**EXPT NO: 3 DATE: 12/11/21**

**QUEUES**

**AIM:**

1) Write a program to implement queue using array and perform the following operations

a) Insert

b) Delete

c) Display element at the front

d) Display all elements in the queue

2) Write a program to implement queue using linked list and perform the following operations

a) Insert

b) Delete

c) Display element at the front

d) Display all elements in the queue

3) Write a program to implement queue using circular linked list and perform the following operations

a) Insert

b) Delete

c) Display all elements in the queue

4) Write a program to implement circular queue and perform the following operations

a) Insert

b) Delete

c) Display all elements in the queue

5) Write a program to convert decimal number to binary number using queue

**THEORY:**

A queue is an ordered list of elements from which an element may maybe removed at one end, called the front of the queue and into an element may be inserted at the other end, called the rear of the queue. The behavior of a queue is if first in first out, so it is called as FIFO (First In First Out) data structure.

In a queue, the insertion operation is known as Enqueue and the deletion is known as deque.

If the insertion operation is attempted and there is no enough space in the queue, then this situation is called overflow and new elements cannot be inserted. If queue is empty and delete operation is attempted, then this situation I called as queue underflow.

**Array implementation of queue**

In stack, both the operations were performed at the same end, so we had to take only one variable top. but here operations are performed at different ends so we have to take two variables to keep track of both the ends. We will take the variable named rear and front , where rear will hold the index value of the last added element and front will old the value of the first added element.

1. Initially when eh queue is empty, the values of both front and rear will be -1.
2. For insertion, the value of rear is incremented by 1 and the element is inserted at the new rear position.
3. For deletion, the element at the front position is deleted and the value of front is incremented by 1.
4. When insertion done in an initially empty queue, i.e. If the value of front is -1, then value of front is made 0.

**NOTE:**

1) At anytime the number of elements in the queue is equal to

**(rear-front+1)**, except initially empty

**Conditions for queue to be Empty**

1. front == -1
2. front == rear+1

**Conditions for queue to be Full**

1. rear == MAX-1

**Advantage:**

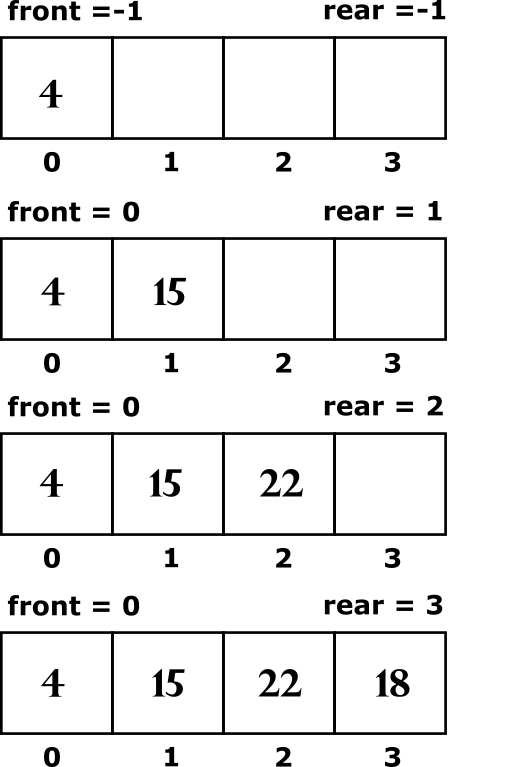
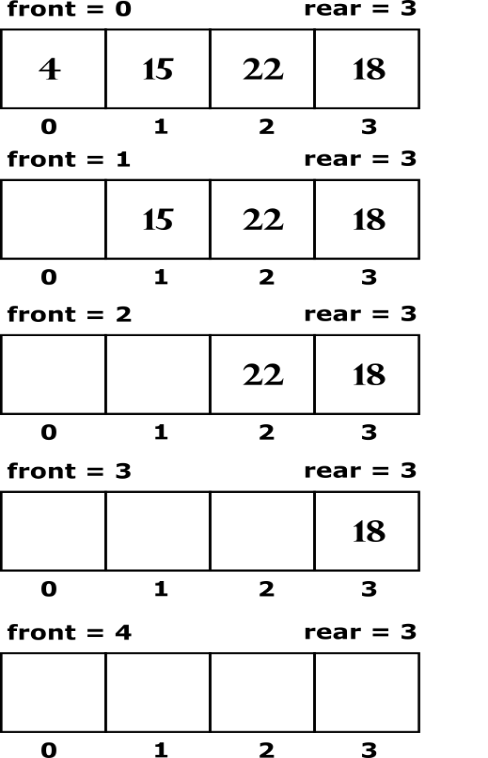
Correctly implemented array-based queue can be much faster as compared to the Linked List based implementation.

**Disadvantage:**

Memory wastage: After a certain dequeuing, the deleted element spaces will be wasted. But this problem can be solved using a Circular Queue.

