Date / /

UNIT -2

Stack & Queue

★ **Stack :-**

1- Stack is a data structure used to store a collection of objects. Individual items can be added and stored in a stack using a push operation.

2- Objects are retrieved using a pop operation.

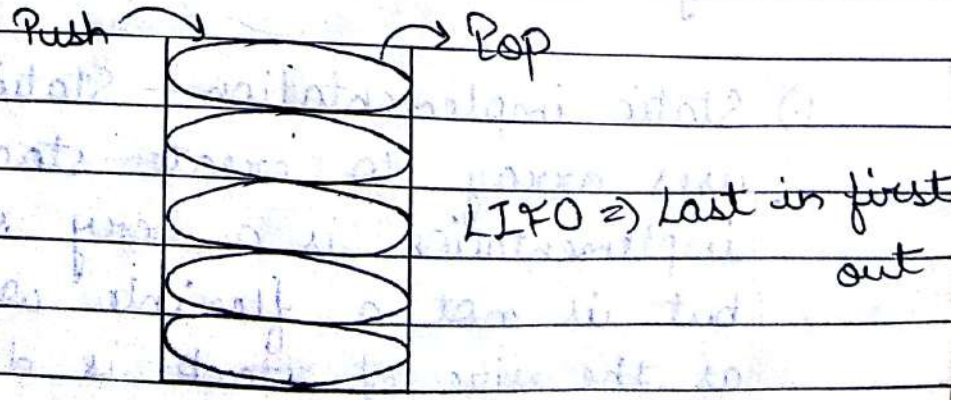
3- When an object is added to a stack, it is placed on the top of all previously entered items. A stack in which items are removed from the top in LIFO.

4- Stacks have several applications in computer programming. Stack can be used to retrieve recently used object from a cache memory.

