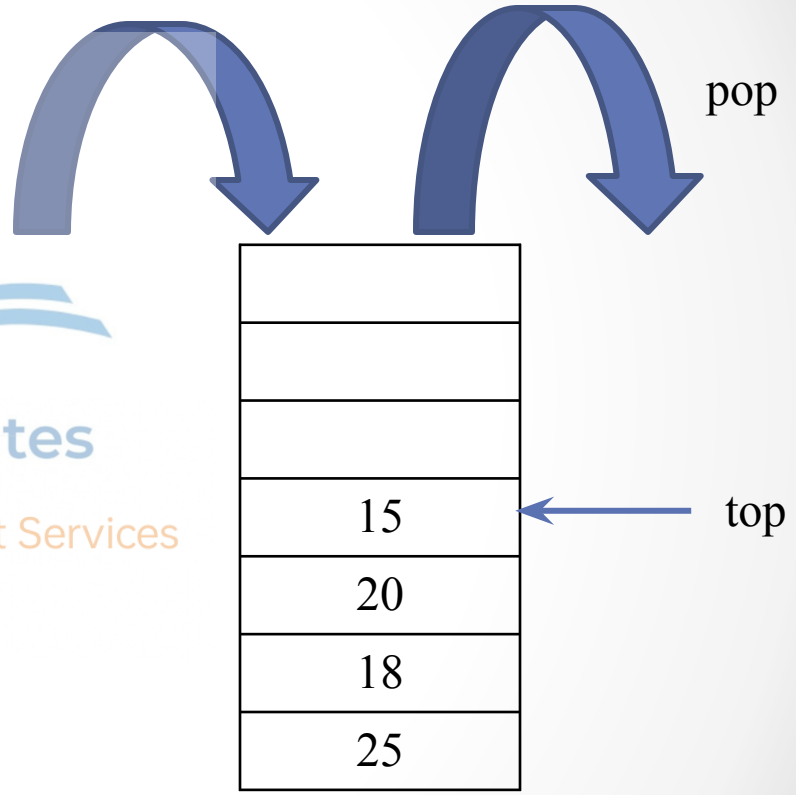
Stacks and Queues

Dr. Jyoti
Professor
Department of Computer Science & Engineering
GJUST, Hisar

Stack

- It is a non-primitive linear data structure in which insertions and deletions take place from a single end.
- This end is called as ‘top of the stack’
- It works on principal Last-in-First-Out (LIFO)i.e. the element which is inserted in the last in removed first.
- Insertion operation is called ‘PUSH’ operation and deletion operation is called ‘POP’ operation.



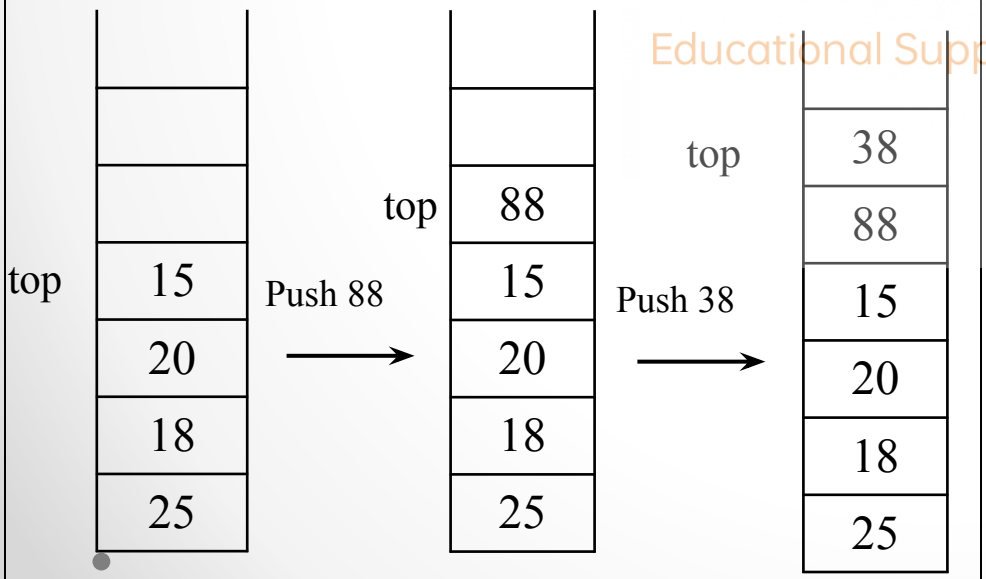
Implementation of Stacks

- Static Implementation
  - For static implementation of stacks arrays are used. But it is not a flexible technique as the size of stack is fixed.
- Dynamic Implementation
  - For dynamic implementation of stacks linked lists are used for storing stacks in memory.

Static Implementation (Push Operation)

Let stack[maxsize] is an array.

1. If top=maxsize-1, then print overflow.
2. Set top=top+1
3. Set stack[top]=item
4. Exit.

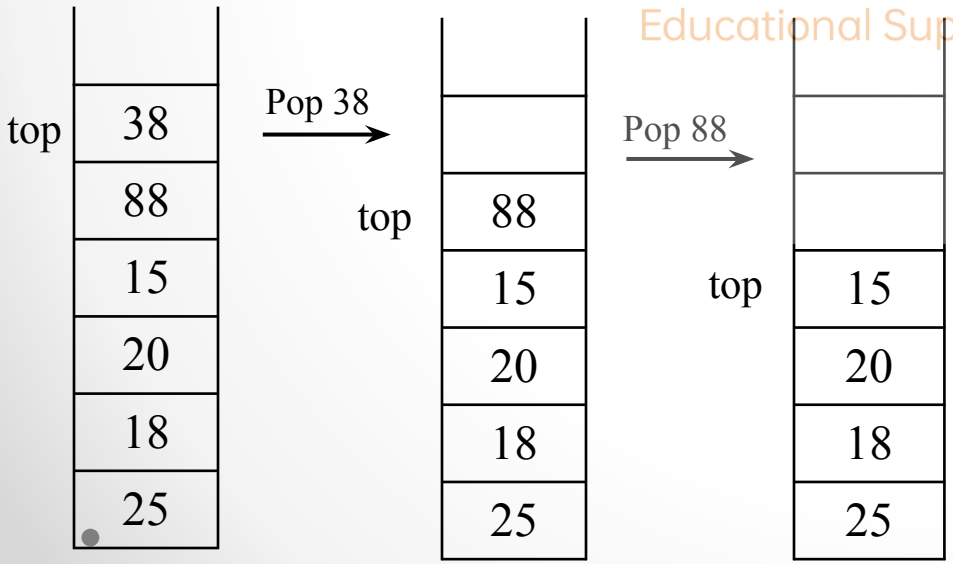



```
void push()
{
int item;
if (top==maxsize-1)
{
   printf("\n" The stack is full");
   return;
}
else
 { printf("Enter Element to be inserted");
   scanf("%d",&item);
   top=top+1;
   stack[top]=item;
 }
 return;
}
```

Static Implementation (Pop Operation)

Let stack[maxsize] is an array.