**Array based queue implementation illustration**

Insert: 4, 15, 22, 18 Delete operation



**Linked List implementation of queue**

Queue can also be implemented using a Linked Lists.

struct node {

int info;

struct node \*link;

};

We will take beginning of linked list as front and end of linked list as rear. So, to insert an item in our queue we will add a node at the end of the list and to delete an item from the queue we will delete a node from the beginning of the list. We will maintain two pointers names front and rear, where front will point to the first node of the linked list and rear will point to the last node the linked list.

**Conditions for queue to be Empty**

1. front == NULL

**Conditions for queue to be Full**

1) temp =(struct node\*)malloc(sizeof(struct node));

Malloc() returns NULL if the memory is full

**Advantage:**

1) Dynamic Data structure: A linked List is a dynamic arrangement so it can grow and shrink at runtime by allocating and deallocating memory. So, there is no need to give the initial size of the queue

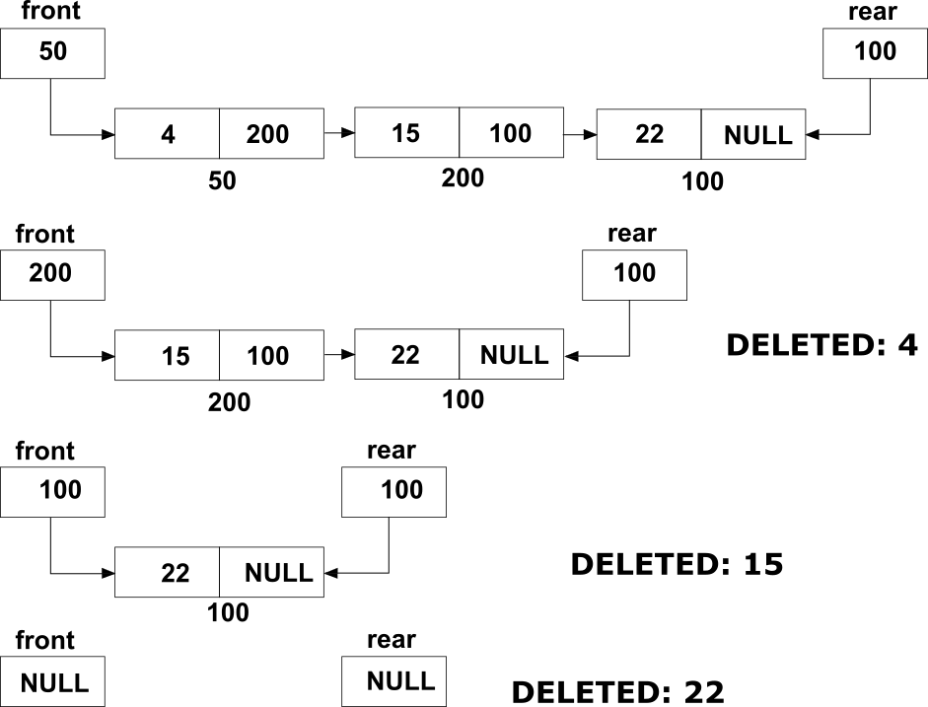
2) Easy implementation

**Disadvantage:**

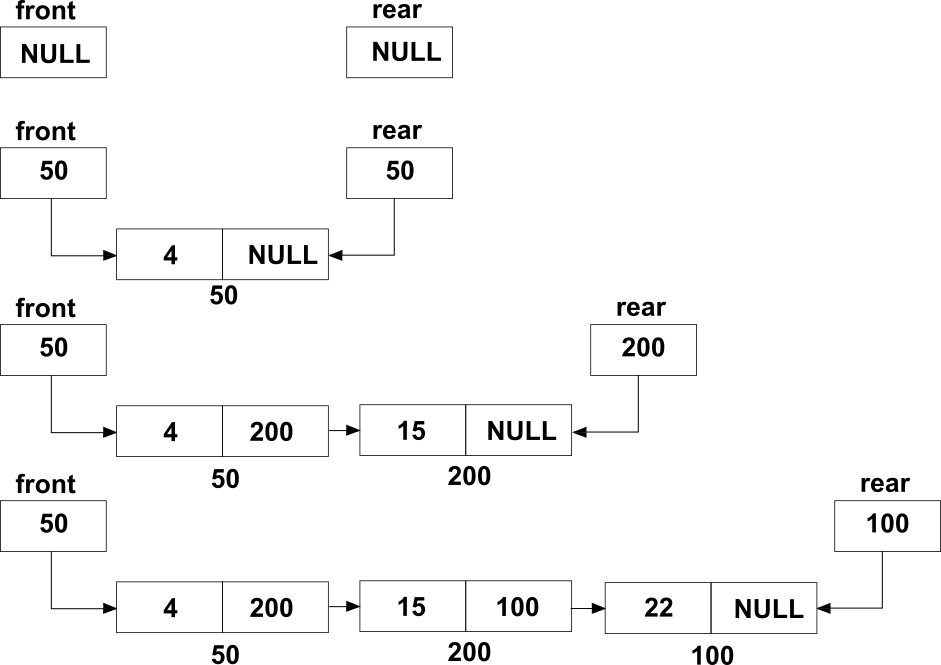
1) Memory Usage: More memory is required in the linked list as compared to an array. Because in a linked list, a pointer is also required to store the address of the next element and it requires extra memory for itself.

