Data Structure

GATEWAY CLASSES

Unit-II

Stacks & Queues

ONE SHOT

KAPIL SIR

DATA STRUCTURE
Syllabus UNIT-2

II	**Stacks:** Abstract Data Type, Primitive Stack operations: Push & Pop, Array and Linked Implementation of Stack in C, Application of stack: Prefix and Postfix Expressions, Evaluation of postfix expression, Iteration and Recursion- Principles of recursion, Tail recursion, Removal of recursion Problem solving using iteration and recursion with examples such as binary search, Fibonacci numbers, and Hanoi towers. Tradeoffs between iteration and recursion. **Queues:** Operations on Queue: Create, Add, Delete, Full and Empty, Circular queues, Array and linked implementation of queues in C, Dequeue and Priority Queue.

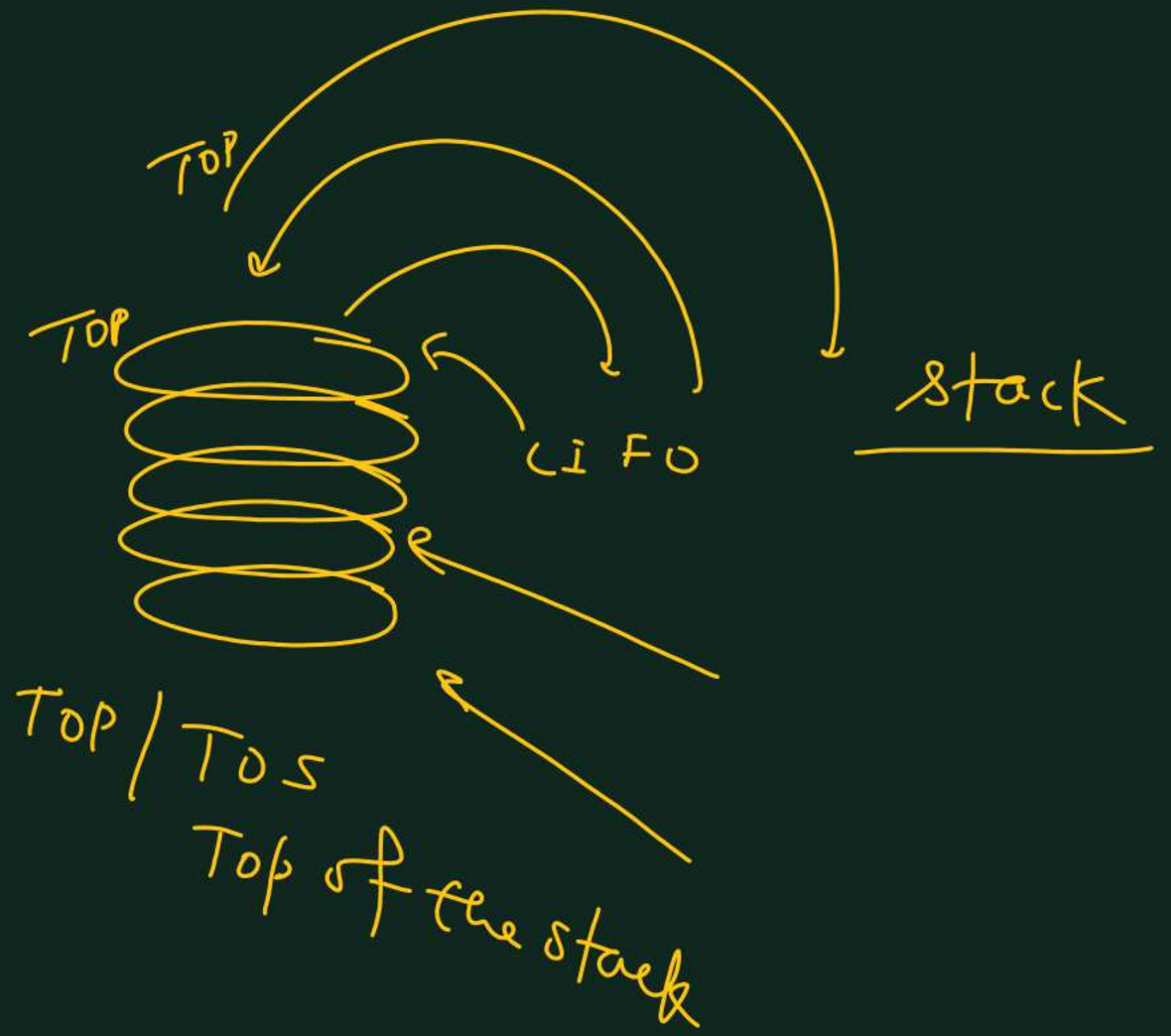
What is Stack Data Structure?

- We know that linked list is a collection of nodes and there are different types of possible operations which can be performed on linked list data structure.
- Any item can be inserted to any position of linked list and in the same manner any item can be deleted from any position of the linked list.
- So in case of linked list which is a dynamic data structure, there is no restriction to perform any operation.
- Stack is another non primitive data structure in which it is possible to create the restriction for performing the operation on the data structure.
- Stack works on LIFO property.

GATEWAY CLASSES

LIFO

FIFO



..What is Stack Data Structure?

LIFO

- Stack is a linear data structure in which operation of insertion and deletion can be performed only at one end and that end is known as top of the stack.

TOS/TOP

- A common model of a stack is plates in a marriage party. Fresh plates are pushed onto the top and popped from the top.

remove

insert

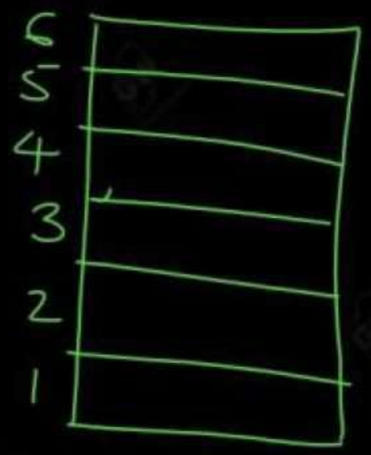
- The operation of insertion of an element is known as PUSH operation and the operation of deletion of an element is known as POP operation.

GATEWAY CLASSES

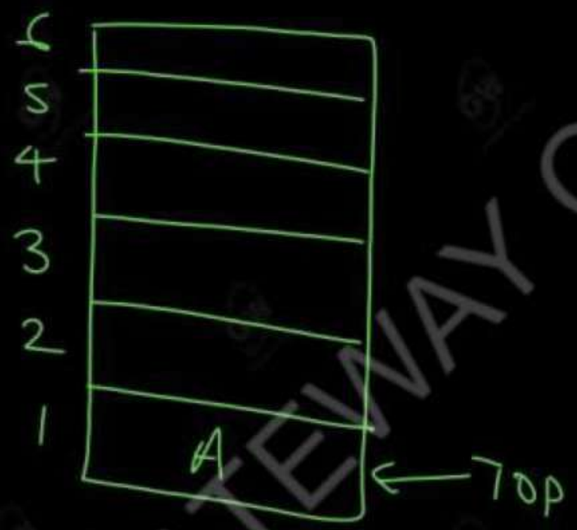
Example.1 :- Consider the following example where we have an empty STACK which can contain the following 6 elements:-

Elements are :- A, B, C, D, E, F

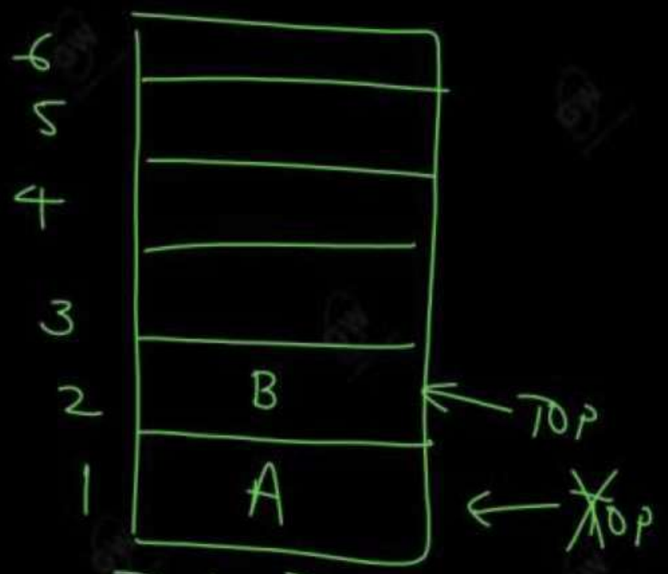
After pushing elements one by one the STACK will be as follows:-



STACK: Initial Empty
TOS = 0
TOP = 0
(a)



STACK: PUSH 'A'
OR STACK[TOP] = PUSH 'A'
TOP = TOP + 1 = 0 + 1 = 1
(b)



TOP = TOP + 1 = 1 + 1 = 2
STACK[TOP] = PUSH 'B'
(c)

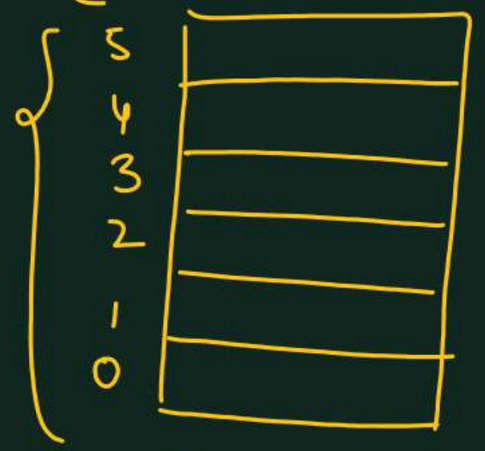
GATEWAY CLASSES

Insert = PUSH
Delete = POP

A, B, C, D E, F

STACK[6]

(a)



Initially empty

PUSH

POP X

UNDERFLOW

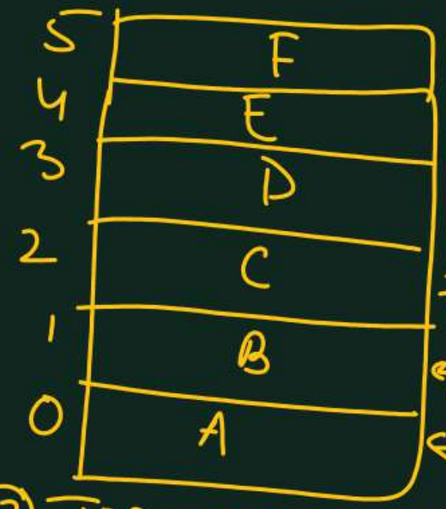
TOP

TOS = -1

PUSH

① if (TOS == MAXSIZE - 1)
{
"OVERFLOW"
}

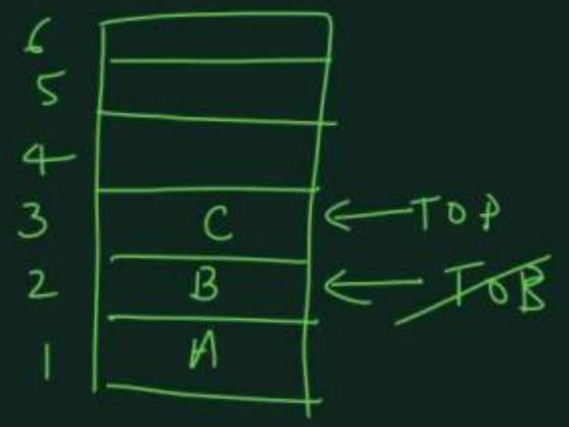
(b) PUSH 'A'



TOS
~~TOS~~
~~TOS~~
~~TOS~~
~~TOS~~
~~TOS~~

② TOS = TOS + 1 = -1 + 1 = 0

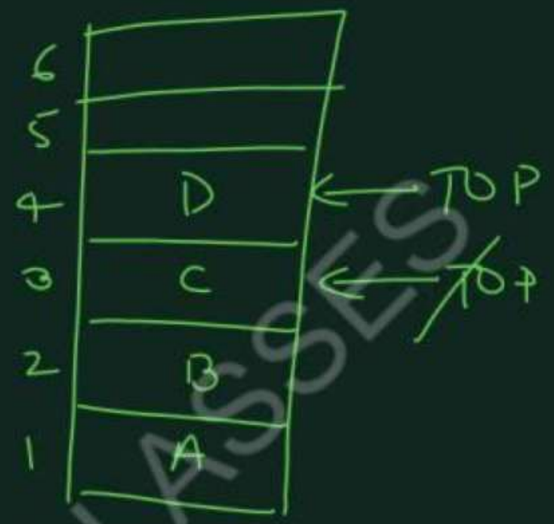
③ STACK[TOS] = ITEM
= 'A'



TOP++ ; TOP = TOP+1
= 2+1
= 3

STACK[TOP] = PUSH 'C'

(d)



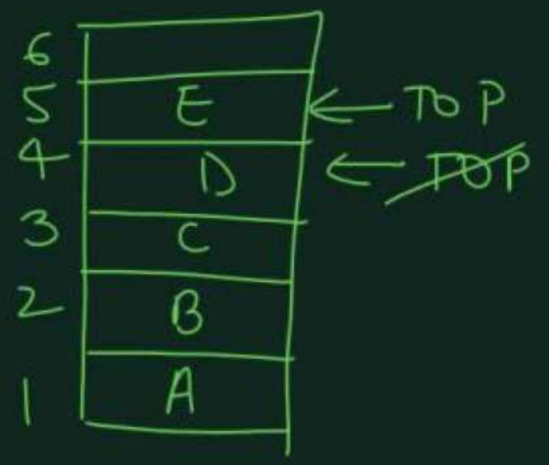
TOP++ ; TOP = TOP+1
= 3+1 = 4

STACK[TOP] = PUSH 'D'

STACK[4] = 'D'

(e)

GATEWAY CLASSES

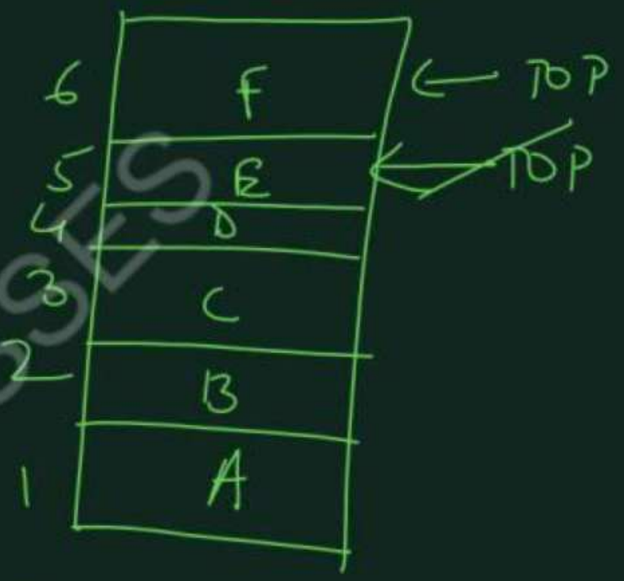


TOP++ ; TOP = TOP+1
= 4+1=5

STACK[TOP] = PUSH 'E'

STACK[5] - E

(f)



TOP++ ; TOP = TOP+1
= 5+1=6

STACK[TOP] = PUSH 'F'

i.e. STACK[6] = 'F'

(g)

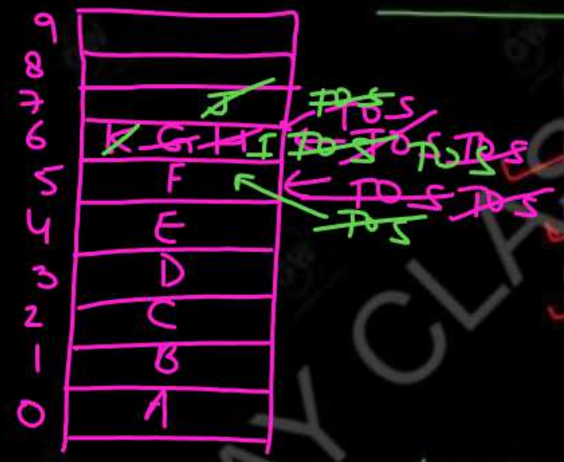
GATEWAY CLASSES

Example.2 :- Consider the following STACK of characters where STACK can contain 10 items i.e. 10 memory cells are available.

STACK:- A, B, C, D, E, F, G, -, -, -

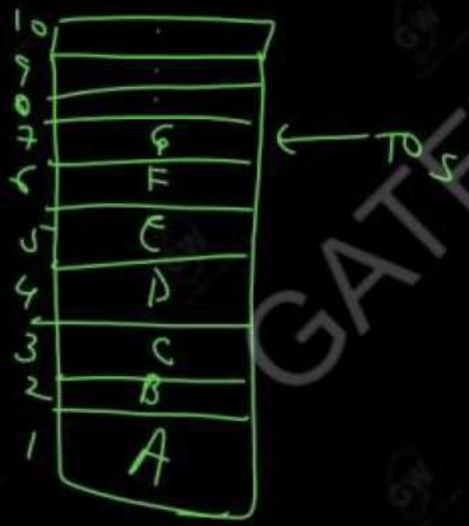
Describe the STACK as the following operations takes place. Here PUSH means insertion of an element and POP means deletion of an element. Write the final STACK after performing all operations. List the popped items.