1. If top<0, then print underflow.
2. Set item=stack[top]
3. Set top=top-1
4. Exit.




```
int pop()
{
int item;
if (top == -1)
{
   printf("\n" The stack is empty");
   return;
}
else
 {
   item=stack[top]
   top=top-1;
 }
  return(item);
}
```

Dynamic Implementation (Push Operation)

```
struct stack
{  int info;
   struct stack *next;
}
typedef struct stack st;
st *top=NULL;
```

```
void push()
{
   st *temp;
   temp=(st*)malloc(sizeof(st));
   printf("\n Enter the value");
   scanf("%d",&temp ->info);
   temp ->next=top;
   top=temp;
   return;
}
```

```
int pop()
 { st *ptr;
   int item;
   ptr = top;
   if (top = = NULL)
   { printf("\n Stack is Empty");
     return;
   }
   else
   { top=top ->next;
     item=ptr->info;
     free(ptr);
    }
    return(item);
 }
```

```
void display()
{ st *ptr;
  ptr=top;
  while(ptr-> next!=NULL)
  { printf("\n No.=%d", ptr->info);
    ptr=ptr->next;
   }
}
```

Applications of Stacks

Function Calls

- In Programming, whenever you make a call from one function to the another function. The address of the calling function gets stored in the Stack. So, when the called function gets terminated. The program control move back to the calling function with the help of the address which was stored in the Stack. So, Stack plays the main role when it comes to Calling a Function from other Function.

Expression Conversion and Evaluation

- There are 3 types of expression we use in Programming, which are Infix Expression, Prefix Expression and Postfix Expression.Infix Expression is represented as X + Y. Prefix Expression is represented as +XY and Postfix Expression is represented as XY+. In order to evaluate these expressions in Programming, a Data Structure called Stack is used.Similarly, Stack is also used for Converting one expression into another. For example, converting Infix to Postfix or Infix to Prefix.

Parenthesis Checking

- In Programming, we make use of different type of parenthesis, like – (,), {, }, which are used for opening and closing a block of code. So, these parenthesis get stored in Stack and control the flow of our program.

String Reversal

- String Reversal is another amazing Application of Stack. Here, one by one each character of the Stack get inserted into the Stack. So, the first character of the Stack is on the bottom of the Stack and the last character of the String is on the Top of the Stack. After performing the pop operation in Stack, we get the String in Reverse order.

Syntax Parsing

- As many of the Programming Languages are context-free languages. So, Stack is also heavily used for Syntax Parsing by most of the Compilers.

Memory Management

- Memory Management is the important function of the Operating System. Stack also plays the main role when it comes to Memory Management.

Function Call

- Programs compiled from High Level Language make use of stack frame for working memory of each procedure or function invocation.
- When a procedure or function is called, a number of words are pushed onto a program stack. When procedure or function returns, this frame of data is popped off the stack and used to track back the path where it sent the called function. This data may consists of – current contents of the caller function, the address of the instruction just next to the call instruction and list of parameters.
- In recursive functions it is used to save postponed decisions. A well-defined recursive function must possess the following two properties:
  - There must be some base criteria, for which the procedure does not call itself.
  - Each time the procedure calls itself, it must be closer to the base criteria.

For Example:

```
Factorial(FACT, N)
if N=0, then set FACT=1 and return.
Call FACTORIAL(FACT, N-1)
Set FACT = N*FACT
Return
```

N! = N * (N-1)! And 0!=1 Assume N=4

4! = 4 * 3!

3 * 2!

2 * 1!

1 * 0!

1

Polish Notation

Arithmetic operation has constants and operators. It has three types of notations – Infix, Prefix and Postfix.

Infix – e.g. A+B. Evaluated using rules of BODMAS.

Prefix – e.g. +AB. It is also called Polish notation in the honor of a Mathematician

Postfix – e.g. AB+. It is also called Reverse Polish notation.

- The fundamental property of Polish Notation is that the order in which the operations are to be performed is completely determined by the positions of the operators and operands in the expression.
- This is most suitable and universally accepted for a computer to calculate any expression.
- Any expression entered into computer is first converted into postfix notation, stored in a stack and then evaluated.
- There is no need of operator precedence rules.

Transforming Infix Expression into Postfix Expression

The following algorithm transforms the infix expression 'Q' into its equivalent postfix expression 'P'. This algorithms uses a stack to temporarily hold operators and left parenthesis. The operators are removed from the stack and operands from Q to construct postfix expression P from left to right.

We begin by pushing a left parenthesis '(' onto stack and adding a right parenthesis ')' at the end of Q. The algorithm is completed when stack is empty.

POLISH(Q,P)

1. Push '(' onto STACK and add ')' at the end of Q.
2. Scan Q from left to right and repeat steps 3 to 6 for each element of Q until STACK is empty.
3. If an operand is encountered, add it into P.
4. If a left parenthesis '(' is encountered, push it onto STACK.
5. If an operator Θ is encountered, then:
   - Repeatedly pop from STACK each that operator which has the same precedence as or higher precedence than Θ and add to P
   - Add Θ to STACK.
6. If right parenthesis ')' is encountered, then:
   - Repeatedly pop from STACK and add to P each operator until a left parenthesis '(' is encountered.
   - Remove the left parenthesis '(' is encountered. [Do not add the left parenthesis '(' to P]
7. Exit

