**PROGRAM-5**

***Q. Design and implement a given type of queue in C (Ordinary queue/Circular queue) using Array implementation and linked list implementation. Also demonstrate the working with suitable inputs. Display appropriate messages in case of exceptions.***

**“Queues Implementation using Linked List”**

* **Theory:**

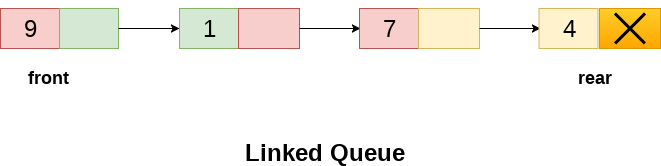
Because queues are employed in large-scale applications, the array implementation is inapplicable. Using a linked list to build a queue is an alternative to using an array.

Each node in a linked queue is made up of two components, the data part and the link part. Every item in the queue points to the item in the memory that comes right after it.

Two pointers—the front pointer and the rear pointer—are kept in memory for the linked queue. The addresses of the first and last elements in the queue are contained in the front pointer and the rear pointer, respectively.

At the back and the front, respectively, are where insertions and deletions are made. When both the front and the back of the queue are NULL, the queue is empty.

The linked representation of queue is shown in the following figure.



* **Algorithm:**

**Insertion:**

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

**Deletion:**

Step 1: IF FRONT = NULL

Write " Underflow "

Go to Step 5

[END OF IF]

Step 2: SET PTR = FRONT

Step 3: SET FRONT = FRONT -> NEXT

Step 4: FREE PTR

Step 5: END

//----------------------------------------------------------------------------------

* **Code:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next; // structure for creating a node

};

struct node \*front; //front and rear are declared

struct node \*rear;

void insert();

void delete();

void display(); //function prototypes

//-------------------------------------------------------------------------------

void main()

{

int choice;

while(choice!=4)

{

printf("------------------------Main Menu---------------\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n1.Insert an element\n2.Delete an element\n3.Display\n4.Exit\n");

printf("Enter your choice:"); //user enters his choice

scanf("%d",&choice);

switch(choice)

{

case 1: insert();

break;

case 2: delete(); //switch case for operations

break;

case 3: display();

break;

case 4: exit(0);

break;

default: printf("please enter valid choice\n");

}

}

}

//---------------------------------------------------------------------------------

void insert()

{

struct node \*ptr;

int item;

ptr=(struct node \*)malloc(sizeof(struct node));

//memory is allocated dynamically. A chunk of memory from heap is allocated.

if(ptr==NULL)

{

printf("Overflow\n");

}

else

{

printf("enter an value:"); // user enters the value to be inserted

scanf("%d",&item);

ptr->data=item; // data is entered

if(front==NULL)

{

front=ptr;

rear=ptr;

//if front is the last node then make front and rear as ptr.

front->next=NULL;

rear->next=NULL;

// the links of front and rear are made as NULL

}

else

{

rear->next=ptr;

rear=ptr;

rear->next=NULL;

}

}

printf("value inserted\n"); //value is inserted

}

//--------------------------------------------------------------------------------

void delete()

{

int item;

struct node \*ptr;

if(front==NULL)

{

printf("Underflow\n"); //no elements are present in queue

}

else

{

ptr=front; //front is pointed to ptr

front=front->next; // front is traversed

free(ptr); //ptr is freed

}

printf("Item is deleted\n"); //item is deleted.

