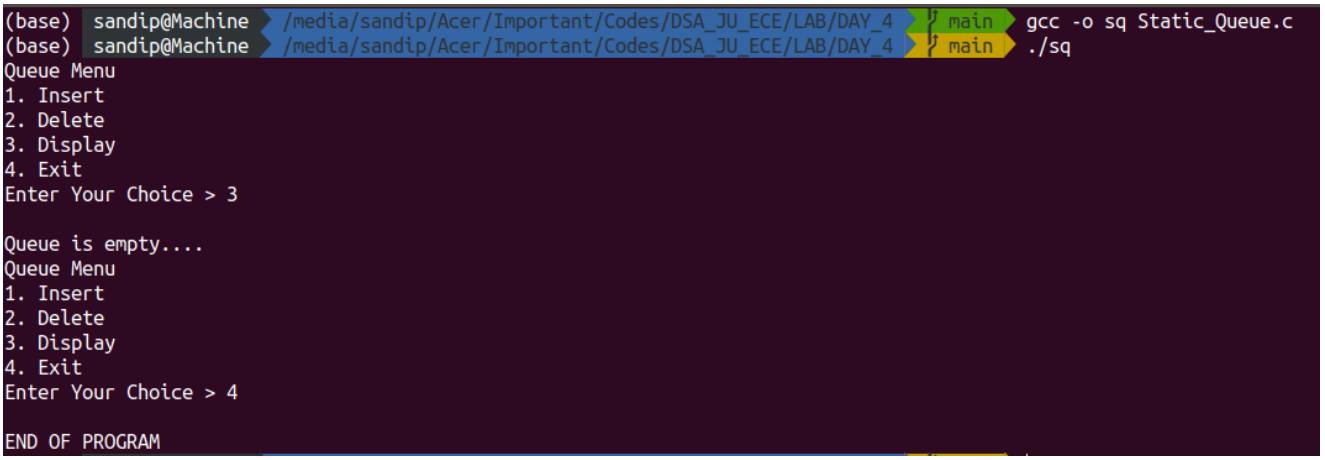
```
}
// enters number
void enqueue(){
     int no;
     printf("Enter Number to be inserted to Queue > ");
     scanf("%d",&no);

     if(rear < MAX_SIZE-1){
        ++(rear);
          queue[rear] = no;
          if(front == -1)
          front = 0;
     }else{
          printf("\nQueue overflow...\n");
     }
}

// removes number
void dequeue(){
     if(front == -1){
          printf("\nQueue Underflow...\n");
          return;
     }else{
          printf("Deleted Item --> %d\n",queue[front]);
     }
     if(front == rear){
          front=-1;
          rear=-1;
     }else{
        front++;
     }
}

// prints queue
void displayQueue(){
     int i;
     if(front == -1){
          printf("\nQueue is empty....\n");
          return;
     }
    printf("Queue : [front] ");
     for(i = front; i < rear; i++)
          printf("%d --> ",queue[i]);
    printf("%d [rear]\n", queue[rear]);
}
```

```
void displayMenu(){
    printf("Queue Menu\n");
     printf("1. Insert\n");
     printf("2. Delete\n");
     printf("3. Display\n");
     printf("4. Exit\n");
     printf("Enter Your Choice > ");
}
```

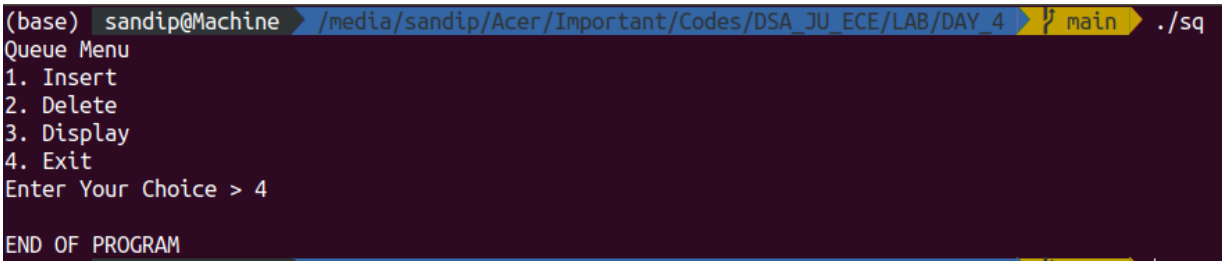
==========
=

OUTPUTS [OS : (UBUNTU) LINUX, COMPILER : GCC]

