

UNDERGRADUATE LEARNING SERIES / UNIT II

DATA STRUCTURES STACK & QUEUE

Expressions से Circular Queue तक

Hindi-English explanations के साथ

14 अध्यायों में Unit II की पूर्ण तैयारी

STACK

EXPRESSIONS

QUEUE / DEQUE

Hindi + English Edition

Algorithms · Diagrams · Dry Runs · MCQs

पुस्तक का परिचय

Unit II / How to Use This Book

यह पुस्तक दिए syllabus को cover करती है: Stack definition, PUSH/POP, array और linked-list implementation; infix, prefix, postfix representation एवं stack conversion; postfix evaluation; recursion using stack; queue definition और types-priority queue, circular queue, double ended queue; INSERT, DELETE, TRAVERSE; array/linked-list implementation और applications.

14 chapters में Hindi-English concepts, pseudocode, comparison tables, diagrams, solved expression examples, boundary cases और 40 MCQs with answer key हैं. अंतिम chapter expression-workbench project और queue-lab checklist देता है.

Learning method

पहले representation draw करें. हर operation के बाद top/front/rear/count लिखें. Expressions में tokens, precedence और associativity देखें. Answer key से पहले MCQs solve करें.

Conventions

Programming language specify नहीं है; algorithms language-independent pseudocode हैं. Assignment $\leftarrow$, missing reference NIL. "report error" current operation immediately ends करता है. Array stack top last occupied index है; circular queue rear next insertion slot है. दोनों conventions को mix न करें.

All circular queue/deque algorithms use count; capacity $C > 0$, all C slots usable. Alternate unused-slot convention अलग explain की है. Expressions binary operators $+$, $-$, $*$, $/$, $^$ use करती हैं. $^$ mathematical exponentiation, right-associative है; unary minus and functions require extended parser. Numeric division real arithmetic assume करती है.

Syllabus Mapping

Topic	Chapters
Stack definition, operations, implementations	1-3
Notation, conversions, postfix evaluation	4-7
Recursion using stack	8
Queue definition, array/circular/linked queue	9-11
Priority queue and deque	12-13
Revision, MCQs and practical work	14

Complexity tables state representation assumptions; allocator costs and arbitrary-precision arithmetic are excluded unless discussed. Diagrams are conceptual, not actual memory addresses.

विषय-सूची

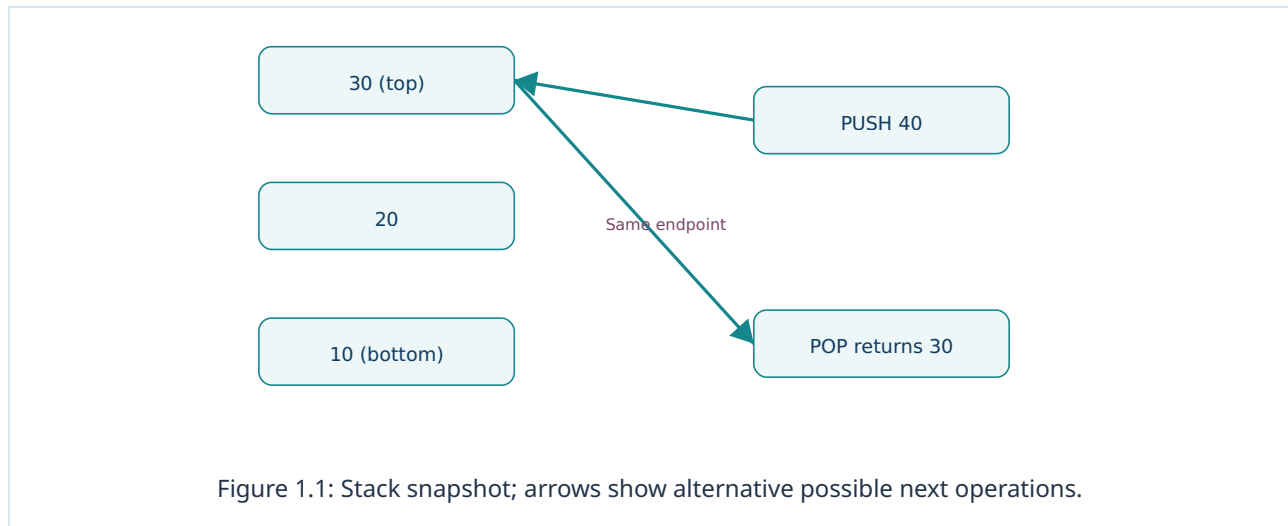
Contents / Chapters 1-14

01. Stack Definition एवं Operations	4
02. Array से Stack Implementation	6
03. Linked List से Stack Implementation	7
04. Infix, Prefix एवं Postfix Representation	9
05. Infix से Postfix Conversion using Stack	11
06. Infix से Prefix Conversion using Stack	13
07. Postfix Expression Evaluation	14
08. Recursion using Stack	16
09. Queue Definition, Operations एवं Types	18
10. Array Queue एवं Circular Queue	19
11. Linked List से Queue Implementation	21
12. Priority Queue	23
13. Double Ended Queue (Deque)	25
14. Complete Revision एवं Unit Test	27

UNIT II / CHAPTER 01

अध्याय 1: Stack Definition एवं Operations

One endpoint और LIFO discipline



1.1 Stack ADT

Stack एक linear abstract data type है जिसमें insertion और removal एक ही endpoint, **top**, पर होते हैं। Last In, First Out (LIFO): सबसे बाद में रखा गया अभी मौजूद item सबसे पहले निकलता है। Plates का ढेर useful analogy है, लेकिन ADT का व्यवहार array या linked-list representation से independent है।

1.2 Core Operations

Operation	Effect	Error condition
PUSH(x)	x नया top बनता है	Bounded stack full
POP()	Top remove और return	Empty stack
PEEK()/TOP()	Top return, no removal	Empty stack
isEmpty()	Empty status	None
size()	Current element count	None

Overflow = capacity उपलब्ध न होने पर insertion failure. Underflow = empty structure से removal/access failure. Dynamic linked stack में fixed-array-style full index नहीं, पर memory allocation failure संभव है।

1.3 Dry Run

Operation	Bottom → Top	Returned
PUSH(10)	[10]	-
PUSH(20)	[10,20]	-

Operation	Bottom → Top	Returned
PEEK	[10,20]	20
POP	[10]	20
PUSH(30)	[10,30]	-
POP	[10]	30
POP	[]	10

PEEK size नहीं बदलती. POP exactly one element हटाती है. एक और POP underflow होगा.