**Linked List based queue implementation illustration**

Insert: 4, 15, 22



Delete operation



**Circular Queue**

We know that when queue is implemented as an array, insertion is not possible after the rear reaches the last position of the array. There may be vacant positions in the array but they can’t be utilized. To overcome limitations, we use the concept of circular queue.

We can think of an array to be logically circular. So that the two ends of the array wrap up to make a circle.

Now after the (n-1)th position, 0th position occurs, if we want to insert an element we can insert it in the 0th position.

**Conditions for a Circular queue to be Empty**

1) front == -1

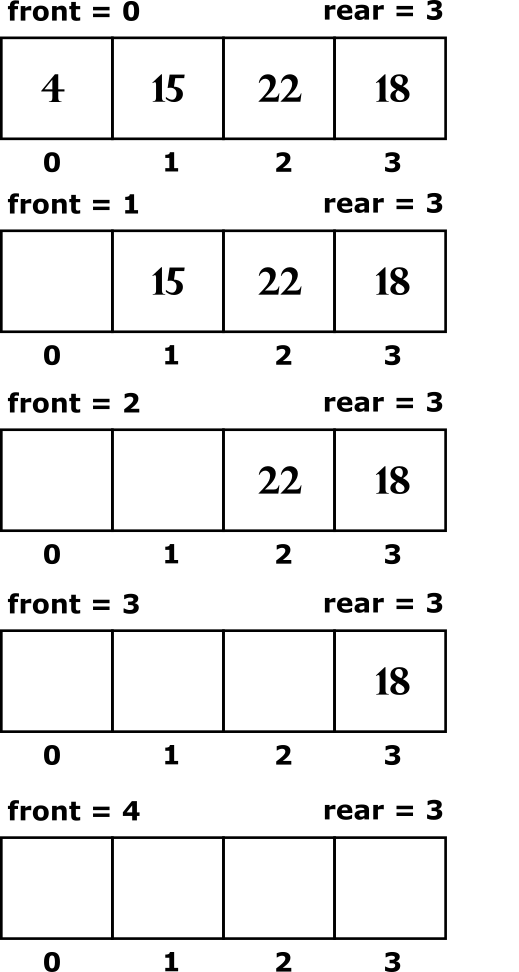
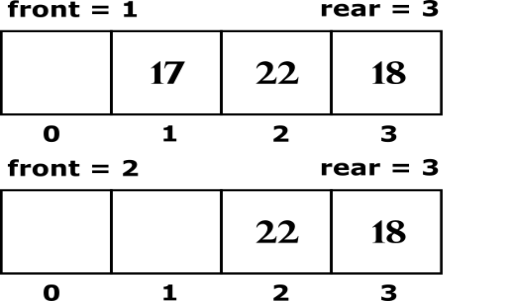
**Conditions for a Circular queue to be Full**

1) front == 0 && rear=MAX-1

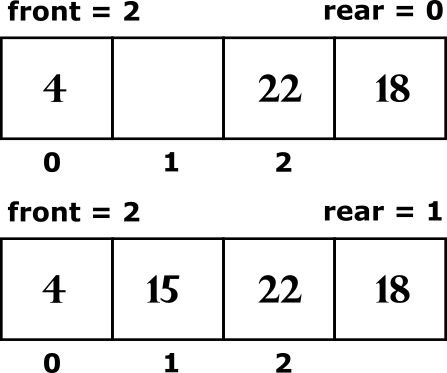
2) front == rear+1

**Circular queue illustration**

Insert: 17, 10, 22, 18 Delete: 17, 10



Insert 4, 15



**FUNCTIONS**

**void insert()**

The objective of this function is to insert the elements into the queue array. It first checks for the overflow and then subsequently inserts the elements into the queue

**void del()**

The del() function deletes the front element of the queue array. It first checks for the empty queue condition and subsequently deletes the front element.

**int peek()**

The front element of the queue is found and returned through this function.

**void display()**

It displays all the element present in the queue array.

**int isFull()**

Checks for the overflow condition of a queue (In case of array implementation of a queue)

**int isEmpty()**

Checks for the under-overflow condition of a queue

(In case of array implementation of a queue and Linked List implementation)

**Program Design**

1. Reversing The Queue

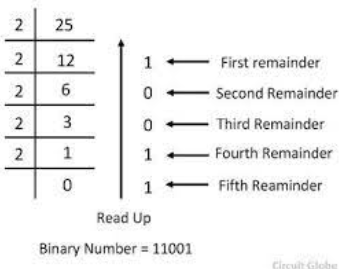
The fundamental approach used to reverse a queue is to use a stack.