1. **Empty Queue Printing to check garbage value printed or not**



2. **Check whether exiting properly**



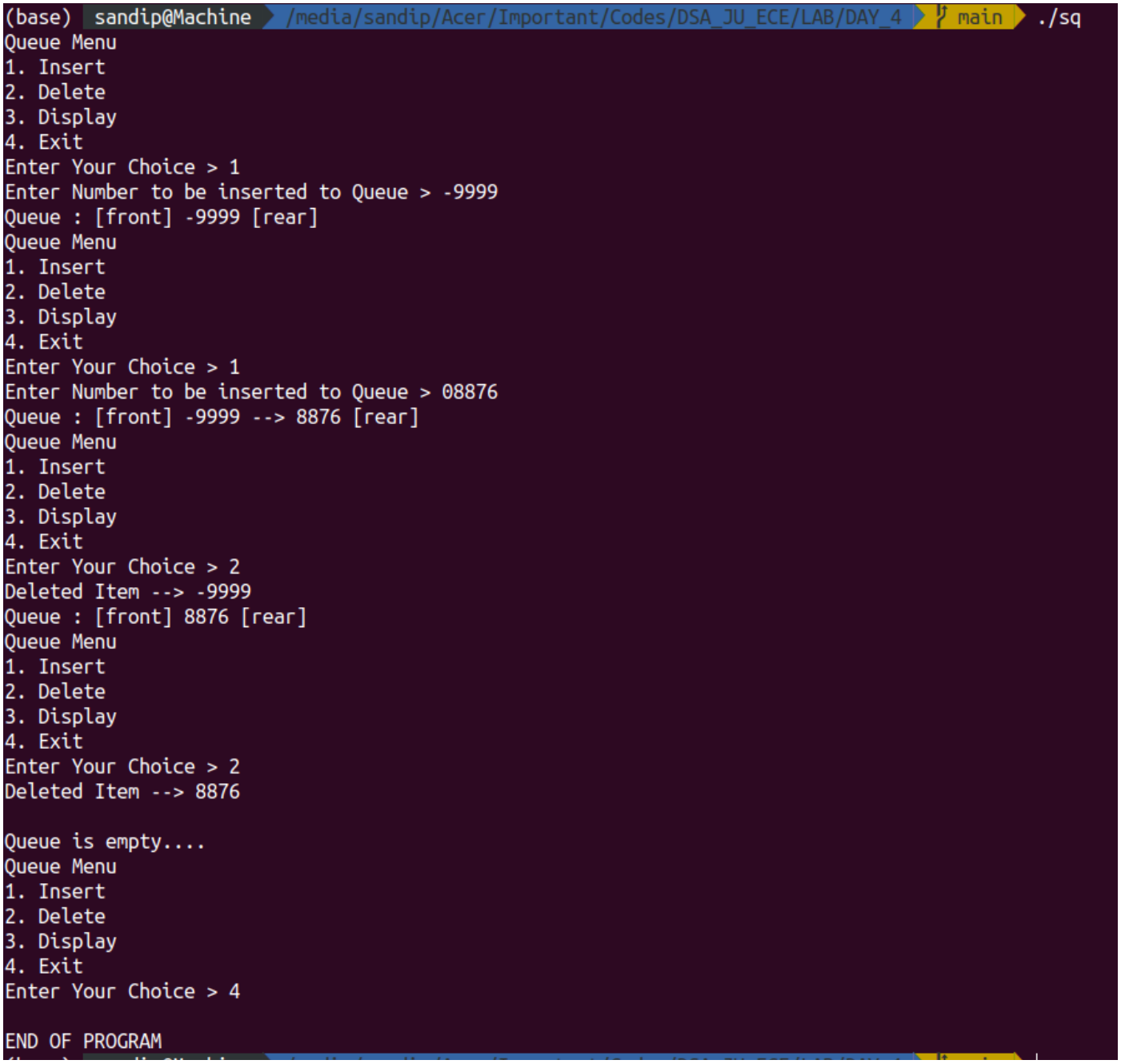
3. **Insertion, Displaying of value and Queue Underflow(Removing value from empty queue) [PTO]**

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ./sq
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 1
Enter Number to be inserted to Queue > 10
Queue : [front] 10 [rear]
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 3
Queue : [front] 10 [rear]
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 2
Deleted Item --> 10

Queue is empty....
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 2

Queue Underflow...

Queue is empty....
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 4

END OF PROGRAM
```

4. **Queue Overflow[Insert more value than MAXSIZE]**

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ./sq
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 1
Enter Number to be inserted to Queue > 100
Queue : [front] 100 [rear]
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 1
Enter Number to be inserted to Queue > 100
Queue : [front] 100 --> 100 [rear]
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 1
Enter Number to be inserted to Queue > 2022
Queue : [front] 100 --> 100 --> 2022 [rear]
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 1
Enter Number to be inserted to Queue > 10223

Queue overflow...
Queue : [front] 100 --> 100 --> 2022 [rear]
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 3
Queue : [front] 100 --> 100 --> 2022 [rear]
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice > 4

END OF PROGRAM
```

5. **Insertion and Deletion operation, clearly showing last in first out structure of queue (PTO)**



Conclusion

Thus, we have experimented with Queue, checked their LIFO structure. We have also experimented thoroughly, showing underflow, overflow and display conditions. Thus our program is thoroughly tested.