1.4 Applications

Undo history, expression conversion/evaluation, balanced delimiters, function-call management और depth-first exploration stack use करते हैं. Browser back behavior को stack से model किया जा सकता है; full back/forward design में multiple histories चाहिए.

1.5 Contracts एवं Invariant

Operation success के बाद stored sequence और top consistent होने चाहिए. Failed push/pop state को corrupt न करे. Return value और success/error status अलग रखें; legitimate value -1 होने पर -1 को अकेले error sentinel बनाना ambiguous है.

Exam point: Stack का rule LIFO है; memory allocation scheme नहीं. “Stack ADT” और “program call-stack memory” related applications हैं, identical definitions नहीं.

Practice: PUSH(A),PUSH(B),POP,PUSH(C),PEEK,POP,POP का returned sequence लिखें. **Answer:** B,C,C,A; PEEK वाला C remove नहीं हुआ था.

UNIT II / CHAPTER 02

अध्याय 2: Array से Stack Implementation

Capacity, top और boundary checks

2.1 Representation

Array $S[0..C-1]$, capacity $C > 0$. $top = -1$ initially. Valid occupied indices $0..top$; $size = top + 1$. Full when $top = C - 1$. Empty when $top = -1$. इस convention को size-based top convention से mix न करें.

```
PUSH(S, C, top, x)
    if top = C-1
        return (top, false)
    top ← top+1
    S[top] ← x
    return (top, true)

POP(S, top)
    if top = -1
        report underflow
    value ← S[top]
    top ← top-1
    return (top, value)

PEEK(S, top)
    if top = -1
        report underflow
    return S[top]
```

report इस पुस्तक में error return/exception और immediate termination of that operation को दर्शाता है। Caller updated top को रखे। Removed slot की bytes physical array में रह सकती हैं, पर वह logical stack का हिस्सा नहीं। Reference-based implementations unused slot clear करके object retention रोक सकती हैं।

2.2 Capacity 3 Example

PUSH5 → top0; PUSH8 → top1; PUSH2 → top2. Fourth push fails and top remains2. POP returns2, top1. PUSH9 writes index2. Result bottom-to-top[5,8,9].

2.3 Traverse

Top-to-bottom display:

```
for i ← top downto 0
    visit S[i]
```

Display $\Theta(n)$, non-destructive. Repeated POP display destructive है unless items subsequently restored.

2.4 Complexity एवं Resizing

Fixed-capacity push/pop/peek $\Theta(1)$, array allocated space $\Theta(C)$. Dynamic array can resize by copying n items: individual resize push $\Theta(n)$, geometric-growth push amortized $\Theta(1)$. Amortized is sequence average bound, not probabilistic average-case behavior.

2.5 Common Errors

Check full before incrementing top. Check empty before indexing. C is count; highest valid index $C - 1$. $top = -1$ is state marker, not usable array index. Return updated top after operation in pass-by-value pseudocode.

Practice: $C=4, top=2$. How many items and free slots? **Answer:** 3 items, 1 free slot. After two successful pops $top=0, size=1$.

UNIT II / CHAPTER 03

अध्याय 3: Linked List से Stack Implementation

Head reference ही top



Figure 3.1: Linked stack; the head holds the newest item.

3.1 Node एवं Representation

Each node contains data,next. top points to first node; empty top=NIL. Next links lead toward older elements. Head insertion and deletion preserve LIFO without traversing the list.

```
PUSH_LINKED(top, value)
    node ← allocate node
    if allocation fails
        report allocation failure
    node.data ← value
    node.next ← top
    top ← node
    return top

POP_LINKED(top)
    if top = NIL
        report underflow
    victim ← top
    value ← victim.data
    top ← victim.next
    release victim if manually managed
    return (top, value)
```

Automatic memory-managed languages explicit release नहीं माँगतीं; unreachable nodes eventually reclaimed हो सकती हैं. Manual-memory implementations links पढ़ने के बाद ही victim free करें.

3.2 PEEK एवं TRAVERSE

PEEK returns top.data after nonempty check. Traverse current=top; while current≠NIL visit current.data,current=current.next. Traversal visits newest→oldest and costs $\Theta(n)$.

3.3 Empty एवं Singleton Cases

First push sets `node.next=NIL` and `top=node`. Sole node pop sets `top=NIL`. No special predecessor search needed. Failed allocation leaves old top unchanged.

3.4 Array versus Linked Stack

Feature	Fixed array	Linked nodes
Capacity	Predetermined C	Grows with allocated nodes
Extra per item	No next reference	Next reference + node overhead
Locality	Usually good	Often weaker
Full condition	<code>top=C-1</code>	Allocation can fail
Push/pop	$\Theta(1)$	$\Theta(1)$ links, ignoring allocator cost
Allocated space	$\Theta(C)$	$\Theta(n)$ nodes

3.5 Memory Numerical

Assume value 4 bytes, next 8 bytes, no padding/runtime overhead. $n=20$ linked nodes use $20 \times 12 = 240$ bytes plus top reference. Array capacity 20 values uses 80 bytes plus index/control fields. Actual platform memory sizes differ; assumptions लिखें.

Practice: Draw `top→30→20→10→NIL`. PUSH 40 then POP twice. **Answer:** Returns 40, 30; final top points 20.

UNIT II / CHAPTER 04

अध्याय 4: Infix, Prefix एवं Postfix Representation

Operator position और expression tree meaning

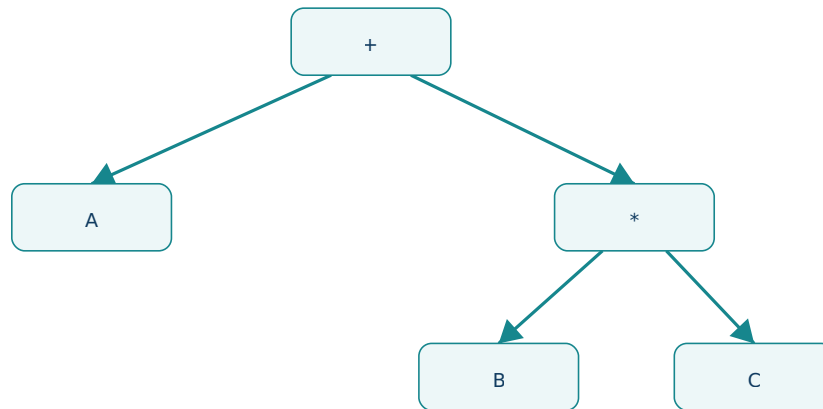


Figure 4.1: $A + B * C$. Preorder gives prefix; postorder gives postfix.