Since the nature of **Queue** is **First In First Out (FIFO)** and the nature of a **Stack** is **Last In Last Out (LIFO),** we can transfer the element of queue into a stack and then pop the contents of the stack back into the queue array and hence the resultant queue will be the reverse version of the original queue.

2. Decimal to Binary Conversion

Step to perform the above-mentioned conversion

1. Write down the number
2. Divide it by 2 and note the remainder
3. Divide the quotient obtained by 2 and note the remainder
4. Repeat the same process till we get 0 as the quotient
5. Write the values of all the remainders starting from the bottom to the top.



This method is translated into a C- program. The remainders are stored into the Queue array. The queue is then reversed using a stack. The reverse queue is will yield the binary equivalent.

**PSEUDOCODES**

Array Based Implementation

**int isEmpty()**

1. if front== -1||front==rear+1

1.return 1

2.else

1. return 0

**int isFull()**

1. if top ==MAX-1

1.return 1

2.else

1. return 0

**void insert (int item)**

1. if (isFull())

1. Output “Queue Overflow”

2. return

2. if (front==-1)

1. front=0

3. rear++

4. Qarray[rear]=item

**int del()**

1. declare int item

2. if(isEmpty())

1. Output “Queue Underflow”

2. exit(1)

3. item=Qarray[front]

4. front++

5. return item

**Void display()**

1. declare int i

2. if(isEmpty())

1. Output “Queue is Empty”

2. return

3. for(i=front;i<=rear;i++)

1.Output Qarray[i]

**int peek()**

1. if(isEmpty())

1. Output “Queue Underflow”

2. exit(1)

2. return(Qarray[front])

Linked List Based Implementation

**void insert (int item)**

1. Declare struct node\*temp

2. temp=(struct node\*)malloc(sizeof(struct node))

3. if temp==NULL

1. Output “No Enough Space”

2. return

4. temp->info=item

5. temp->link=NULL

6. if(front==NULL)

1. front=temp

7. else

1. rear->link=temp

2. rear=temp

**int isEmpty()**

1. if front==NULL

1.return 1

2.else

1. return 0

**int peek()**

1. if(isEmpty())

1. Output Queue Underflow

2. exit(1)

2. return(front->info)

**int del()**

1. Declare struct node\*temp

2. int item

3. if(isEmpty())

1. Output “Queue Underflow”

2. exit(1)

4. temp=front

5. item=temp->info

6. front=front->link

7. free(temp)

8. return item

**Void display()**

1. Declare struct node \*p

2. p=front

3. if(isEmpty())

1. Output Queue Underflow

2. exit(1)

4. Output Queue Elements

5. while(p!=NULL)

1. Output p->info

2. p=p->link

Linked List Based Circular Queue

**int isEmpty()**

1. if front==NULL

1.return 1

2.else

1. return 0

**int del()**

1. Declare struct node\*temp

2. Declare int item

3. if(isEmpty())

1. Output Queue Underflow

2. exit(1)

4. if(rear->link==rear)

1. temp=rear

2. rear=NULL

5. else

1. temp=rear->link

2. rear->link=rear->link->link

6. item=temp->info

7. free(temp)

8. return item

**int peek()**

1. if(isEmpty())

1. Output Queue Underflow

2. exit(1)

2. return rear->link ->info

**void display()**

1. struct node\*p

2. if(isEmpty())

1. Output “Queue Is Empty”

2. return

4. Output “Queue”

5. p=rear->link

6. do

1. Output p->info

2. p=p->link

while(p!=rear->link)

**void insert (int item)**

1. struct node\*temp

2. temp=(struct node\*)malloc(sizeof(struct node))

3. if(temp==NULL)

1. Output Queue Overflow

2. exit(1)

4. if(isEmpty())

1. rear=temp

2. temp->link=rear

5. else

1. temp->link=rear->link

2. rear=temp

Array Based Circular Queue

**int isEmpty()**

1. if front== -1

1.return 1

2.else

1. return 0

**void display()**

1. Declare int i

2. if(isEmpty())

1. Output “Queue Is Empty”

2. return

3. Output “Queue”

4. i=front

5. if(front<=rear)

1. while(i<=rear)

1. Output Cqueue[i++]

else

1. while(i<=max-1)

2. Output Cqueue[i++]

3. i=0

4. while(i<=rear)

5. Output Cqueue[i++]

**int isFull()**

1. if ((front==0&&rear==max-1||front==rear+1))

2. return 1

3. else

4. return 0

**void insert(int item)**

1. if(isFull())

1. Output Queue Overflow

2. return

3. if(front==-1)

4. front=0

5. if(rear==max-1)

6. rear=0

7. else

8. rear=rear+1

9. Cqueue[rear]=item

**int del()**

1. int item

2. if(isEmpty())

1. Output “Queue Underflow”

2. exit(1)

3. item=Cqueue[front]

4. if front==rear

1. front = -1

2. rear = -1

5. else if(front==max-1)

1. front=0

else

1. front=front+1

7. return item

**int peek()**

1. if(isEmpty())

1. Output Queue Underflow

2. exit(1)