Example

Q: A + (B * C - (D / E ^ F) * G) * H

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

S.No.	Symbol Scanned	STACK	Expression P
1	A	(	A
2	+	(+	A
3	(	(+ (	A
4	B	(+ (	A B
5	*	(+ (*	A B C
6	C	(+ (*	A B C
7	-	(+ (-	A B C *
8	(	(+ (- (	A B C *
9	D	(+ (- (	A B C * D
10	/	(+ (- (/	A B C * D
11	E	(+ (- (/	A B C * D E
12	^	(+ (- (/ ^	A B C * D E
13	F	(+ (- (/ ^	A B C * D E F
14	)	(+ (-	A B C * D E F ^ /
15	*	(+ (- *	A B C * D E F ^ /
16	G	(+ (- *	A B C * D E F ^ / G
17	)	(+	A B C * D E F ^ / G * -
18	*	(+ *	A B C * D E F ^ / G * -
19	H	(+ *	A B C * D E F ^ / G * - H
20	)		A B C * D E F ^ / G * - H * +

Note:

- In row 7, '-' operator sends '*' to P from the STACK and '-' onto the STACK (Step 5(a)).
- In row 14, right parenthesis ')' sends '^' and '/' operators to P and then removes the left parenthesis '('.
- In row 20, right parenthesis ')' sends '*' and '-' operators to P and then removes the left parenthesis '('.

Evaluation of Postfix Notation

Suppose ‘P’ is an arithmetic expression written in Postfix notation and scanned from left to right. The following algorithm uses a STACK to hold operands. As an operator appears in the ‘P’ during scanning, the topmost operands are popped off and calculated applying the encountered operator. Then the result is again pushed onto the STACK. Finally, there will be a single value at the end of the process in the STACK.

1. Add a right parenthesis ‘)’ at the end of ‘P’. [This acts as a sentinel]
2. Scan ‘P’ from lest to right and repeat steps 3 to 4 for each element of P until the sentinel ‘)’ is encountered.
3. If an operand is encountered, put it on to the STACK.
4. If an operator Θ is encountered, then:
   a) Remove the top two elements of the SATCK, where A is the top element and B is the next to the top element.
   b) Evaluate B Θ A.
   c) Place the result of (b) back on to the SATCK.
5. Set VALUE equal to the top element on the STACK.
6. Exit

Example

P : A B C + * D E / -)

5 6 2 + * 12 4 / -)

Symbol Scanned	STACK
5	5
6	5 6
2	5 6 2
+	5 8
*	40
12	40 12
4	40 12 4
/	40 3
-	37
)	

Tower of Hanoi - Rules

- Only one disk may be moved at a time (only the top disk on any peg may be moved to any other peg)
- A larger disk can be placed on a smaller (*smaller disk can not be placed on larger*)

Tower of Hanoi – Plates Move

1.

From Tower A to Tower B

2.

From Tower A to Tower C

3.

From Tower B to Tower C

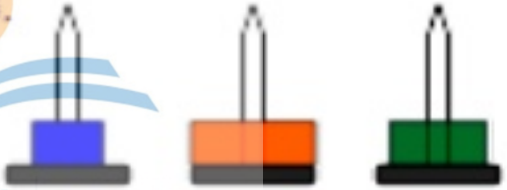
4.

From Tower A to Tower B

5.

From Tower C to Tower A

6.

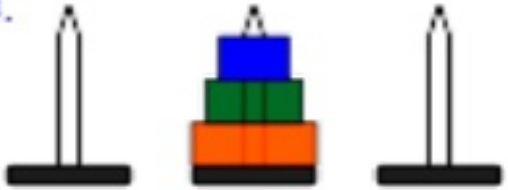


From Tower C to Tower B

7.

From Tower A to Tower B

8.



Problems for Practice

Convert the following infix expressions into their equivalent postfix notation and evaluate the postfix expressions using given values. Draw proper tables for your answers.

A*(B+D)/E-F*(G+H/K)

A=5, B=3, D=3, E=1, F=2, G=5, H=10, K=2

(A+B)*C+D/(E+F*G)-H

A=5, B=3, C=3, D=6, E=2, F=1, G=4, H=10

A*(B+D)/E-F*(G+H/K)

A=5, B=3, D=3, E=1, F=2, G=5, H=10, K=2

S.No.	Symbol Scanned	STACK	Expression P
1	A	(	A
2	*	(*	A
3	(	(* (	A
4	B	(* (	A B
5	+	(* (+	A B
6	D	(* (+	A B D
7	)	(*	A B D +
8	/	(/	A B D + *
9	E	(/	A B D + * E
10	-	(-	A B D + * E /
11	F	(-	A B D + * E / F
12	*	(- *	A B D + * E / F
13	(	(- * (	A B D + * E / F
14	G	(- * (	A B D + * E / F G
15	+	(- * (+	A B D + * E / F G
16	H	(- * (+	A B D + * E / F G H
17	/	(- * (+ /	A B D + * E / F G H
18	K	(- * (+ /	A B D + * E / F G H K
19	)	(- *	A B D + * E / F G H K / +
20	)		A B D + * E / F G H K / + * -

Symbol Scanned	STACK
5	5
3	5 3
3	5 3 3
+	5 6
*	30
1	30 1
/	30
2	30 2
5	30 2 5
10	30 2 5 10
2	30 2 5 10 2
/	30 2 5 5
+	30 2 10
*	30 20
-	10
)	

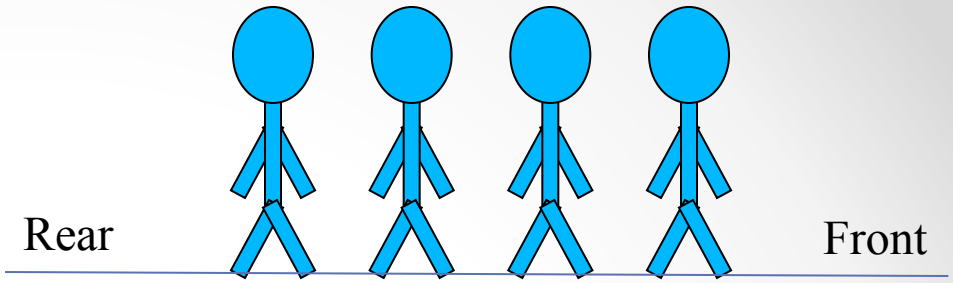
Q=(A+B)*C+D/(E+F*G)-H)