(PTO)

2. Implement a DYNAMIC memory queue. You should implement the two primitive operations, namely, insert and delete with proper guards.You should experiment with a set of insert and delete operations. Show the contents of your queue after each operation.

```
================================================================================
=
// Dynamic Queue - Sandip Dutta
# include <stdio.h>
# include <stdlib.h>
# include <limits.h>

// Node for Dynamic Queue
struct Node{
     int data;
     struct Node *next;
}*front, *rear;

// enqueue - puts in queue
// Since dynamic queue, so no OVERFLOW
void enqueue(int data){
    struct Node *newNode;
    newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if(rear == NULL || front == NULL) front = newNode;
    else rear -> next = newNode;

    rear = newNode;
}

// dequeue - removes from queue
int dequeue(){
    struct Node *temp;
    int number;

    if(front == NULL || rear == NULL){
        printf("\n![WARNING] UNDERFLOW. Returning Garbage Value");
        return INT_MIN;
    }else{
        temp = front;
        number = temp -> data;
        front = front->next;
        free(temp);
    }
    return number;
}
```

```
// display
void display(){
    struct Node *temp;
    temp = front;

    if(front == NULL||rear == NULL){
        printf("\nQueue is empty");
        return;
    }

    if(front == rear){
        printf("\nQueue [front] %d [rear]\n", front->data);
        return;
    }

    printf("Queue [front] ");

    while(temp -> next != NULL){
        printf("%d -> ", temp -> data);
        temp = temp -> next;
    }
    printf("%d [rear]\n", rear -> data);
}

// main
int main(){
    int choice, number;
    rear = NULL;
    front = NULL;

    while(1){
        printf("\nQueue Menu");
        printf("\n1. Insert");
        printf("\n2. Delete");
        printf("\n3. Display");
        printf("\n4. Exit");
        printf("\nEnter your choice > ");

        scanf("%d", &choice);

        switch(choice){

            case 1:
                printf("Enter The Element Value : \n");
                scanf("%d",&number);
                enqueue(number);
                printf("\nQueue after insertion : ");
                display();
                break;
```

```
            case 2:
                number = dequeue();
                printf("\nThe deleted element -> %d",number);
                printf("\nQueue after deletion");
                display();
                break;

            case 3:
                printf("\nQueue : ");
                display();
                break;

            case 4:
                printf("EXITING...");
                exit(0);
                break;

            default:
                printf("\n SELECT AGAIN!");
        }
    }
    return 0;
}
```

==========

=

OUTPUT : [OS : (UBUNTU) LINUX, COMPILER : GCC] [PTO]

NOTE : AS DYNAMIC QUEUE, NO OVERFLOW SHOWN

1. **Proper functioning of insert and delete, clearly showing LIFO structure**

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ± ./dq

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 1
Enter The Element Value :
1

Queue after insertion :
Queue [front] 1 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 1
Enter The Element Value :
2

Queue after insertion : Queue [front] 1 -> 2 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 2

The deleted element -> 1
Queue after deletion
Queue [front] 2 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 2

The deleted element -> 2
Queue after deletion
Queue is empty
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 4
EXITING...%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ±
```

2. Proper functioning of INSERT operation

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4  main ± ./dq

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 1
Enter The Element Value :
1

Queue after insertion :
Queue [front] 1 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 1
Enter The Element Value :
2

Queue after insertion : Queue [front] 1 -> 2 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 1
Enter The Element Value :
3

Queue after insertion : Queue [front] 1 -> 2 -> 3 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 1
Enter The Element Value :
4

Queue after insertion : Queue [front] 1 -> 2 -> 3 -> 4 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 4
EXITING...%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4  main ±
```

3. Proper functioning of INSERT and DELETE

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ± ./dq

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 1
Enter The Element Value :
100

Queue after insertion :
Queue [front] 100 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 1
Enter The Element Value :
2000

Queue after insertion : Queue [front] 100 -> 2000 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 2

The deleted element -> 100
Queue after deletion
Queue [front] 2000 [rear]

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 4
EXITING...%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ±
```

4. Exiting Properly

```
(base) ✘ sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ± ./dq

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 4
EXITING...%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ±
```

5. UNDERFLOW condition check

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ± ./dq

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 2

![WARNING] UNDERFLOW. Returning Garbage Value
The deleted element -> -2147483648
Queue after deletion
Queue is empty
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 4
EXITING...%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ±
```

6. Empty queue printing

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ± gcc -o dq Dynamic_Queue.c
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ± ./dq

Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 3

Queue :
Queue is empty
Queue Menu
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice > 4
EXITING...%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_4 main ±
```

Conclusion

Thus, we have experimented with dynamic Queue, checked their LIFO structure. We have also experimented thoroughly, showing underflow and display conditions. Overflow cannot be tested as for that we need to give so many inputs that program memory exhausts. Thus our program is thoroughly tested.

(PTO)