(a) POP(STACK,ITEM)
(b) PUSH(STACK,H)
(c) POP(STACK,ITEM)
(d) PUSH(STACK,K)
(e) POP(STACK,ITEM)
(f) PUSH(STACK,I)
(g) PUSH(STACK,J)
(h) POP(STACK,ITEM)

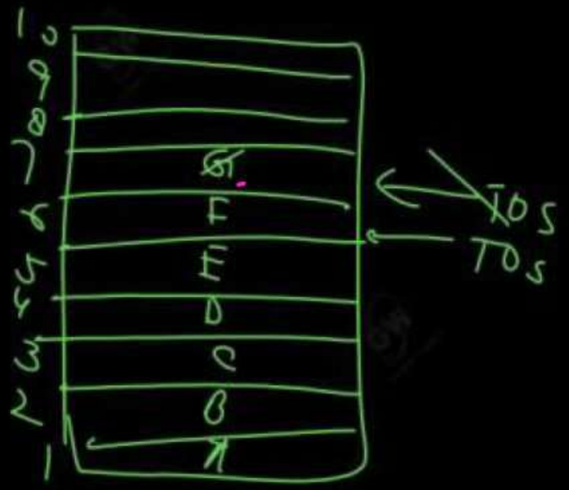


Solution:- 10 items

Initial Stack TOS = 7



(a)



ITEM = STACK[TOS]
= STACK(7)
ITEM = 'G'
TOS = TOS − 1
= 7 − 1
= 6

GATEWAY CLASSES

(a) POP (STACK, ITEM)

(1) if (TOS == -1)

UNDERFLOW | NOT POSSIBLE TO DELET THE ITEM.

if (6 == -1) F

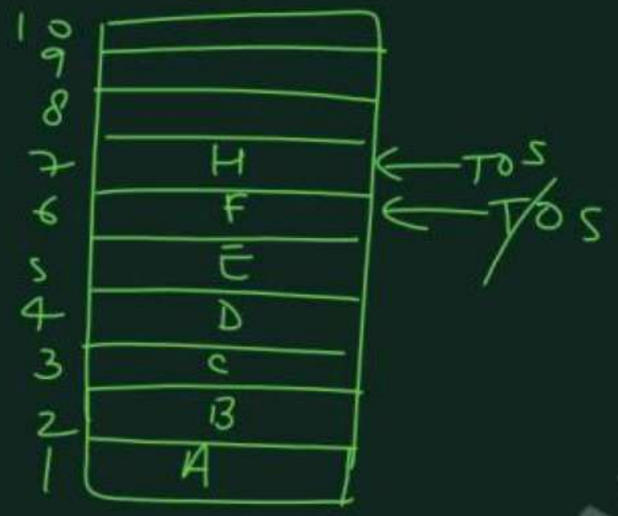
(2) POPPED_ELEMENT = STACK[TOS]

= STACK[6]

= 'G'

(3) TOS = TOP - 1

(b) PUSH (STACK, H)

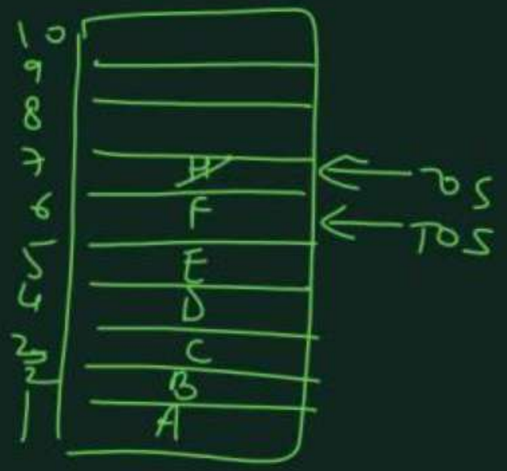


TOS = TOS + 1
= 6 + 1
= 7

STACK [TOS] = ITEM

STACK [7] = H

(c) POP (STACK, ITEM)



ITEM = STACK[TOS]

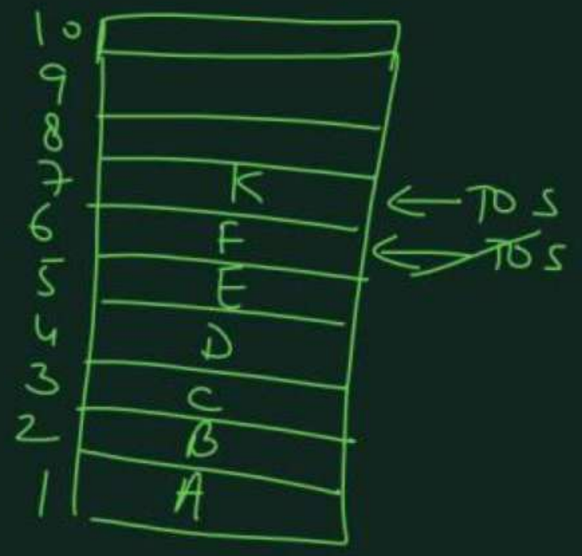
ITEM = STACK[7]
= H

TOS = TOS - 1

TOS = 7 - 1 = 6

GATEWAY CLASSES

(d) PUSH (STACK, K)

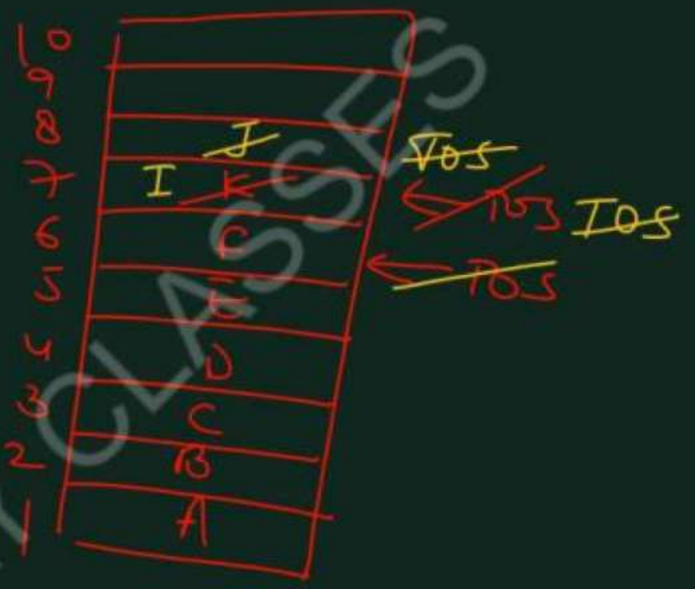


STACK

TOS++

STACK[TOS] = ITEM
= K

(c) POP (STACK, ITEM)



ITEM = STACK[TOS]
= K
TOS = TOS − 1
= 7 − 1 = 6

item = 'J'

PUSH
POP } TOS

GATEWAY CLASSES

PUSH, POP — TOS

PUSH — you should have the item.

POP — TOP Element → Item, TOS = TOS − 1

Stack full

N = 10

TOS = 10

PUSH " OVERFLOW "

Stack empty

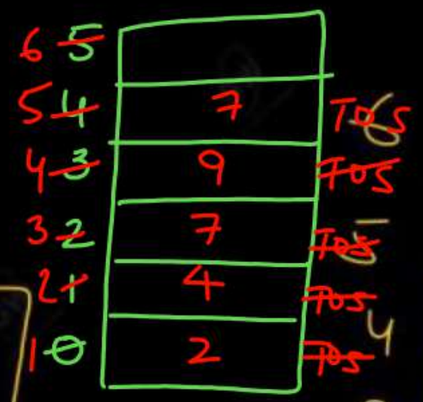
TOP = ≯ 0

POP " UNDERFLOW "

GATEWAY CLASSES

EXAMPLE. 6 :- Suppose STACK is allocated N=6 memory cells and initially STACK is empty, or, in other words, TOP=0. Find the output of the following module:

1. Set P = 2 and Q = 5.
2. Call PUSH(STACK,P)
   Call PUSH(STACK,4)
   Call PUSH(STACK,Q+2)
   Call PUSH(STACK,9)
   Call PUSH(STACK,P+Q).
3. Repeat while TOP=0;
   Call POP(STACK,DATA)
   Write : DATA.
   [End of Loop]
4. Return.



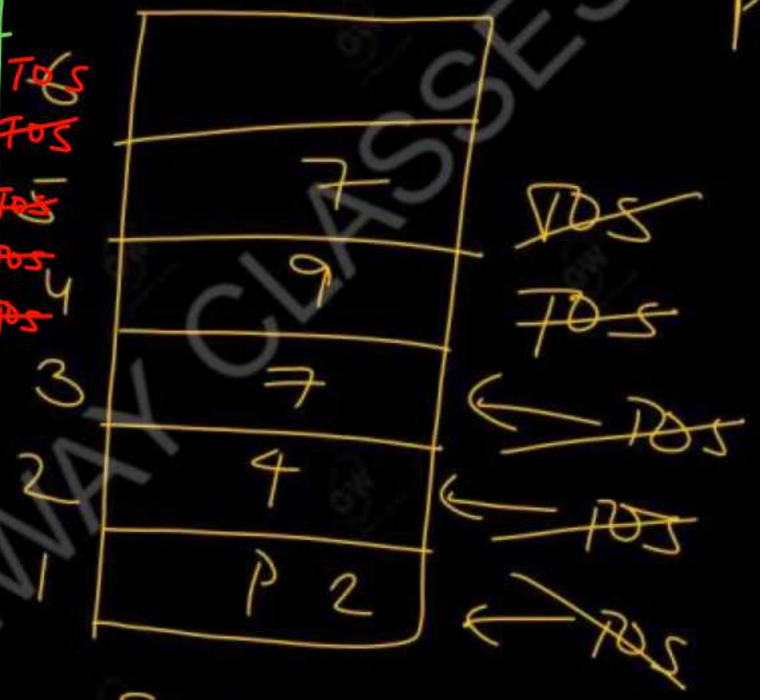
STACK

TOS = 0

① P = 2
② Q = 5

7 9 7 4 2

N = 6



STACK

TOP = 0

P = 2, Q = 5

5+2 = 7

7, 9, 7, 4, 2

Ans

GATEWAY CLASSES

Representation of Stack

A stack can be implemented by 2 ways:-

1. Static implementation of Stack
2. Dynamic implementation of Stack

PUSH / P P

(Linked list)

1. Static implementation of Stack:- Stack can be implemented in memory by using arrays. In this technique of stack implementation, the size of the stack (array) is defined during design time. It is not the efficient way, in terms of memory usage. At run time, there is no way to increase or decrease the size of the array.

2. Dynamic Implementation of stack:- Dynamic implementation of stack is possible by using the concept of linked list data structure. It is a dynamic data structure where the size of data structure can be increased or decreased very easily at run time.

GATEWAY CLASSES

Possible Operations on Stack:

Two important operations which can be performed on Stack data structure are as follows:-

1. PUSH
2. POP

Array

To be inserted

Top of the stack

ALGORITHM:- PUSH (STACK, ITEM, TOS, MAXSIZE) → Array is fixed, maxsize

This algorithm is used to insert an element ITEM to the top of the stack (TOS) of STACK which is of size MAXSIZE.

Step 1: - [Check for the Overflow]

(i) if (TOS == MAXSIZE – 1) F

(ii) {

STACK

(iii) Write " Overflow" or "Insertion is not possible" or "PUSH operation is not possible".

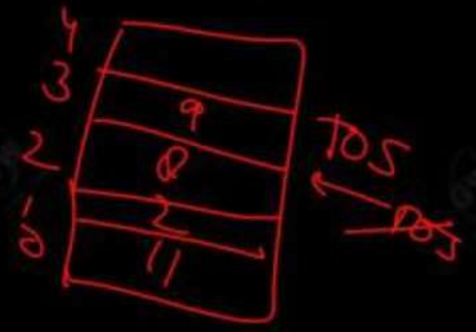
(iv) exit

(v) }

Step 2:- TOS = TOS + 1.

Step 3:- STACK [TOS] = ITEM.

Step 4:- Stop



GATEWAY CLASSES

ALGORITHM:- POP (STACK, ITEM, TOS)

This algorithm is used to delete an element ITEM from the top of the stack (TOS) of STACK.

Step 1: - [Check for Underflow]

(i) if (TOS == - 1)

(ii) {

(iii) Write " Underflow" or "Deletion is not possible" or "POP operation is not possible".

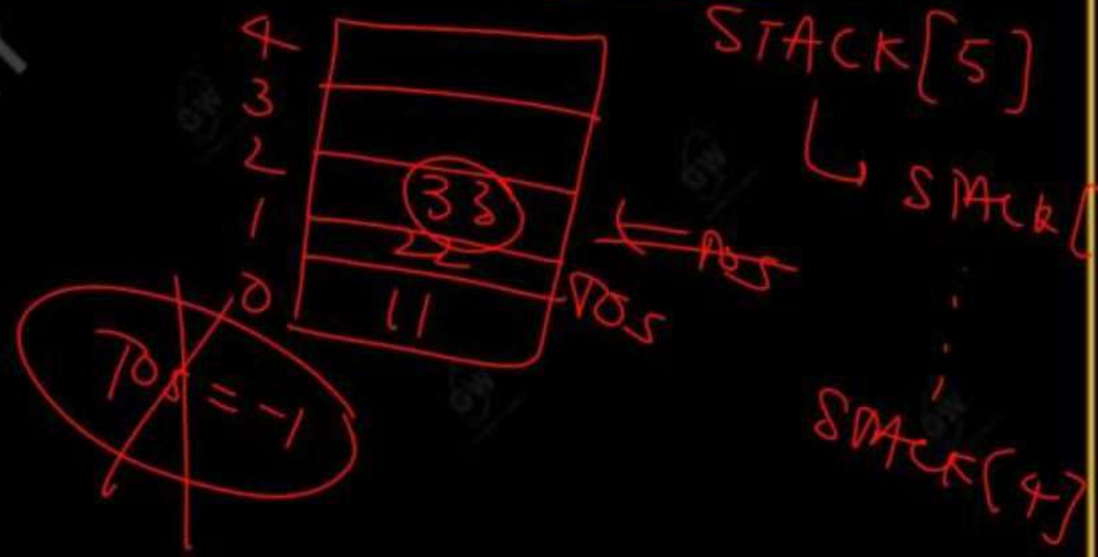
(iv) exit

(v) }

Step 2:- ITEM = STACK [TOS]

Step 3:- TOS = TOS - 1.

Step 4:- Stop



GATEWAY CLASSES

Push and Pop Operations Implementation in C or Array

Implementation of Stack in C

```
#include<stdio.h>
int top=-1;
int stack[20];
void push (int *, int, int);
int pop ( int * );
void display ( int *);
void main()
{
int data, ch,size;
clrscr();
printf("\n enter the size of stack:");
scanf("%d",&size);
if(size>20)
{
printf("\n Invalid size, it can not be grater than
20");
getch();
exit();
}
do
{
printf(" \n\n Available options are as follows:");
printf("\n\n \t Press 1 to Push an element to the
stack");
printf("\n\n \t Press 2 to Pop the element from the
stack");
printf("\n\n \t Press 3 to Quit from the
application");
printf("\n\n enter your choice:");
scanf("%d",&ch);
```

/* array of 20 elements */

FP

GATEWAY CLASSES

```
switch(ch)
{
case 1:
printf("\n\n enter the element to push to
stack:");
scanf("%d",&data);
push( stack, size, data);
printf("\n\n after performing the Push
operation the stack is as follows:");
display(stack);
break;
case 2:
data=pop(stack);
printf("\n\n the popped data is:%d",data);
printf("\n\n after performing Pop operation the
stack is as follows:");
display(stack);
break;
case 3:
exit();
}
}while(ch!=3);
getch()
}
```

17

/* funⁿ call */

GATEWAY CLASSES

```
/* Push Operation */
void push (int *S, int size, int d)
{
if(top== size - 1)
printf("\n Insertion is not possible due to stack full i.e. Stack Overflow");
else
{
top=top+1;
S[top]=d;
}
}
/* Pop Operation */
int pop(int *S)
{
int popped_element;
if(top==-1)
{
printf("\n Deletion is not possible i.e. Stack Underflow");
return 0;
}
else
{
popped_element = S[top];
top--;
}
return popped_element;
}
/* Display Operation */
void display(int s[])
{
int i;
if(top==-1)
printf("\n\n Stack is empty");
else
{
for(i=top;i>=0;i--)
printf("\n %d",s[i]);
}
}
```

GATEWAY CLASSES

Evaluation of Arithmetic Expression

At the time of evaluation of arithmetic expression some precedence rules are to be kept in mind.

(i) Highest Precedence – Exponent Operator ↑ ^ Exp

(ii) Next Highest Precedence – Multiplication and Division Operators * /

(iii) Lowest Precedence – Addition and Subtraction Operators + –

Example.1 :- Evaluate 2 ^ 3 + 4 ^ 2 – 12 / 6. (Ans=22)

2 ↑ 3

2 ^ 3

2 exp 3

2^{2^3}

2^3

Solution:

2 ^ 3 + 4 ^ 2 – 12 / 6

8 + 4 ^ 2 – 12 / 6

8 + 16 – 12 / 6

8 + 16 – 2

24 – 2 = 22

GATEWAY CLASSES

$\underline{2 \wedge 3} + 4 \wedge 2 - 12 / 6$

$\wedge$ Exponent

$8 + \underbrace{4 \wedge 2} - 12 / 6$

$8 + 16 - \underbrace{12 / 6}$

$\underbrace{8 + 16} - 2$

$\underbrace{24 - 2} = 22$ Ans

Example.2 :- Evaluate each of the following parenthesis free arithmetic expressions.

(a) 5 + 3 ^ 2 – 8 / 4 * 3 + 6 (ans = 14)

(b) 6 + 2 ^ 3 + 9 / 3 – 4 * 5 (ans = -3)

Solution -

(a)

5 + 3 ^ 2 - 8 / 4 * 3 + 6

5 + 9 - 8 / 4 * 3 + 6

5 + 9 - 2 * 3 + 6

5 + 9 - 6 + 6

14 - 6 + 6

8 + 6

14

5 + 3 ^ 2 - 8 / 4 * 3 + 6

5 + 9 - 8 / 4 * 3 + 6

5 + 9 - 2 * 3 + 6

5 + 9 - 6 + 6

14 - 6 + 6

8 + 6

= 14

GATEWAY CLASSES

(b) $6 + \underbrace{2 \wedge 3} + 9 / 3 - 4 * 5$

$6 + 8 + \underbrace{9 / 3} - 4 * 5$

$6 + 8 + 3 - \underbrace{4 * 5}$

$\underbrace{6 + 8} + 3 - 20$

$14 + 3 - 20$

$17 - 20$

$= -3$

GATEWAY CLASSES

POLISH AND REVERSE POLISH NOTATIONS

Any expression can be represented in the following 3 notations:-

1. A + B – Infix Notation
2. + A B – Prefix Notation (Polish Notation)
3. A B + – Postfix notation (Reverse Polish Notation)

Notation

A + B = Infix

+ A B = Prefix | POLISH NOTATION

A B + = Postfix (REVERSE POLISH NOTATION)

Example 3:- Convert infix (A+B) * C to prefix.
(ans= *+ABC)

Solution -

(A + B) * C

[+AB] * C

*+ABC Ans

(A + B) * C

+AB * C

*+ABC Ans

GATEWAY CLASSES

Example. 4:- Convert infix A + (B * C) to prefix.

Solution –

A + (B * C)

A + *BC

+ A * BC

Ans

A + (B * C)

A + [*BC]

+ A * BC Ans

Example. 5:- Convert infix (A + B) / (C - D) to prefix.

Solution –

(A+B) / (C-D)

[+AB] / [-CD]

/ +AB - CD Ans

(A+B) / (C-D)

[+AB] / [-CD]

/ +AB - CD Ans

S^2 +AB = X

-CD = Y

Pre x / y

/ x y

/ +AB CD

GATEWAY CLASSES

Example. 6:- Give postfix for A + [(B + C) + (D + E) * F] / G.