A=5, B=3, C=3, D=6, E=2, F=1, G=4, H=10

S.No.	Symbol Scanned	STACK	Expression P
1	(	((	
2	A	((	A
3	+	((+	A
4	B	((+	A B
5	)	(	A B +
6	*	(*	A B +
7	C	(*	A B + C
8	+	(+	A B + C *
9	D	(+	A B + C * D
10	/	(+ /	A B + C * D
11	(	(+ / (	A B + C * D
12	E	(+ / (	A B + C * D E
13	+	(+ / (+	A B + C * D E
14	F	(+ / (+	A B + C * D E F
15	*	(+ / (+ *	A B + C * D E F
16	G	(+ / (+ *	A B + C * D E F G
17	)	(+ /	A B + C * D E F G * +
18	-	(-	A B + C * D E F G * + / +
19	H	(-	A B + C * D E F G * + / + H
20	)		A B + C * D E F G * + / + H -

Symbol Scanned	STACK
5	5
3	5 3
+	8
3	8 3
*	24
6	24 6
2	24 6 2
1	24 6 2 1
4	24 6 2 1 4
*	24 6 2 4
+	24 6 6
/	24 1
+	25
10	25 10
-	15
)	

Queues



- It is a non-primitive linear data structure in which new elements are inserted at one end called 'Rear' end and deletions take place from a another end called 'Front' end. Insertions are restricted to rear end and deletions restricted to front end.
- It is logically of First-in-First-Out (FIFO) nature i.e. the element which is inserted first is removed first.

- Rear = Rear +1
  when an element is inserted into the QUEUE
- Front = Front +1
  when an element is deleted from the QUEUE
- Rear = NULL and Front = NULL,
  QUEUE is empty
- Rear = 0 and Front =0 or Rear = Front <> NULL
  QUEUE has only one element

Implementation of Queues

- Static Implementation
  - For static implementation of stacks arrays are used. But it is not a flexible technique as the size of queue is fixed.
- Dynamic Implementation
  - For dynamic implementation of stacks linked lists are used for storing queues in memory.

Types of Queues

- Simple Queue
- Circular Queue
- Priority Queue
- Deque

Insertion in a Queue (Static Implementation)

QINSERT(QUEUE, N, FRONT, REAR, ITEM)

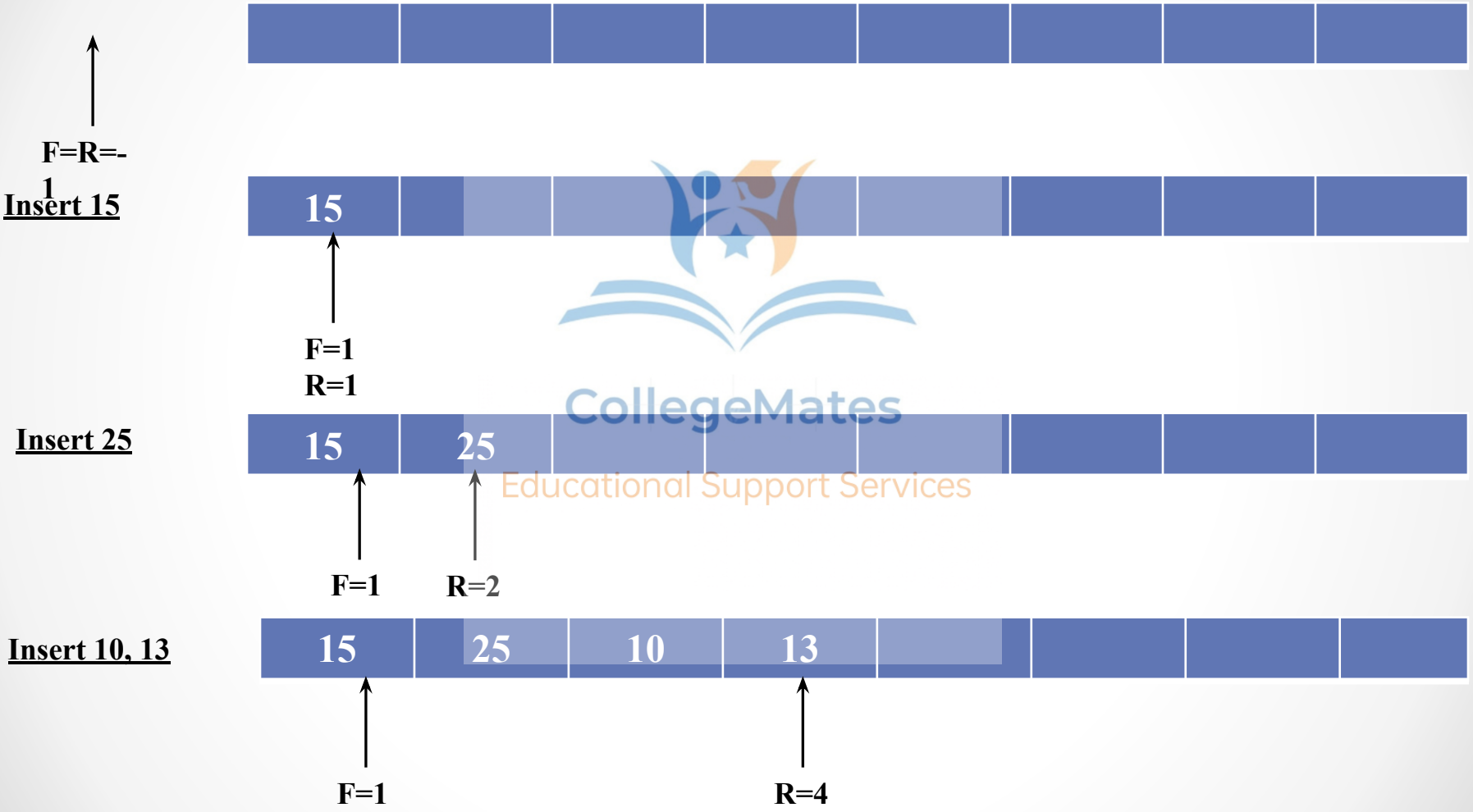
1. If FRONT = 1 and REAR = N
   then overflow and return.
2. If FRONT = NULL, then
   FRONT = 1 and REAR = 1
   else
   REAR = REAR + 1
3. Set QUEUE[REAR] = item
4. Return.

