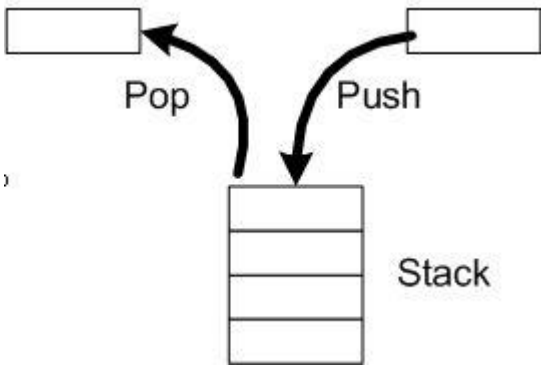
Unit-2: STACK and QUEUE

STACK [LIFO]

Basic Concept of Stack:



Definition:

Stack is **First-In-Last-Out** [FILO] or **Last-In-First-Out** [LIFO] Structure.

Explanation:

- Stack is Data Structure used to store the data in such a way that element inserted into the stack will be removed at last.
- A stack is a list of elements in which an element may be inserted or deleted only at one end called Top of Stack [TOS].
- Just take real time example, suppose we have created stack of the book like this shown in the following fig –

How Books are arranged in Stack? :

1. Books are kept one above the other
2. Book which is inserted first is Taken out at last.(Brown)
3. Book which is inserted Lastly is served first.(Light Green)

Common Example:

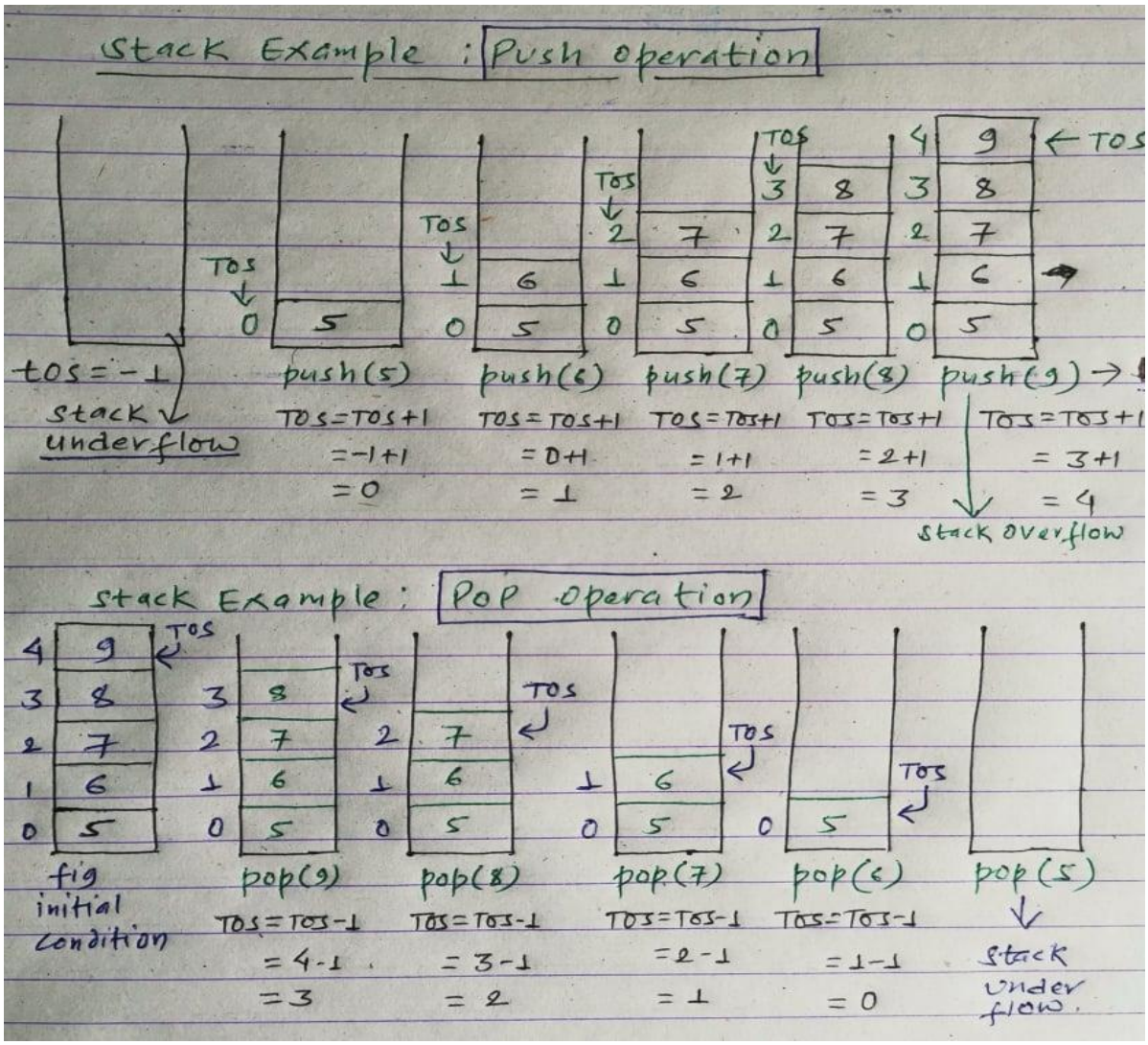
Suppose at your home you have multiple chairs then you put them together to form a vertical pile. From that vertical pile the chair which is placed last is always removed first.

