MOST IMPORTANT QUESTIONS 🔥

7-Mark Questions

Define Stack and explain operations.

Explain stack implementation using array.

Explain infix to postfix conversion.

Explain postfix evaluation.

What is recursion?

Define Queue and explain operations.

Explain circular queue.

Explain DQUEUE.

Explain priority queue.

Differentiate between stack and queue.

RGPV 7-mark format answers.

1. Define Stack and explain operations.

Stack ek linear data structure hai jisme insertion aur deletion sirf ek end se hota hai. Is end ko **TOP** kehte hain. Stack **LIFO** principle follow karta hai.

LIFO = Last In First Out

Matlab jo element last me insert hota hai, wahi sabse pehle delete hota hai.

Example: plates ka stack. Jo plate last me rakhi hai, wahi pehle niklegi.

Main operations of stack:

1. Push: Stack me new element insert karna push kehlata hai.

```
push(10)
push(20)
push(30)
```

```
Stack = 10, 20, 30
TOP = 30
```

2. Pop: Stack ke top element ko delete karna pop kehlata hai.

```
pop()

Removed element = 30
```

3. Peek / Top: Top element ko bina delete kiye dekhna.

```
peek() = 30
```

4. isEmpty: Check karta hai stack empty hai ya nahi.

5. isFull: Check karta hai stack full hai ya nahi.

Overflow: Jab stack full ho aur push operation karein.

Underflow: Jab stack empty ho aur pop operation karein.

Applications:
Recursion, expression conversion, expression evaluation, undo/redo, browser back button.

Conclusion: Stack ek important data structure hai jo LIFO rule follow karta hai aur many computer applications me use hota hai.

---

2. Explain stack implementation using array.

Stack ko array ke help se implement kiya ja sakta hai. Array implementation me ek fixed-size array use hota hai aur ek variable `top` use hota hai jo stack ke top element ko point karta hai.

Initially:

```
int stack[100];
int top = -1;
```

Yaha `top = -1` ka matlab stack empty hai.

Push Operation Algorithm:

```
Step 1: If top == size - 1
        Print Overflow
Step 2: Else
        top = top + 1
Step 3: stack[top] = item
Step 4: Stop
```

Pop Operation Algorithm:

```
Step 1: If top == -1
        Print Underflow
Step 2: Else
        item = stack[top]
Step 3: top = top - 1
Step 4: Stop
```

Example:

```
Push 10 → top = 0
Push 20 → top = 1
Push 30 → top = 2
```

```
Stack:
10
20
30
```

Now pop:

```
Pop removes 30
top becomes 1
```

Advantages of array implementation:

Simple, easy to implement, fast access.

Disadvantages:

Fixed size, memory wastage possible, overflow occurs when array is full.

Conclusion: Array implementation of stack is simple and efficient, but size is fixed.

---

3. Explain infix to postfix conversion.

Infix expression me operator operands ke beech hota hai.

Example:

```
A + B
```

Postfix expression me operator operands ke baad hota hai.

Example:

```
A B +
```

Computer ke liye postfix expression evaluate karna easy hota hai, isliye infix ko postfix me convert kiya jata hai.

Operator precedence:

```
^      highest
* /    medium
+ -    lowest
```

Algorithm for infix to postfix conversion using stack:

```
Step 1: Expression ko left to right scan karo.
Step 2: Agar operand aaye, output me add karo.
Step 3: Agar '(' aaye, stack me push karo.
Step 4: Agar operator aaye:
        stack ke top par higher/equal precedence operator ho to pop karke output me add karo.
        phir current operator push karo.
Step 5: Agar ')' aaye:
        '(' milne tak stack se pop karke output me add karo.
Step 6: End me stack ke remaining operators pop karke output me add karo.
```

Example:

```
Infix: A + B * C
```

Scan:

A → output

- → stack

  B → output

- → stack because * has higher priority

  C → output

  End → pop operators

```
Postfix: A B C * +
```

Another example:

```
Infix: (A + B) * C
Postfix: A B + C *
```

Why stack is used?