(ANS= ABC+DE+F*+G/+)

Sol –

$A + [(B+C) + (D+E) * F] / G$

$A + [BC+ + DE+ * F] / G$

$A + [BC+ + DE+F*] / G$

$A + BC+DE+F*+ / G$

① ② ③

$A + BC+DE+F*+G /$

$ABC+DE+F*+G/+$ Ans

Example.7:- Give postfix for A + B - C.

Sol

$A + B - C$

$AB+ - C$

$-B+C$ – Ans

GATEWAY CLASSES

$$A + [(B + C) + (D + E) * F] / G$$

$$A + [\ [BC+] \ + \ [DE+] \ * F] / G$$

$$A + [\ \underbrace{[BC+]} \ + \ \underbrace{[DE+F*]}\ ] / G$$

$$A + [\ BC+DE+F*+\ ] / G$$

$$A + [\ BC+DE+F*+G\ /\ ]$$

$$A\ BC+DE+F*+G+$$

Ans

Example. 8:- Convert the expression A * B + C / D to postfix form.

Sol

A * B + C / D

[AB*] + C / D

[AB*] + [CD/]

AB*CD/+

A * B + C / D

AB* + C / D

AB* + CD/

AB*CD/+

Ans

Example. 9:- Give postfix for A * B + C.

Sol

A * B + C

AB* + C

x + y

AB*C+ Ans

A * B + C

[AB*] + C

AB*C+

GATEWAY CLASSES

Example. 10:- Give postfix form for A + B / C - D.

Sol

A + B / C — D

A + BC/ — D

ABC/+ — D

ABC/+D-

Ans

A+B/C-D

A+[BC/]-D

[ABC/+]-D

ABC/+D-

Ans

Example. 11:- Give postfix for (A + B) / (C – D).

Sol

(A+B)/(C-D)

[AB+]/[CD-]

AB+CD-/

Ans

(A+B)/(C-D)

[AB+]/[CD-]

AB+CD-/

Ans

GATEWAY CLASSES

Example. 12:- Give postfix form for (A + B) * C / D + E ^ F / G.
(Answer: - A B + C * D / E F ^ G / +)

Sol

$(A+B) * C/D + E \wedge F / G$

$\underbrace{AB+} * C/D + \underbrace{E \wedge F} / G$

$\underbrace{AB+} * \underbrace{C}/D + \underbrace{EF\wedge} / G$

$\underbrace{AB+C*} / \underbrace{D} + \underbrace{EF\wedge} / G$

$\underbrace{AB+C*D/}_{x} + \underbrace{EF\wedge}_{y} / \underbrace{G}_{z}$

$\underbrace{AB+C*D/} + \underbrace{EF\wedge G/}$

$AB+C*D/EF\wedge G/+$ Ans

GATEWAY CLASSES

$(A+B) * C / D + E \wedge F / G$

$[AB+] * C / D + \underline{E \wedge F} / G$

$\underline{[AB+]} * \underline{C} / D + [EF\wedge] / G$

$[AB+C*] / D + [EF\wedge] / G$

$[AB+C*D/] + \underline{[EF\wedge]} / \underline{G}$

$\underline{[AB+C*D/]} + \underline{[EF\wedge G/]}$

$AB+C*D/EF\wedge G/+$ Ans

Evaluation of Postfix Expression

Suppose P is an arithmetic expression written in postfix notation. The following algorithm which uses a STACK to hold operands, evaluates P.

ALGORITHM : This algorithm finds the value of an arithmetic expression E written in postfix notation.

Step 1 : Add a right parenthesis ")" at the end of E

Step 2 : Scan E from left to right and repeat steps 3 and 4 for each element of E until the right parenthesis ")" is encountered.

Step 3 : If an operand is encountered, put it on to the STACK.

Step 4 : If an operator * is encountered, then

(a) Remove the top two elements of STACK, where A is the top element and B is the next to top element.

(b) Perform B*A.

(c) Place the result of (b) back on to the STACK. [End of if structure]

[End of step 2 loop]

Step 5 : Set value equal to the top element on STACK.

Step 6 : Exit.

GATEWAY CLASSES

Example.1:- Consider the following arithmetic expression E written in postfix notation:-

Postfix E: 5 , 2 , 3 , + , * , 12 , 6 , / , -

Evaluate E- (i) Using Tabular Method. (ii) Using individual stacks for each step. (ans=23)
(iii) Convert this postfix expression into its equivalent infix using tabular method.
(Ans=5*(2+3)-12/6)

Solution:-

SYMBOL SCANNED(E)		STACK	
5	5	5	5
2	5, 2	5, 2	5, 2
3	5, 2, 3	5, 2, 3	5, 2, 3
+	5, 5	5, 5	5, (2+3)
*	25	25	5 * (2+3)
12	25, 12	25, 12	5*(2+3), 12
6	25, 12, 6	25, 12, 6	5*(2+3), 12, 6
/	25, 2	25, 2	5*(2+3), (12/6)
-	23	23	5*(2+3)-(12/6)
)			

GATEWAY CLASSES

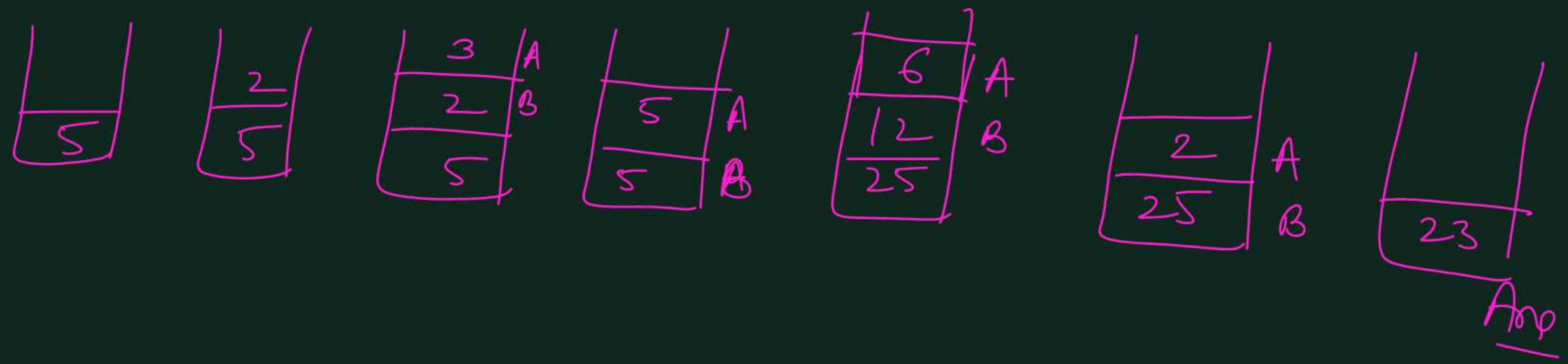
$$5 * (2+3) - (12/6)$$

$$\underbrace{5 * 5} - 2$$

$$25 - 2$$

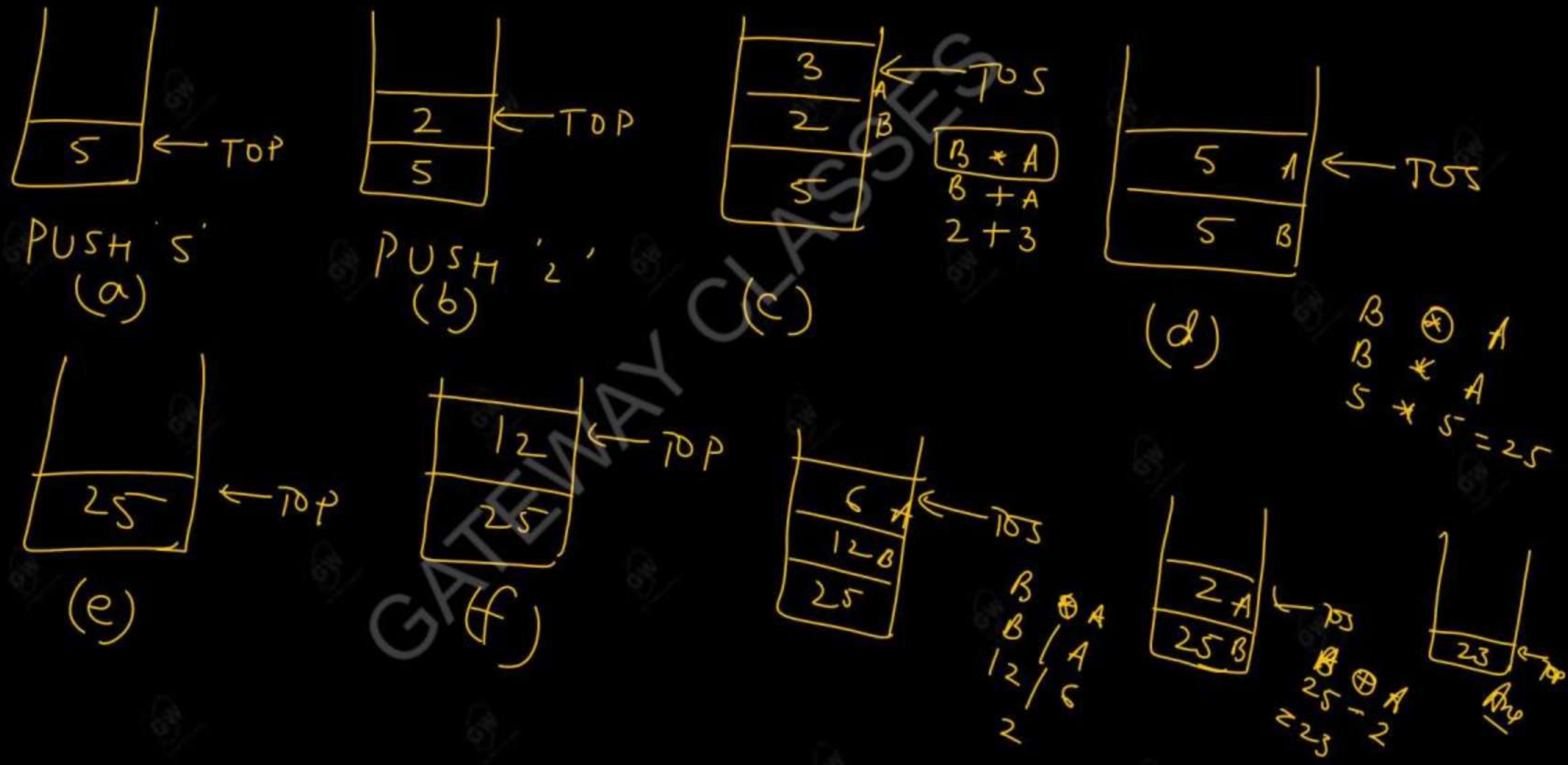
$$23$$

Ans


5
2
5
3
2
5
A
B
5
5
A
B
6
12
25
A
B
2
25
A
B
23
Ans

(II) Through individual stacks for each STACK:

E: 5 , 2 , 3 , + , * , 12 , 6 , / , -



GATEWAY CLASSES

(iii) Conversion of this postfix expression to infix form using tabular method:-

E: 5 , 2 , 3 , + , * , 12 , 6 , / , -

SYMBOL SCANNED (E)	STACK
5	5
2	5, 2
3	5, 2, 3
+	5, (2+3)
*	5 * (2+3)
12	5 * (2+3), 12
6	5 * (2+3), 12, 6
/	5 * (2+3), (12/6)
-	5 * (2+3) - (12/6)
	Ans

GATEWAY CLASSES

$$5 * (2 + 3) - (12 / 6)$$

$$\underbrace{5 * 5}_{} - 2$$

$$25 - 2$$

$$23$$

Ans

GATEWAY CLASSES

Q. $A+(B*C-(D/E\uparrow F)*G)*H$

Infix — find postfix.

Symbol scanned (Q)	stack	Postfix (P)
A	(	A
+	(+	A
(	(+(	A
B	(+((	AB
*	(+((	AB
C	(+((	ABC
−	(+((	ABC*
(	(+((	ABC*
D	(+((	ABC*D
/	(+((	ABC*D
E	(+((	ABC*DE
↑	(+((	ABC*DE
F	(+((	ABC*DEF
)	(+(	ABC*DEF↑/
*	(+(*	ABC*DEF↑/
G	(+(*	ABC*DEF↑/G
)	(+	ABC*DEF↑/G*−
*	(+*	ABC*DEF↑/G*−H
H		ABC*DEF↑/G*−H*+

ANS

Transforming Infix into Postfix Expression

(Reverse Polish Notation)

The following algorithm transform the infix expression Q into its equivalent POSTFIX expression P. This algorithm uses a STACK to temporarily hold operators and left parenthesis. The postfix expressions, P will be constructed from left to right using the operands from Q and the operators which are removed from STACK. We begin by pushing a left parenthesis onto the STACK and adding a right parenthesis at the end of Q. The algorithm is completed when the STACK empty.

Q → P

Infix → Postfix

GATEWAY CLASSES

ALGORITHM : POLISH (Q, P) Suppose Q is an arithmetic expression written in infix notation. This algorithm find the equivalent postfix expression P.

Q infix
P postfix

STEP 1 : PUSH "(" onto STACK, and add ")" to end of Q.

STEP 2 : Scan Q from left to right and repeat steps 3 to 6 for each element of Q until the stack is empty.

STEP 3 : If an operand is encountered, add it to P .

STEP 4 : If a left parenthesis is encountered, push it onto STACK.

STEP 5 : If an operator * is encountered then-

(a) Repeatedly POP from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than *.

(b) Add * to stack. [End of if structure]

STEP 6 : If a right parenthesis is encountered then,

(a) Repeatedly POP from STACK and add to P each operator (on the top of stack) until a left parenthesis is encountered.

(b) Remove the left parenthesis (do not add the left parenthesis to P)

[End of if structure] [End of step 2 loop]

STEP 7 : Exit

GATEWAY CLASSES

Example.2:- Translate, each infix expression into its equivalent postfix expression:

(a) (A – B) * (D / E) (ans= AB-DE/*)

Solution:-

Symbol Scanned	Stack	Postfix(P)
(	(	–
A	(	A
–	(–	A
B	(–	AB
)	(	AB–
*	(*	AB–
(	*(	AB–
D	*(	AB–D
/	*(/	AB–D
E	*(/	AB–DE
)	(*	AB–DE/
)	–	AB–DE/* Ans

GATEWAY CLASSES

Example.4:- Consider the following arithmetic expression P, written in postfix notation:-

P : 12 , 7 , 3 , - , / , 2 , 1 , 5 , + , * , +

(a) Translate P, by tabular method, into its equivalent infix expression.(ans=12/(7-3)+2*(1+5))

(b) Evaluate the infix expression. (ans=15)

Solution: - (a)

P	Equivalent Infix Notation	
12	12	12
7	12, 7	12, 7
3	12, 7, 3 B ⊕ A	12, 7, 3
−	12, 4 (B, A)	12, (7−3)
/	3	12 / (7−3)
2	3, 2	12 / (7−3), 2
1	3, 2, 1	12 / (7−3), 2, 1
5	3, 2, 1, 5	12 / (7−3), 2, 1, 5
+	3, 2, 6	12 / (7−3), 2, (1+5)
*	3, 12	12 / (7−3), 2 * (1+5)
+	15	12 / (7−3) + 2 * (1+5)

GATEWAY CLASSES

$12 / (7-3) + 2 * (1+5)$

$12 / 4 + 2 * 6$

$3 + 2 * 6$

$3 + 12$

15

Ans

GATEWAY CLASSES

Reverse of a String Using Stack Data Structure

```c
#include<stdio.h>
#define MAX 20
int top=-1;
char stack[MAX];
int push(char);
char pop();

void main()
{
char str[20];
int i;
clrscr();
printf("\n enter a string:");
gets(str);
for(i=0;i<strlen(str);i++)
push(str[i]);
for(i=0;i<strlen(str);i++)
str[i]=pop();
printf("\n reversed string is:");
puts(str);
getch();
}

push(char item)
{
if(top==MAX-1)
printf("\n Stack Overflow");
else
{
top++;
stack[top]=item;
}
}

char pop()
{
if(top==-1)
printf("\n stack Underflow");
else
return stack[top--];
}
```

/* stack empty */

RP

"Kamal"