4.1 Three Notations

Notation	Operator position	Example
Infix	Between operands	$A+B$
Prefix/Polish	Before operands	$+ A B$
Postfix/Reverse Polish	After operands	$A B +$

Prefix/postfix token sequences encode grouping without parentheses when operator arity is known. Infix needs precedence, associativity and parentheses rules. “AB+” convenient for single-letter operands, but real multi-digit values require token separation: $12\ 3\ +$.

4.2 Precedence एवं Associativity

Operator	Precedence	Associativity assumed here
$^$ (exponentiation)	Highest	Right
$*, /$	Middle	Left
$+, -$	Lowest	Left

Parentheses force grouping and are not output operators. Here $^$ means mathematical exponentiation; in some programming languages $^$ means XOR. Examples use binary operators only; unary minus, functions and implicit multiplication require extended parsing rules.

4.3 Solved Conversions

Infix	Prefix	Postfix
$A+B*C$	$+ A * B C$	$A B C * +$
$(A+B)*C$	$* + A B C$	$A B + C *$
$A-B-C$	$-- A B C$	$A B - C -$
A^B^C	$^ A ^ B C$	$A B C ^ ^$
$(A+B)/(C-D)$	$/ + A B - C D$	$A B + C D - /$

$A-B-C$ groups $(A-B)-C$ due left associativity. A^B^C groups $A^B(C)$ due right associativity. Rearranging symbols without preserving grouping can change meaning.

4.4 Expression-tree View

For $A+BC$, root $+$, left child A , right child. Multiplication's children B, C . Preorder traversal gives prefix; postorder gives postfix. Fully parenthesized inorder gives the infix grouping. Tree method is conceptual support; syllabus stack conversion remains primary.

Practice: Convert $A(B+C)-D$. Answer: * Prefix - $* A + B C D$; postfix $A B C + * D -$.

UNIT II / CHAPTER 05

अध्याय 5: Infix से Postfix Conversion using Stack

Operator stack और output sequence

5.1 Algorithm

Scan tokens left-to-right. Operand directly output. Left parenthesis push. Right parenthesis pops until matching left parenthesis, then discards that left parenthesis. Operator pops higher-precedence operators; equal precedence pops only when incoming operator is left-associative.

```

INFIX_TO_POSTFIX(tokens)
  operators ← empty stack
  output ← empty token list
  for token in tokens
    if token is operand
      append token to output
    else if token = "("
      push token
    else if token = ")"
      while stack nonempty and peek() ≠ "("
        append pop() to output
      if stack empty
        report mismatched parentheses
      pop() // discard "("
    else if token is recognized binary operator
      while stack nonempty and peek() is operator
        and (precedence(peek()) > precedence(token)
            or (precedence(peek()) = precedence(token)
                and token is left-associative))
        append pop() to output
      push token
    else
      report invalid token
  while stack nonempty
    if peek() is parenthesis
      report mismatched parentheses
    append pop() to output
  return output

```

Assume a syntactically valid binary expression after tokenization. Parenthesis checks alone do not validate missing operands, adjacent operators or implicit multiplication. Production parser should validate token-order grammar too.

5.2 Dry Run: A+B*C

Token	Stack bottom→top	Output
A	[]	A
+	[+]	A
B	[+]	A B

Token	Stack bottom→top	Output
*	[+,*]	A B
C	[+,*]	A B C
End: pop *,+	[]	A B C * +

5.3 Parenthesis Example

(A+B)C: push(; outputA; push+; outputB; on) pop+ and discard(; push; outputC; drain*. Output A B + C *. Parentheses are not part of postfix output.

5.4 Associativity Trap

For $A^B C$, incoming second $^$ does not pop equal $^$ because incoming operator is right-associative. End draining produces A B C $^ ^$. For A-B-C, incoming second- pops equal- $\rightarrow$ A B - C -.

5.5 Complexity

Each token processed once, each operator pushed and popped at most once. $\Theta(n)$ token-processing time and $O(n)$ auxiliary operator/output storage for n tokens. Tokenization cost and string-building choices are additional implementation details.

Practice: Convert $A+B*(C-D)/E$. **Answer:** A B C D - * E / +.

UNIT II / CHAPTER 06

अध्याय 6: Infix से Prefix Conversion using Stack

Right-to-left scan और associativity reversal

6.1 Reliable Procedure

One method: reverse token order, swap parentheses, perform a modified postfix conversion, then reverse output tokens. Reversal is of tokens, not characters: operand123 must remain123. During modified conversion, original associativity is reversed for equal-precedence handling.

Original operator associativity	Modified scan's equal-precedence action
Left (+, -, *, /)	Do not pop equal precedence
Right (^)	Pop equal precedence

Other conversion rules same as chapter5. Simply reverse and reuse ordinary postfix rule unchanged can produce incorrect grouping for repeated operators.

6.2 Pseudocode

```

INFIX_TO_PREFIX(tokens)
    reversed ← reverse token order
    for each token in reversed
        swap "(" with ")" and ")" with "("
    middle ← modified postfix conversion(reversed)
    // equal-precedence: pop iff ORIGINAL operator
    // is right-associative; higher still pops
    return reverse token order of middle

```

Input uses valid binary grammar and precedence from chapter4. A second method builds an expression tree and emits preorder; it avoids the reversal-rule pitfall but needs its own parser.

6.3 Worked Example: A+B*C

Reversed: CB+A. Modified postfix: C B * A +. Reverse token output: + A * B C *. Operator scope means A plus productBC.

6.4 Repeated Subtraction

A-B-C means(A-B)-C. Reverse C-B-A; equal- is not popped in modified conversion. Intermediate C B A - - ; reverse → - - A B C. Incorrect - A - B C meansA-(B-C), which is different.

6.5 Repeated Exponentiation

A^B^C meansA^(B^C). Reverse C^B^A; equal^ is popped in modified conversion. Intermediate C B ^ A ^ ; reverse → ^ A ^ B C.