Chair which was placed first is removed last. In this way we can see how stack is related to us.

Two basic operations associated with the stack are:

1. **Push:** This operation used to insert an element into a stack.
2. **Pop:** This operation used to delete an element from a stack.

Example of stack:



Algorithm of stack:

Step 1: Declare necessary variables
E.g. size=10, TOS= -1, stack[size]

Step 2: for "**Push Operation**"

- Check stack is full or not
  - if (stack is full) i.e TOS=size-1
    - Display "Stack is Overflow"
  - else i.e stack is not full
    - Read the data/element to be stored

- Increase TOS by 1 i.e. TOS==TOS+1
- stack[TOS]=new data

Step 3: for **"Pop Operation"**

- Check stack is empty or not
  - if (stack is empty) i.e TOS<0
    - Display "Stack is Underflow"
  - else i.e stack is not empty
    - Display value of stack[TOS]
    - Decrement TOS by 1 i.e. TOS=TOS-1

Step 4: Repeat step 2 and 3 according to the user's choice.
Step 5: Stop.

/* Stack lab number 1*/

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define max 5
int tos=-1,stack[max];
void push()
	{
		if(tos==max-1)
			printf("Stack is full\n");
		else
		{
			int a;
			printf("Enter the data to push in a stack:");
			scanf("%d",&a);
			tos=tos+1;
			stack[tos]=a;
		}
	}
void pop()
	{
		if(tos<0)
			printf("Stack is empty\n");
		else
```

```
		{
			printf("The data pop from a stack is: %d",stack[tos]);
			tos=tos-1;
		}