0,1,2,3,4

/* func^n call */

0,1,2,3,4

0	1	2	3	4
k	a	m	a	K

/* */ → multiline

→ single line

str[0] = e

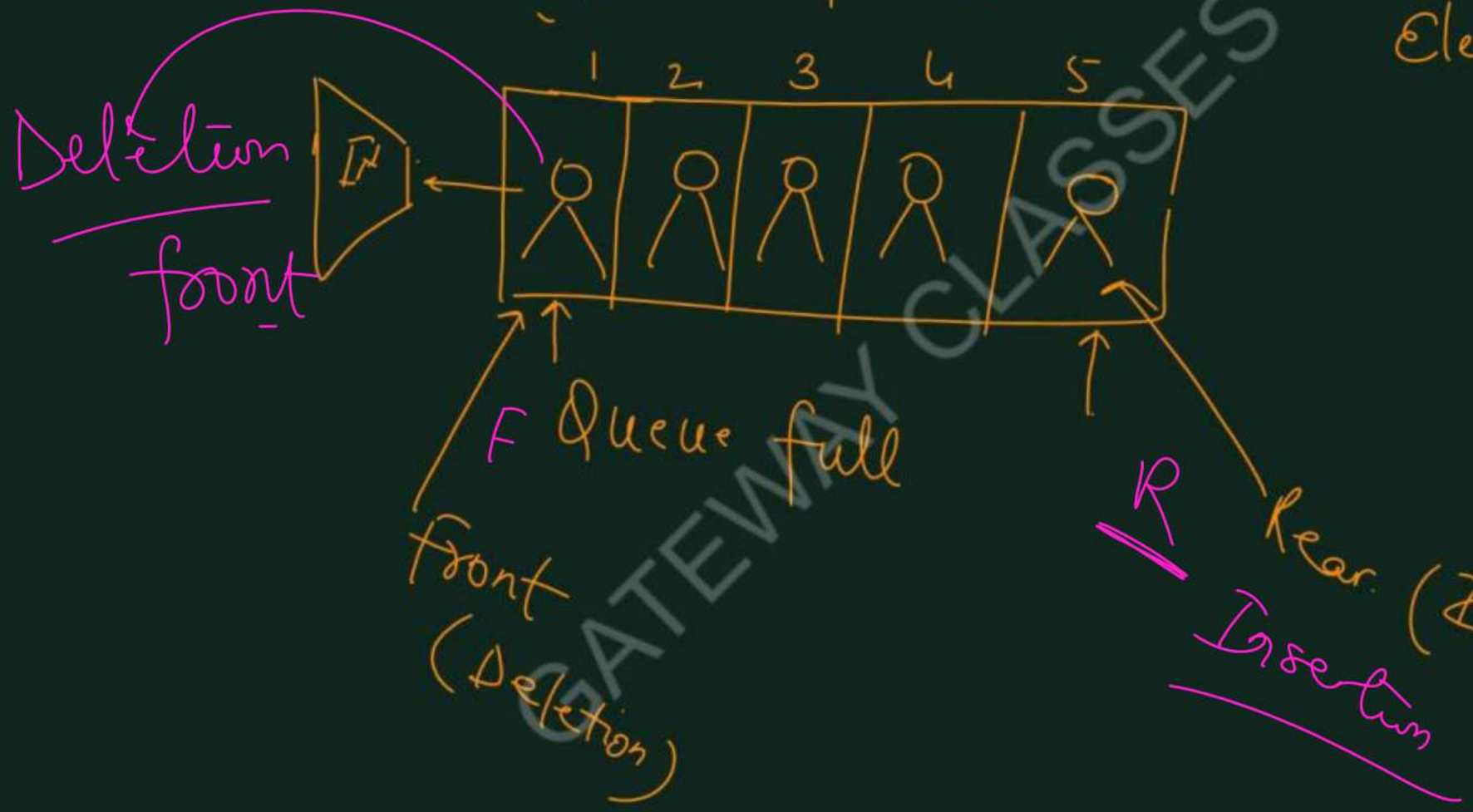
str[1] = a

GATEWAY CLASSES

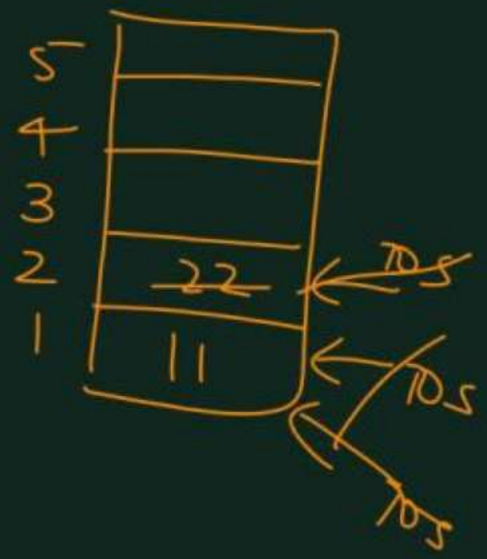
Queue FIFO

FIFO

5, memory cells



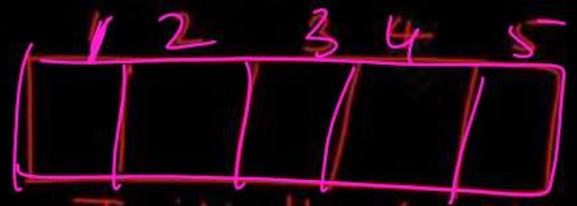
Electricity Bill



GATEWAY CLASSES

Example.1:- Consider the following empty queue, with 5 memory cells as:-

Queue:



Initially empty (front = Rear = 0)

Empty:- (1)
F = R = 0

As the queue is empty Front and Rear end both are at 0.

(i) Insert A -

Q:-

Rear

Insert A

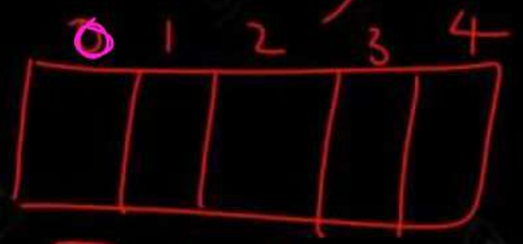
front

Rear = Rear + 1
= 0 + 1
= 1

Queue[Rear] = A

if (front = 0)
set front = 1

Queue



Initially empty

Array
f = R = -1

~~(ii) Insert B -~~

~~(iii) Insert C -~~

if (R == 0)
set R = 1

Q[R] = item

if (f == 0)
set f = 1

Del = f
Ins = R

GATEWAY CLASSES

(II) Insert B

Queue:

1	2	3	4	5
A	B			

FRONT ~~REAR~~ R

Rear = Rear + 1
= 1 + 1 = 2

Queue[Rear] = 'B'

if (front == 0) F

GATEWAY CLASSES

(iii) Insert C

queue:

1	2	3	4	5
A	B	C		
F	~~R~~	R		

$R = R + 1 = 2 + 1$

$= 3$

$Queue[R] = C$

GATEWAY CLASSES

(iv) Insert D –

Queue

1	2	3	4	5
A	B	C	D	

F, ~~R~~, R

$R = R + 1 = 3 + 1 = 4$

Queue[R] = D

(v) Insert E -

Queue

1	2	3	4	5
A	B	C	D	E

maxsize

F, ~~R~~, R

$R = R + 1 = 4 + 1 = 5$

Queue[R] = E

if (R == maxsize - 1)

OVERFLOW

Queue full – if (f == 0) && (R == maxsize - 1)

in 'C' Program

Now, the queue is full because the rear end is equal to the maximum size of the array.

~~(vi) Delete A -~~

1	2	3	4	5
~~A~~	B	C	D	E

~~f~~, R

~~(vii) Delete B -~~

item = Q[f]

item = A

f = f + 1

if ((f == 0) && (R == maxsize - 1))

Queue full

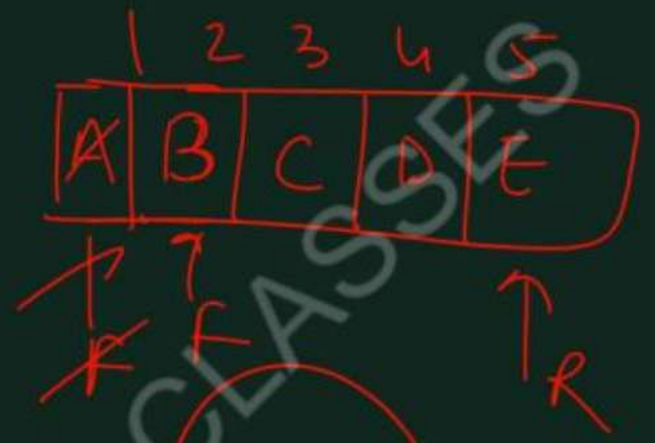
0	1	2	3	4
A	B	C	D	E

f, R

GATEWAY CLASSES

(VI) Delete A

Queue



item = Queue[f]

item = A

f = f + 1

GATEWAY CLASSES

Delete B

	1	2	3	4	5
Queue		~~B~~	C	D	E
		~~F~~	F		R

item = Queue[f]

item = B

f = f + 1

{ int a = 2;

b = a;

GATEWAYCLASSES

Algorithms - Queue Insertion and Deletion Operations

Algorithm : Queue Insertion or Enqueue

This algorithm is used to insert an element ITEM to the Rear end of the Queue which is of n elements (maximum size of Queue).

Step 1:- [Check for the Overflow]

If (Rear == n) then

Print " Overflow" or "Insertion operation is not possible as Queue is full".

[End of if structure]

if (Rear == maxsize) if (R == maxsize - 1)

if (Rear == maxsize-1)

OVERFLOW

UNDERFLOW

Step 2:- Set: Rear = Rear + 1

Step 3:- Set: Queue [Rear] = ITEM.

Step 4:- if (Front == 0) then

Set : Front = 1. {if the first ITEM is inserted to the Queue}

[End of if structure]

Step 5:- Exit

GATEWAY CLASSES

Algorithm : Queue Deletion or Dequeue

This algorithm is used to delete an element ITEM from the Front end of the Queue which is of n elements (i.e. maximum size of Queue).

Step 1:- [Check for Underflow]

If (Front == 0) then

Print " Underflow" or "Deletion operation is not possible as queue is empty".

[End of if structure]

Step 2:- ITEM = Queue [Front]

Step 3:- if (Front == Rear) then

{

Set: Front = 0.

Set: Rear = 0.

}

else

{

Set : Front = Front + 1.

}

Step 5:- Exit

if (f == -1)

Queue

item = Queue[f]

item = 33

GATEWAY CLASSES

Implementation of Insertion & Deletion Operations of Queue Data Structure in C

```
/*Insertion and Deletion operations to the Ordinary Queue*/
#include<stdio.h>
void queue_insertion(int);
void queue_deletion();
void traverse();
int queue[20];
int front=0,rear=0,item;
int size;
void main()
{
int ch;
clrscr();
printf("\n Enter the size of the queue:");
scanf("%d",&size);
if(size>20)
{
printf("\n Invalid size of queue, it should be less than 20
because the size of the array is 20");
getch();
exit();
}
do
{
printf("\n Your choices are as follows:");
printf("\n\n Press 1 for Insertion of an element to the Queue");
printf("\n\n Press 2 for Deletion of an element from the
Queue");
printf("\n\n Press 3 for Exit from the Program:");
printf("\n\n Enter your choice:");
scanf("%d",&ch);
```

-1 -1

Queue[0]
↓
Queue[19]
f = R = -1

5

1 2 3

GATEWAY CLASSES

Implementation of Insertion & Deletion Operations of Queue Data Structure in C

```
switch(ch)
{
case 1:
queue_insertion(size);
printf("\n After insertion the queue as follows");
traverse();
break;
case 2:
queue_deletion();
printf("\n After deletion the queue is as follows:");
traverse();
break;
case 3:
exit();
}
printf("\n Do you want to continue, press 1 for
continue and 0 for end:");
scanf("%d",&ch);
}while(ch!=0);
getch();
}
```

GATEWAY CLASSES

Implementation of Insertion & Deletion Operations of Queue Data Structure in C

```
void queue_insertion(int s)
{
printf("\n Enter the item to be inserted to the queue:");
scanf("%d",&item);
if(rear<s)
{
rear++;
queue[rear]=item;
if(front==0)
front=1;
}
else
printf("\n Insertion is not possible:");
}

void queue_deletion()
{
if(front==0)
{
printf("\n Underflow");
return;
}
else
{
item=queue[front];
printf("\n Deleted item is =%d",item);
}
if(front==rear)
{
front=0;
rear=0;
}
else
front=front+1;
}

void traverse()
{
int i;
if(front==0)
return;
for(i=front;i<=rear;i++)
printf("\n %d",queue[i]);
}
```

3 5

11, 22, 33

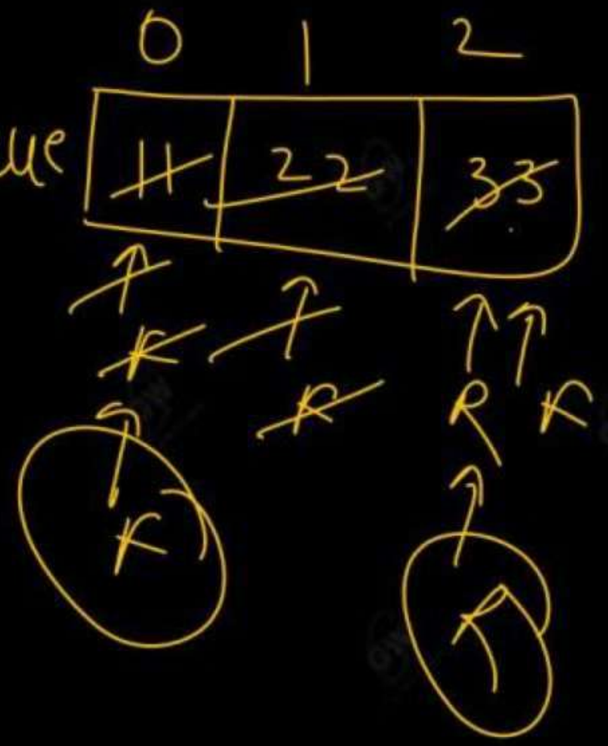
1<3 T

2<3 T

2<2 F

S-1

OVERFLOW



GATEWAY CLASSES

Limitations of Simple or Ordinary Queue

There are some limitations associated with simple or ordinary queue.

Consider the following initially empty queue of 6 memory cells:-

0	1	2	3	4	5
11	22	33	44	55	66

Queue:

full if F = 0 && R = maxsize-1

We analyze the problem associated with simple queue with a series of insertion and deletion operations performed on the simple queue.

For example, now insert 11, 22, 33, 44, 55, 66 in this queue one by one-

~~Del~~ Insert

GATEWAY CLASSES

item = Q[front]

f = f + 1

0	1	2	3	4
~~11~~	~~12~~	13	14	15

Queue Empty

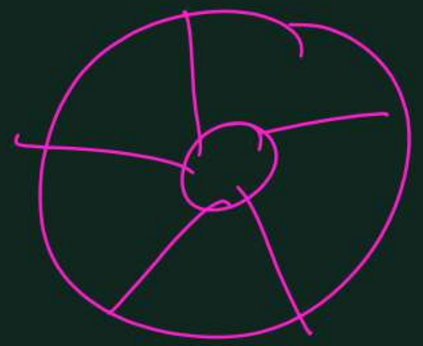
f = R = -1

Overflow

limitation

if (f == -1)

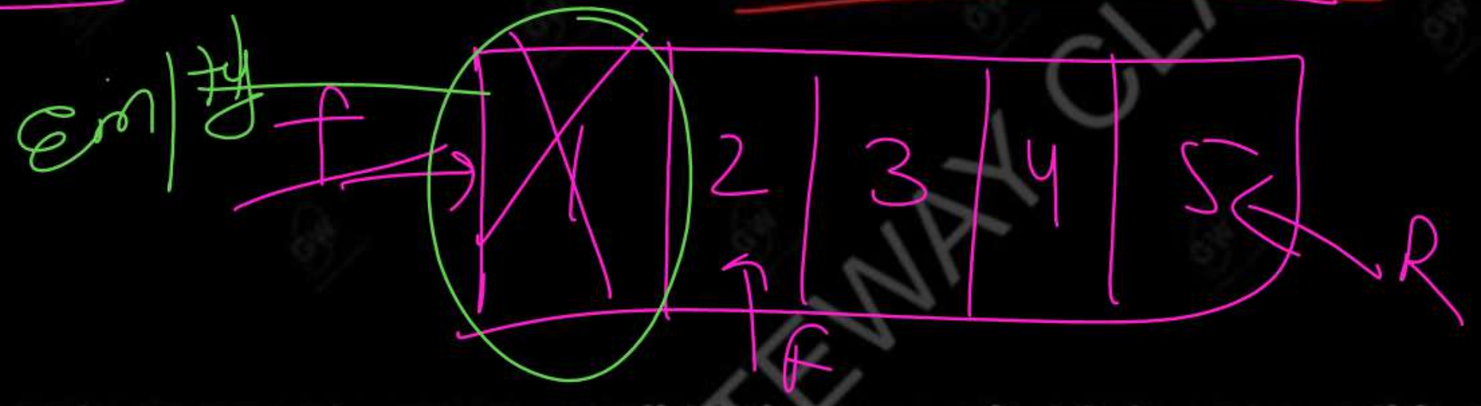
Underflow



Limitations of Ordinary Queue

We know that, in ordinary queue the elements are deleted from the Front end and inserted to the Rear end of the queue.

Suppose one element is deleted from the Front end of the queue when the queue is full as mentioned in the previous example. The deletion operation can be performed like this-



Now, there is one empty memory cell in the queue but it is not possible to insert new element to the queue because it is pre-fixed that insertion of element in the simple queue is possible only from the Rear end of the queue and in this example when the queue is full the Rear is on the last position of the queue that is Rear is equal to the maximum size of the queue because the queue is full.

GATEWAY CLASSES

Limitations of Ordinary Queue

So, space is available but insertion of new element in simple queue is not possible because of its limitation.

If we will try to insert an element in this queue, it will give message " Overflow" which is not correct.

This limitation can be removed by using the next version of queue data structure which is known as Circular Queue.

Circular Queue

GATEWAY CLASSES

Limitations of Ordinary Queue

Consider the following empty queue of 6 memory cells:-

0	1	2	3	4	5

(i) 1 2 3 4 5 6

Now insert 11, 22, 33, 44, 55, 66 in this queue-

maxsize = 6

Front = (Deletion) = −1
Rear = (Insertion) = −1
} NULL

0	1	2	3	4	5 (maxsize−1)
11	22	33	44	55	66

Front

0	1	2	3	4	5
	22	33	44	55	66

F R

Rear maxsize−1

GATEWAY CLASSES

0	1	2	3	4	5
		33	44	55	66
	~~F~~	T			R

item = Q[F]

item = 22

f = f + 1

= 1 + 1 = 2

GATEWAY CLASSES

Limitations of Ordinary Queue

- In ordinary queue the elements are deleted from the front end and inserted to the Rear end of the queue.
- Suppose one element is deleted from the Front end of the queue when the queue is full as mentioned above.
- Now, there is one empty memory cell but it is not possible to insert new element to the queue because it is fixed that insertion of element is possible only from the Rear end of the queue and in this example when the queue is full the Rear is on the last position of the queue that is Rear is equal to the maximum size of the queue because the queue is full.

  maxsize-1
- So, space is available but insertion of new element is not possible because of its limitation. This limitation can be overcome by using the next version of queue which is known as Circular Queue.