6.6 Numeric Check

For8-3-2, correct grouping gives3. Prefix- - 8 3 2 gives3, whereas- 8 - 3 2 gives7. Numeric substitution is a useful check, though testing a few values is not a proof of equivalence.

Practice: (A+B)/(C-D)→/ + A B - C D. Trace reversal, swapped parentheses and intermediate postfix explicitly.

UNIT II / CHAPTER 07

अध्याय 7: Postfix Expression Evaluation

Operand stack और operand order

7.1 Algorithm

Scan left-to-right. Push numeric operands. For a binary operator pop right operand first, left operand second; calculate left op right and push result. Validate enough operands. Final stack must contain exactly one value.

```

EVALUATE_POSTFIX(tokens)
  values ← empty stack
  for token in tokens
    if token is numeric operand
      push numericValue(token)
    else if token is recognized binary operator
      if size() < 2
        report missing operand
      right ← pop()
      left ← pop()
      if token = "/" and right = 0
        report division by zero
      push apply(token, left, right)
    else
      report invalid token
  if size() ≠ 1
    report invalid expression
  return pop()

```

Number format, integer versus real division, overflow and exponentiation domain are implementation policies. In this book numerical evaluation uses real arithmetic where division appears unless stated otherwise.

7.2 Solved Evaluation

Expression 5 2 3 * + 4 - :

Token	Stack bottom→top	Action
5	[5]	Push
2	[5,2]	Push
3	[5,2,3]	Push
*	[5,6]	2×3
+	[11]	5+6
4	[11,4]	Push
-	[7]	11-4

Result 7. Same infix 5+2×3-4.

7.3 Operand-order Trap

8 2 / 3 -: right2,left8→4; right3,left4→1. If order reversed, division and subtraction change result. Addition/multiplication examples may hide this mistake because they are commutative.

7.4 Invalid Inputs

2 + insufficient operands; 2 3 final stack has2 values; 5 0 / division by zero; unknown token invalid. Empty expression final stack0 invalid. A lone numeric token can be a valid expression under this grammar.

7.5 Prefix Evaluation Connection

Scan prefix right-to-left. On operator first pop is **left**, second is right, because scan direction reversed. Prefix- 8 3→5. Do not copy postfix operand-order rule blindly.

7.6 Complexity

$\Theta(n)$ token operations, $O(n)$ stack space in worst case, ignoring arbitrary-precision arithmetic cost. For specific bounded-width arithmetic, each operator treated constant time.

Practice: 6 2 / 3 4 * +→15. 2 3 2 ^ ^→512. Show all intermediate stack states.

UNIT II / CHAPTER 08

अध्याय 8: Recursion using Stack

Calls, activation records और unwinding



8.1 Recursion क्या है?

Function अपने आप को directly या indirectly call करे तो recursion है. Base case process रोकती है; recursive step problem को base case की ओर ले जाती है. केवल base case लिखना पर्याप्त नहीं-recursive arguments में progress भी होनी चाहिए.

```

FACTORIAL(n)
  require integer n ≥ 0
  if n = 0
    return 1
  return n × FACTORIAL(n-1)

```

For n=4 calls 4→3→2→1→0; return values 1→1→2→6→24. 0!=1 mathematical convention है.

8.2 Call Stack एवं Frames

Each active call has activation record/frame conceptually storing arguments, local state, return destination and suspended work. Exact runtime layout language/compiler-specific. Most recently called function returns first: LIFO.

Active stage	Pending work
FACTORIAL(4)	4 × result of FACTORIAL(3)
FACTORIAL(3)	3 × result of FACTORIAL(2)
FACTORIAL(2)	2 × result of FACTORIAL(1)
FACTORIAL(1)	1 × result of FACTORIAL(0)
FACTORIAL(0)	Return 1, begin unwinding

8.3 Explicit Stack Simulation

```

FACTORIAL_WITH_STACK(n)
  require integer  $n \geq 0$ 
  S ← empty stack
  for value ← n downto 1
    push value
  result ← 1
  while S nonempty
    result ← pop() × result
  return result

```

Same result, explicit pending values replace recursive calls. Both $\Theta(n)$ multiplications under unit-cost arithmetic and $O(n)$ stack storage. Simple iterative factorial needs $\Theta(1)$ auxiliary storage, so recursion is not always necessary.

8.4 Recursion Applications

Nested structures, tree traversals, divide-and-conquer and backtracking express naturally with recursive definitions. For recursive linked-list traversal, each node call may keep pending work. Printing before recursion gives forward order; printing after recursion gives reverse order.

```

PRINT_REVERSE(node)
  if node = NIL
    return
  PRINT_REVERSE(node.next)
  visit node.data

```

For 10 → 20 → 30 outputs 30, 20, 10; $\Theta(n)$ time and $O(n)$ call-stack depth.

8.5 Stack Overflow एवं Traps

Very deep recursion can exceed runtime call-stack limit. Stack overflow here differs from bounded ADT overflow implementation detail but both involve unavailable stack capacity. Tail-call optimization is not universally guaranteed; do not assume every tail-recursive algorithm constant-space. Naive Fibonacci repeats subproblems; recursion does not automatically mean linear complexity.

Practice: Trace FACTORIAL(3), count active frames at base case: 4 including $n=0$. Explain missing-base-case and nonprogressing-recursion errors.

UNIT II / CHAPTER 09

अध्याय 9: Queue Definition, Operations एवं Types

FIFO और service order

9.1 Queue ADT

Ordinary queue insertion rear पर और removal front पर करती है। First In, First Out (FIFO): earliest still-present item leaves first. INSERT/enqueue, DELETE/dequeue and TRAVERSE syllabus operations हैं। PEEK/front reads oldest item without removal.

9.2 Operations एवं Contracts

Operation	Successful effect	Boundary
enqueue(x)	x rear पर जोड़ें	Full bounded queue
dequeue()	Front remove and return	Empty queue
peekFront()	Front inspect	Empty queue
traverse()	FIFO order visit	Empty produces no visits

Example enqueue10,enqueue20,dequeue→10,enqueue30,dequeue→20. Result remaining[30]. Stack with same insert/removal sequence would remove20 then30 instead.

9.3 Types of Queues

Type	Rule/feature	Application
Simple FIFO queue	Arrival order	Waiting jobs
Circular queue	Slots wrap for reuse	Bounded buffer
Priority queue	Priority governs removal	Scheduled tasks
Deque	Both endpoints supported	Sliding-window methods

Circular queue is often a representation of FIFO behavior. Priority queue changes service rule. Deque broadens permitted endpoint operations. These are not four identical FIFO contracts.