	}
int main()
	{
		int choice;
		clrscr();
		do
		{
			printf("\nEnter your choice \n1.Push \n2.Pop
\n3.Exit\n");
			scanf("%d",&choice);
			switch(choice)
			{
				case 1:
					push();
					break;
				case 2:
					pop();
					break;
				case 3:
					exit(0);
			}
			}while(choice<=3);
		getch();
		return 0;
	}
```

Applications of Stack

- Reversing a String
- Parentheses Matching
- Arithmetic Expressions
  - Infix to Postfix conversion
  - Infix to Prefix conversion
  - Evaluation of Postfix arithmetic expressions
- Matching HTML tags
- Number conversions from
  - Decimal to other base (2,8,16)

Questions:

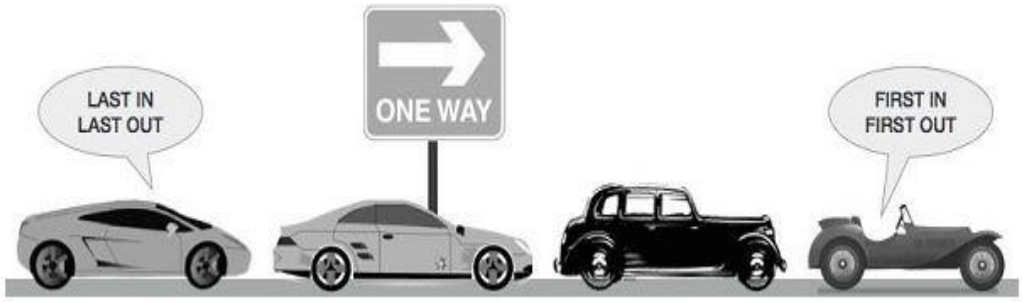
1. Explain the concept of stack with an example. 6 marks
2. Write an algorithm of stack. 6 marks
3. Write down the application of stack. 3 marks

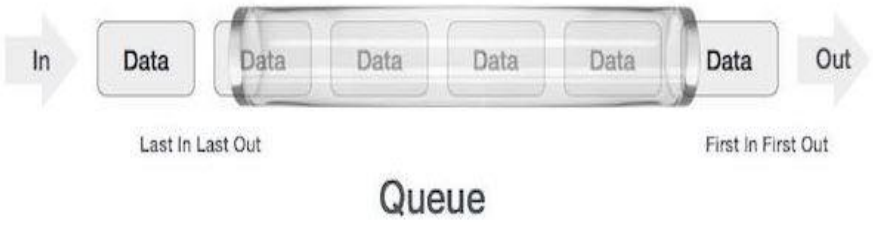
Note:

- **Do above questions in a note copy.**
- **Make a lab report of Stack in a4 paper.**

Queue: [FIFO]

- Queue is an abstract data structure, somewhat similar to Stacks.
- Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other end is used to remove data (dequeue).
- **The insertion end is called rear and deletion end is called front.**
- Queue follows First-In-First-Out [FIFO] methodology.
- **Queue example:**





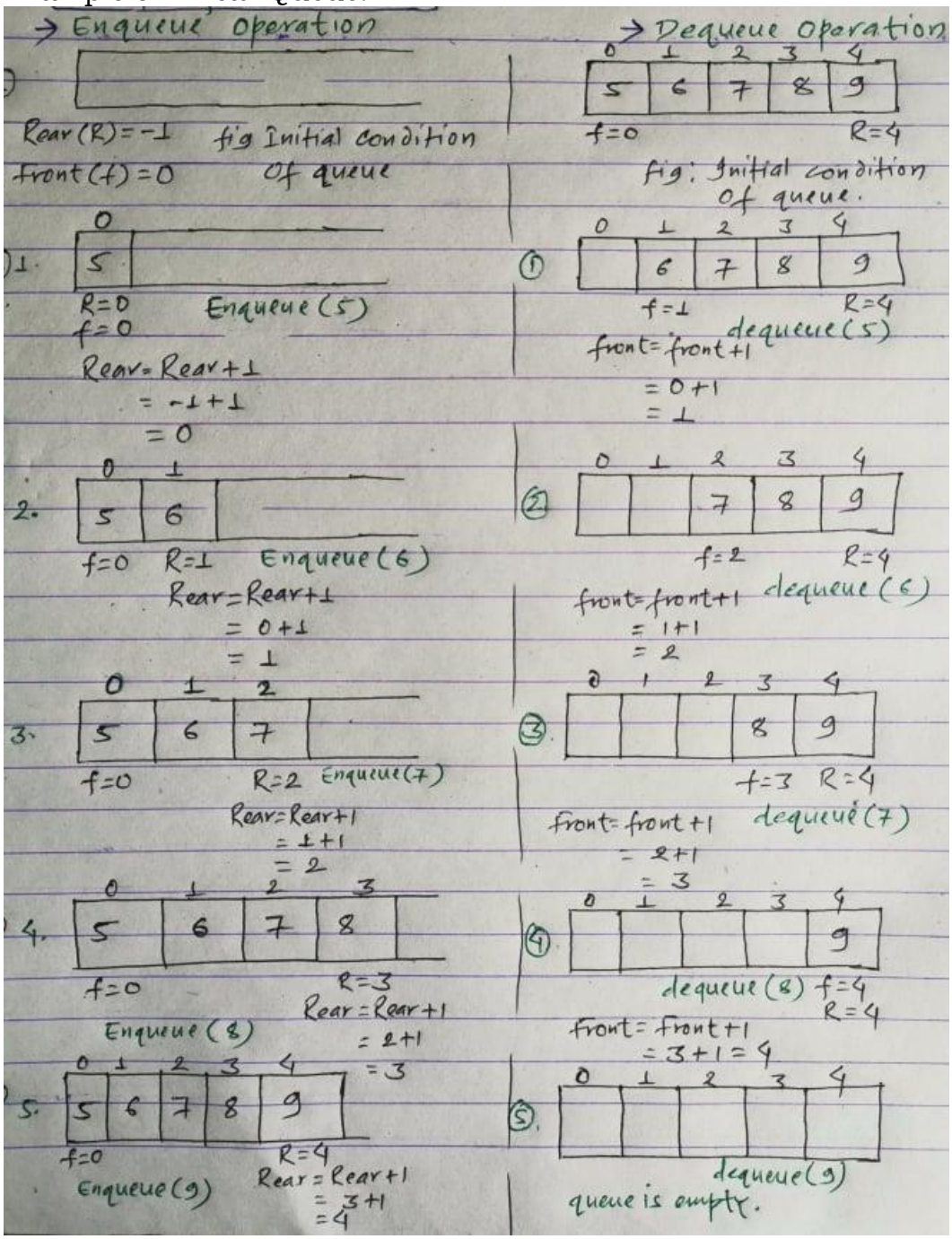
Queue

- A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first.
- More real-world examples can be seen as queues at the movies, train, airways, token/ticket windows and bus-stops.
- The difference between stacks and queues is in removing. In a stack we remove the item the most recently added but where as in a queue, we remove the item the least recently added.

Types of queue:

1. Linear queue: What is linear queue?

- It is linear list of elements or data.
- Insertion is performed from **REAR** end
- Deletion is performed from **FRONT** end.
- Insertion operation is also known as **ENQUEUE**.
- Deletion operation is also known as **DEQUEUE.**
- Queue follows First-In-First-Out [FIFO].
- A good example of a linear queue is any queue of consumers in a bank for a resource where the consumer that came first is served first.
- Example of Linear Queue:



Operations on Linear Queue

There are two operations that can be performed on a linear queue:

- **Enqueue:** The enqueue operation inserts the new element from the rear end.
- **Dequeue:** The dequeue operation is used to delete the existing element from the front end of the queue.

Algorithm of Linear queue: FIFO