Deletion from a Queue (Static Implementation)

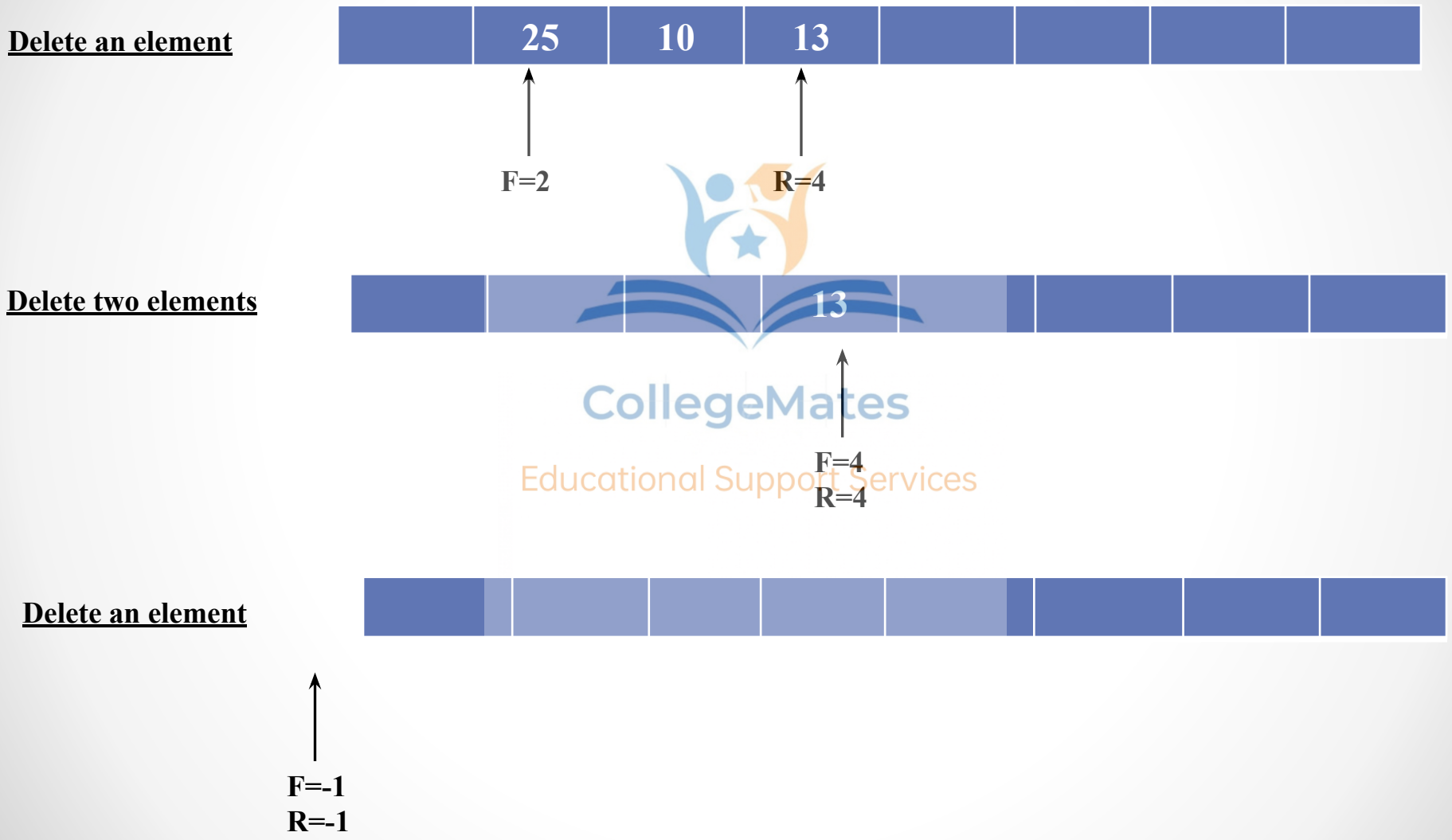
QDELETE(QUEUE, N, FRONT, REAR, ITEM)

1. If FRONT = NULL
   then underflow and return.
2. Set item = QUEUE[FRONT]
3. If FRONT = REAR, then (there is only one element)
   Set FRONT = NULL and REAR = NULL
   else
   FRONT = FRONT + 1
4. Return.

Example



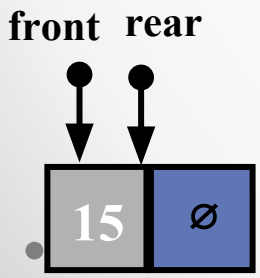
Example

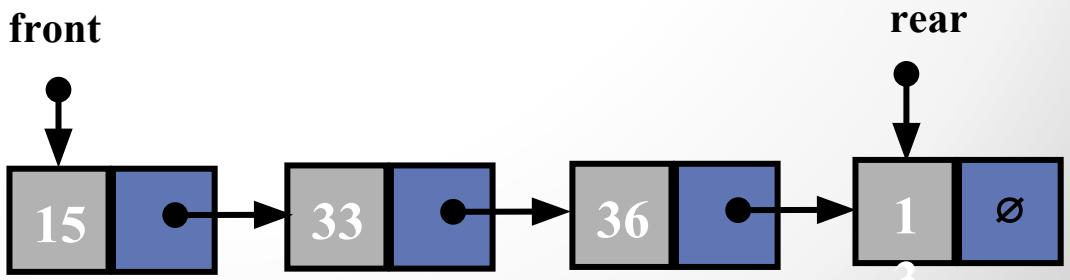


Insertion in a Queue (Dynamic Implementation)

```
struct queue_type
{
  int info;
  struct queue_type
*next;
}
typedef struct
queue_type queue;
queue *rear, *front;
void add();
int dele();
void traverse();
```