2. return(front->info)

Decimal To Binary Conversion

**void insert(int item)**

1. Declare struct node\*temp

2. temp=(struct node\*)malloc(sizeof(struct node))

3. if(temp==NULL)

1. Output No Enough Space

2. return

4. temp->bite=item

temp->link=NULL

5. if(front==NULL)

1. front=temp

else

1. rear->link=temp

7. rear=temp

**void display()**

1. struct node\*temp

2. while(front!=NULL)

1. Rpush(front->bits)

2 temp=front

3. front=front->link

4. free(temp)

3. Rpop()

4. declare struct node\*p=front

5. while(p!=NULL)

1. Output p->bits

2. p=p->link

**void Rpop()**

1. declare struct reversestack \*temp,\*p

2. p=top

3. while(p!=NULL)

1. insert(p->data)

2. p=p->link

**void Rpush(int x)**

1. struct reverseStack \*temp

2. temp=(struct node\*)malloc(sizeof(struct node))

3. temp->data=x

4. temp->link=top

5. top=temp

**void convert(int dec)**

1. while(dec!=0)

1. insert(dec%2)

2. dec=dec/2

**CODES: 1)**

#include<stdio.h>

#include<stdlib.h>

#define MAX 5

int Qarray[MAX];

int rear=-1;

int front=-1;

int isFull()

{

if(rear==MAX-1)

return 1;

else

return 0;

}

int isEmpty()

{

if(front==-1 ||front==rear+1)

return 1;

else

return 0;

}

void insert(int item)

{

if(isFull())

{

printf("QUEUE OVERFLOW\n");

return;

}

if(front==-1)

front=0;

rear++;

Qarray[rear]=item;

}

int del()

{

int item;

if(isEmpty())

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

item=Qarray[front];

front++;

return item;

}

int peek()

{

if(isEmpty())

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

return(Qarray[front]);

}

void display()

{

int i;

if(isEmpty())

{

printf("QUEUE IS EMPTY\n");

return;

}

printf("QUEUE IS : ");

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

{

printf("%d ",Qarray[i]);

}

printf("\n\n");

}

int main()

{

int choice,item;

while(1)

{

printf("\n\nARRAY IMPLEMENTATION OF A QUEUE\n");

printf("1: TO INSERT ELEMENTS INTO THE QUEUE\n");

printf("2: TO DELETE ELEMENTS FROM THE QUEUE\n");

printf("3: TO DISPLAY THE FRONT ELEMENT OF THE QUEUE\n");

printf("4: DISPLAY ALL THE ELEMENTS OF THE QUEUE\n");

printf("\nENTER YOUR CHOICE\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("ENTER THE ELEMENT: ");

scanf("%d",&item);

insert(item);

break;

case 2:

item=del();

printf("%d deleted",item);

break;

case 3:

item=peek();

printf("FRONT ELEMENT: %d",item);

break;

case 4:

display();

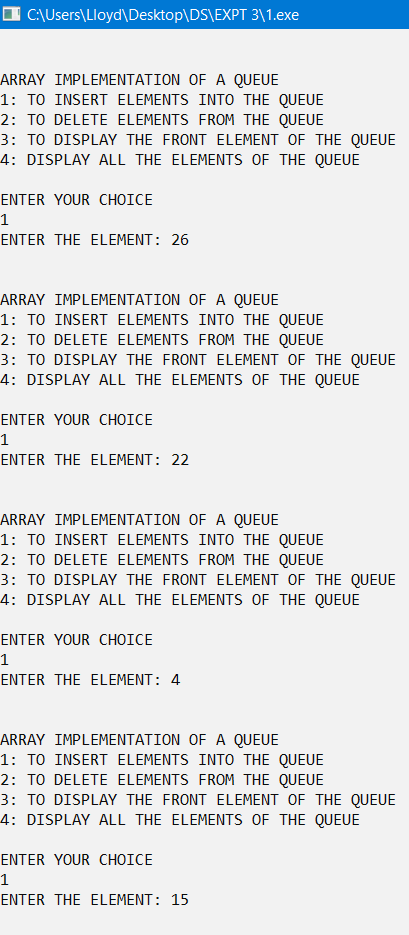
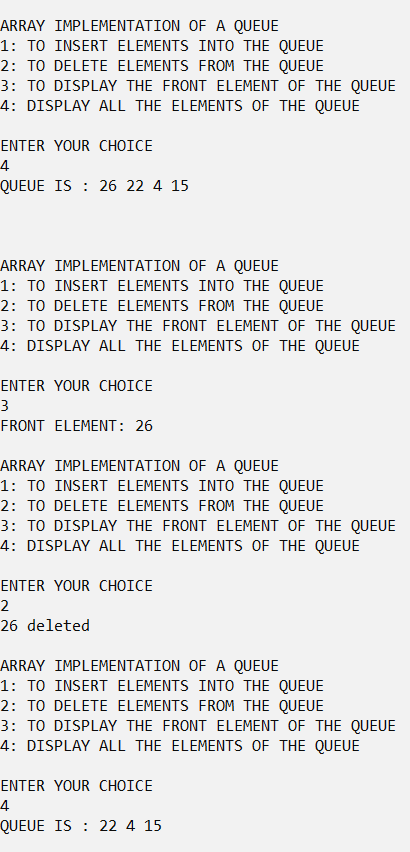
break;