9.4 Applications

Print spool, arrival-order requests, task queues, producer-consumer buffers and BFS. Real systems may add priorities, fairness, dropping policies or concurrency controls. Scheduling is not automatically FIFO for every operating system.

9.5 Traverse versus Delete

Traversal inspects each current item in FIFO order without necessarily removing it. Repeated dequeue empties queue and is destructive. Circular physical index order differs from logical FIFO order after wraparound.

9.6 Error Handling

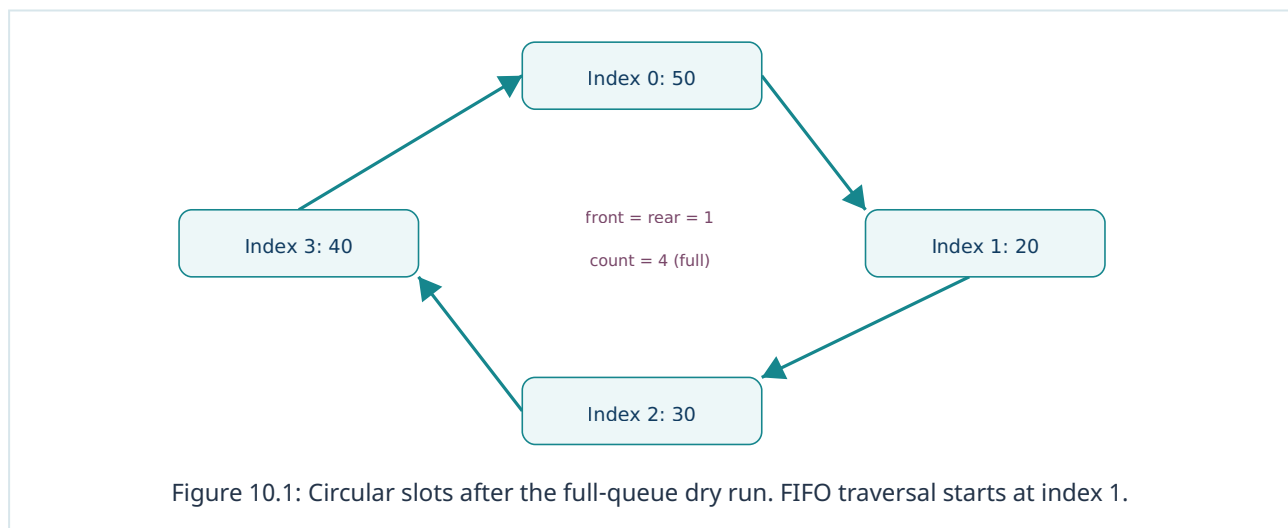
Failed insertion should preserve queue state. Empty dequeue must report underflow before reading. Status/exception can disambiguate failures from legitimate values. Multi-thread shared queue needs synchronization beyond the sequential algorithms in this book.

Practice: EnqueueA,B,C; dequeue twice; enqueueD. **Answer:** ReturnsA,B; queue[C,D], frontC,rearD logically.

UNIT II / CHAPTER 10

अध्याय 10: Array Queue एवं Circular Queue

INSERT, DELETE, TRAVERSE और wraparound



10.1 Simple Linear Array Queue

Use $\text{front}=0, \text{rear}=-1$ initially, rear last occupied position. Empty iff $\text{front} > \text{rear}$. Enqueue checks $\text{rear} = C-1$, then increments rear and stores. Dequeue checks empty, reads $Q[\text{front}]$, increments front. On becoming empty reset $\text{front}=0, \text{rear}=-1$ in this convention.

For partially emptied linear queue, earlier slots may remain unused while rear reaches $C-1$. This is **false overflow** relative to total free storage. Shifting items reuses slots but costs $\Theta(n)$ per shift. Circular indexing avoids that movement.

10.2 Circular Count-based Model

$Q[0..C-1]$, $C > 0$; $\text{front} = \text{rear} = \text{count} = 0$ initially. front points next removal; rear points next insertion. Empty $\text{count} = 0$, full $\text{count} = C$. All C slots usable. $\text{front} = \text{rear}$ can occur both empty and full, distinguished by count.

```

ENQUEUE(Q, x)
    if count = C
        report overflow
    Q[rear] ← x
    rear ← (rear+1) mod C
    count ← count+1

DEQUEUE(Q)
    if count = 0
        report underflow
    value ← Q[front]
    front ← (front+1) mod C
    count ← count-1
    return value
  
```

```

TRAVERSE(Q)
  for k ← 0 to count-1
    index ← (front+k) mod C
    visit Q[index]

```

State fields belong to queue object and updates persist. Enqueue/dequeue $\Theta(1)$; traversal $\Theta(n)$ current items; allocated space $\Theta(C)$.

10.3 Dry Run: C=4

Operation	front	rear	count	Logical FIFO
Initial	0	0	0	[]
enqueue10,20,30	0	3	3	[10,20,30]
dequeue	1	3	2	[20,30]
enqueue40	1	0	3	[20,30,40]
enqueue50	1	1	4	[20,30,40,50]

Physical slots now[50,20,30,40]. Traversal indices1,2,3,0. Another enqueue fails; queue unchanged.

10.4 Alternative One-unused-slot Convention

Without count, one design emptyfront=rear and full(rear+1)modC=front, usable capacityC-1. That full formula does **not** apply to the all-slots count-based design. Declare convention before solving MCQs/numericals.

10.5 Boundary Cases

C=1 count-based queue supports exactly one item: indices stay0, count distinguishes states. When count drops0, front/rear may both be a nonzero index; no reset required. Modulo is defined because C>0.

Practice: Continue C=4 trace: dequeue twice then enqueue60. **Answer:** queue[40,50,60],front3,rear2,count3.