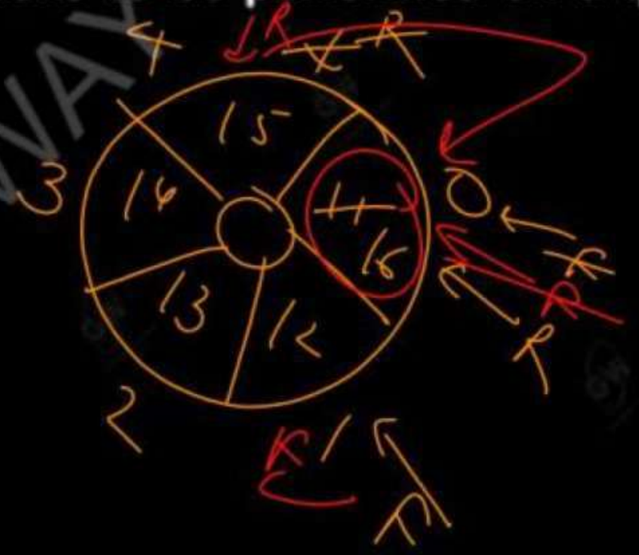
GATEWAY CLASSES

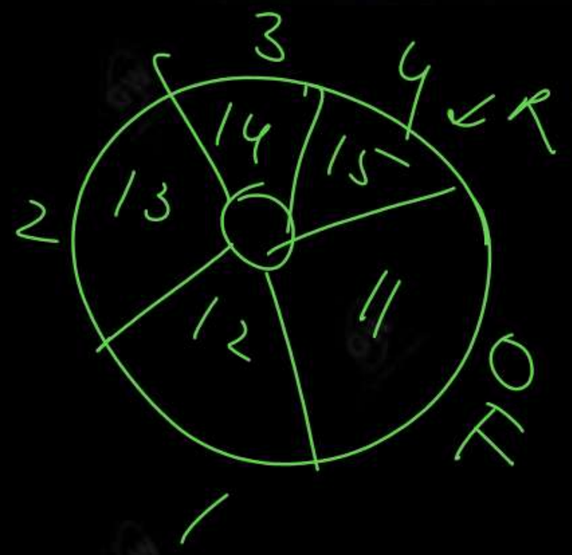
Circular Queue

- Circular queue is form of a queue where there is no end of queue.
- Suppose we have one queue of 5 elements, it means front represent the first element of the queue and rear represent the last element of the queue. In circular queue, rear can move to the first location according the requirement.
- This concept of circular queue can be understood by the following example:-

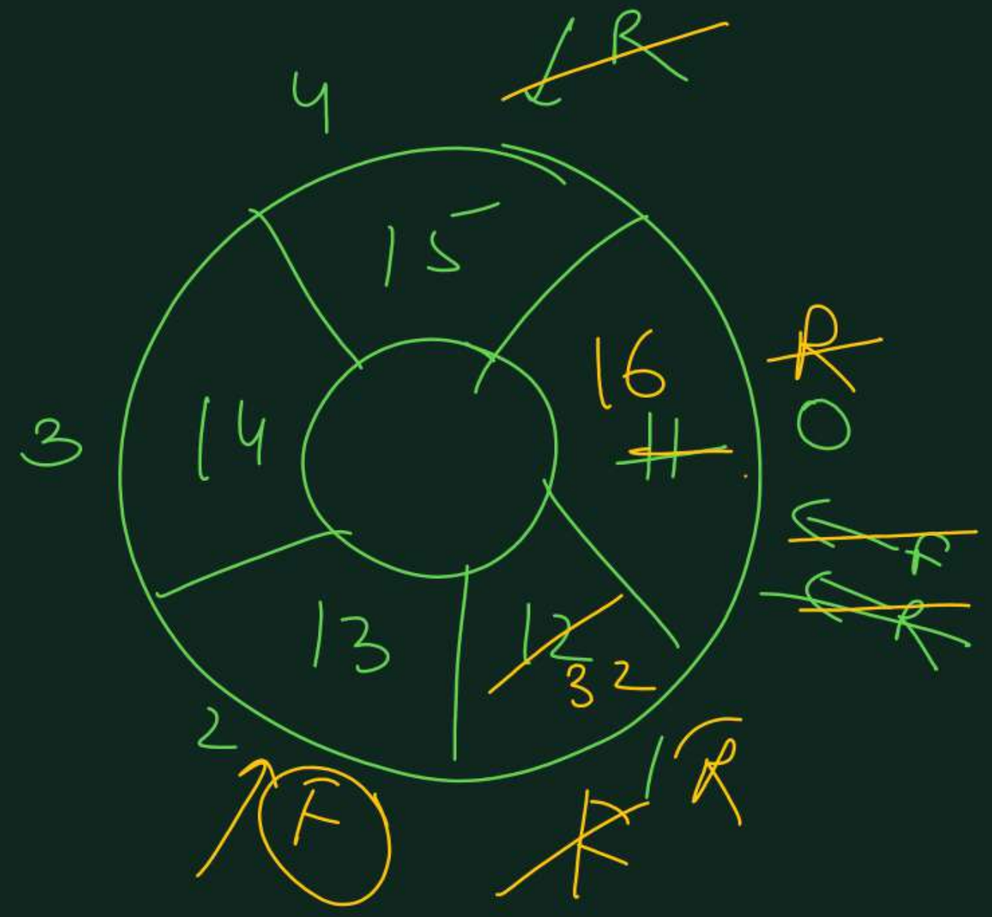
Example 1:- Consider the following operations to be performed on a queue when it is circular with 5 memory cells:-

~~(a) Queue is initially empty-~~





GATEWAY CLASSES



if (f == 0 && r == maxsize-1)

||

(f == R+1)

1 == 0+1

1 == 1

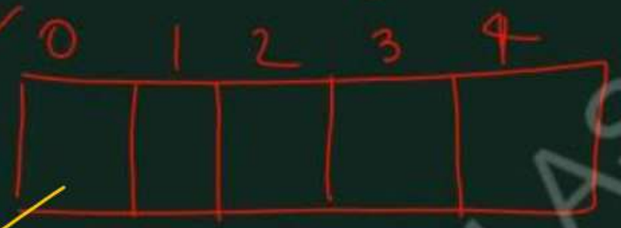
T

2 == 1+1

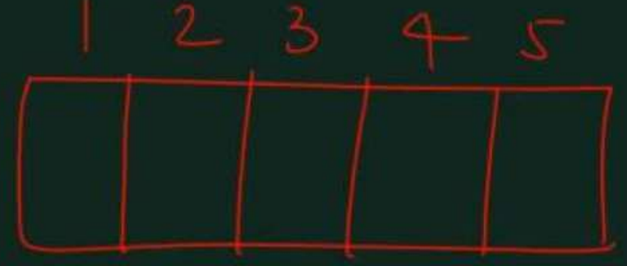
2 == 2 Full

(a) Queue is initially empty :-

size = 5



Front = -1
Rear = -1



front = 0
Rear = 0

GATEWAY CLASSES

(a)A is inserted to the queue-

Circular Queue :

0	1	2	3	4
A				

front, Rear

(2) Queue[Rear] = A,

(1) Rear = Rear+1
= -1+1
= 0

if (front == -1) then
set front = 0

(c) B and C are inserted to the queue-

(i) B

0	1	2	3	4
A	B			

f R R

R = R+1
= 0+1 = 1
Q[R] = B

GATEWAY CLASSES

(ii) C

0	1	2	3	4
A	B	C		
F	~~R~~	R		

$R = R + 1$

$= 1 + 1 = 2$

$Q[R] = C$

GATEWAY CLASSES

(d) D is inserted to the queue-

0	1	2	3	4
A	B	C	D	
F		R	R	

$R = R + 1$
$= 2 + 1 = 3$
$Q[R] = D$

0	1	2	3	4
~~F~~	~~G~~	C	D	E
~~R~~	~~R~~	~~F~~	~~R~~	~~R~~
	R	F		

(e) Two items are deleted from the queue-

0	1	2	3	4
~~A~~	~~B~~	C	D	
~~R~~	~~R~~	F	R	

$F = 0$ && R=maxsize-1
$\parallel$
$f == R+1$
$2 == 1+1$
$2 == 2$ T

GATEWAY CLASSES

(f) E , F and G are inserted to the queue-

0	1	2	3	4
F	G	C	D	E
~~F~~	~~R~~	F	~~R~~	~~R~~

R F

(i) E

R = R+1
= 3+1
= 4

Q[R] = E

(ii) F

R = R+1
= 4+1 = 5 X

if (R == maxsize-1)
4 == 5-1
4 == 4 T

&& f != 0
set R = 0
Q[R] = F

(iii) G

R = R+1
Q[R] = G

Simple

(it should be noted here if it is an ordinary queue then insertion of F and G is not possible which is the limitation of ordinary queue

GATEWAY CLASSES

X is inserted to the Queue : OVERFLOW
{Because f is equal to the R+1}

(g) Two items are deleted from the queue-

0	1	2		4
F	G	~~C~~	~~D~~	E
R	~~F~~	~~F~~	~~F~~	F

if(F == R+1)

2 == 1+1

2 == 2 True

(f) Two items H and I are inserted-

0	1	2	3	4
~~F~~	G	H	I	~~E~~
~~F~~	~~F~~ ~~R~~	~~F~~ ~~R~~	~~F~~ ~~R~~	~~F~~ ~~R~~

if F == R+1

F == 3+1

F == 4 True

GATEWAY CLASSES

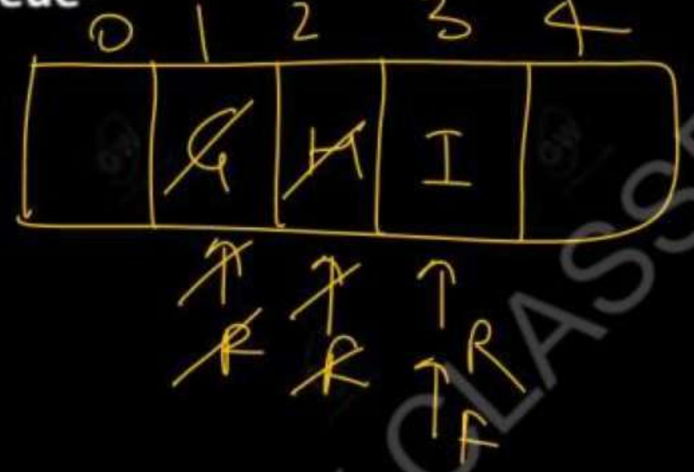
(i) I is inserted to the queue-

Now insertion is not possible because queue is full.

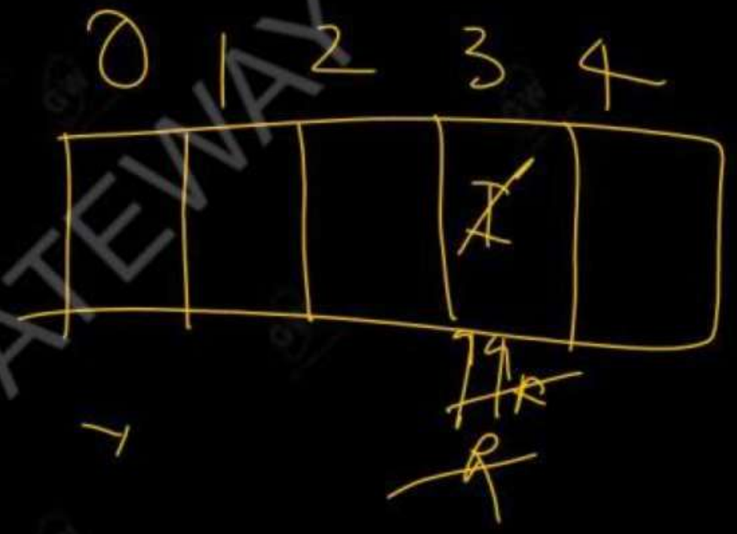
(j) Two items are deleted from the queue-

GATEWAY CLASSES

(k) Two items are deleted from the queue-



(l) One item is deleted from the queue-



item = Q[f]
item = I
if (f == R)
set f = R = -1

GATEWAY CLASSES

(m) One item is deleted from the queue-

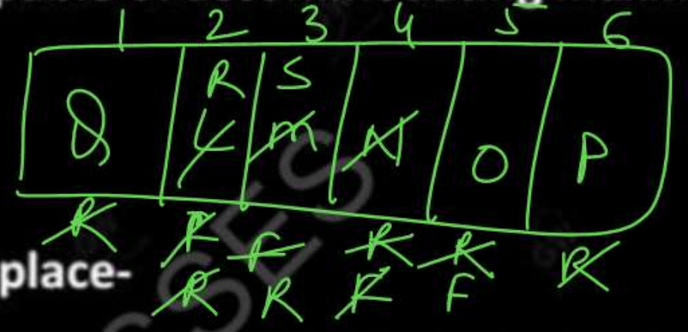
Now deletion is not possible because underflow.

GATEWAY CLASSES

Example 2:- Consider the following circular queue capable of accommodating maximum 6 elements:

Rear = 4, Front = 2

Queue :- _ , L , M , N , _ , _ .

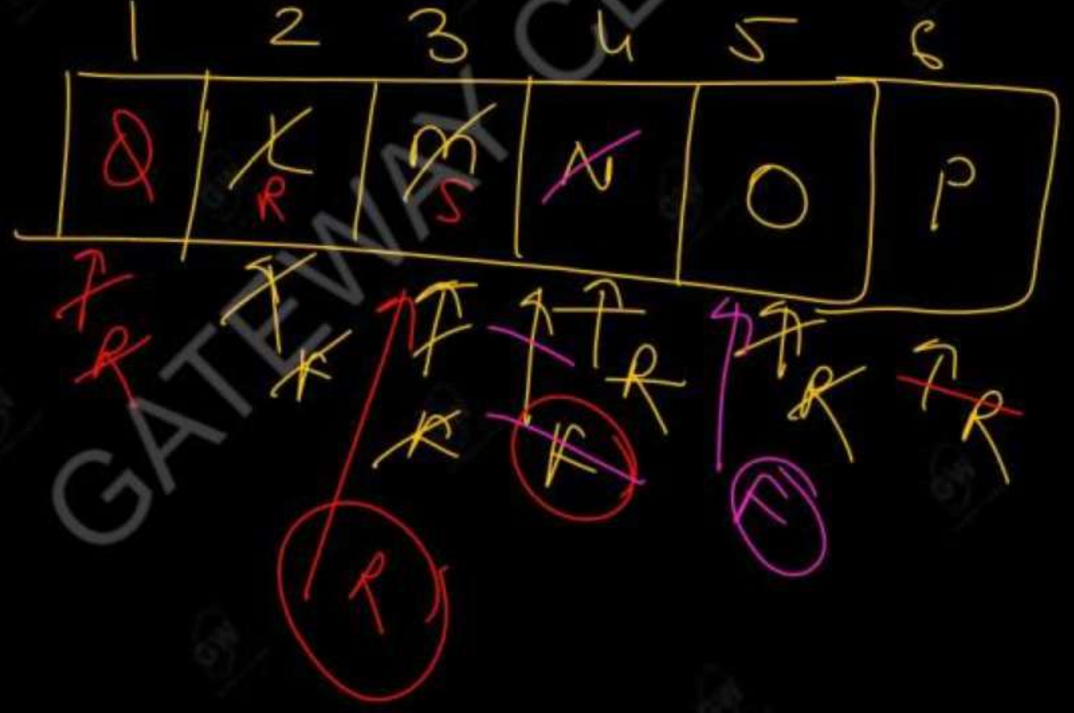


Describe the queue as the following operations take place-

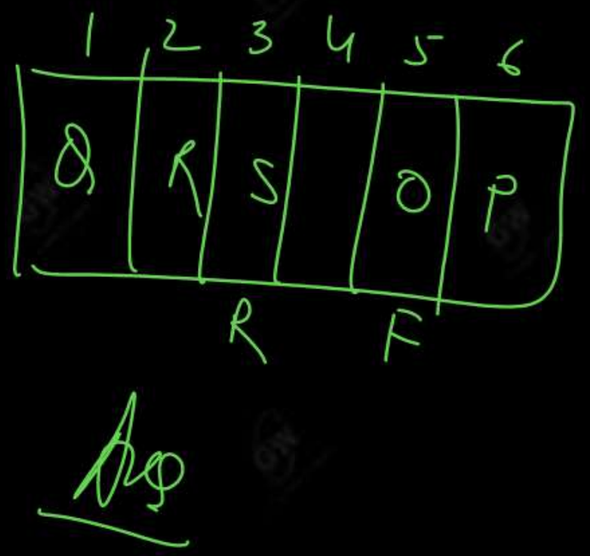
(a) Add O (b) Add P (c) Delete 2 letters (d) Add Q, R, S (e) Delete one letter

Solution:-

Initial Queue:-



(a) Add O



Ans

GATEWAY CLASSES

(b) Add P

(c) Delete 2 letters

GATEWAY CLASSES

(d) Add Q, R, S

(e) Delete one letter

GATEWAY CLASSES

Algorithms Circular Queue Insertion and Deletion

Algorithm : Circular Queue Insertion

This algorithm is used to insert an element ITEM to the circular queue.

Step 1:- if (Front == 0 && Rear == MaxSize-1) || (Front == Rear + 1) then

Write "Overflow" or "Insertion of element is not possible as queue is full"

[end of if structure]

Step 2:- if Front == -1 then

Set Front = 0 and Rear = 0

Queue[Rear] = ITEM.

Step 3:- else if Rear == Maxsize-1 then

Set Rear = 0

Queue [Rear] = ITEM.

Step 4:- else

Rear = Rear + 1

Queue [Rear] = ITEM.

Step 5:- Exit

GATEWAY CLASSES

```
R = R + 1
Q[R] = item
if (f == -1)
    set f = f + 1
```

GATEWAY CLASSES

Algorithm : Circular Queue Deletion

This algorithm is used to delete an element ITEM from the circular queue.

Step 1:- if Front == - 1 then

Write "Underflow" or "Deletion operation is not possible as queue is empty".

[end of if structure]

Step 2:- [Read ITEM from the Front of the Queue]

ITEM = Queue [Front].

Step 3:- if Front == Rear then

Set Front = -1 and

Set Rear = -1.

Step 4:- else if Front = Maxsize then

Set: Front = 0.

Step 5:- else

Set Front = Front + 1.