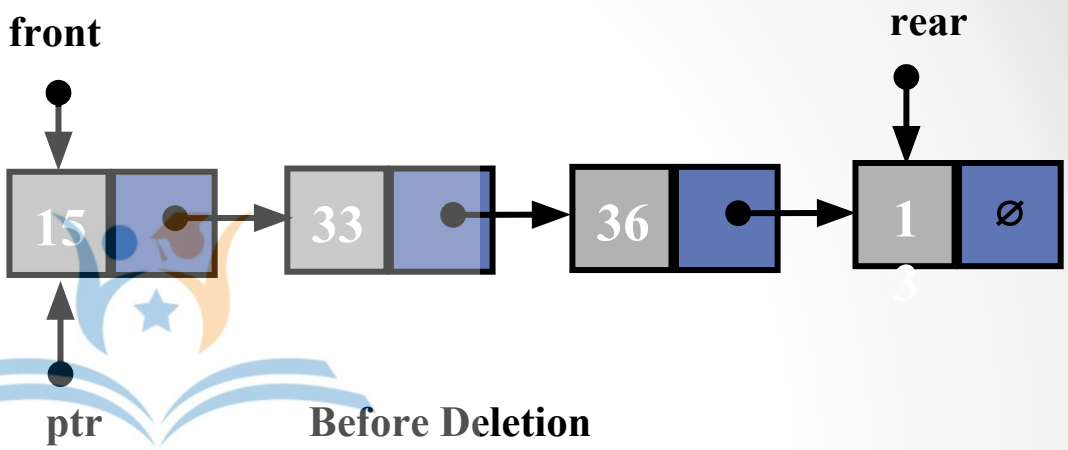
```
void add()
{
 queue *temp;
 temp=(queue*)malloc(sizeof(queue));
 printf("Enter the data");
 scanf("%d", &temp->info);
 temp->next=NULL;
 if(front==NULL)
   front=rear=temp;
 else
 {
  rear->next=temp;
   rear=temp;
 }
}
```

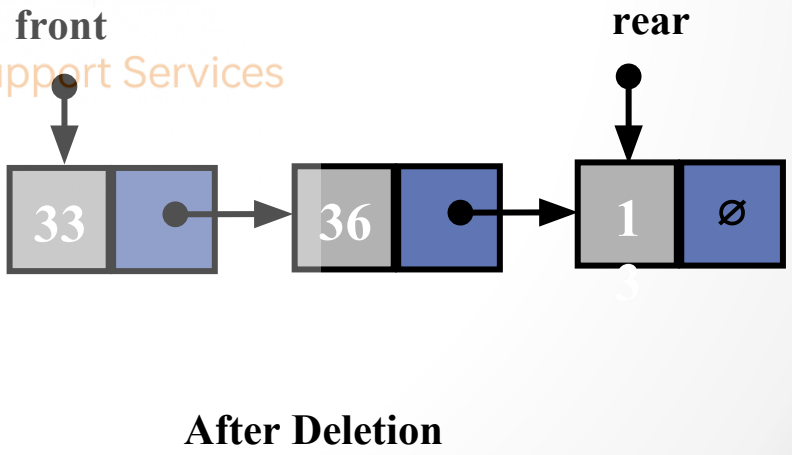





Deletion in a Queue (Dynamic Implementation)

```
int del()
{
 queue *ptr;
 int value;
 if(front==NULL)
 {
  printf("Queue is Empty");
  return(0);
 }
 else
 {
  ptr=front;
  value=front->info;
  front=front->next;
  free(ptr);
 }
 return(value);
}
```





Circular Queue

- Circular queue are used to remove the drawbacks of simple queue
- Both the front and the rear pointers wrap around the beginning of the queue.
- It is also called as “Ring Buffer”.

Insertion in a Circular Queue (Static Implementation)

QINSERT(QUEUE, N, FRONT, REAR, ITEM)

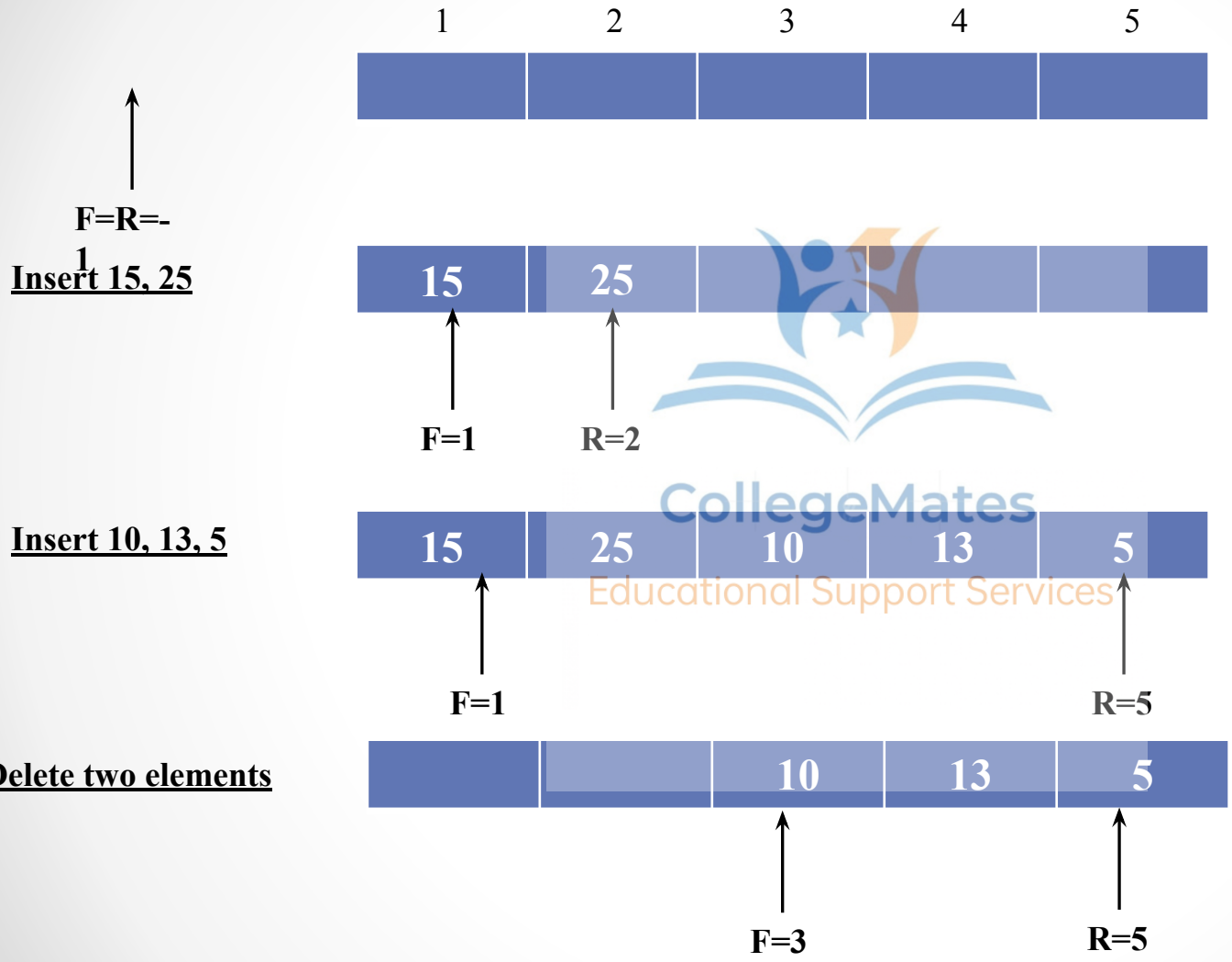
1. If FRONT = 1 and REAR = N or FRONT=REAR+1
   then overflow and return.
2. If FRONT = NULL, then
   FRONT = 1 and REAR = 1
   elseif REAR=N then
   set REAR=1
   else
   REAR = REAR + 1
3. Set QUEUE[REAR] = item
4. Return.