Stack operators ko temporarily store karta hai aur precedence ke according unhe output me bhejta hai.

Conclusion: Infix to postfix conversion stack ka important application hai, jo expression evaluation ko easy banata hai.

4. Explain postfix evaluation.

Postfix expression me operator operands ke baad likha jata hai.
Postfix evaluation ke liye stack ka use kiya jata hai.

Example:

```
23*5+
```

Meaning:

```
(2*3)+5
```

Algorithm for Postfix Evaluation

```
Step 1: Expression ko left to right scan karo.

Step 2: Agar operand aaye to stack me push karo.

Step 3: Agar operator aaye:
        top ke do operands pop karo.
        operation perform karo.
        result ko stack me push karo.

Step 4: Expression complete hone par stack ka final value answer hota hai.
```

Example

Expression

```
23*5+
```

Stepwise Evaluation

Step 1

2 → push

Stack:

```
2
```

Step 2

3 → push

Stack:

```
2 3
```

Step 3

- operator

Pop:
3 and 2

Perform:

```
2 × 3 = 6
```

Push 6

Stack:

```
6
```

Step 4

5 → push

Stack:

```
6 5
```

Step 5

- operator

Pop:
5 and 6

Perform:

```
6 + 5 = 11
```

Push 11

Stack:

11

Final Answer:

11

Applications

- Calculator
- Compiler design
- Expression evaluation

Advantages

- No brackets required
- Faster evaluation
- Easy for computers

Conclusion

Postfix evaluation stack ka important application hai jisme operands ko stack me store karke operators ke according result calculate kiya jata hai.

---

5. What is Recursion?

Recursion ek process hai jisme function khud ko hi call karta hai.

Recursive function tab tak repeatedly call hota hai jab tak stopping condition satisfy nahi hoti.

Syntax

```
function()
{
    if(condition)
        return;

    function();
}
```

Example: Factorial using Recursion

```
int fact(int n)
{
    if(n==0)
        return 1;

    return n * fact(n-1);
}
```

Working

Suppose:

```
fact(4)
```

then:

```
4 × fact(3)
3 × fact(2)
2 × fact(1)
1 × fact(0)
```

Since:

```
fact(0) = 1
```

Final Answer:

```
4 × 3 × 2 × 1 = 24
```

Types of Recursion

1. Direct Recursion

Function directly khud ko call kare.

---

2. Indirect Recursion

Ek function dusre function ko call kare aur second function first ko call kare.

Advantages

- Code simple hota hai
- Tree and graph problems me useful
- Complex problems ko easy banata hai

Disadvantages

- Extra memory use hoti hai
- Stack overflow ho sakta hai
- Slow ho sakta hai

Applications

- Tree traversal
- Factorial
- Fibonacci series
- Tower of Hanoi

Conclusion

Recursion ek powerful technique hai jisme function repeatedly khud ko call karta hai aur stack memory ka use hota hai.

---

6. Define Queue and explain operations.

Queue ek linear data structure hai jo FIFO principle follow karta hai.

FIFO means:

```
First In First Out
```

Matlab jo element pehle insert hoga wahi pehle delete hoga.

Insertion rear side se hota hai aur deletion front side se hota hai.

Real Life Example

- Ticket counter
- Printer queue
- Railway reservation

Queue Operations

---

1. Enqueue

Queue me new element insert karna.

Example:

```
Enqueue(10)
Enqueue(20)
Enqueue(30)
```

Queue:

```
10 20 30
```

---

2. Dequeue

Front element delete karna.

Example:

```
Dequeue()
```

Deleted element:

```
10
```

Queue becomes:

```
20 30
```

---

3. Peek / Front

Front element ko dekhna without deleting.

---

4. isEmpty()

Check karta hai queue empty hai ya nahi.