}

//------------------------------------------------------------------------------

void display()

{

struct node \*ptr;

ptr=front;

if(front==NULL)

{

printf("Empty queue"); // queue has no elements

}

else

{

printf("Elements of the queue are...\n");

while(ptr!=NULL)

{

printf("%d\n",ptr->data);

//until ptr’s next is not equal to next it keeps on printing the data part.

ptr=ptr->next;

}

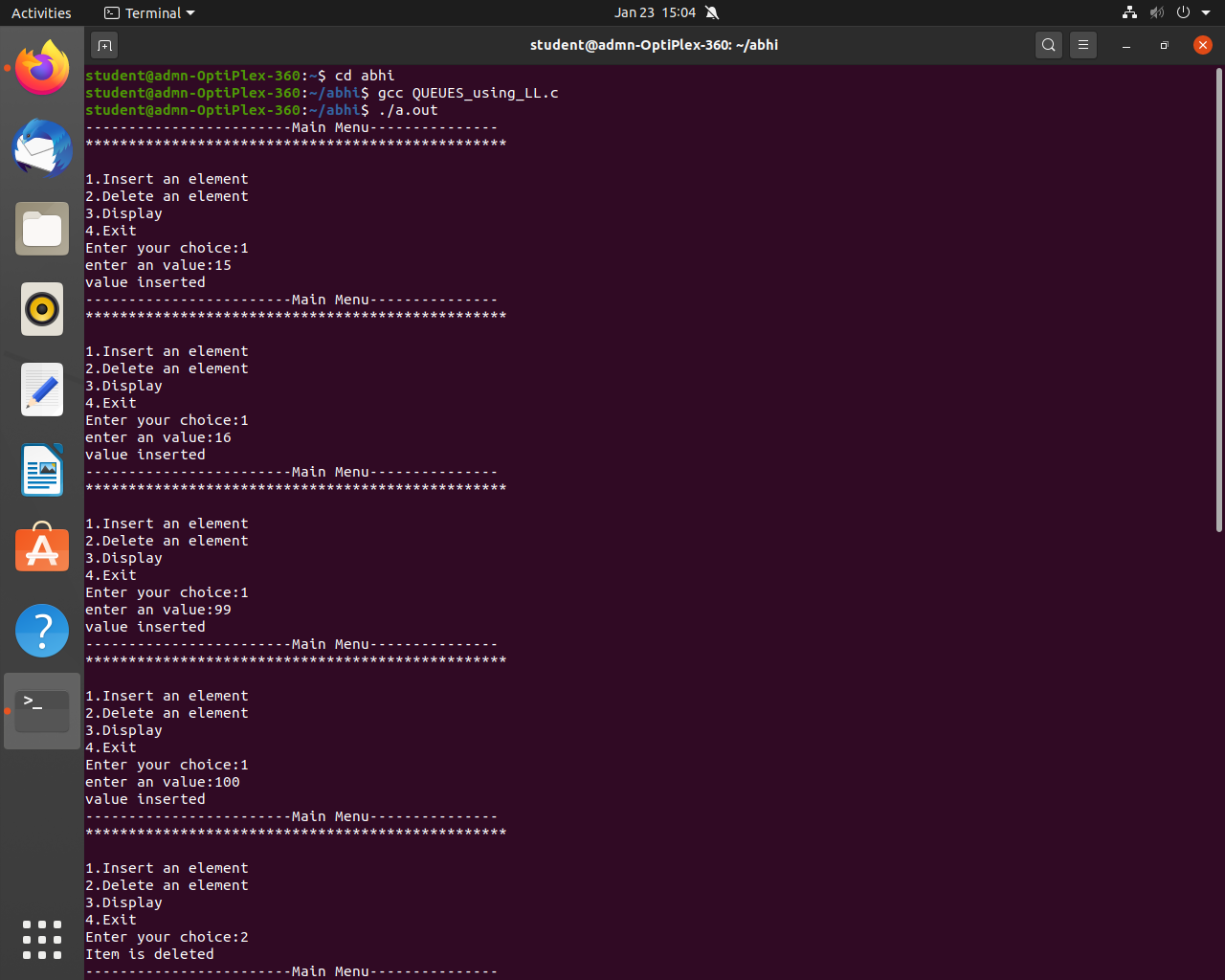
}

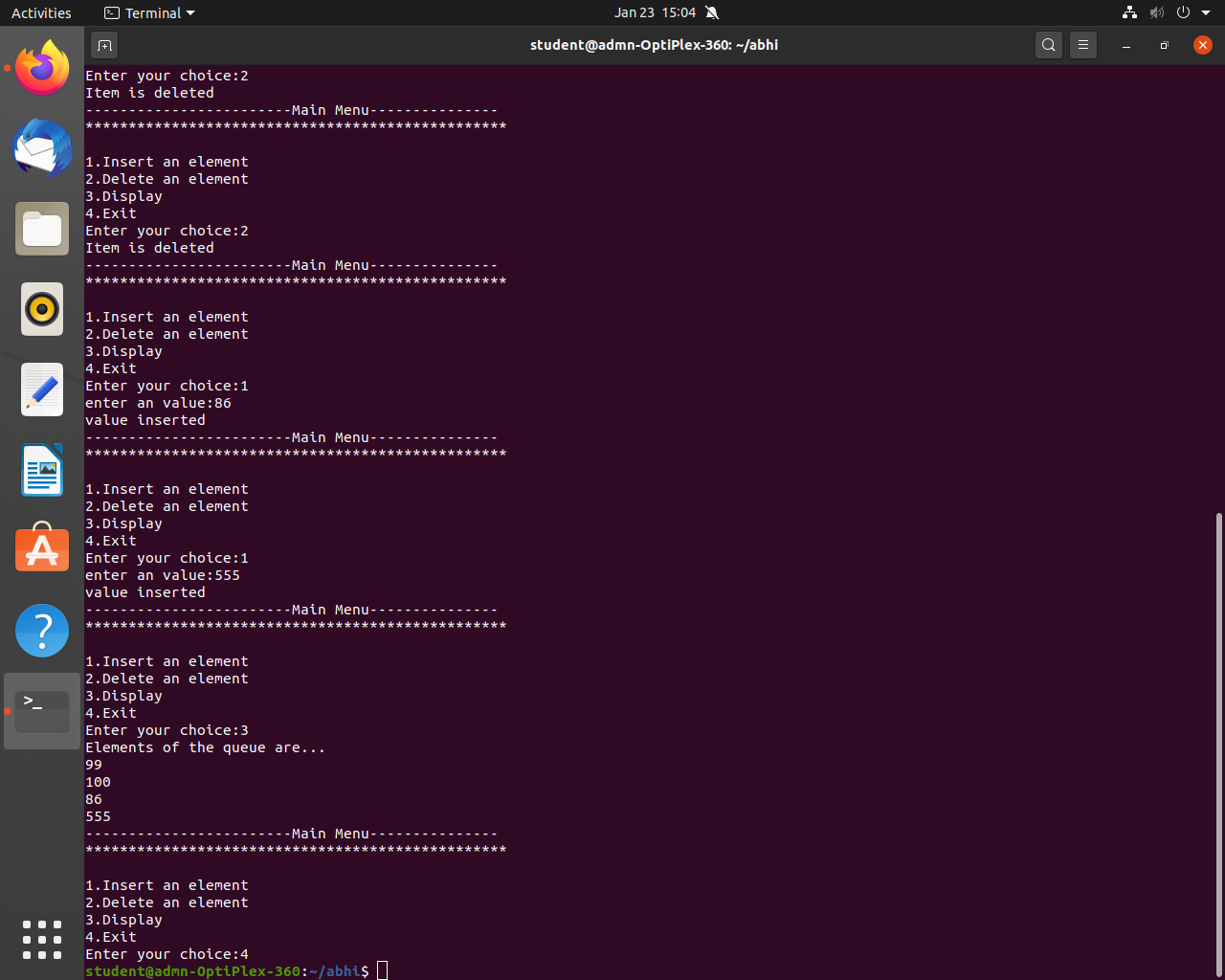
}

//---------------------------------------------------------------------------

//END OF THE PROGRAM

**“SCREENSHOTS OF OUTPUT”**





//----------------------------------------------------------------------------