}

}

}

**OUTPUT:**

****

**2)**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node \*link;

}\*front=NULL,\*rear=NULL;

int isEmpty()

{

if(front==NULL)

return 1;

else

return 0;

}

void insert(int item)

{

struct node \*temp;

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

if(temp==NULL)

{

printf("NO ENOUGH MEMORY\n");

return;

}

temp->info=item;

temp->link=NULL;

if(front==NULL)

front=temp;

else

rear->link=temp;

rear=temp;

}

int del()

{

struct node \*temp;

int item;

if(isEmpty())

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

temp=front;

item=temp->info;

front=front->link;

free(temp);

return item;

}

int peek()

{

if(isEmpty())

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

return(front->info);

}

void display()

{

struct node \*p;

p=front;

if(isEmpty())

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

printf("QUEUE ELEMENTS: ");

while(p!=NULL)

{

printf("%d ",p->info);

p=p->link;

}

printf("\n");

}

int main()

{

int choice,item;

while(1)

{

printf("\n\nLINKED LIST IMPLEMENTATION OF A QUEUE\n");

printf("1: TO INSERT ELEMENTS INTO THE QUEUE\n");

printf("2: TO DELETE ELEMENTS FROM THE QUEUE\n");

printf("3: TO DISPLAY THE FRONT ELEMENT OF THE QUEUE\n");

printf("4: DISPLAY ALL THE ELEMENTS OF THE QUEUE\n");

printf("\nENTER YOUR CHOICE\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("ENTER THE ELEMENT: ");

scanf("%d",&item);

insert(item);

break;

case 2:

item=del();

printf("%d deleted",item);

break;

case 3:

item=peek();

printf("FRONT ELEMENT: %d",item);

break;

case 4:

display();

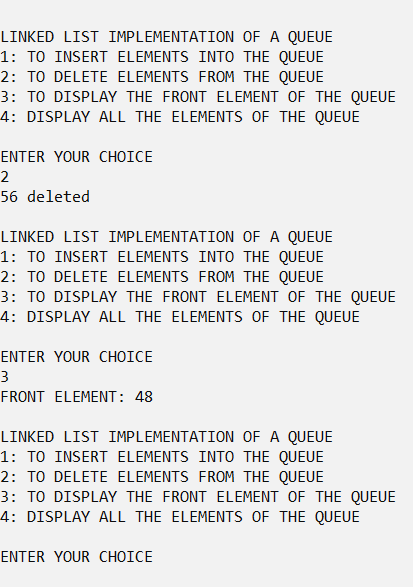
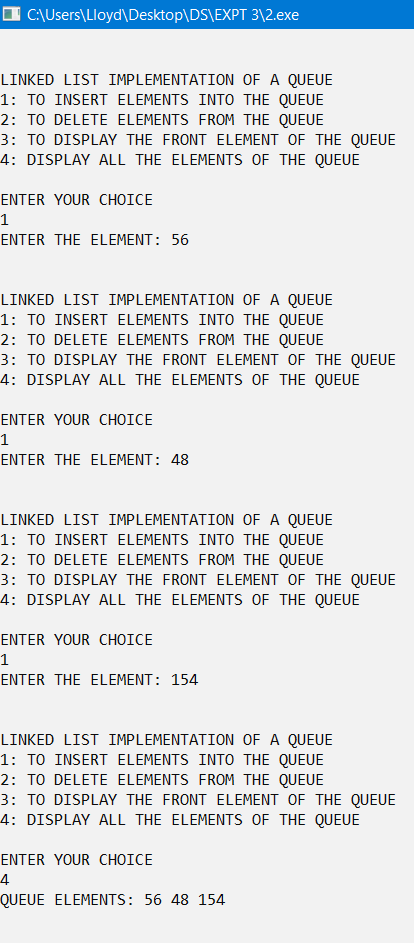
break;

}

}

}

**OUTPUT:**

****

**3)**

#include<stdio.h>

#include<stdlib.h>

struct node{

int info;

struct node \*link;

}\*rear=NULL;

int isEmpty()

{

if(rear==NULL)

return 1;

else

return 0;

}

void insert(int item)

{

struct node \*temp;

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

temp->info=item;

if(temp==NULL)

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

if(isEmpty())

{

rear=temp;

temp->link=rear;

}

else{

temp->link=rear->link;

rear->link=temp;

rear=temp;

}

}

int del()

{

int item;

struct node \*temp;

if(isEmpty())

{

printf("QUEUE UNDERFLOW \n");

exit(1);

}

if(rear->link==rear)

{

temp=rear;

rear=NULL;

}

else{

temp=rear->link;

rear->link=rear->link->link;