Step 6:- Exit

GATEWAY CLASSES

Insertion and Deletion of an element in Circular Queue

```
/*Insertion and Deletion operations to the Circular Queue*/
#include<stdio.h>
#define MAX 5
void insertion();
void deletion();
void display();
int cqueue[MAX];
int front=-1,rear=-1;
void main()
{
int choice;
clrscr();
while(1)
{
printf("\n Your choices are as follows:");
printf("\n\n 1:Insertion");
printf("\n\n 2:Deletion");
printf("\n\n 3:Display");
printf("\n\n 4:Quit");
printf("\n\n Enter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
insertion();
break;
case 2:
deletion();
break;
case 3:
display();
break;
case 4:
exit(1);
default:
printf("\n This choice is not available:");
}
}
getch();
}
```

GATEWAY CLASSES

```
void insertion()
{
int item;
if((front==0 && rear==MAX-1) || (front==rear+1))
{
printf("\n Queue is Overflow");
return;
}
if(front==-1)    /*if queue is empty*/
{
front=0;
rear=0;
}
else if(rear==MAX-1) /*rear is at the last position*/
rear=0;
else
rear=rear+1;
printf("\n Enter the item to be inserted to the queue:");
scanf("%d",&item);
cqueue[rear]=item;
}
```

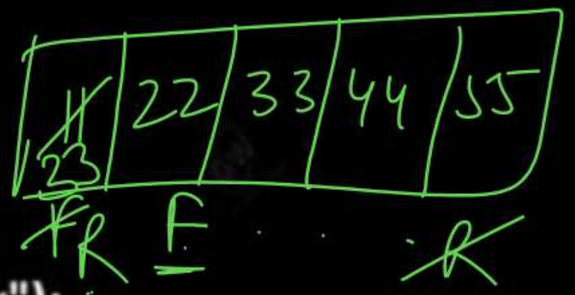
so Queue is full.

```
void deletion()
{
int item;
if(front==-1)
{
printf("\n Queue Underflow");
return;
}
item=cqueue[R];
printf("\n Deleted item from queue is =%d",cqueue[front]);
if(front==rear)
/* Queue has only one element*/
{
front=-1;
rear=-1;
}
else
if(front==MAX-1)
front=0;
else
front=front+1;
}
```

GATEWAY CLASSES

```
void display()
{
int front_pos=front;
int rear_pos=rear;
if(front==-1)
{
printf("\n queue is empty");
return;
}
printf("\n queue elements are as follows:");

if(front_pos<=rear_pos)
while(front_pos<=rear_pos)
{
printf("\n%d",cqueue[front_pos]);
front_pos++;
}
else
{
while(front_pos<=MAX-1)
{
printf("\n%d",cqueue[front_pos]);
front_pos++;
}
front_pos=0;
while(front_pos<=rear_pos)
{
printf("\n%d",cqueue[front_pos]);
front_pos++;
}
}
printf("\n");
}
```



GATEWAY CLASSES

Double Ended Queue

- In a DEQUE, both insertion and deletion operations can be performed at either end of the DEQUE i.e. any item can be inserted from both front or rear ends and in the same manner the operation of deletion of an element can also be performed from both front and rear ends of the DEQUE.
- Consider the following figure to understand the concept:-

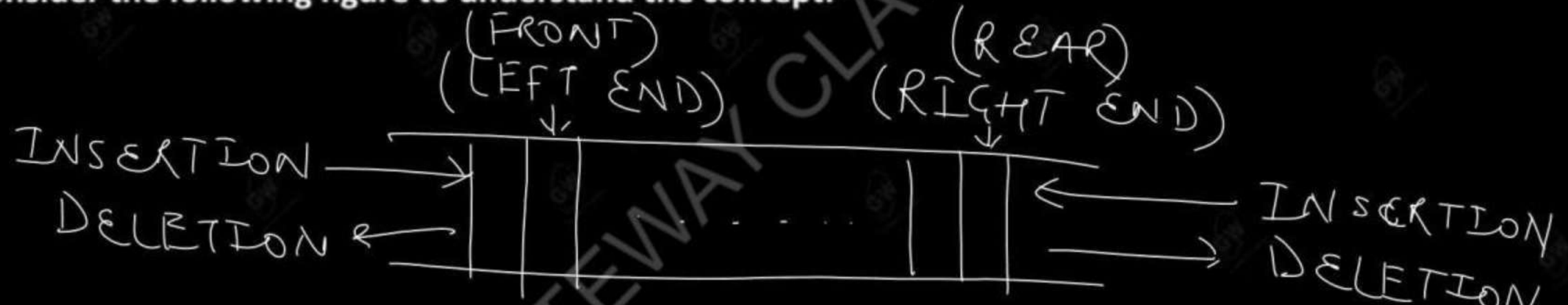


FIG.-A VIEW OF DEQUE

- DEQUE can be represented by using doubly linked list or a circular linked list.

GATEWAY CLASSES

Types of DEQUE

DEQUE can be of 2 types-

1. Input Restricted DEQUE (INPUT / INSERTION = Only one end, Del – Both ends)
2. Output Restricted DEQUE (Output / Deletion = Only one end, Insertion = Both ends)

1. **Input Restricted DEQUE –** In Input – restricted DEQUE, the item can be deleted from both the ends of the DEQUE but insertion can be performed only at one end of the DEQUE.

Consider the following figure to understand the concept-

Fig:- Input Restricted DEQUE

GATEWAY CLASSES

2. Output - Restricted DEQUE – In Output – restricted DEQUE, the item can be inserted from both the ends of the DEQUE but operation of deletion can be performed only at one end of the DEQUE.

Consider the following figure to understand the concept-

Fig:- A view of output Restricted Deque.

GATEWAY CLASSES

Possible Operations on Deque

The possible operations which can be performed on DEQUE are as follow:-

1. Insertion of an element at the rear (right) end of the DEQUE
2. Insertion of element at the front (left) end of the DEQUE
3. Deletion of an element from the rear (right) end of the DEQUE
4. Deletion of an element from the front (left) end of the DEQUE

GATEWAY CLASSES

Example 1:- Consider the following queue (deque) of characters where DEQUE is a circular array which is allocated 6 memory cells:

F R

LEFT = 2 RIGHT = 4 DEQUE : _ , A , C , D , _ , _

Describe the DEQUE while the following operations take place-

(a) F is added to the right of the DEQUE

(b) Two letters on the right are deleted

(c) K, L and M are added to the left of the DEQUE

(d) One letter on the left is deleted

(e) R is added to the right of the DEQUE

(f) S is added to the left of the DEQUE

FRONT LEFT

REAR RIGHT

1	2	3	4	5	6
K	A	C	~~D~~	~~F~~ M	L

left ~~left~~ left ~~right~~ right ~~right~~ right ~~left~~ left ~~left~~ left

Note ① Insert (Right) Right ++

② Deletion (Right) Right --

③ Insertion (left) left --

④ Delete (left) left ++

GATEWAY CLASSES

Solution:-

(a) F is added to the right of the DEQUE-

Initial Deque:-

1	2	3	4	5	6
	A	C	D		
	↑ left		↑ Right		

1	2	3	4	5	6
	A	C	D	F	
				↑ Right	

if (left == 0 && right == Max size) || F = R+1 L = Right+1

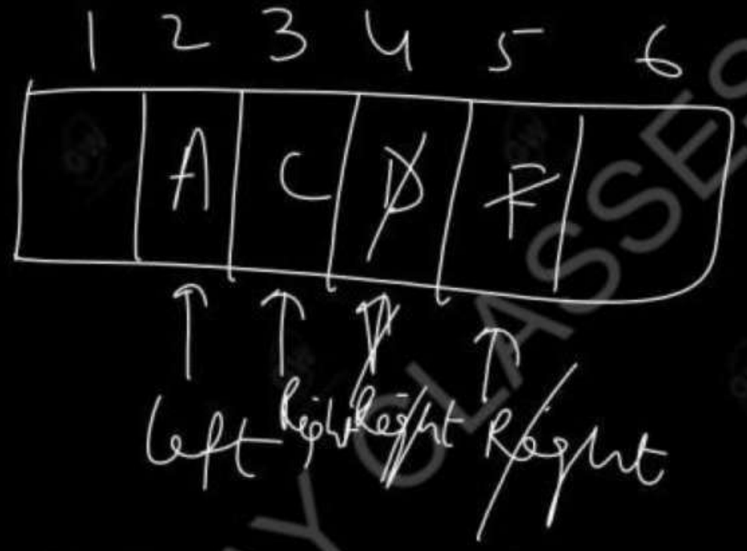
Right = Right + 1
= 4+1 = 5

Deque[R] = item

Deque[5] = 'F'

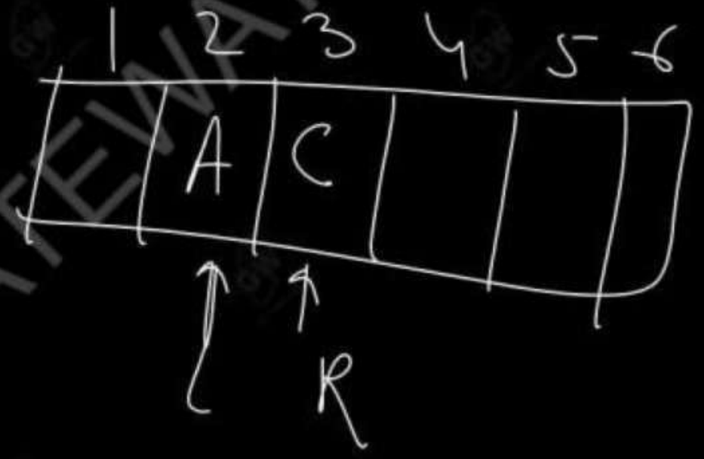
GATEWAY CLASSES

(b) Two letters on the right are deleted



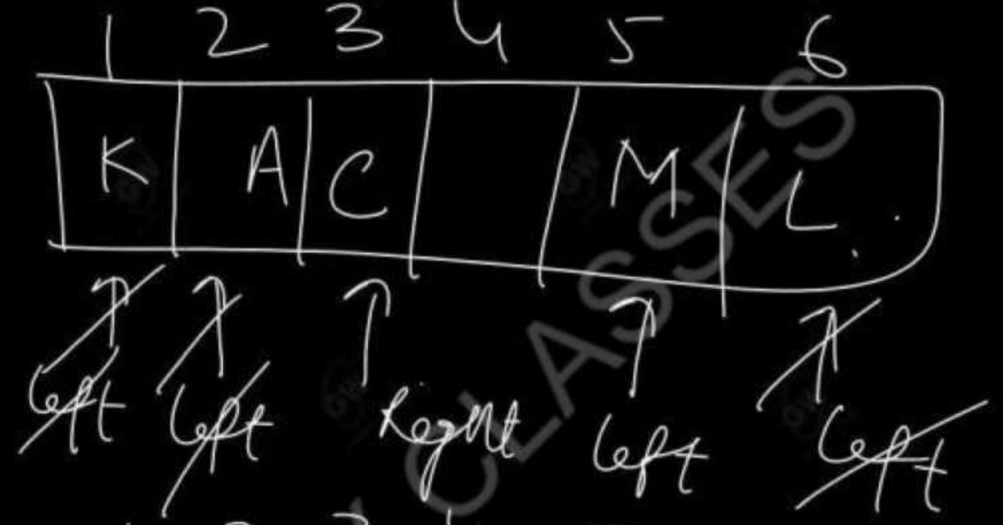
item = Deque[Right]

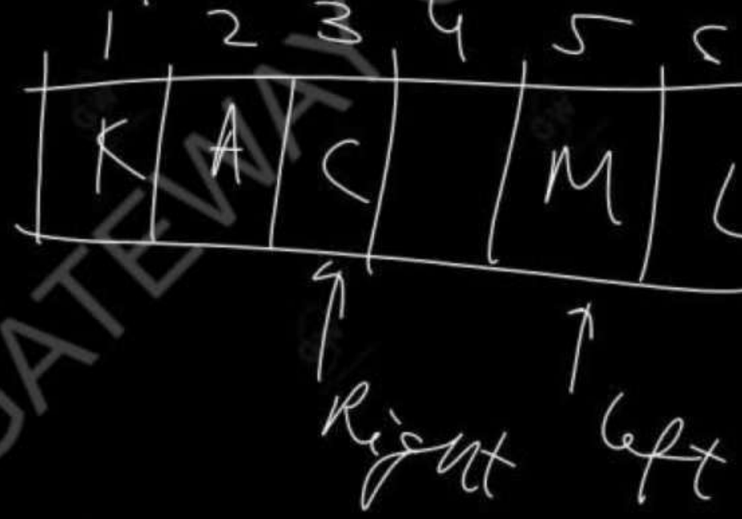
Right = Right - 1



GATEWAY CLASSES

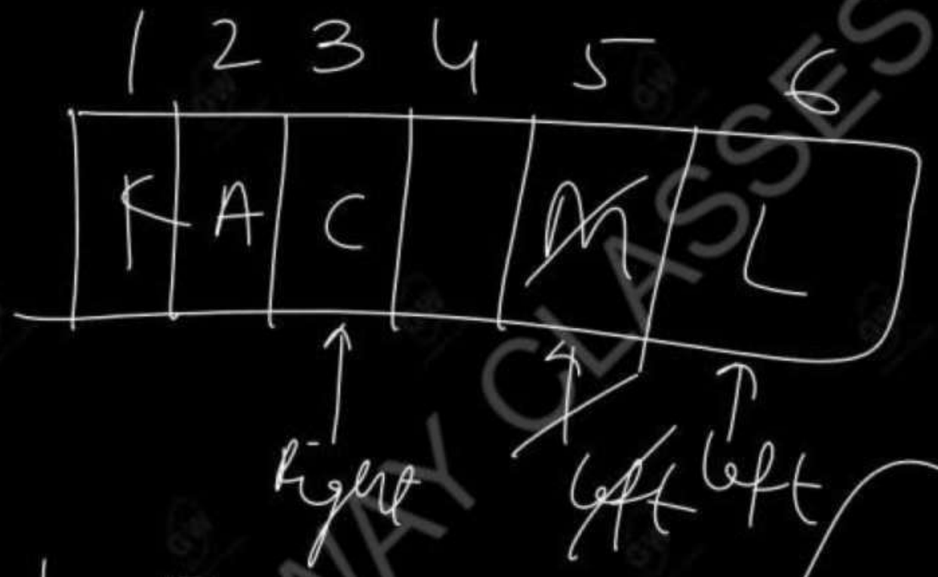
(c) K, L and M are added to the left of the DEQUE

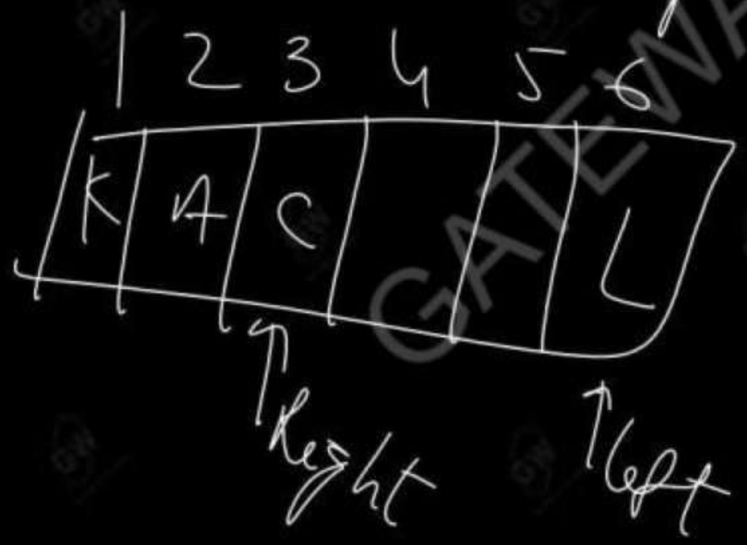




GATEWAY CLASSES

(d) One letter on the left is deleted





data = deque[left]
data = M
left = left + 1

GATEWAY CLASSES

(e) R is added to the right of the DEQUE

1	2	3	4	5	6
K	A	C	R		L

Right

~~Right~~ Left

1	2	3	4	5	6
K	A	C	R		L

Right Left

Right = Right + 1
= 3 + 1 = 4

Deque[Right] = item

Deque[4] = R

GATEWAY CLASSES

(f) S is added to the left of the DEQUE

1	2	3	4	5	6
K	A	C	R	S	L

Right left ~~left~~

left = Right + 1
5 = 4 + 1
5 = 5 ✓ T

1 | 6

1	2	3	4	5	6
K	A	C	R	S	L

Right left

left = left − 1
= 6 − 1 = 5

Deque[left] = S
Deque[5] = S

GATEWAY CLASSES

Algorithm: Insertion DEQUE Right (rear) Side

This algorithm is used to insert an element at the right side of the DEQUE.

Step 1:- Read "Item" to be inserted to the DEQUE.

Step 2:- [Check for overflow]

If (left==0 && right == maxsize – 1) || (left == right + 1) then

Write " Overflow" or "DEQUE is full"

[End of if structure]

Step 3: - if (~~left~~ Right == -1)

Set left = 0.

Set right = 0

Step 4: - else if (right = maxsize -1) then

Set right = 0

else

right = right + 1.

Step 5: - DEQUE[right] = item.

Step 6: - Exit

GATEWAY CLASSES

Algorithm: Insertion DEQUE Left (front) Side

This algorithm is used to insert an element at the left side of the DEQUE.

Step 1:- Read "Item" to be inserted to the DEQUE.

Step 2:- [check for overflow]

If (left==0 && right == maxsize – 1) || (left == right + 1) then

Write " Overflow" or "DEQUE is full"

[End of if structure]

Step 3: - if (left == - 1)

Set left = 0.

Set right = 0

Step 4: - else if (left == 0) then

Set left = maxsize - 1

else

left = left - 1.

Step 5: - DEQUE [left] = item.

Step 6: - Exit

GATEWAY CLASSES

Algorithm: Deletion DEQUE Right (rear) Side

This algorithm is used to delete an element from the right side of the DEQUE.

Step 1:- if left = = -1 then

Write " underflow" or "DEQUE is empty"

[End of if structure]

Step 2:- item = DEQUE [right]. ← Deletion

Step 3: - if (left == right) then

Set left = -1.

Set right = -1

Step 4: - else if (right == 0) then

Set right = maxsize -1

else

right = right - 1.

Step 5: - Exit

GATEWAY CLASSES

Algorithm: Deletion in DEQUE from the Left (front) Side

This algorithm is used to delete an element from the ~~right~~ left side of the DEQUE.

Step 1:- if left = = -1 then

Write " underflow" or "DEQUE is empty"

[End of if structure]

Step 2:- item = DEQUE [left].

Step 3: - if (left == right) then

Set left = -1.

Set right = -1

Step 4: - else if (left == maxsize - 1) then

Set left = 0

else

left = left + 1.

Step 5: - Exit

GATEWAY CLASSES

Today's Target

- What is Priority Queue Data Structure?
- Types of Priority Queue
- Representation of Priority Queue
- Application of Queues Data structure
- Difference between Stack and Queue Data Structures
- AKTU PYQs

GATEWAY CLASSES

AKTU PYQs

Q.1 Differentiate between deque and priority queue. (AKTU 2014-15, 5 Marks)

Q.2 Write short note on Priority Queue. (AKTU 2016-17, 5 Marks)

Q.3 What do you mean by priority queue? (AKTU 2017-18, 2 Marks)

Q.4 What do you mean by priority queue? Explain the types to maintain the priority queue in memory. (AKTU 2018-19, 7 Marks)

Q.5 Define priority queue? Give one application of priority queue. (AKTU 2019-20, 2 Marks)

Q.6 Differentiate between circular queue and priority queue. (AKTU 2020-21, 2 Marks)

Q.7 Define priority queue and its uses. (AKTU 2021-22, 2 Marks)

GATEWAY CLASSES

Priority Queue

- Many applications requires priority queues rather than the simple FIFO strategy.
- A priority queue is another type of queue data structure in which elements can be inserted or deleted based on the priority i.e. each element in the priority queue is assigned a priority number and the element are inserted on the priority bases instead of using the order of arrival.
- If the priority is same than the FIFO order is used.