Deletion from a Circular Queue (Static Implementation)

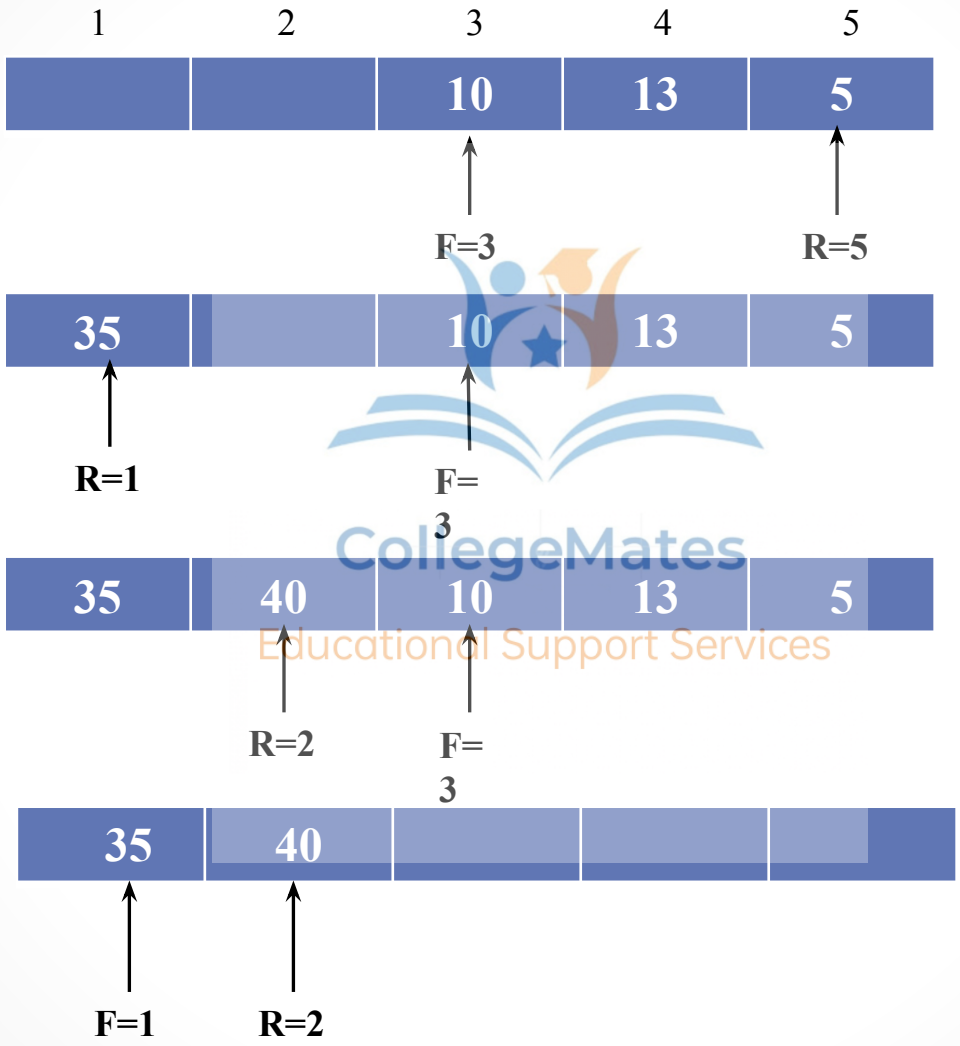
QDELETE(QUEUE, N, FRONT, REAR, ITEM)

1. If FRONT = NULL
   then underflow and return.
2. Set item = QUEUE[FRONT]
3. If FRONT = REAR, then (there is only one element)
   Set FRONT = NULL and REAR = NULL
   elseif FRONT=N then
   Set FRONT=1
   else
   FRONT = FRONT + 1
4. Return.