}

item=temp->info;

free(temp);

return item;

}

void display(){

struct node \*p;

if(isEmpty())

{

printf("QUEUE IS EMPTY\n");

return;

}

printf("QUEUE IS: ");

p=rear->link;

do{

printf("%d ",p->info);

p=p->link;

}while(p!=rear->link);

printf("\n");

}

int peek(){

if(isEmpty())

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

return rear->link->info;

}

int main()

{

int choice,item;

while(1)

{

printf("\n\nLINKED LIST IMPLEMENTATION OF A CIRCULAR QUEUE\n");

printf("1: TO INSERT ELEMENTS INTO THE QUEUE\n");

printf("2: TO DELETE ELEMENTS FROM THE QUEUE\n");

printf("3: TO DISPLAY THE FRONT ELEMENT OF THE QUEUE\n");

printf("4: DISPLAY ALL THE ELEMENTS OF THE QUEUE\n");

printf("\nENTER YOUR CHOICE\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("ENTER THE ELEMENT: ");

scanf("%d",&item);

insert(item);

break;

case 2:

item=del();

printf("%d deleted",item);

break;

case 3:

item=peek();

printf("FRONT ELEMENT: %d",item);

break;

case 4:

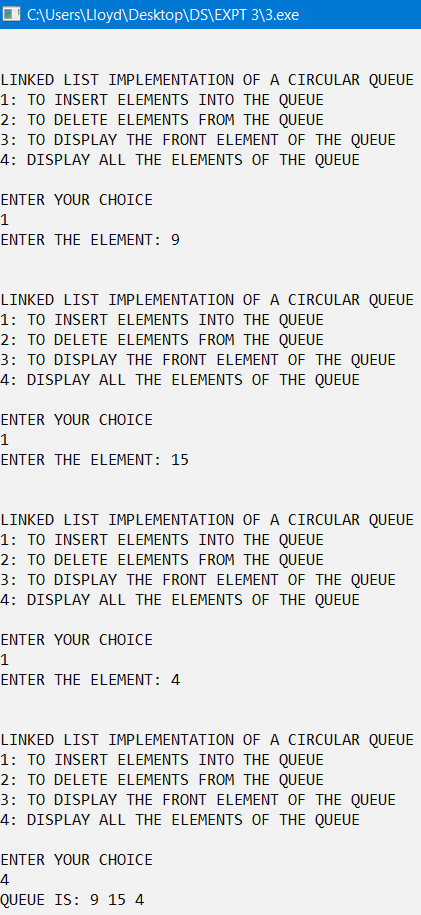
display();

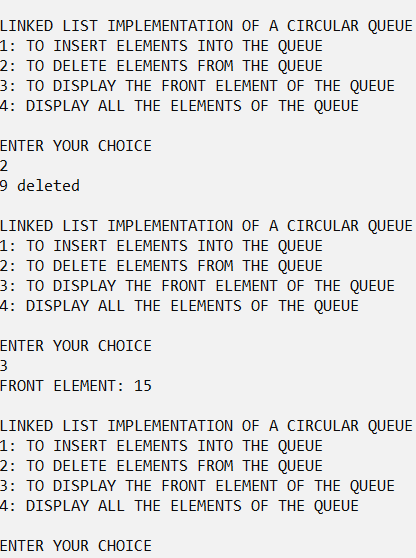
break;

}

}

}

**OUTPUT:**

****

**4)**

#include<stdio.h>

#include<stdlib.h>

#define max 5

int Cqueue[max];

int front = -1;

int rear = -1;

int isEmpty()

{

if(front==-1)

return 1;

else

return 0;

}

int isFull()

{

if((front==0 && rear==max-1)||(front==rear+1))

return 1;

else

return 0;

}

int peek()

{

if(isEmpty())

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

return Cqueue[front];

}

void display()

{

int i;

if(isEmpty())

{

printf("QEUEU IS EMPTY \n");

return;

}

printf("QUEUE IS: ");

i=front;

if(front<=rear)

{

while(i<=rear)

printf("%d ",Cqueue[i++]);

}

else{

while(i<=max-1)

printf("%d ",Cqueue[i++]);

i=0;

while(i<=rear)

printf("%d ",Cqueue[i++]);

}

printf("\n\n");

}

void insert(int item)

{

if(isFull())

{

printf("QUEUE OVERFLOW\n");

return;

}

if(front == -1)

front=0;

if(rear==max-1)

rear=0;

else

rear=rear+1;

Cqueue[rear]=item;

}

int del()

{

int item;

if(isEmpty())

{

printf("QUEUE UNDERFLOW\n");

exit(1);

}

item=Cqueue[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else if(front== max-1)

{

front=0;

}

else

front=front+1;

return item;

}

int main()

{

int choice,item;

while(1)