1. Declare necessary variable:
   size=5, front=0, rear=-1, queue[size], count=0

2. For **Enqueue** operation:
   Check queue full or not
   If queue is full i.e. **rear==size-1**

   Display "**The queue is full**" message.
   Stop.

   else **or if queue is not full**

   Read the data/element to be stored
   Increment rear by 1 **i.e. rear=rear+1**
   queue[rear]=newdata
   Increment count by 1 **i.e. count++**

3. To enqueue next data, repeat step 2.

4. For **Dequeue** operation:
   Check queue is empty or not
   If queue is empty i.e. **rear<front**

   Display "**The queue is empty**" message.
   Stop.

   else **or if queue is not empty**
   Display the value of **queue[front]**
   Increment front by 1 **i.e front=front+1**
   Decrement count by 1 **i.e. count--**

5. To dequeue next data, repeat step 4.
6. Stop

Drawback of Linear Queue

The linear queue suffers from serious drawback that performing some operations, we cannot insert items into queue, even if there is space in the queue. Suppose we have queue of 5 elements and we insert 5 items into queue, and then delete some items, then queue has space, but at that condition we cannot insert items into queue.

/*Linear queue i.e lab number 2*/

Method 1:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define max 5
int rear=-1,front=0,count=0;
int queue[max];
void enqueue()
{
        if(rear==max-1)
                printf("The queue is full\n");
        else
        {
                int a;
                printf("Enter the data to be inserted\n");
                scanf("%d",&a);
                rear=rear+1;
                queue[rear]=a;
                count++;
        }
}

void dequeue()
{
        if(rear<front)
                printf("The queue is empty\n");
        else
        {
                printf("The data deleted from the queue is %d\n",queue[front]);
                front++;
                count--;
        }
}
int main()
{
        int choice;
        clrscr();
```

```
        do
        {
        if(count==0)
                {
                        rear=-1;
                        front=0;
                }
        printf("\nEnter your choice:\n1.Enqueue\n2.Dequeue\n3.Exit\n");
        scanf("%d",&choice);
        switch(choice)
                {
                        case 1:
                                enqueue();
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                exit(0);
                }
        }while(choice<=3);
        getch();
        return 0;
}
```