---

5. isFull()

Check karta hai queue full hai ya nahi.

Queue Representation

```
FRONT → 10 20 30 ← REAR
```

Applications

- CPU scheduling
- Printer spooling
- BFS traversal
- Network buffering

Advantages

- Fair processing
- Easy implementation

Disadvantages

- Fixed size in arrays
- Memory wastage possible

Conclusion

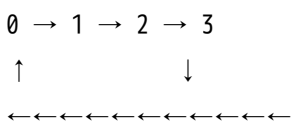
Queue ek important FIFO data structure hai jo scheduling and buffering applications me widely use hota hai.

7. Explain Circular Queue.

Circular Queue ek special type ki queue hai jisme last position first position se connected hoti hai.

Normal queue me memory wastage hota hai because deleted positions reuse nahi hoti. Circular queue is problem ko solve karta hai.

Structure of Circular Queue



Last position first se connected hoti hai.

FIFO Principle

Circular queue bhi FIFO follow karta hai.

```
First In First Out
```

Operations

1. Enqueue

Element insert karna.

---

2. Dequeue

Element delete karna.

Condition for Full Queue

```
(front == rear + 1)
```

OR

```
(front == 0 && rear == size-1)
```

Condition for Empty Queue

```
front == -1
```

Example

Suppose queue size = 5

Insert:

```
10 20 30 40
```

Delete two elements:

```
10 and 20 removed
```

Now free spaces available at beginning.

Circular queue reuses these spaces.

Advantages

- Better memory utilization
- No memory wastage
- Faster operations

Disadvantages

- Implementation slightly complex

Applications

- CPU scheduling
- Keyboard buffering
- Traffic systems
- Printer queue

Conclusion

Circular queue memory wastage problem ko solve karta hai aur queue implementation ko efficient banata hai.

---

8. Explain DQUEUE.

DQUEUE ka full form hai:

```
Double Ended Queue
```

Isme insertion aur deletion dono ends se possible hota hai.

Structure

```
FRONT ←→ Queue ←→ REAR
```

Operations

1. Insert Front

Front side insertion.

---

2. Insert Rear

Rear side insertion.

---

3. Delete Front

Front side deletion.

---

4. Delete Rear

Rear side deletion.

Types of DQUEUE

---

1. Input Restricted DQUEUE

Insertion sirf ek end se hota hai.

Deletion dono ends se ho sakta hai.

---

2. Output Restricted DQUEUE

Deletion sirf ek end se hota hai.

Insertion dono ends se possible hai.

Advantages

- Flexible operations
- Better memory usage

Disadvantages

- Complex implementation

Applications

- Browser history
- Undo/Redo operation
- CPU scheduling

Conclusion

DQUEUE ek flexible data structure hai jisme insertion aur deletion dono ends se possible hote hain.

---

9. Explain Priority Queue.

Priority Queue ek special queue hai jisme har element ki ek priority hoti hai.

Higher priority wala element pehle process hota hai.

Example

Hospital emergency system.

High critical patient ko pehle treatment diya jata hai.

Types of Priority Queue

1. Ascending Priority Queue

Lower value = higher priority.

---

2. Descending Priority Queue

Higher value = higher priority.

Operations

1. Insertion

Element insert according to priority.

---

2. Deletion

Highest priority element remove hota hai.

Example

Elements:

```
A(1)
B(3)
C(2)
```

Priority order:

```
B → C → A
```

Advantages

- Important tasks execute first
- Efficient scheduling

Disadvantages

- Complex implementation
- Extra processing required

Applications

- CPU scheduling
- Hospital management
- Network routing
- Printer management

Conclusion

Priority queue ek important scheduling data structure hai jisme high priority elements ko pehle process kiya jata hai.