GATEWAY CLASSES

View of Priority Queue

FRONT PQ ↓ REAR PQ ↓

Elements			A	B	C	.....	X	.....	Z
Priority No.			P_1	P_2	P_3		P_i		P_n

GATEWAY CLASSES

One way list representation

In this representation each node of the linked list will have 3 fields.

1. An information field i.e. INFO
2. A priority number i.e. PRN
3. An address to the next node i.e. LINK

X Z 1 2 24

INFO	PRN	NEXT

NODE REPRESENTATION TO REPRESENT PRIORITY Queue through one-way list representation

- A node X precedes a node Y in the list when X has higher priority than Y or when both have same priority but X was inserted to the list before Y.
- Priority number will operate in the usual way i.e. the lower the priority number, the higher the priority.

GATEWAY CLASSES

The first element in the list is the element with maximum priority. So it is very simple to delete an element from the priority queue. It takes only O(1) time.

Algorithm: PRIORITY QUEUE DELETION

1. ITEM = INFO[FIRST]
2. DELETE FIRST NODE FROM THE LIST
3. PROCESS ITEM
4. STOP

GATEWAY CLASSES

Adding an element to priority queue is much more complicated then deleting an element from the queue, because we need to find the correct place to insert the element.

Suppose we add an ITEM with priority number N to a priority queue than-

(a) Traverse the list until finding a node X whose priority number exceeds N.

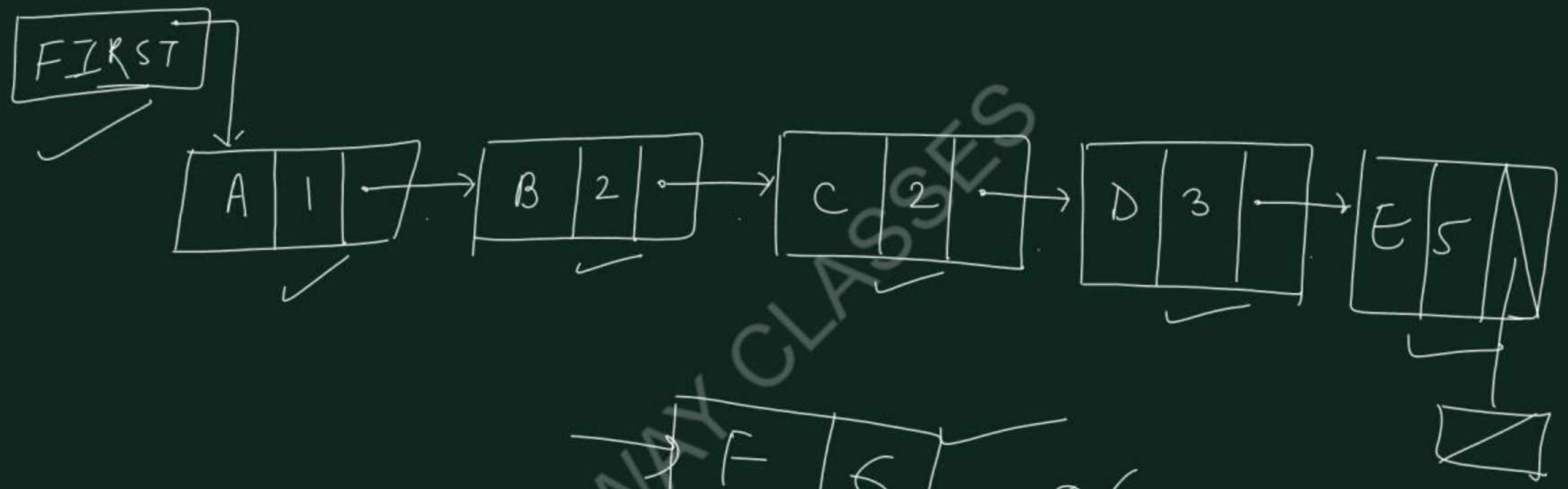
(b) Insert the node ITEM in front of X.

(c) If no such node is found , insert ITEM as the last node of the list.

So, insert needs O(n) time.

Consider the following priority queue with 5 nodes.

GATEWAY CLASSES

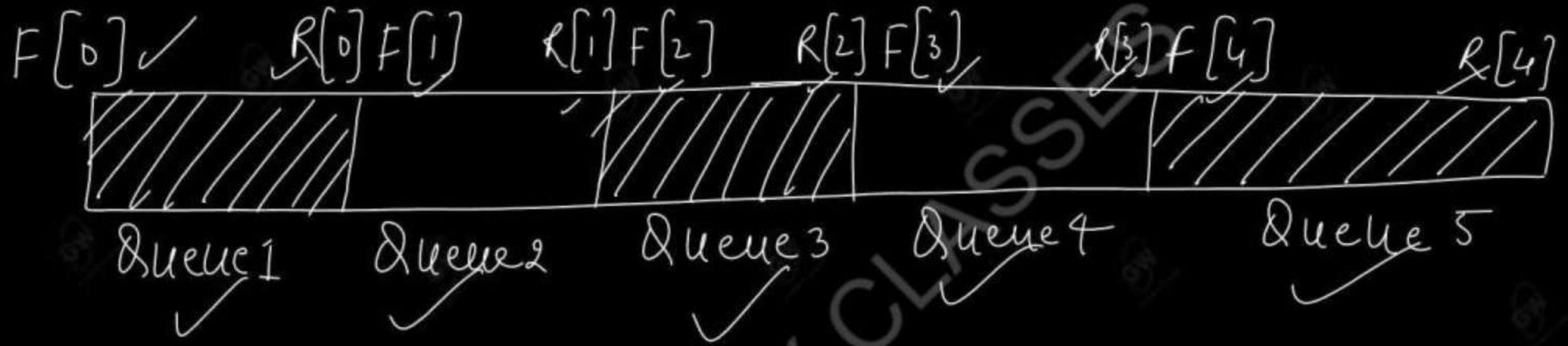


$O(n)$

GATEWAY CLASSES

Multiple Queue Representation of Priority Queue

Consider the following representation:-



- In this representation, multiple queues are used in a same array queue[] of size n.
- Each queue i is allocated a predefined space bounded by array indices.
- In order to preserve multiple queues, there should be multiple front and multiple rear ends. So, here a vector front and rear denotes the indices for each queue.

GATEWAY CLASSES

Heap Representation of a Priority Queue

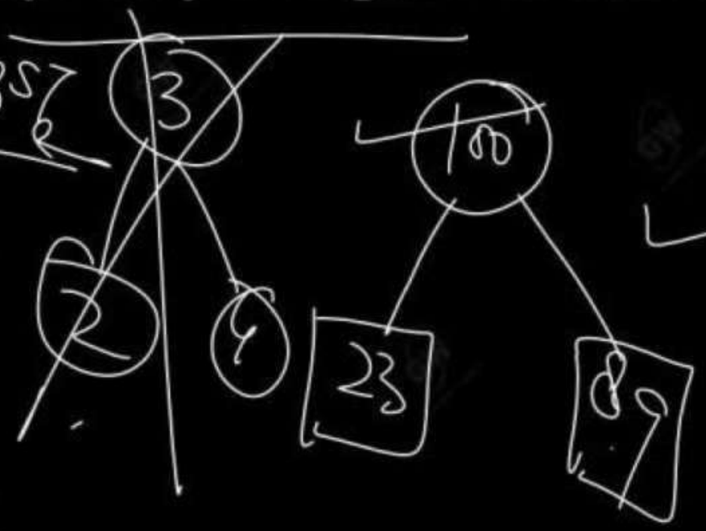
- A heap is a complete binary tree with additional property – the root node is either smallest or largest from its children. → Max heap.
- If root node of the heap is smallest from its children, it is called min heap. ↑ min heap
- If root node is largest from its children, it is called max heap.
- A priority queue having highest priority for lower number can be represented using min heap and a priority queue having the highest priority for higher number can be represented using max heap.

Heap

Max

Min

BST



Min heap

GATEWAY CLASSES

Recursion

- When a function call to itself that function is known as recursive function.
- Recursion is a process of expressing a function in terms of itself.
- We can also define recursion as a process in which a function calls itself with reduced input and has a base condition to stop the process, i.e. in order to solve a problem recursively, the following two conditions must be satisfied:-

(a) First, the problem must be written in a recursive form, and

(b) Second the problem statement must include a stopping condition.

GATEWAY CLASSES

1. Direct
2. Indirect
3. Tail
4. nontail

```
ABC()
{
    ABC()
}
```

XYZ()

{

ABC()

}

Indirect

ABC()

{

XYZ()

}

Example:2. Write a program in C to calculate the factorial of an integer using recursion.

Solution:-

```
/*TO CALCULATE THE FACTORIAL OF A NUMBER USING RECURSION*/

int factorial(int);
#include<stdio.h>
void main()
{
int a,fact;
clrscr();
printf("\n enter a number to calculate the factorial of that number:");
scanf("%d",&a);
fact=factorial(a);
printf("\n factorial of %d=%d",a,fact);
getch();
}

int factorial(int a)
{
if(a==0)
return 1;
else
return(a*factorial(a-1));
}
```

/* fun' prototype */

/* funcall */

Base criteria

5 == 0 F
4 == 0 F
3 == 0 F
2 == 0 F
1 == 0 F
0 == 0 T

5 = 120

a = 5
5 * factorial(4)
5 * 4 * factorial(3)
5 * 4 * 3 * factorial(2)
5 * 4 * 3 * 2 * factorial(1)
5 * 4 * 3 * 2 * 1 * factorial(0)
5 * 4 * 3 * 2 * 1 * 1
= 120

GATEWAY CLASSES

```
int factorial (int a)
{
  if (a==0)
   return 1;
  else
   return (a * factorial(a-1)),
}
```

5 4 3 2 1 0

5==0 F
4==0 F

5 * factorial(4)
5 * 4 * factorial(3)
5 * 4 * 3 * factorial(2)
5 * 4 * 3 * 2 * factorial(1)
5 * 4 * 3 * 2 * 1 * factorial(0) = 120

Definition & Principle of Recursion:-

- A function is said to be recursively defined, if a function containing either a Call statement to itself or a Call statement to a second function that may eventually result in a Call statement back to the original function.

Direct

A recursive function must have the following properties:-

1. There must be certain criteria, called base criteria for which the function does not call itself.

2. Each time the function does call itself (directly or indirectly), the argument of the function must be closer to a base value.

GATEWAY CLASSES

① Direct

ABC()

{

ABC()

}

② Indirect Recursion

ABC()

{

XYZ()

{

XYZ()

{

ABC()

}

GATEWAY CLASSES

Example 3:- Write a program in C to print the reverse of a multi-digit number using recursion.

Solution:-

Note:- The static variables are able to preserve their previous value, and one doesn't need to initialize them again.

```
int reverse_digit(int);
#include<stdio.h>
void main()
{
int num,result;
clrscr();
printf("\n enter a multi-digit number:");
scanf("%d",&num);
result=reverse_digit(num);
printf("\n The reverse of the number
is:%d",result);
getch();
}
int reverse_digit(int a)
{
int b;
static int r=0;
if(a==0)
return a;
else
{
b=a%10;
r=r*10+b;
reverse_digit(a/10);
}
return(r);
}
```

/* funⁿ prototype */

1243

3421

Input:- 1243

Output:- 3421

1234==0 F
123==0 F
12==0 F
1==0 F
0==0 T

1243/10=124
124/10=12
12/10=1
1/10=0

GATEWAY CLASSES

1243

```
int reverse_digit (int a)
{
    int b;
    static int r=0;
    if (a==0) F
     return a;
     else
     {
        b = a % 10;
        r = r * 10 + b;
        reverse_digit(a/10);
     }
     return(r);
}
```

$1234 \% 10 = 4$

$1234 / 10 = 123$

$r = 0$

$1243 == 0$ F

$b = 1243 \% 10 = 3$

$r = 0 * 10 + 3$

$= 3$

reverse_digit (~~123~~ 1243 / 10 = ~~123~~ 124)

$a = \cancel{123}\ 124$

$124\ \cancel{123} == 0$ F

$b = \cancel{3}\ 4$

$r = 3 * 10 + \cancel{3}\ 4 = 34$

return — (12)

Example 4:- Write a program in C to find the sum of all digits of a multi-digit number using recursion.

Solution:-

```
int sum_digits(int);
#include<stdio.h>
void main()
{
int num,result;
clrscr();
printf("\n enter a multidigit number:");
scanf("%d",&num);
result=sum_digits(num);
printf("\n The reverse of the number
is:%d",result);
getch();
```

sum

/* fun^n call statement */

10

Input:- 1243

Output:- 3421

output:- 1+2+4+3 = 10

```
}
int sum_digits(int a)
{
int b;
static int sum=0;
if(a==0)
return a;
else
{
b=a%10;
sum=sum+b;
sum_digits(a/10);
}
return(sum);
}
```

1243

function header

1243==0 F
124==0 F
12==0 F
1==0 F
0==0 T

a=1243
sum=0
b=1243%10=3
sum=0+3=3
sum_digits(1243/10)
124

a=124
b=124%10=4
sum=3+4=7
sum_digits(124/10)
12

a=12
b=12%10=2
sum=7+2=9
sum_digits(12/10)
1

GATEWAY CLASSES

2 marks

Example -5 Write a program in c to multiply two natural numbers using recursion.

Solution:-

```
int multiply(int,int); /* func^n Prototype */
#include<stdio.h>
void main()
{
int num1,num2,result;
clrscr();
printf("\n enter the first number:");
scanf("%d",&num1);
printf("\n enter the second number:");
scanf("%d",&num2);
result=multiply(num1,num2);
printf("\n The multiplication of %d and %d =
%d",num1,num2,result);
getch();
}
int multiply(int i, int j)
{
if(i==0, j==0)
return 0;
if(j==1)
return i;
else
return(multiply(i,j-1)+i);
}
```

5 × 4 = 20

20 5 4 5 4 20

5 1
5 3
5 4

return(multiply(5,3)+5)
return(multiply(5,2)+5+5)
return(multiply(5,1)+5+5+5)
5+5+5+5
20

GATEWAY CLASSES

Example 6: Write a program in C to print the Fibonacci series using recursion.

Solution:-

```
void fibonacci_series(int);  /* funⁿ Prototype or funⁿ Declaration */
#include<stdio.h>
void main()
{
int counter,first=0,second=1;
clrscr();
printf("\n how many first term of fibonacci series you want to print:");
scanf("%d",&counter);  5
if(counter>2)  T
{
printf("\n %d",first);
printf("\n %d",second);  3
fibonacci_series(counter-2);
}
else if(counter==2)
{
printf("\n %d",first);
printf("\n %d",second);
}
else if(counter==1)
printf("\n %d",first);
else
printf("\n Fibonacci series printing is not possible");
getch();
}
```

0
1

GATEWAY CLASSES

```
void fibonacci_series(int a)
{
int b;
static int first=1,second=1;
if(a==0)
printf("\n The series ends here");
else
{
b=first+second;
first=second;
second=b;
printf("\n %d",b);
fibonacci_series(a-1);
}
}
```

Fibonacci series -

0, 1, 1, 2, 3, 5, ...

I 5 terms

Output:

0, 1, 1, 2, 3

a = 3 first = 0 second = 1

b = 0 + 1 = 1

first = 1

second = 1

a = 2

b = 1 + 1 = 2

first = 1

second = 2

b = 1 + 2 = 3

first = 2

second = 3

0
1
1
2
3

The series ends here

GATEWAY CLASSES

```
void fibonacci_series (int a)
{
 int b;
 static int first=0, second=1;
 if (a==0)
   { printf("\n The series ends here");
   } else
    {
      b = first + second;
       first = second;
         second=b;
      printf("%d", b);
    fibonacci_series (a-1);
  }  }
```

Example – 7: Write a program in C to calculate the power of a given number using recursion.

```c
int power(int,int);
#include<stdio.h>
void main()
{
int a,b,result;
clrscr();
printf("\n enter a base value:");
scanf("%d",&a);
printf("\n enter a power value:");
scanf("%d",&b);
result=power(a,b);
printf("\n The %d raised to %d =%d",result);
getch();
}
int power(int a,int b)
{
static int r=1;
if(b==0)
return 1;
else
{
r=r*a;
power(a,b-1);
}
return(r);
}
```

/* funⁿ prototype */

$3^3 = 27$

a, b, result

$r = 1 * 3 = 3$

power(3, 2)

$r = 3 * 3 = 9$

power(3, 1)

$r = 9 * 3 = 27$

power(3, 0)

GATEWAY CLASSES

Example 1:- Write a program in C to print the table of a number using recursion.

Solution –

```
int table ( int , int );
#include<stdio.h>
void main()
{
int a,b,result;
clrscr();
printf("\n enter a number for which the table is to be printed:");
scanf("%d",&a);
for(i=1;i<=10;i++)
printf("\n %d * %d = %d",a,i,table(a,i));
getch();
}
int table(int a,int i)
{
if(i==1)
return a;
else
return(table(a,i-1)+a);
}
```

5×1=5
5×2=10
5×3=15
5×4=20
10

table(5,1)+5
5+5=10
table(5,2)+5
table(5,1)+5+5
5+5+5=15

GATEWAY CLASSES

Example 2 : - Write a program in C to reverse an array of n elements using recursion.

Solution:-

Input:-

0	1	2	3	4
11	22	33	44	55

Output:

0	1	2	3	4
55	44	33	22	11

```
void main()
{
int a[20],i,n;
printf("\n How many numbers you want to input:");
scanf("%d",&n);
if(n>20)
{
printf("\n Invaid input, size must be less then 20");
getch();
exit();
}
/*Input array for n elements */
for(i=0;i<n;i++)
{
printf("\n enter number %d:",i+1);
scanf("%d",&a[i]);
}
printf("\n the reverse array is:");
reverse(a,n-1);  /* function call */
getch();
}
```

5-1
4

GATEWAY CLASSES

```
void reverse(int a[],int n)
{
If(n==-1)
return;
printf("\n%d",a[n]);
reverse(a,n-1);
}
}
```

4 == -1 F

-1 == -1 T

else {

/* fun call */

addr

Output -

a [4] = 55

[3] = 44

[2] = 33

[1] = 22

[0] = 11

GATEWAY CLASSES

Greatest Common Divisor (GCD) of Two numbers

- The greatest common divisor of two integers m and n is the largest common divisor of m and n and is denoted by GCD (m, n). This divides both m and n such that the remainder is zero.
- The greatest common divisor (GCD), also known as "Greatest Common Factor (GCF)" or "Highest Common Factor (HCF)", of two numbers is the greatest factor that divides both the numbers.
- Euclid's Method: To find the greatest common divisor of two positive integers with the help of Euclid's algorithm is as follows:
- The recursive definition for finding an GCD of two numbers is:-

$$GCD(m,n) = \begin{cases} GCD(n,m) & if\ n > m \\ m & if\ n = 0 \\ GCD(n, m \bmod n) & if\ m > n \end{cases}$$

GATEWAY CLASSES

Example 3:- Write a C program to find the GCD of two numbers using recursion.