/*Linear queue i.e lab number 2*/

Method: 2

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define max 5
int rear=0,front=0;
int queue[max];
void enqueue()
{
        if(rear==max)
                printf("The queue is full\n");
        else
        {
                int a;
                printf("Enter the data to be inserted\n");
                scanf("%d",&a);
                queue[rear]=a;
                rear=rear+1;
        }
}
```

```
void dequeue()
{
        if(rear==front)
                printf("The queue is empty\n");
        else
        {
                printf("The data deleted from the queue is %d\n",queue[front]);
                front++;
        }
}

int main()
{
        int choice;
        clrscr();
        do
        {
        printf("\nEnter your choice:\n1.Enqueue\n2.Dequeue\n3.Exit\n");
        scanf("%d",&choice);

        switch(choice)
                {
                        case 1:
                                enqueue();
                                break;

                        case 2:
                                dequeue();
                                break;
                        case 3:
                                exit(0);
                }
        }while(choice<=3);
        getch();
        return 0;
}
```

2. Circular queue:

What is circular queue?

- As we know that in a queue, the front pointer points to the first element while the rear pointer points to the last element of the queue.

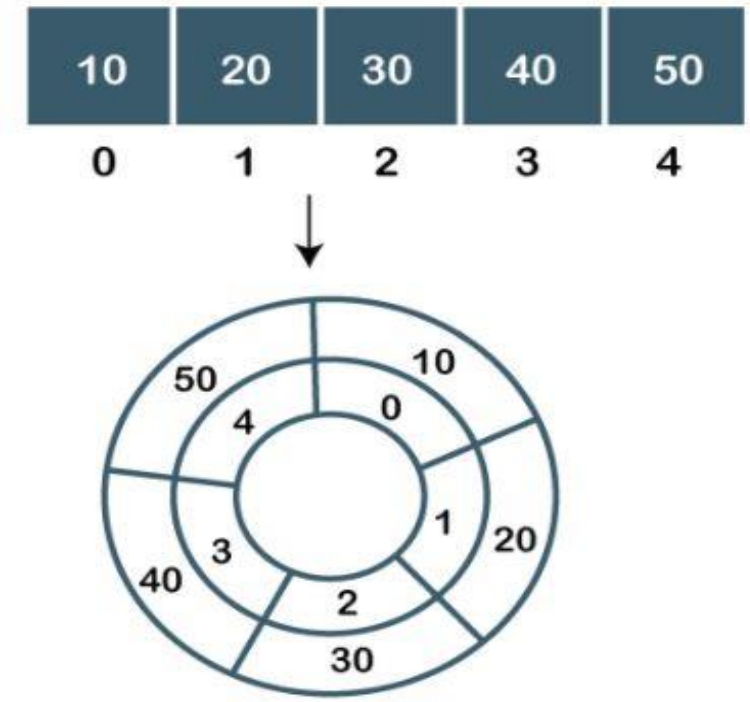
- The problem that arises with the linear queue is that if some empty cells occur at the beginning of the queue then we cannot insert new

element at the empty space as the rear cannot be further incremented.

- A circular queue is also a linear data structure like a normal queue that follows the FIFO principle but it does not end the queue; it connects the last position of the queue to the first position of the queue. If we want to insert new elements at the beginning of the queue, we can insert it using the circular queue data structure.

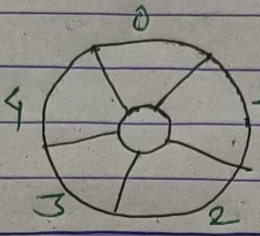
- In the circular queue, when the rear reaches the end of the queue, then rear is reset to zero. It helps in refilling all the free spaces.

- Circular queue fig:



- **Circular queue example:**

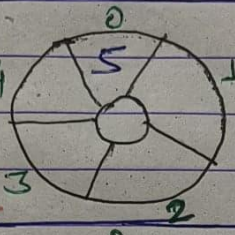
Enqueue Operation in Circular queue



front (f) = 0
Rear (r) = ~~0~~ -1
count = 0
size = 5

fig: initial condition

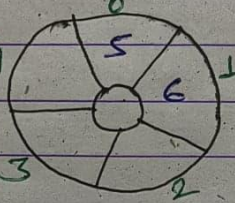
① Enqueue(5)
f = 0
r = 0
c = 1



r = (r+1) % 5
= (-1+1) % 5
= 0 % 5
= 0

5)0(0
0
0

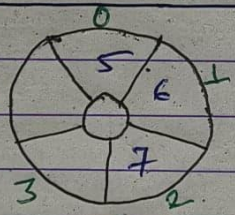
② Enqueue(6)
f = 0
r = 1
c = 2



r = (r+1) % 5
= 1 % 5
= 1

5)1(0
0
1

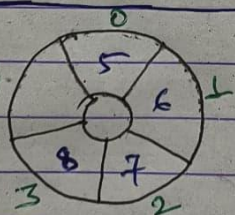
③ Enqueue(7)
f = 0
r = 2
c = 3



r = (r+1) % 5
= (1+1) % 5
= 2 % 5
= 2

5)2(0
0
2

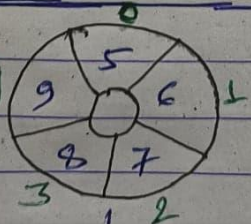
④ Enqueue(8)
f = 0
r = 3
c = 4



r = (r+1) % 5
= (2+1) % 5
= 3 % 5
= 3

5)3(0
0
3

⑤ Enqueue(9)
f = 0
r = 4
c = 5



r = (r+1) % 5
= (3+1) % 5
= 4 % 5
= 4

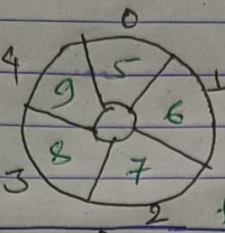
5)4(0
0
4

queue overflow

fig: final condition.

Dequeue Operation in circular queue

f=0
r=4
c=5
s=5

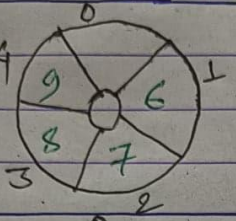


if(c==s)
queue overflow/full

fig: Initial condition

dequeue(5)

Fig ① f=1, r=4, c=4

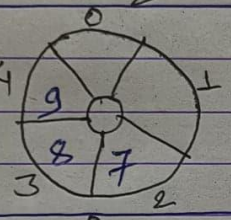


f=(f+1)%.5
=(0+1)%.5
=1%.5
=1

5)1(0
0
1

dequeue(6)

Fig ② f=2, r=4, c=3

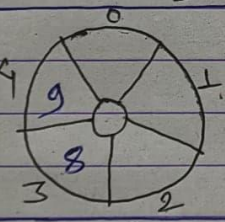


f=(f+1)%.5
=(1+1)%.5
=2%.5
=2

5)2(0
0
2

dequeue(7)

Fig ③ f=3, r=4, c=2

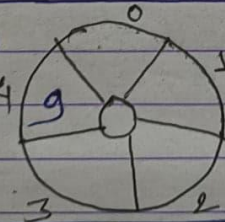


f=(f+1)%.5
=(2+1)%.5
=3%.5
=3

5)3(0
0
3

dequeue(8)

Fig ④ f=4, r=4, c=1

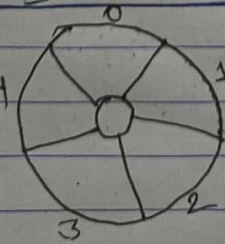


f=(f+1)%.5
=(3+1)%.5
=4%.5
=4

5)4(0
0
4

dequeue(9)

Fig ⑤ f=0, r=4, c=0



f=(f+1)%.5
=(4+1)%.5
=5%.5
=0

5)5(1
5
0

if(c==0) queue empty.

Algorithm of Circular queue: FIFO

1. Declare necessary variable:
   i.e size=5, front=0, rear=-1, queue[size], count=0
2. For **Enqueue** operation:
   Check queue full condition **i.e count==size**
   If queue is full

   Display "**The queue is full**" message.
   Stop.
   else
   Read the data to be inserted [DTBI]
   Increment the rear by 1 **i.e rear=(rear+1)%size;**
   Store DTBI in queue[rear] **i.e queue[rear]=newdata**
   Increment the count by 1 **i.e count++;**

3. To enqueue next data, repeat step 2.

4. For **Dequeue** operation:
   Check queue empty condition **i.e count==0**
   If queue is empty

   Display "**The queue is empty**" message.
   Stop.
   else
   Display the value of **queue[front]** i.e Display "**The data deleted from the queue is %d**" message.
   Increment the front by 1 **i.e front=(front+1)%size;**
   Decrement the count by 1 **i.e count--;**

5. To dequeue next data, repeat step 4.