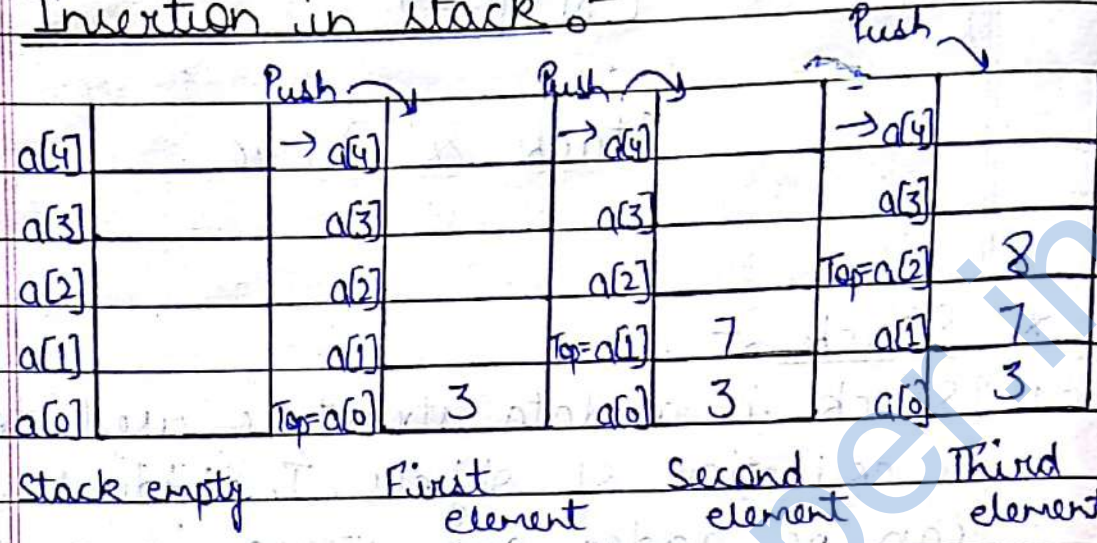
Example -

1- Plates in a marriage

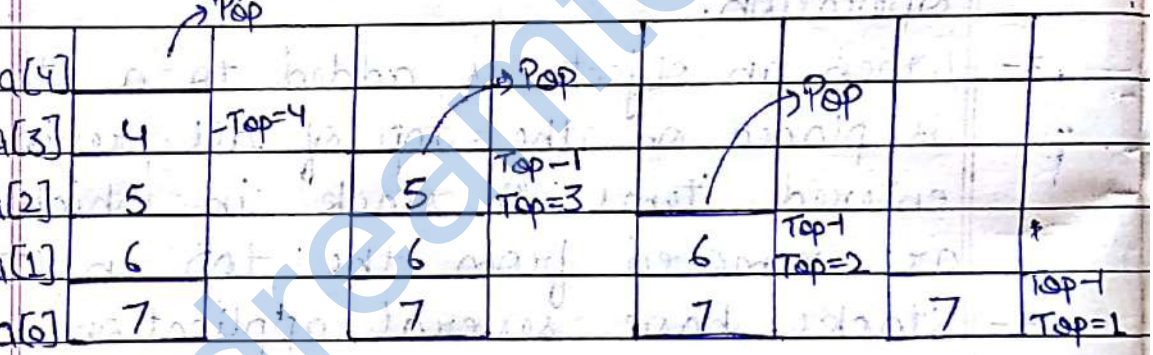
2- Coins stacker



* Insertion in stack :-



* Deletion in stack :-



* Stack implementation :-

1.) Static implementation
2.) Dynamic implementation

1.) Static implementation — Static implementation uses array to create stack. Static implementation is a very simple technique but is not a flexible way of creation, as the size of stack is declared during program design, after that size cannot

be varied.

2.) Dynamic implementation – Dynamic implementation is also called link-list representation and uses the pointers to implement the stack type of data structure.

* Inserting an item :-

Algorithm –

Step-1 Initialize
Set top = -1
Step-2 Repeat step 3 to 5 until top < maxsize - 1
Step-3 Read item
Step-4 Set top = top + 1
Step-5 Set stack[top] = item
Step-6 Print stack overflow

Function –

```
int stack[5], top = -1;
void push()
{
int item;
if (top < 4)
{
printf("Enter the no");
scanf("%d", &item);
```

```
top = top + 1;
stack [top] = item;
}
else
printf (" stack overflow");
getch();
}
```

* Pop operation :-

Pop (stack [max size], item);

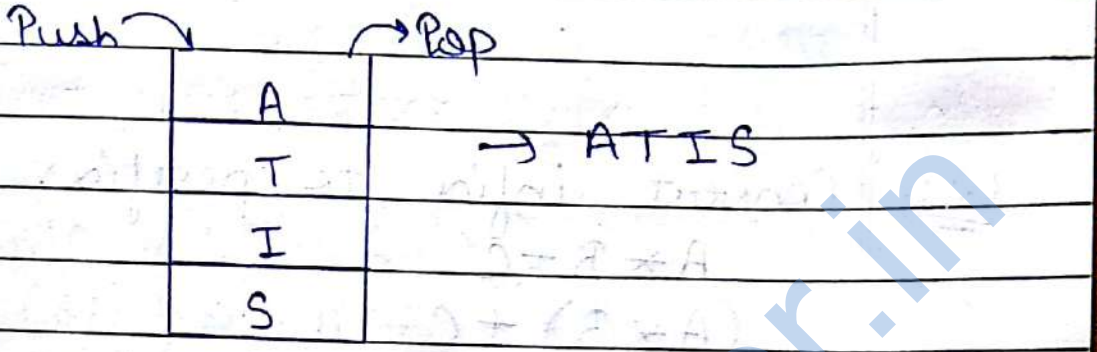
Step 1 - Repeat step 2 to 4 until top > 0
Step 2 - set item = stack [top]
Step 3 - set top = top - 1
Step 4 - print, no deletion
Step 5 - print stack underflow

Function -

```
void pop()
{
int item;
if (top > 0)
{
item = stack [top];
top = top - 1;
printf (" No deletion = %d", item);
}
else
printf (" stack underflow");
}
```

* Application of stack :-

1.) Reversing a string -



• Infix to postfix conversion of expressions-

Expressions -

A + B	AB +	+ AB
Infix	Postfix	Prefix

Rules -

1- Parenthesize the expression starting from left to right.
2- During parenthesizing the expression, the operands associated with operator having higher precedence first parenthesized.
3- The sub expression which has been converted into postfix is to be treated as a single operand.
4- Once the expression is converted to postfix form, remove the parenthesis.