---

10. Differentiate between Stack and Queue.

Stack	Queue
Follows LIFO	Follows FIFO
Last inserted removed first	First inserted removed first
Insertion and deletion at same end	Insertion and deletion at different ends
Uses TOP	Uses FRONT and REAR
Operations: Push, Pop	Operations: Enqueue, Dequeue
Simpler structure	More scheduling oriented
Used in recursion	Used in CPU scheduling
Example: Plates stack	Example: Ticket queue

Similarities

- Both are linear data structures
- Both can be implemented using array or linked list
- Both store data sequentially

Applications of Stack

- Expression evaluation
- Recursion
- Browser backtracking

Applications of Queue

- Scheduling
- Printer spooling
- BFS traversal

Conclusion

Stack and Queue dono important linear data structures hain, but Stack LIFO follow karta hai while Queue FIFO follow karta hai.

14-Mark Questions
Explain stack implementation with algorithms.
Explain infix to postfix conversion using stack with example.
Explain postfix expression evaluation with example.
Explain recursion and stack memory.
Explain queue implementation with algorithms.
Explain circular queue with advantages.
Differentiate stack and queue in detail.
Explain DQUEUE and Priority Queue with applications.

1. Explain Stack Implementation with Algorithms

Introduction

Stack is a linear data structure in which insertion and deletion are performed from only one end called **TOP**.

Stack follows:

```
LIFO = Last In First Out
```

Means jo element last me insert hota hai, wahi sabse pehle delete hota hai.

Example: plates ka stack.

Stack Operations

Main operations:

1. Push
2. Pop
3. Peek
4. isEmpty
5. isFull

Stack Implementation Using Array

Array implementation me stack ka size fixed hota hai.

```
int stack[100];
int top = -1;
```

Here:

```
top = -1 means stack is empty
```

PUSH Algorithm

Push operation stack me element insert karta hai.

```
Algorithm PUSH(stack, item)

Step 1: If top == size - 1
        Print "Stack Overflow"
        Exit

Step 2: top = top + 1

Step 3: stack[top] = item

Step 4: Stop
```

POP Algorithm

Pop operation stack ke top element ko delete karta hai.

```
Algorithm POP(stack)

Step 1: If top == -1
        Print "Stack Underflow"
        Exit

Step 2: item = stack[top]

Step 3: top = top - 1
```

```
Step 4: Return item
```

PEEK Algorithm

```
Algorithm PEEK(stack)

Step 1: If top == -1
        Print "Stack Empty"

Step 2: Else
        Return stack[top]
```

Example

Push:

```
Push 10
Push 20
Push 30
```

Stack:

```
10
20
30  ← TOP
```

Pop:

```
Pop removes 30
```

Now stack:

```
10
20  ← TOP
```

Stack Implementation Using Linked List

In linked list implementation, stack size is dynamic.

Each node contains:

```
Data + Next Pointer
```

Top pointer points to first node.

Push in Linked List

```
1. Create new node
2. Store data
3. newNode->next = top
4. top = newNode
```

Pop in Linked List

```
1. Check if top == NULL
2. temp = top
3. top = top->next
4. delete temp
```

Advantages of Stack

- Easy to implement
- Useful in recursion
- Used in expression conversion
- Used in undo/redo operation

Disadvantages

- Array stack has fixed size
- Stack overflow possible
- Stack underflow possible

Applications

- Recursion
- Function calls
- Infix to postfix conversion
- Postfix evaluation
- Browser back button

Conclusion

Stack is an important linear data structure that follows LIFO principle. It can be implemented using array and linked list. Stack is widely used in expression evaluation, recursion and memory management.

---

2. Explain Infix to Postfix Conversion Using Stack with Example

Introduction

Expression conversion is an important application of stack.

There are three types of expressions:

1. Infix Expression

Operator is written between operands.

```
A + B
```

2. Postfix Expression

Operator is written after operands.

```
A B +
```

3. Prefix Expression

Operator is written before operands.

```
+ A B
```

Why Convert Infix to Postfix?

Computers find postfix expression easier to evaluate because postfix does not require brackets or operator precedence rules during evaluation.