UNIT II / CHAPTER 11

अध्याय 11: Linked List से Queue Implementation

Front और rear references

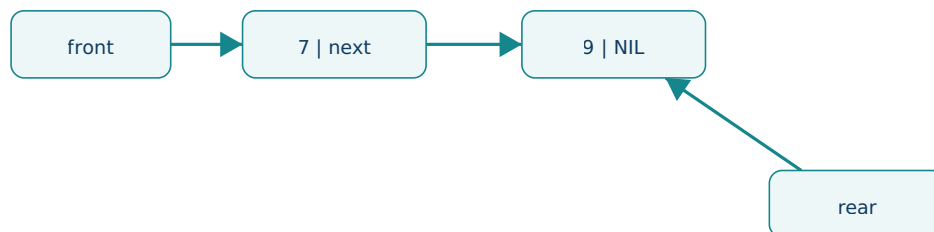


Figure 11.1: Linked queue. Enqueue after rear; dequeue the front node.

11.1 Representation

front/head oldest node; rear/tail newest node. Empty front=rear=NIL. Nonempty rear.next=NIL. Singly nodes data,next. Maintaining rear pointer avoids traversal on insertion.

```

ENQUEUE_LINKED(front, rear, value)
    node ← allocate node
    if allocation fails
        report allocation failure
    node.data ← value
    node.next ← NIL
    if rear = NIL
        front ← node
        rear ← node
    else
        rear.next ← node
        rear ← node
    return (front, rear)

DEQUEUE_LINKED(front, rear)
    if front = NIL
        report underflow
    victim ← front
    value ← victim.data
    front ← victim.next
    if front = NIL
        rear ← NIL
    release victim if manually managed
    return (front, rear, value)
  
```

Allocation failure happens before old state changes. Sole-node removal resets both references; keeping old rear creates dangling/stale reference and corrupts next enqueue.

11.2 Traverse

current=front; while current≠NIL visit data and follow next. $\Theta(n)$, $\Theta(1)$ auxiliary. PEEK requires front≠NIL. Linked queue space $\Theta(n)$, plus per-node/reference overhead.

11.3 Dry Run

Empty → enqueue 7 sets both to 7. enqueue 9 links 7.next=9, rear 9. dequeue returns 7, front 9, rear 9. dequeue returns 9, both NIL. enqueue 11 afterwards starts valid new singleton queue.

11.4 Comparison

Aspect	Circular array	Linked queue
Bounded capacity	Fixed C	Unless explicit bound, allocation-limited
Slot reuse	Modulo indices	New nodes/released nodes
Per-item link	None	next reference
enqueue/dequeue	$\Theta(1)$	$\Theta(1)$ links, allocator ignored
Traverse	Logical modulo order	Follow next
Space	$\Theta(C)$	$\Theta(n)$

11.5 Queue Invariants

Empty iff both endpoints NIL. Nonempty front reachable along next links to rear. rear.next=NIL. Node sequence follows successful enqueue order minus dequeued prefix. Count if tracked equals node count.

Practice: Explain why queue enqueue at list head with dequeue at same head becomes a stack. Which endpoints must differ for FIFO?

UNIT II / CHAPTER 12

अध्याय 12: Priority Queue

Priority-based removal और tie policy

12.1 Definition

Priority queue items have priorities and removal selects highest-service-priority item. Min-priority convention uses smallest numeric key first; max-priority convention largest first. State convention explicitly. Ordinary FIFO ignores priority; equal-priority ties may preserve arrival order if stable policy is specified.

12.2 Example: Lower Number = Higher Priority

Arrival	Item	Priority
1	A	3
2	B	1
3	C	2
4	D	1

With stable equal-key policy, dequeue sequence B,D,C,A. Without specified tie policy B versus D order is not guaranteed. The word “priority” alone does not define numeric direction or tie behavior.

12.3 Unsorted Array Implementation

Append item with priority and arrival sequence. Dequeue scans all items to find smallest(priority, arrival) pair, removes it, shifts suffix to preserve chosen storage order.

```

DELETE_MIN_PRIORITY(items, n)
  if n = 0
    report underflow
  best ← 0
  for i ← 1 to n-1
    if items[i].priority < items[best].priority
      best ← i
    else if priorities equal
      and items[i].arrival < items[best].arrival
        best ← i
  value ← items[best]
  shift items after best one position left
  return (value, n-1)

```

Append $\Theta(1)$ if capacity available; deletion $\Theta(n)$. Swap-with-last deletion can avoid shifting after selection, but stable ties still require arrival metadata if physical order changes.

12.4 Sorted Array, Linked List एवं Heap

Representation	Insert	Remove best
Unsorted array	$\Theta(1)$, available capacity	$\Theta(n)$
Sorted array with best at end	Worst $\Theta(n)$ shifting	$\Theta(1)$
Sorted linked list best at head	Worst $\Theta(n)$ locate position	$\Theta(1)$
Binary heap	$O(\log n)$	$O(\log n)$

Heap peek $O(1)$; building arbitrary heap bottom-up $O(n)$. Heap is additional implementation overview here, not required detailed tree theory. Table excludes resizing/allocation cost as stated.

12.5 Applications एवं Fairness

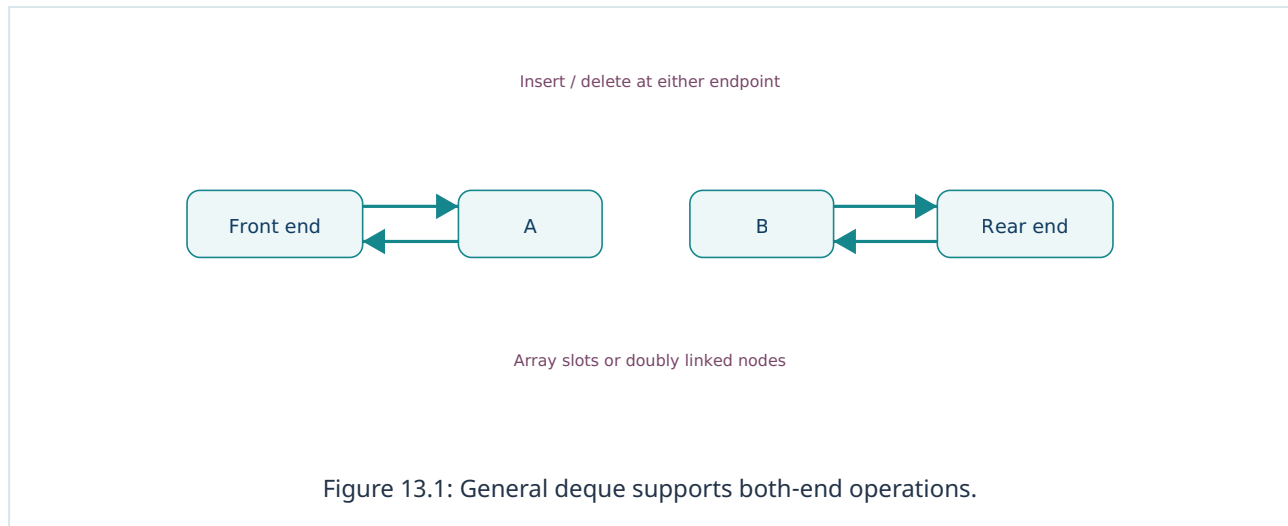
Urgent tasks, event simulation, shortest-path frontier selection. Starvation can happen if lower-priority items continually wait; aging changes priority over time. Priority queue alone does not enforce every fairness policy.