Date / /

Operator precedence –

1. Exponential operator ^
2. Multiplication / Division * / /
3. Addition / subtraction + / –

Q1. Convert infix to postfix.

A * B + C

(A * B) + C

(AB *) + C Let AB * = T

T + C

TC +

AB * C +

Q2. A + B / C – D

A + (B / C) – D

A + (BC /) – D Let BC / = T

A + T – D

(A + T) – D

(AT +) – D AT + = U

U – D

UD –

AT + D –

ABC / + D –

A + (B / C) – D

Symbol	Stack	Postfix
A		A
+	+	A
(	+(	A

Date / /

B	+(	AB
/	+(/	AB
C	+(/	ABC
)	+	ABC/
−	−	ABC/+
D	−	ABC/+D−

Q3. (A+B/C * (D+E) − F)

(A + (B/C) * (D+E) − F)

(A + (BC/) * (D+E) − F) BC/ = T

(A + T * (D+E) − F)

(A+T * (DE+) − F) DE+ = P

(A+T * P − F)

(A + (T*P) − F)

(A + (TP*) − F)

(ATP*+) − F

ATP*+F−

ABC/DE+*+F−

((A+(B/C)*(D+E)) − F)

Symbol	Stack	Postfix
(	(	
(	((	
A	((	A
+	((+	A
(	((+(	A
B	((+(	AB
/	((+(/	AB
C	((+(/	ABC
)	((+	ABC/

*	((+*	ABC/
(	((+*(	ABC/
D	((+*(	ABC/D
+	((+*(+	ABC/D
E	((+*(+	ABC/DE
)	((+*	ABC/DE+
)	(	ABC/DE+*+
−	(−	ABC/DE+*+
F	(−	ABC/DE+*+F
)		ABC/DE+*+F−

- Infix to Prefix conversion −

1- Reverse the given string.
2- Convert into postfix.
3- Reverse the output.

Qus1: A + B/C − D (Infix to prefix)
Reversing the string
D − (C/B) + A

Symbol	Stack	Prefix
D		D
−	−	D
(	−(	D
C	−(	DC
/	−(/	DC
B	−(/	DCB
)	−	DCB/

+	−	DCB/+
A	−	DCB/+A−

−A+/BCD (Prefix)

Ques2. A+(B*C−(D/E*F)*G)

Reversing the string -

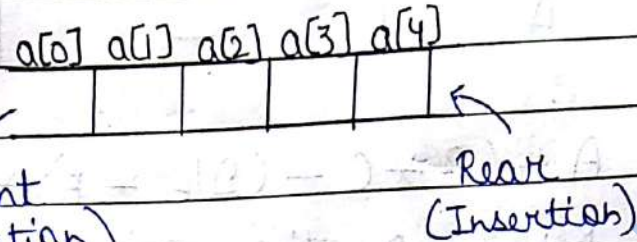
(G*(F*(E/D))−(C*B)+A

Symbol	Stack	Prefix
(	(	
G	(	G
*	(*	G
(	(*(	G
F	(*(	GF
*	(*(*	GF
(	(*(*(	GF
E	(*(*(	GFE
/	(*(*(/	GFE
D	(*(*(/	GFED
)	(*(*	GFED/
−	(*(−	GFED/*
C	(*(−	GFED/*C
*	(*(−*	GFED/*C
B	(*(−*	GFED/*CB
)	(*	GFED/*CB*−
+	(*+	GFED/*CB*−
A	(*+	GFED/*CB*−A*+

+*A−*BC*/DEFG ⇒ Prefix

Date / /

* Queue :- FIFO ⇒ First in first out



"A queue is a non-primitive linear data structure. It is an homogeneous collection of elements in which new elements are added at one end called the rear and the existing elements are deleted from other end called the front end.