Operator Precedence

Operator	Priority
^	Highest
* , /	Medium
+ , -	Lowest

Associativity

Operator	Associativity
^	Right to Left
* / + -	Left to Right

Algorithm for Infix to Postfix

```
Algorithm INFIX_TO_POSTFIX(expression)

Step 1: Initialize empty stack.

Step 2: Scan expression from left to right.
```

```
Step 3: If scanned symbol is operand,
        add it to postfix expression.

Step 4: If scanned symbol is '(',
        push it into stack.

Step 5: If scanned symbol is operator,
        pop operators from stack having higher or equal precedence
        and add them to postfix expression.
        Then push current operator.

Step 6: If scanned symbol is ')',
        pop from stack until '(' is found.

Step 7: After complete scanning,
        pop remaining operators from stack.

Step 8: Postfix expression is obtained.
```

Example

Convert:

```
A + B * C
```

Stepwise Conversion

Scanned Symbol	Stack	Postfix
A	Empty	A
+	+	A
B	+	AB

Scanned Symbol	Stack	Postfix
*	+ *	AB
C	+ *	ABC
End	Empty	ABC*+

Final Postfix

```
ABC*+
```

Because * has higher precedence than +.

Another Example

Convert:

```
(A + B) * C
```

Stepwise:

```
A and B are inside brackets, so A+B is solved first.
```

Postfix:

```
AB+C*
```

Importance of Stack in Conversion

Stack stores operators temporarily. It helps in maintaining precedence and parenthesis order.

Advantages

- Easy expression evaluation
- No need of brackets in postfix
- Used in compiler design

Applications

- Compiler design
- Expression evaluation
- Calculator programs
- Syntax processing

Conclusion

Infix to postfix conversion is a major application of stack. Stack helps to handle operators, precedence and brackets efficiently.

3. Explain Postfix Expression Evaluation with Example

Introduction

Postfix expression evaluation stack ka important application hai.

Postfix expression me operator operands ke baad likha jata hai.

Example:

```
AB+
```

Computers postfix expressions ko easily evaluate kar sakte hain because:

- no brackets required
- no precedence checking required

Working Principle

Postfix evaluation me:

- operands stack me push hote hain
- operator aane par calculation perform hota hai

Algorithm for Postfix Evaluation

```
Algorithm POSTFIX_EVALUATION(expression)

Step 1: Initialize empty stack.

Step 2: Scan expression left to right.

Step 3: If symbol is operand,
        push into stack.

Step 4: If symbol is operator:
```

```
        pop top two operands.
        perform operation.
        push result into stack.

Step 5: Repeat until expression ends.

Step 6: Final value in stack is answer.
```

Example

Evaluate:

```
23*5+
```

Meaning:

```
(2 × 3) + 5
```

Stepwise Evaluation

Symbol	Operation	Stack
2	Push	2
3	Push	2 3
*	2×3=6	6
5	Push	6 5
+	6+5=11	11

Final Answer

11

Advantages

- Faster evaluation
- No precedence handling required
- No brackets needed

Applications

- Compiler design
- Calculator programs
- Expression evaluation systems

Conclusion

Postfix evaluation stack ka important application hai jisme operands stack me push hote hain aur operators calculation perform karte hain.

---

4. Explain Recursion and Stack Memory

Introduction

Recursion ek process hai jisme function khud ko hi repeatedly call karta hai.

Recursive function tab tak call hota rehta hai jab tak stopping condition satisfy nahi hoti.

Syntax of Recursive Function

```
function()
{
    if(condition)
        return;

    function();
}
```

Example: Factorial Program

```
int fact(int n)
{
    if(n==0)
        return 1;

    return n * fact(n-1);
}
```

Working of Recursion

Suppose:

```
fact(4)
```

then:

```
4 × fact(3)
3 × fact(2)
2 × fact(1)
1 × fact(0)
```

Base condition:

```
fact(0)=1
```

Final answer:

```
24
```

Stack Memory in Recursion

Recursion me har function call stack memory me store hoti hai.