Practice: Max-priority keys [A2, B5, C1, D5], stable ties $\rightarrow$ B, D, A, C.

UNIT II / CHAPTER 13

अध्याय 13: Double Ended Queue (Deque)

Both ends पर insert/delete



13.1 Deque ADT

Deque (pronounced “deck”) double ended queue है. insertFront, insertRear, deleteFront, deleteRear and endpointpeeks देती है. It can model stack or queue by restricting used operations. It does not mean two unrelated queues.

13.2 Restricted Types

Variant	Allowed insertion	Allowed deletion
General deque	Both ends	Both ends
Input-restricted deque	One designated end	Both ends
Output-restricted deque	Both ends	One designated end

Choose designated end and state it. Terms input/output refer to insertion/removal, not console I/O.

13.3 Circular Array Count-based Implementation

front=rear=count=0; rear next insertion at rear, front current front; C>0. Same convention as chapter10.

```

INSERT_FRONT(x)
    if count = C
        report overflow
    front ← (front-1+C) mod C
    Q[front] ← x
    count ← count+1

INSERT_REAR(x)
    if count = C
        report overflow
  
```

```

    Q[rear] ← x
    rear ← (rear+1) mod C
    count ← count+1

DELETE_FRONT()
    if count = 0
        report underflow
    x ← Q[front]
    front ← (front+1) mod C
    count ← count-1
    return x

DELETE_REAR()
    if count = 0
        report underflow
    rear ← (rear-1+C) mod C
    x ← Q[rear]
    count ← count-1
    return x

```

Invariant $\text{rear} = (\text{front} + \text{count}) \bmod C$. Adding C before modulo avoids negative-index problems in languages whose remainder may be negative. All four endpoint operations $\Theta(1)$; traverse uses chapter10 formula.

13.4 Dry Run: $C=4$

insertRear10 $\rightarrow$ [10]; insertRear20 $\rightarrow$ [10,20]; insertFront5 $\rightarrow$ [5,10,20]; deleteRear returns 20 $\rightarrow$ [5,10]; deleteFront returns 5 $\rightarrow$ [10]. Final front0, rear1, count1 with this trace. Draw physical indices to check wraparound.

13.5 Linked Deque एवं Applications

Doubly linked list with maintained head/tail supports both-end insertion/deletion in $\Theta(1)$ link operations. Singly list removal from rear normally needs $\Theta(n)$ predecessor search. Palindrome comparisons, sliding-window algorithms and work scheduling may use deque. Sliding-window maximum uses a monotonic deque plus index-expiry rules, not ordinary FIFO alone.

Practice: Implement stack using only insertRear/deleteRear; FIFO queue using insertRear/deleteFront.

UNIT II / CHAPTER 14

अध्याय 14: Complete Revision एवं Unit Test

Definitions, algorithms और exam preparation

14.1 Master Comparison

Structure	Removal order	Endpoints	Example
Stack	Most recent first	Top only	Undo
FIFO queue	Oldest first	Rear insert/front remove	Print jobs
Priority queue	Best priority first	Representation-dependent	Urgent scheduling
Deque	User-selected endpoint	Both ends	Endpoint operations

14.2 Formula एवं Invariant Sheet

Array stack $\text{emptytop} = -1, \text{fulltop} = C-1, \text{size} = \text{top} + 1$. Linked stack $\text{emptytop} = \text{NIL}$. Linked queue $\text{emptyfront} = \text{rear} = \text{NIL}$. Count-based circular queue $\text{emptycount} = 0, \text{fullcount} = C$. $\text{Nextindex} = (\text{index} + 1) \bmod C$. $\text{Previousindex} = (\text{index} - 1 + C) \bmod C$. Logical kth item $Q[(\text{front} + k) \bmod C]$. $\text{Dequerear} = (\text{front} + \text{count}) \bmod C$.

Expression rules: higher precedence pops; equal pops in ordinary postfix conversion only if incoming operator left-associative. In reversed prefix method, equal-precedence rule flips. Postfix evaluation first popright; right-to-left prefix evaluation first popleft.

14.3 Solved Mixed Questions

Q1: PUSH2, PUSH5, POP, PUSH7, POP, POP → returned 5, 7, 2.

Q2: Enqueue2, enqueue5, dequeue, enqueue7, dequeue, dequeue → returned 2, 5, 7.

Q3: $(A - B) / (C + D) \rightarrow \text{postfix } A \ B \ - \ C \ D \ + \ /$; prefix $/ \ - \ A \ B \ + \ C \ D$.

Q4: $9 \ 3 \ / \ 2 \ 4 \ * \ + \rightarrow 3 + 8 = 11$.

Q5: $C = 5, \text{front} = 4, \text{rear} = 2, \text{count} = 3$. Occupied indices 4, 0, 1. Dequeue twice → $\text{front} = 1, \text{rear} = 2, \text{count} = 1$. EnqueueX → $\text{rear} = 3, \text{count} = 2$. No shifting.

14.4 Unit Test: 40 MCQs

One best answer per question. Convention is this book's count-based circular queue unless stated otherwise.