```
/* Circular queue lab number 3*/
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define size 5
int front=0,rear=-1,count=0;
int queue[size];

void enqueue()
    {
        if(count==size)
                {
                        printf("Queue is full\n");
                }
        else
                {
                        int data;
                        printf("Insert a data into a circular queue:\n");
                        scanf("%d",&data);
                        rear=(rear+1)%size;
                        queue[rear]=data;
                        count++;
                }
    }
void dequeue()
     {
        if(count==0)
                {
                        printf("Queue is empty\n");
                }
        else
                {
                        int data;
                        data=queue[front];
                        printf("The dequeue data is: %d",data);
                        front=(front+1)%size;
                        count--;
                }
    }
int main()
{
        int choice;
        clrscr();
        do
        {
        printf("\nEnter your choice\n1.Enqueue\n2.Dequeue\n3.Exit\n");
                scanf("%d",&choice);
                switch(choice)
                {
```

```
                case 1:
                        enqueue();
                        break;

                case 2:
                        dequeue();
                        break;

                case 3:
                        exit(0);
                }
        }while(choice<=3);
        getch();
        return 0;
}
```

Differentiate between linear and circular queue.

Basis of comparison	Linear Queue	Circular Queue
Meaning	The linear queue is a type of linear data structure that contains the elements in a sequential manner.	The circular queue is also a linear data structure in which the last element of the Queue is connected to the first element, thus creating a circle.