This memory is called:

```
Call Stack
```

Working of Call Stack

Suppose:

```
fact(3)
```

Stack storage:

```
fact(3)
fact(2)
fact(1)
fact(0)
```

After completion:

functions reverse order me return karte hain.

Advantages of Recursion

- Code simple hota hai
- Complex problems easy ho jate hain
- Tree and graph problems me useful

Disadvantages

- Extra stack memory required

- Stack overflow possible
- Execution slower ho sakta hai

Applications

- Factorial
- Fibonacci series
- Tree traversal
- Tower of Hanoi
- Graph algorithms

Difference Between Recursion and Iteration

Recursion	Iteration
Function calls itself	Uses loops
Uses stack memory	Less memory
Simple code	Faster execution
Risk of stack overflow	No stack overflow

Conclusion

Recursion ek powerful programming technique hai jo stack memory ka use karti hai. Recursive calls call stack me store hoti hain aur base condition par terminate hoti hain.

---

5. Explain Queue Implementation with Algorithms

Introduction

Queue ek linear data structure hai jo FIFO principle follow karta hai.

```
FIFO = First In First Out
```

Insertion rear side se hota hai aur deletion front side se hota hai.

Queue Representation

```
FRONT → 10 20 30 ← REAR
```

Queue Operations

1. Enqueue
2. Dequeue
3. Peek
4. isEmpty
5. isFull

Queue Implementation Using Array

Declaration:

```
int queue[100];
int front = -1;
int rear = -1;
```

ENQUEUE Algorithm

```
Algorithm ENQUEUE(item)

Step 1: If rear == size-1
        Print Overflow

Step 2: Else if front == -1
        front = 0

Step 3: rear = rear + 1

Step 4: queue[rear] = item
```

DEQUEUE Algorithm

```
Algorithm DEQUEUE()

Step 1: If front == -1 OR front > rear
        Print Underflow

Step 2: item = queue[front]

Step 3: front = front + 1

Step 4: Return item
```

Example

Enqueue:

```
10, 20, 30
```

Queue:

```
FRONT → 10 20 30 ← REAR
```

Dequeue:

```
10 removed
```

Queue becomes:

```
FRONT → 20 30 ← REAR
```

Queue Using Linked List

Each node contains:

- data
- next pointer

Front points to first node.

Rear points to last node.

Advantages

- Dynamic size
- Better memory utilization

Disadvantages

- Pointer memory required

Applications

- CPU scheduling
- Printer queue
- BFS traversal
- Keyboard buffering

Conclusion

Queue is an important FIFO data structure used in scheduling and buffering applications. It can be implemented using arrays and linked lists.

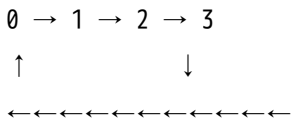
6. Explain Circular Queue with Advantages

Introduction

Circular Queue ek special type ki queue hai jisme last position first position se connected hoti hai.

Normal queue me memory wastage hota hai because deleted positions reuse nahi hoti. Circular queue is problem ko solve karta hai.

Structure of Circular Queue



Last position first position se connected hoti hai.

FIFO Principle

Circular queue bhi FIFO follow karta hai.

```
First In First Out
```

Operations

1. Enqueue

Element insert karna.

---

2. Dequeue

Element delete karna.

Conditions

Queue Empty

```
front = -1
```

Queue Full

```
(front == rear + 1)
```

OR

```
(front == 0 && rear == size-1)
```

ENQUEUE Algorithm

```
Step 1: Check overflow condition.

Step 2: If queue empty:
        front = rear = 0

Step 3: Else:
        rear = (rear + 1) % size

Step 4: Insert element.
```

DEQUEUE Algorithm

```
Step 1: Check underflow condition.

Step 2: Delete front element.

Step 3: If front == rear
        front = rear = -1

Step 4: Else
        front = (front + 1) % size
```

Example

Insert:

```
10 20 30 40
```

Delete:

```
10 and 20 removed
```

Now circular queue reuses empty spaces.