- Stack service rule: A. LIFO B. FIFO C. Priority only D. Alphabetical
- Stack insertion operation: A. ROTATE B. PUSH C. DELETE_FRONT only D. DEQUEUE
- PEEK changes stack size? A. Removes all B. Decreases by 1 C. No D. Always doubles
- Empty POP causes: A. Overflow B. Wraparound C. Sorting D. Underflow
- CapacityC array stack full: A. $\text{top} = C-1$ B. $\text{top} = -1$ C. $\text{top} = C+1$ D. $\text{top} = 0$ always
- Array stack initialtop: A. 1 always B. -1 C. C D. $n+1$
- Linked stack's top commonly is: A. Tail only always B. Capacity number C. Head reference D. Middle index

8. Linked POP singleton makes top: A. Two nodes B. Old dangling node C. C-1 D. NIL
9. Fixed-array PUSH time: A. $\Theta(1)$ B. $\Theta(n^2)$ always C. $\Theta(n)$ always D. Impossible
10. Geometric dynamic-array PUSH has: A. No copies ever B. Amortized $\Theta(1)$ C. Every push worst $\Theta(1)$ guaranteed D. $\Theta(n^3)$
11. Infix example: A. A B + B. A B C. A+B D. + A B
12. Prefix example: A. A+B B. A B + C. AB D. + A B
13. Postfix example: A. A B + B. + A B C. A+B D. +AB+
14. A+B*C postfix: A. A B C + * B. A B C * + C. A B + C * D. + A * B C
15. A-B-C grouping: A. A-(B+C) B. A+B+C C. (A-B)-C D. A-(B-C)
16. Mathematical[^] associativity assumed: A. Left B. None C. FIFO D. Right
17. A[^]B[^]C postfix: A. A B C [^] [^] B. A B [^] C [^] C. [^] [^] A B C D. A B C + +
18. Left parenthesis on operator stack is: A. Division operator B. Discarded after matching C. Always output D. Numeric operand
19. Ordinary postfix conversion on equal precedence pops when incoming operator is: A. Operand B. Parenthesis only C. Left-associative D. Right-associative
20. Reversed prefix method should reverse: A. Digits inside every number B. Only operands C. Only brackets D. Tokens
21. In reverse-method prefix conversion, equal original left operators: A. Do not pop equal B. Always pop equal C. Drop operands D. End parse
22. In postfix binary evaluation first pop is: A. Ignored B. Right operand C. Left operand D. Operator
23. 8 2 / gives: A. 6 B. 10 C. 4 D. 0.25
24. 2 3 as complete postfix expression is: A. Always 5 B. Always 6 C. Stack empty D. Invalid extra operands
25. Recursion termination requires: A. Base case and progress B. Only endless calls C. No return D. FIFO
26. FACTORIAL(0): A. Undefined in this algorithm B. 1 C. 0 D. -1
27. Function-call return order follows: A. Priority only B. Random-only C. LIFO D. FIFO
28. FIFO queue removes from: A. Rear always B. Top C. Both always D. Front
29. Queue inserts at: A. Rear B. Front always C. Middle always D. Top only
30. Circular next index: A. Data value B. $(i+1) \bmod C$ C. $i+C$ always D. $i-1$ always
31. Count-based circular full: A. $\text{front}=\text{rear}$ alone B. $\text{rear}=-1$ C. $\text{count}=C$ D. $\text{count}=0$
32. One-unused-slot capacity with array C: A. $C+1$ B. $2C$ C. Always C D. $C-1$
33. Count-based $\text{front}=\text{rear}$ can indicate: A. Empty or full B. Only empty C. Only full D. Invalid always
34. Linked queue sole deletion must: A. Reverse nodes B. Reset both endpoints C. Keep rear dangling D. Duplicate head
35. Stable minpriority keys A3, B1, C2, D1 remove: A. D, A, C, B B. C, A, B, D C. B, D, C, A D. A, B, C, D
36. Unsorted-array priority removal needs: A. Always $\Theta(1)$ selection B. No comparison C. Recursion mandatory D. Scan for best
37. Deque allows: A. Both-end operations B. Only top C. Only priority D. No deletion
38. Input-restricted deque restricts: A. Memory to one byte B. Insertion to one end C. Deletion to one end D. All traversal
39. Doubly linked deque known end deletion: A. Impossible B. Always array shift C. $\Theta(1)$ links D. $\Theta(n^2)$ always
40. Circular deque previous index safely uses: A. $i-C$ always B. $i+1$ only C. C/i D. $(i-1+C) \bmod C$

उत्तर-कुंजी:

1-A, 2-B, 3-C, 4-D, 5-A, 6-B, 7-C, 8-D, 9-A, 10-B, 11-C, 12-D, 13-A, 14-B, 15-C, 16-D, 17-A, 18-B, 19-C, 20-D, 21-A, 22-B, 23-C, 24-D, 25-A, 26-B, 27-C, 28-D, 29-A, 30-B, 31-C, 32-D, 33-A, 34-B, 35-C, 36-D, 37-A, 38-B, 39-C, 40-D.

14.5 5-mark एवं 10-mark Questions

5 marks: Stack ADT; overflow/underflow; postfix notation; call-stack role; false overflow; input/output-restricted deque.

10 marks: 1. Array and linked stack implementation with algorithms, diagrams and comparison. 2. Infix-to-postfix conversion: precedence, associativity and complete dry run. 3. Infix-to-prefix reversal method and repeated-operator traps. 4. Postfix evaluation with validation and operand-order example. 5. Recursion using stack: factorial trace and explicit-stack simulation. 6. Array/linked queue INSERT,DELETE,TRAVERSE with boundary cases. 7. Circular queue conventions, formulas and dry run. 8. Priority queue versus deque with representations and applications.

14.6 Mini Project: Expression Workbench

Build menu-based program in your chosen language: tokenize valid binary infix, convert to postfix/prefix, evaluate numeric postfix, display each stack state. Support +, -, *, /, ^ and parentheses. Use chapter4 rules; separate unary minus until parser extended. Multi-digit numbers must remain one token. State real versus integer arithmetic.

Test	Expected
Infix8-3-2	Prefix - - 8 3 2; value3
Infix2^3^2	Postfix2 3 2 ^ ^; value512
Infix(2+3)*4	Postfix2 3 + 4 *; value20
Postfix5 0 /	Division-by-zero error
Postfix2 +	Missing-operand error
Postfix2 3	Extra-operands error
Mismatched(2+3	Parenthesis error

14.7 Queue Lab एवं Final Checklist

Implement capacity4 circular queue and deque. Test empty delete, full insertion, wraparound, singleton and capacity1. Compare logical traversal with physical storage. For linked queue test deletion of sole node followed by new insertion. For priority queue explicitly test tied keys.

Can you explain every state variable? Trace output without guessing? Preserve invariants on failed operations? Distinguish token reversal from character reversal? State complexity conditions? These checks cover the unit's core written and objective questions.