{

printf("\n\nARRAY IMPLEMENTATION OF A CIRCULAR QUEUE\n");

printf("1: TO INSERT ELEMENTS INTO THE QUEUE\n");

printf("2: TO DELETE ELEMENTS FROM THE QUEUE\n");

printf("3: TO DISPLAY THE FRONT ELEMENT OF THE QUEUE\n");

printf("4: DISPLAY ALL THE ELEMENTS OF THE QUEUE\n");

printf("\nENTER YOUR CHOICE\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("ENTER THE ELEMENT: ");

scanf("%d",&item);

insert(item);

break;

case 2:

item=del();

printf("%d deleted",item);

break;

case 3:

item=peek();

printf("FRONT ELEMENT: %d",item);

break;

case 4:

display();

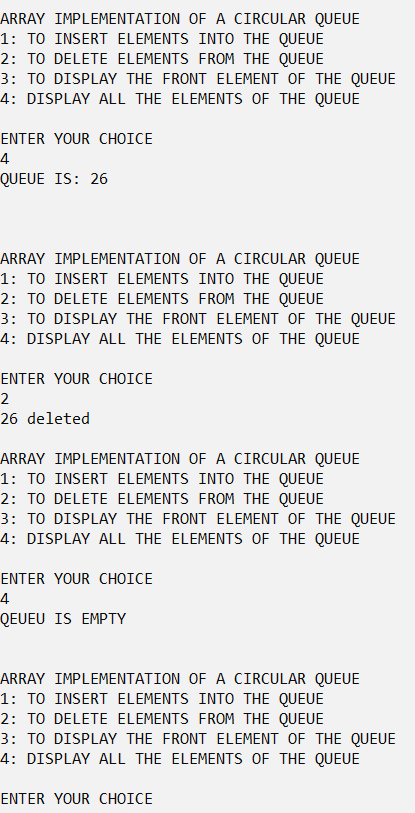
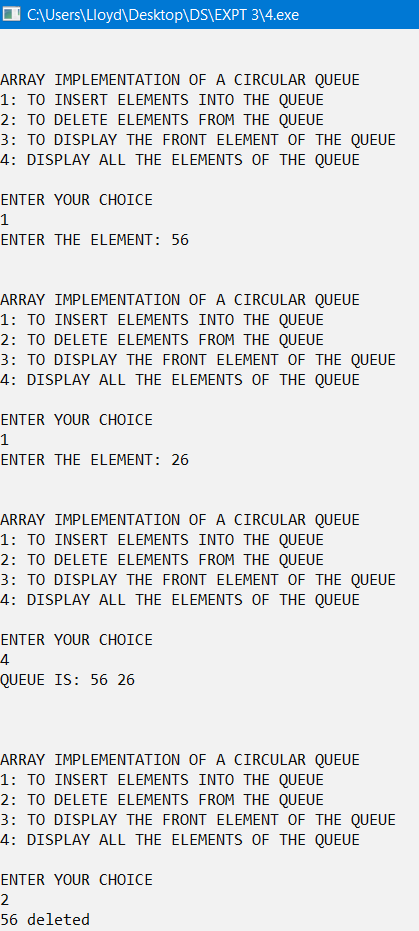
break;

}

}

}

**OUTPUT:**

****

**5)**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int bits;

struct node \*link;

}\*front=NULL,\*rear=NULL;

struct reverseStack

{

int data;

struct reverseStack \*link;

}\*top=NULL;

void insert(int item)

{

struct node \*temp;

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

if(temp==NULL)

{

printf("NO ENOUGH MEMORY\n");

return;

}

temp->bits=item;

temp->link=NULL;

if(front==NULL)

front=temp;

else

rear->link=temp;

rear=temp;

}

void convert(int dec)

{

while(dec!=0)

{

insert(dec%2);

dec=dec/2;

}

}

void Rpush(int x)

{ struct reverseStack \*temp;

temp=(struct reverseStack\*)malloc(sizeof(struct reverseStack));

temp->data=x;

temp->link=top;

top=temp;

}

void Rpop()

{

struct reverseStack \*temp,\*p;

p=top;

while(p!=NULL)

{

insert(p->data);

p=p->link;

}

}

void display()

{

struct node \*temp;

while(front!=NULL)

{

Rpush(front->bits);

temp=front;

front=front->link;

free(temp);

}

Rpop();

struct node \*p=front;

while(p!=NULL)

{

printf("%d",p->bits);

p=p->link;

}

}

int main()

{

int decimal;

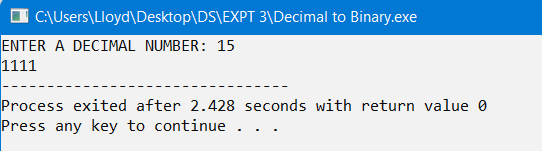
printf("ENTER A DECIMAL NUMBER: ");

scanf("%d",&decimal);

convert(decimal);

display();

}

**OUTPUT:**

**CONCLUSION:**

The given problem statements were successfully compiled and executed.

**LEARNINGS AND FINDINGS:**

1. Concept of Queue data structure.

2. Implementation of Queue.

3. Some applications of Queues.

The Queue data structure’s FIFO nature makes it one of the most widely used data structure in solving problems.