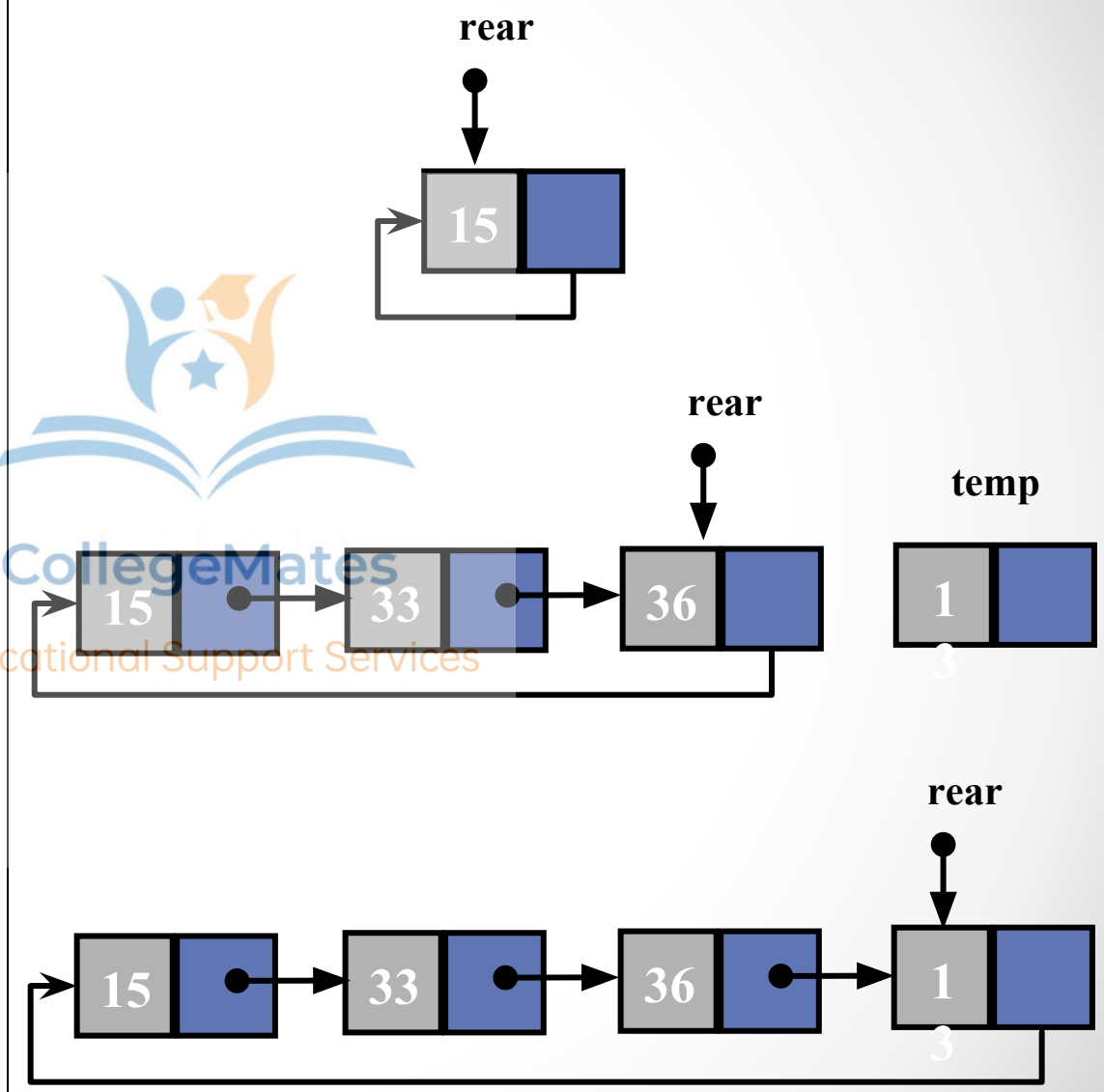
Example



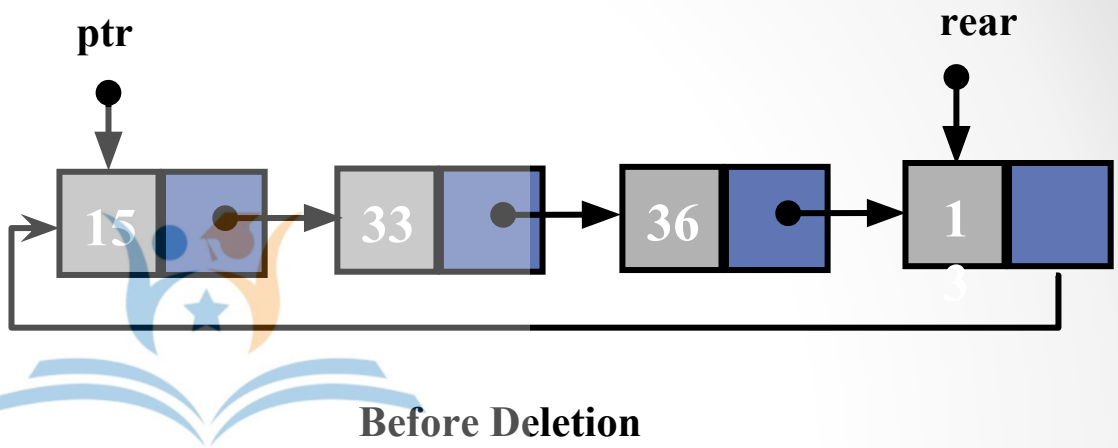
Example



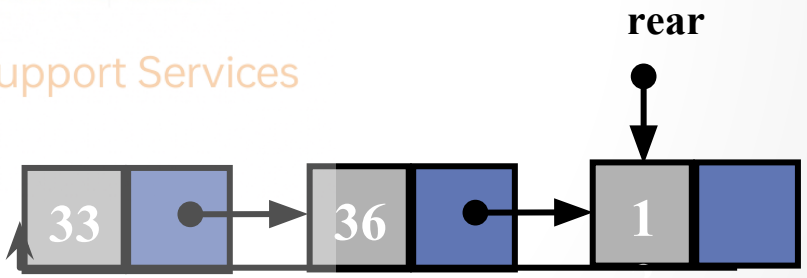
```
void add()
{
 queue *temp;
 temp=(queue*)malloc(sizeof(queue));
 printf("Enter the data");
 scanf("%d", &temp->info);
 temp->next=NULL;
 if(rear==NULL)
{
  rear=temp;
  rear->next=rear;
}
else
 {
   temp->next=rear->next;
   rear->next=temp;
   rear=temp;
 }
}
```



```
int del()
{
 queue *ptr;
 int value;
 if(rear==NULL)
 {
  printf("Queue is Empty");
  return(0);
 }
 else
 {
  ptr=rear->next;
  value=ptr->info;
  rear->next=ptr->next;
  free(ptr);
 }
 return(value);
}
```



Before Deletion



After Deletion

Deques

- Deque stands for double ended queue
- Elements can be inserted or deleted at either end.
- It is of two types:
  - Input restricted deque
    - Insertions take place only at on end and deletions can be performed on both the ends.
  - Output restricted deque
    - Deletions take place only at on end and insertions can be performed on both the ends.

Priority Queue

- It is collection of elements where elements are stored according to their priority levels.
- Insertion and deletions of the elements from the queue are decided by their priority.
- An element of higher priority is processed first.
- Two elements of same priority are processed on first-come-first-served basis.

Applications of Queues

- Data getting transferred between the IO Buffers (Input Output Buffers) e.g. Keyboard Buffers.
- To implement printer spooler so that jobs can be printed in the order of their arrival.
- Implements various aspects of operating system like CPU scheduling and Disk scheduling.
- Managing shared resources between various processes e.g. CPU, memory, I/O devices etc.
- Job scheduling algorithms – In handling jobs in multiuser, multiprogramming environment and time-sharing environment. Ready queue, waiting queue etc.
- Round robin scheduling.
- Priority queues are used to sort heaps.
- Priority queues are used in operating system for load balancing and interrupt handling.
- In traffic light, depending upon the traffic, the colors will be given priority.