Advantages of Circular Queue

- Better memory utilization
- No memory wastage
- Faster operations
- Efficient implementation

Disadvantages

- Complex implementation
- Difficult debugging

Applications

- CPU scheduling
- Keyboard buffering
- Printer queue
- Traffic systems

Conclusion

Circular queue normal queue ki memory wastage problem ko solve karta hai aur efficient queue implementation provide karta hai.

---

7. Differentiate Stack and Queue in Detail

Introduction

Stack and Queue dono linear data structures hain but unka working principle different hota hai.

Difference Between Stack and Queue

Stack	Queue
Follows LIFO	Follows FIFO
Last inserted deleted first	First inserted deleted first
Insertion and deletion at same end	Insertion and deletion at different ends
Uses TOP pointer	Uses FRONT and REAR
Operations: Push and Pop	Operations: Enqueue and Dequeue
Simpler structure	Scheduling oriented structure
Used in recursion	Used in scheduling
Example: Plate stack	Example: Ticket queue

Stack Representation

```
30 ← TOP
20
10
```

Queue Representation

```
FRONT → 10 20 30 ← REAR
```

Applications of Stack

- Recursion
- Expression evaluation
- Browser history
- Undo operation

Applications of Queue

- CPU scheduling
- Printer queue
- BFS traversal
- Network buffering

Similarities

- Both are linear data structures
- Both can be implemented using arrays or linked lists
- Both store data sequentially

Advantages of Stack

- Easy implementation
- Fast insertion/deletion

Advantages of Queue

- Fair scheduling
- Efficient processing

Conclusion

Stack LIFO principle follow karta hai while Queue FIFO principle follow karta hai. Dono data structures computer science me important role play karte hain.

---

8. Explain DQUEUE and Priority Queue with Applications

Introduction

DQUEUE and Priority Queue special types of queues hain jo advanced applications me use hote hain.

DQUEUE

DQUEUE ka full form hai:

```
Double Ended Queue
```

Isme insertion aur deletion dono ends se possible hota hai.

Structure

```
FRONT ←→ Queue ←→ REAR
```

Operations

1. Insert Front

Front side insertion.

---

2. Insert Rear

Rear side insertion.

---

3. Delete Front

Front side deletion.

---

4. Delete Rear

Rear side deletion.

Types of DQUEUE

---

1. Input Restricted DQUEUE

Insertion only one end se hota hai.

Deletion both ends se possible hai.

---

2. Output Restricted DQUEUE

Deletion only one end se hota hai.

Insertion both ends se possible hai.

Advantages of DQUEUE

- Flexible operations
- Better memory usage
- Efficient insertion/deletion

Applications of DQUEUE

- Browser history
- Undo/Redo operation
- CPU scheduling

Priority Queue

Priority Queue ek special queue hai jisme har element ki priority hoti hai.

Higher priority element pehle process hota hai.

Example

Hospital emergency system.

Emergency patient ko pehle treatment diya jata hai.

Types of Priority Queue

1. Ascending Priority Queue

Lower value means higher priority.

---

2. Descending Priority Queue

Higher value means higher priority.

Operations

Insertion

Element priority ke according insert hota hai.

Deletion

Highest priority element delete hota hai.

Example

Elements:

```
A(1)
B(3)
C(2)
```

Priority order:

```
B → C → A
```

Advantages of Priority Queue

- Important tasks processed first
- Efficient scheduling

Applications of Priority Queue

- CPU scheduling
- Hospital systems
- Network routing
- Printer management

Conclusion

DQUEUE flexible queue implementation provide karta hai while Priority Queue priority-based processing provide karta hai. Dono advanced computer applications me widely used hote hain.