Insertion and Deletion	In linear queue, insertion is done from the rear end, and deletion is done from the front end.	In circular queue, the insertion and deletion can take place from any end.
Memory space	The memory space occupied by the linear queue is more than the circular queue.	It requires less memory as compared to linear queue.
Memory utilization	The usage of memory is inefficient.	The memory can be more efficiently utilized.
Order of execution	It follows the FIFO principle in order to perform the tasks.	It has no specific order for execution.

3. Priority queue:

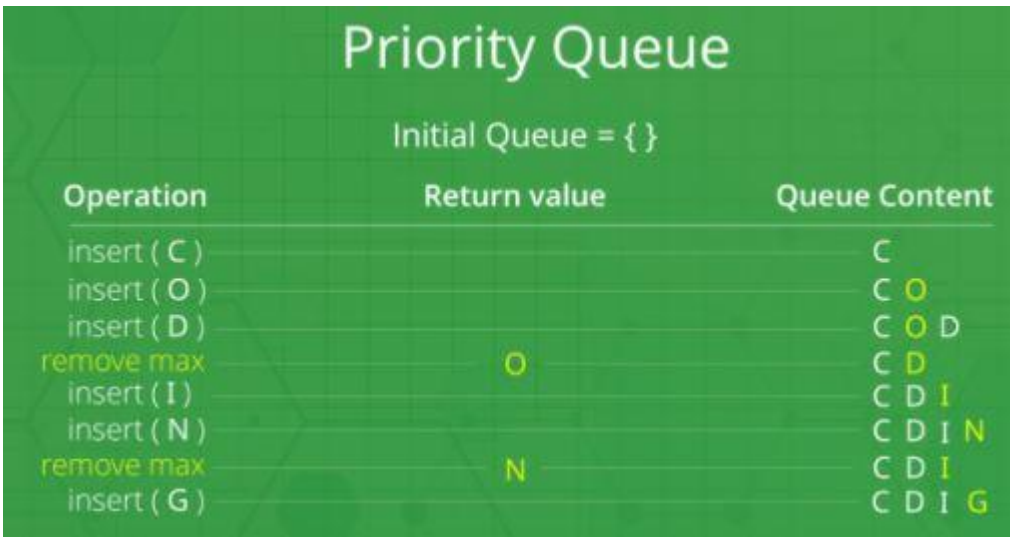
Priority Queue is an extension of queue with following properties.

a. Every item has a priority associated with it.
b. An element with high priority is dequeue before an element with low priority.
c. If two elements have the same priority, they are served according to their order in the queue i.e first come first serve.

So it is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted and processed comes from the rules:

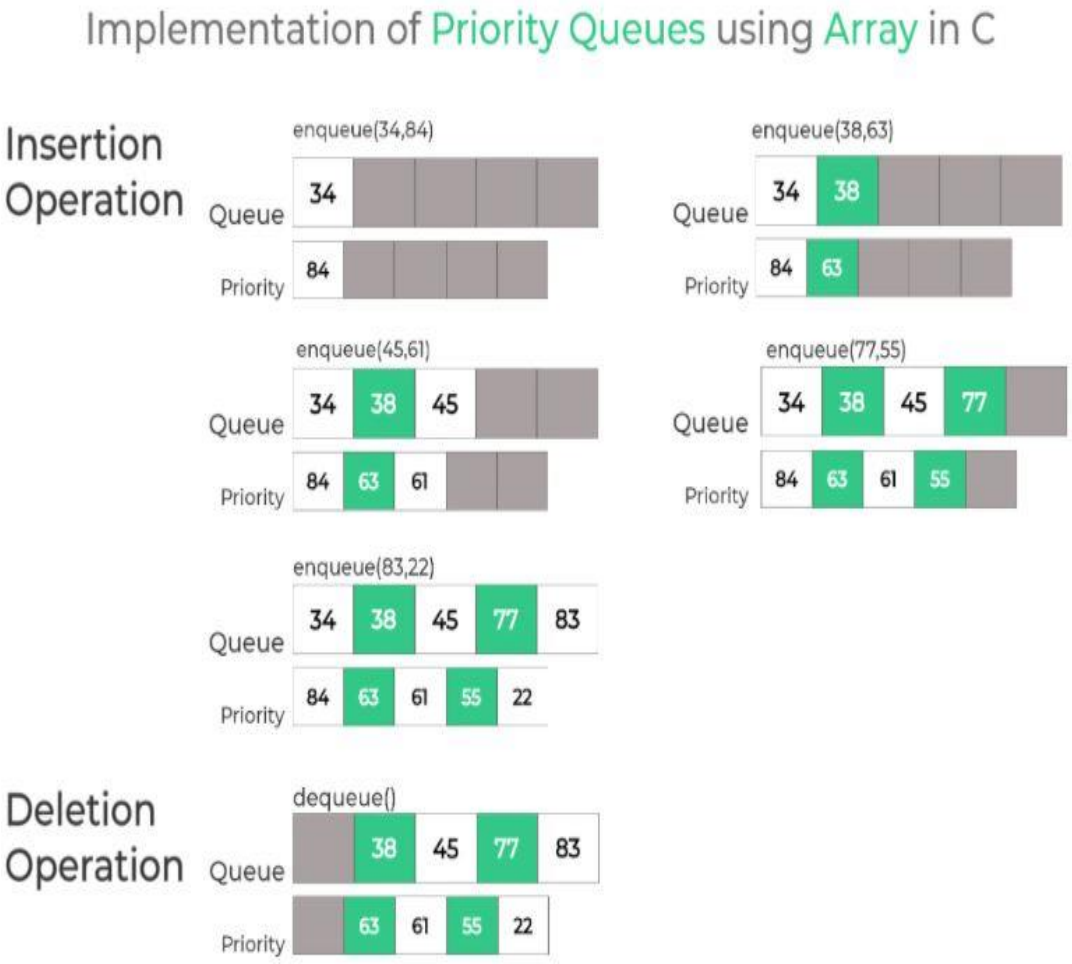
i. An element of higher priority is processed before any element of lower priority.
ii. Two elements with same priority are processed according to the order they were added to the queue.

In the below priority queue, element with maximum ASCII value will have the highest priority.

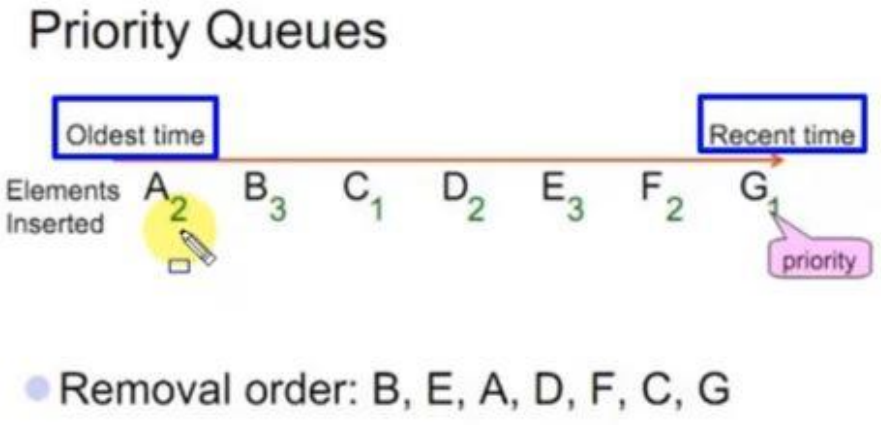


In a queue, the **first-in-first-out rule** is implemented whereas, in a priority queue, the values are removed **on the basis of priority**. The element with the highest priority is removed first.

Example of Priority queue:



Note: If two elements have the same priority then removal of elements will as follows:



Algorithm to implement Priority Queue using Array

- **Enqueue()**