Solution:-

```
int gcd(int,int);
#include<stdio.h>
void main()
{
int a,b,result;
clrscr();
printf("\n enter the first number:");
scanf("%d",&a);
printf("\n enter the second number:");
scanf("%d",&b);
result=gcd(a,b);
printf("\n The gcd of %d and %d =
%d",a,b,result);
getch();
}
int gcd(int a,int b)
{
int result;
if(b==0)
return a;
else
{
result=a%b;
return(gcd(b,result));
}
}
```

GATEWAY CLASSES

EXAMPLE 4. Write a recursive algorithm to implement the following function:-

Ackerman function

$$A(m,n) = \begin{cases} n+1 & if\ m = 0 \\ A(m-1,1) & if\ m\ != 0\ and\ n = 0 \\ A(m-1, A(m, n-1)) & if\ m\ != 0\ and\ n\ != 0 \end{cases}$$

And considering this algorithm, find A (1 , 3). {Ans = 5}

Solution:-

Algorithm A(m,n)

1. If m==0

Return n + 1

2. If n==0

Return call A(m-1,1)

3. Call A((m-1,A(m,n-1))

4. Stop

GATEWAY CLASSES

$$A(m,n) = \begin{cases} n+1 & if\ m = 0 \\ A(m-1,1) & if\ m\ != 0\ and\ n = 0 \\ A(m-1, A(m,n-1)) & if\ m\ != 0\ and\ n\ != 0 \end{cases}$$

Considering this algorithm ,find A (1 , 3). {Ans = 5}

Solution:-

$A(1,3) = A(0, A(1,2))$

$A(1,2) = A(0, A(1,1))$

$A(1,1) = A(0, A(1,0))$

$A(1,0) = A(0,1)$

$A(0,1) = 2$

$A(1,0) = 2$

$A(1,1) = A(0,2)$

$A(0,2) = 3$

$A(1,1) = 3$

$A(1,2) = A(0,3)$

$A(0,3) = 4$

$A(1,2) = 4$

$A(1,3) = A(0,4)$

$A(0,4) = 5$

Ans $A(1,3) = 5$

$A(1,1) = A(0,2)$
$A(0,2) = 3$
$A(1,1) = 3$
$A(1,2) = A(0,3)$

$A(0,3) = 4$
$A(1,2) = 4$
$A(1,3) = A(0,4)$
$A(0,4) = 5$

GATEWAY CLASSES

EXAMPLE 5: Let a and b are positive integers. Suppose a function Q is defined recursively as follows:

$$Q(a,b) = \begin{cases} Q & \text{if } a < b \\ Q(a-b,b)+1 & \text{if } b \le a \end{cases}$$

Find the value of Q(2,3) and Q(14,3). (ans =0,4)

$Q(\overset{a}{2},\overset{b}{3}) = 0$ Ans

(i) $Q(\overset{a}{2},\overset{b}{3}) = 0$

(ii) $Q(14,3) = Q(11,3)+1$

$= Q(8,3)+1+1$

$= Q(5,3)+1+1+1$

$= \underbrace{Q(\overset{a}{2},\overset{b}{3})}_{0}+1+1+1+1$

$= 0+4$

$= 4$

GATEWAY CLASSES

$$Q(\overset{a}{14},\overset{b}{3}) = \underbrace{Q(11,3)}+1$$

$$= \underbrace{Q(8,3)}+1+1$$

$$= \underbrace{Q(5,3)}+1+1+1$$

$$= \underbrace{Q(2,3)}+1+1+1+1$$

$$= 0+4=4$$

EXAMPLE 6: A recursive function f is as follows:-

```
int f ( int x )
{
if ( x < 2 )
return 1;
else
return f ( x – 1 ) + f ( x – 2 );
}
```

Find value of f (5). (ans=8)

Solution:-

F(5)

return f(4) + f(3)

return f(3) + f(2) + f(2) + f(1)

f(2) + f(1) + f(1) + f(0) + f(1) + f(0) + 1

f(1) + f(0) + 1 + 1 + 1 + 1 + 1 + 1

1 + 1 + 6

= 8

GATEWAY CLASSES

DIRECT AND INDIRECT RECURSION

- Recursion is said to be direct if a subprogram calls itself, it is indirect if there is a sequence of more than one sub-program call that eventually calls the first sub-program such as when function A calls function B, which in turn calls A.

GATEWAY CLASSES

Today's Target

- Tail Recursion
- Tower of Hanoi Problem
- AKTU PYQs

GATEWAY CLASSES

TAIL RECURSION

- As we know the process in which function calls itself in function body is called as recursion. But when this called function is the last executed statement in the function body then it is called "tail recursion".
- We use stack to maintain all the functions but here it will not append new function in stack but it will overwrite the value of previous function with the current one.
- So we can say the function call time and stack implementation time will be reduced and it will give better performance.

GATEWAY CLASSES

TAIL RECURSION

- In tail recursion, there is a single recursive call that occurs at the very end of the function implementation.
- In other words, when the recursive call is made, there are no more statements to be executed in the function. The single recursive call is the last thing to happen when the function is executed.

GATEWAY CLASSES

Consider the following example:-

```
void main()
{
function(3);
getch();
}
void function(int n)
{
if(n==0)
return;
else
printf("%d",n);
return function(n-1);
}
```

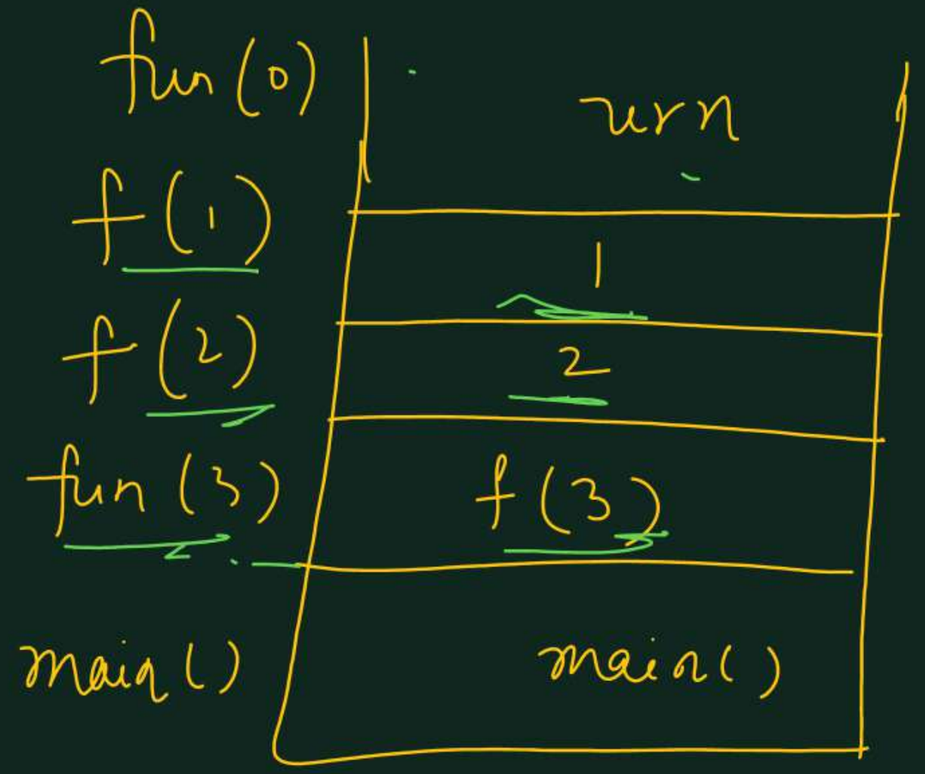
function(1)	return
function(1)	Act. rec. of fun(1)
function(2)	Act. rec. of fun(2)
function(3)	Act. rec. function(3)
main()	Act. rec. main()

GATEWAY CLASSES

Non tail

```
void main()
{ function(3);
}
void function(int n)
{
    if (n==0)
        return ;
    else
        function(n-1);
    print
}
```

1 2 3



THE TOWERS OF HANOI PROBLEM OR
THE PROBLEM OF BANARAS TAMPLE OR
TOWER OF BRAHMA OR
LUCAS TOWER

- Tower of Hanoi is a mathematical puzzle invented by French Mathematician Edouard Lucas in 1883.
- The problem has a historical basis in the ritual of the ancient Tower of Brahma.
- History of Tower of Hanoi:-
- Numerous myths regarding the ancient and mystical nature of the puzzle popped up almost immediately, including a myth about Kashi Vashwanath Temple containing a large room with three poles in it, surrounding by 64 golden disks.
- This story of Indian temple was spread tongue-in-check by a friend of Lucas.

GATEWAY CLASSES

- If the priests were able to move disks at a rate of one per second, using the smaller number of moves, it would take them $2^{64}-1$ seconds or roughly 585 billions (585 arabs) years to finish, which is about 42 times the estimated age of the universe. The universe is approximately 13.8 billion years old.

$2^{64}-1$

GATEWAY CLASSES

In the tower of Hanoi puzzle we have 3 rods and n disks and the objective of the puzzle is to move entire stack of disks from first rod to another rod by obeying the following rules:-

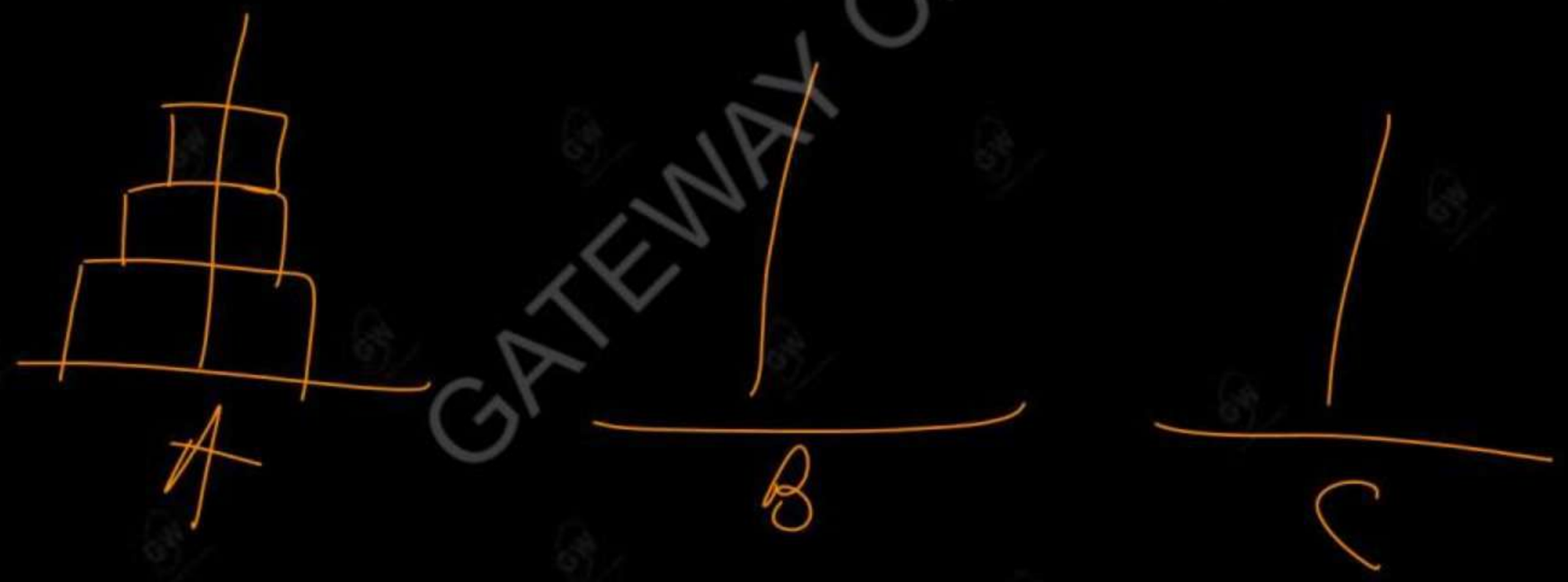
1. Only one disk can be moved at a time.
2. Each move consists of taking upper disk from one of the rods and placing it on top of another rod or an empty rod.
3. No disk can be placed on the top of the disk that is smaller it.

GATEWAY CLASSES

Consider the following figures with 3 rods and 3 disks.

GATEWAY CLASSES

- The requirement is, we have to shift disks from rod A to rod C using rod B by following above mentioned 3 rules.
- To understand this puzzle we will consider 3 disks as:-



GATEWAY CLASSES

Procedure:-

1. Move top n-1 disks from rod ~~a~~ A to rod B (A->B)
2. Move disks ~~from A~~ to C.
3. Move n-1 disks from B->C.

GATEWAY CLASSES

0 1 2 3

Algorithm:- TOH (n,A,B,C)

Where n = number of disks

A = from

B= using

C= to

If (n > 0) T

{

0 1 3 2

TOH(n-1,A,C,B)

Print(From %d to %d",A,C);

TOH(n-1,B,A,C);

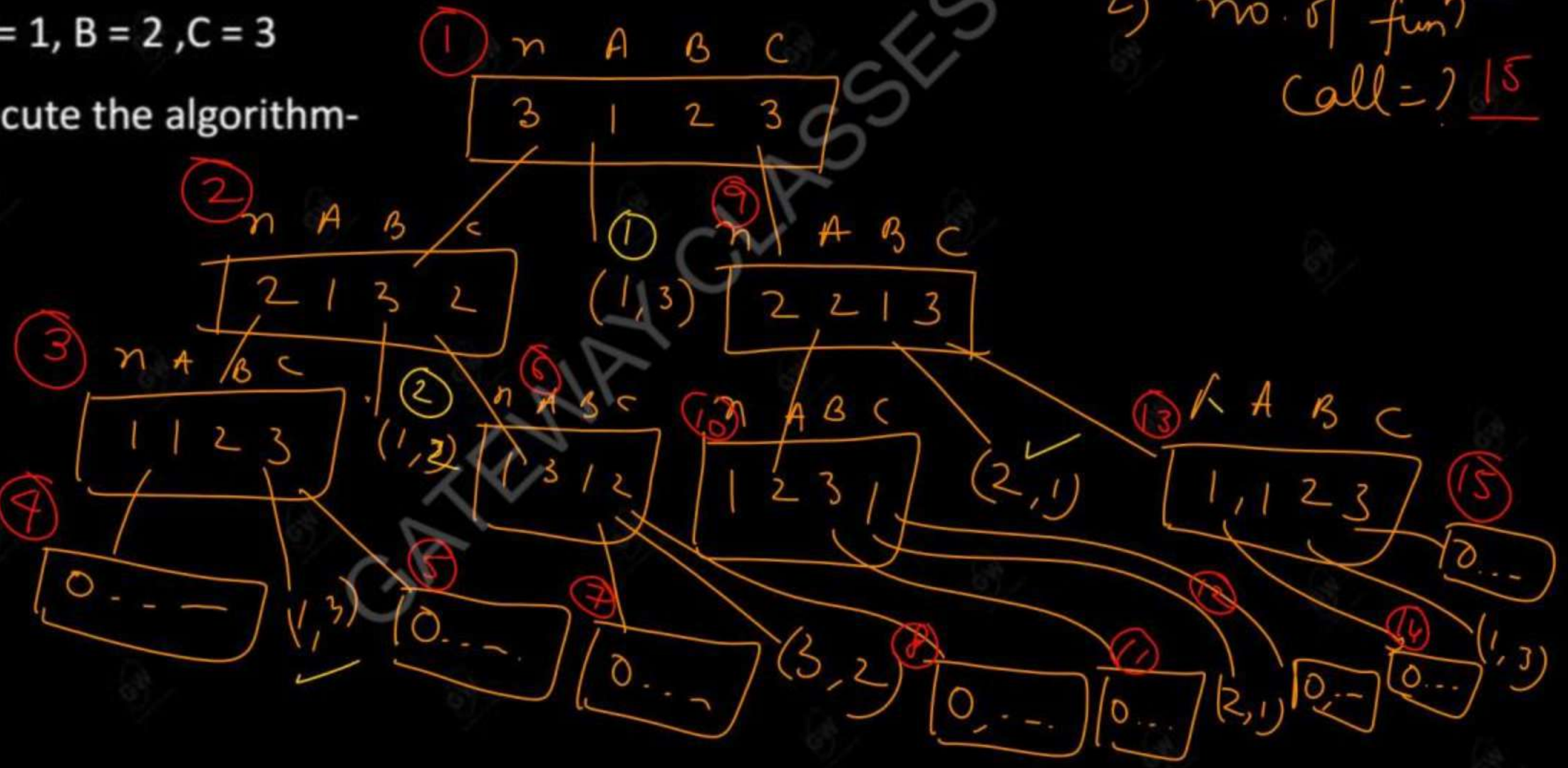
}

GATEWAY CLASSES

Here in our example –

N = 3, A = 1, B = 2 ,C = 3

Now execute the algorithm-



GATEWAY CLASSES

In this example total number of function calls = 15

And sequence of movement is as follows:-

(1,3)

(1,2)

(3,2)

(1,3)

(2,1)

(2,3)

(1,3)

GATEWAY CLASSES

General Formula for number of function calls = $2^{n+1} - 1 = 2^{3+1} - 1 = 2^4 - 1 = 16 - 1 = 15$

Formula for number of movements = $2^n - 1 = 2^3 - 1 = 8 - 1 = 7$

Example:- Say number of disks = 3

Example : Say number of disks = 2 $\longrightarrow$ no. of fun^n call $= 2^{n+1} - 1 = 2^{2+1} - 1 = 8 - 1 = 7$

no. of movement $= 2^n - 1 = 2^2 - 1 = 4 - 1 = 3$

GATEWAY CLASSES

Download Gateway Classes Application
From Google Play store
Link in Description

Next Lecture:- Unit – 3, Searching (Linear and Binary)

Thank You

GATEWAY CLASSES