```
1. IF((Front == 0)&&(Rear == N-1))
2. PRINT "Overflow Condition"
3. Else
4. IF(Front == -1)
5. Front = Rear =0
6. Queue[Rear] = Data
7. Priority[Rear] = Priority
8. ELSE IF(Rear ==N-1)
9. FOR i=Front;i<=Rear;i++)
10. FOR(i=Front;i<=Rear;i++)
11. Q[i-Front] =Q[i]
12. Pr[i-Front] = Pr[i]
13. Rear = Rear-Front
14. Front = 0
15. FOR(i = r;i>f;i−)
16. IF(p>Pr[i])
17. Q[i+1] = Q[i] Pr[i+1] = Pr[i]
18. ELSE
19. Q[i+1] = data Pr[i+1] = p
20. Rear++
```

- **Dequeue()**

```
1. IF(Front == -1)
2. PRINT "Queue Under flow
   condition"
3. ELSE
4. PRINT"Q[f],Pr[f]"
5. IF(Front==Rear)
6. Front = Rear = -1
7. ELSE
8. FRONT++
```

- **Print()**

```
1. FOR(i=Front;i<=Rear;i++)
2. PRINT(Q[i],Pr[i])
```

/*Priority queue program*/

```
#include<stdio.h>
#include<conio.h>
#define N 20
int Q[N],Pr[N];
int r = -1, f = -1;
void enqueue(int data,int p)    //Enqueue function to insert data and
its priority in queue
{
        int i;
        if((f==0)&&(r==N-1))        //Check if Queue is full
                printf("Queue is full\n");
        else
        {
                if(f==-1)       //if Queue is empty
                {
                        f = r = 0;
                        Q[r] = data;
                        Pr[r] = p;

                }
```

```
                else if(r == N-1)     //if there there is some elemets in
Queue
                {
                        for(i=f;i<=r;i++)
                        {
                                Q[i-f] = Q[i]; Pr[i-f] = Pr[i]; r = r-f; f = 0; for(i =
r;i>f;i--)
                                {
                                        if(p>Pr[i])
                                        {
                                                Q[i+1] = Q[i];
                                                Pr[i+1] = Pr[i];
                                        }
                                        else
                                                break;
                                        Q[i+1] = data;
                                        Pr[i+1] = p;
                                        r++;
                                }
                        }
                }
                else
                {
                        for(i = r;i>=f;i--)
                        {
                                if(p>Pr[i])
                                {
                                        Q[i+1] = Q[i];
                                        Pr[i+1] = Pr[i];
                                }
                                else
                                        break;
                        }
                        Q[i+1] = data;
                        Pr[i+1] = p;
                        r++;
                }
        }

}
void print() //print the data of Queue
{
int i;
        for(i=f;i<=r;i++)
        {
                printf("\nElement = %d\tPriority = %d",Q[i],Pr[i]);
        }
}
int dequeue() //remove the data from front
```

```
{
        if(f == -1)
        {
                printf("Queue is Empty\n");
        }
        else
        {
                printf("deleted Element = %d\t Its Priority = %d",Q[f],Pr[f]);
                if(f==r)
                        f = r = -1;
                else
                        f++;
        }
}
int main()
{
        int opt,n,i,data,p;
        clrscr();
        printf("Enter Your Choice:-");
        do{
                printf("\n\n1 Insert the Data in Queue\n2 show the Data
in Queue \n3 Delete the data from the Queue\n0 for Exit\n");
                scanf("%d",&opt);
                switch(opt){
                        case 1:
                                printf("\nEnter the number of data");
                                scanf("%d",&n);
                                printf("\nEnter your data and Priority of
data");
                                i=0;
                                while(i<n){
                                        scanf("%d %d",&data,&p);
                                        enqueue(data,p);
                                        i++;
                                }
                                break;
                        case 2:
                                print();
                                break;
                        case 3:
                                 dequeue();
                                break;
                        case 0:
                                break;
                        default:
                                printf("\nIncorrect Choice");

                }
        }while(opt!=0);
```

```
        return 0;
}
```

Application of queue:

Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in also gets out first while the others wait for their turn, like in the following scenarios:

1. Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.

2. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrives i.e First come first served.

3. Round robin scheduling technique is implemented using queue

4. Real world applications are: checkout at any railways and airways stations, persons waiting line in a bank for cash, ATM card, cheque deposit etc. Line at ticket counter at cinema hall, In Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.

Questions:

1. Explain the concept of queue with an example. 6 marks
2. What is linear queue? Explain with an example. 6 marks
3. Write down the algorithm of linear queue. 6 marks
4. Explain circular with an example. 6 marks
5. Write an algorithm to implement circular queue. 6 marks
6. What is priority queue? Explain with an example. 6 marks
7. Write down the application of queue? 3 marks

Note:

- **Do above questions in a note copy.**
- **Make a lab report of linear queue, circular queue and priority queue in a4 paper.**