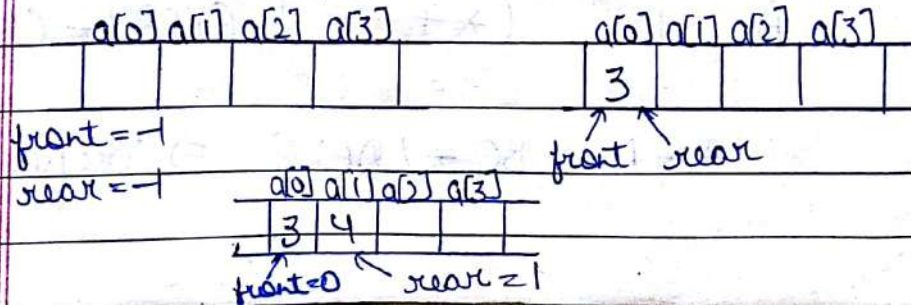
* Implementation of Queue :-

1- Static implementation - With the help of array.

2- Dynamic implementation - With the help of pointers.

* Operations in queue :-

1.) Insertion Operation -



Date / /

Algo-

[Queue, [Maxsize] item]

Step 1- Initialization
set front = -1
set rear = -1

Step 2- Repeat step 3 to 5 until Rear < Maxsize-1

Step 3- Read item

Step 4- if front = = -1 then
front = 0
rear = 0
else
rear = rear + 1
End if

Step 5- set queue[rear] = item

Step 6- Print queue overflow

Queue is declared as int queue[5]; front=-1, rear = -1

```
void queue()
{
int item;
if (rear < 4)
{
printf ("Enter the number");
scanf ("%d", &item);
if (front == -1)
{
front = 0;
rear = 0;
}
```

```
else
{
rear = rear + 1;
}
queue[rear] = item;
}
else
printf("Queue is full");
}
```

2.) Queue deletion –

```
Queue [Maxsize]
```

Step 1- Repeat stages 2 to 4 until front >= 0
Step 2- Set item = queue[front]
Step 3- if front = = rear
set front = -1.
set rear = -1
else
front = front + 1
end if
Step 4- printf ; no item for delete
Step 5- printf, "queue is empty".

Function for deletion in queue –

```
void delete()
{
int item;
if (front != -1)
{
item = queue[front];
```

```
if (front == rear)
{
front = -1;
rear = -1;
}
else
front = front + 1;
printf ("no deleted is :%d", item);
}
else
printf ("Queue is empty");
}
```

* Circular queue :-

1- A circular queue is one in which the insertion of a new element is done at the very first location of the queue if full.

2- A circular queue overcomes the problem of unutilized space in linear queue implemented as array.

3- A circular queue also has a front and a rear to keep the track of the elements to be deleted and therefore to maintain the unique characteristics of the queue.

The below assumptions are made -

1- Front will always be pointing the field element.

2- If front = rear, the queue will be empty.

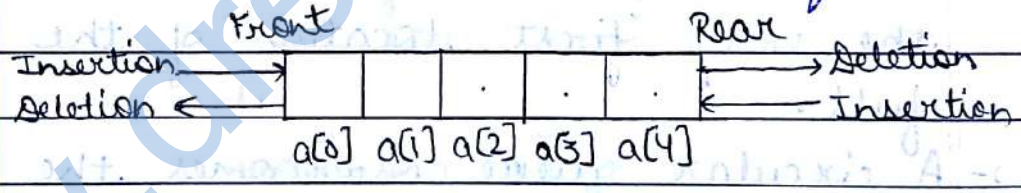
3- Each time a new element is inserted into the queue, the rear is increment -ed by 1.

rear = rear + 1.

4- Each time an element is deleted from the queue, the value of front is incremented by 1.

front = front + 1.

* Double - ended queue :- It is also a homogeneous list of elements in which insertion and deletion operations are performed from both the end i.e. we can insert element from the rear end and from the front end. Hence it is called double ended queue.



* Priority queue :- A priority queue is a collection of elements such that each element has been assigned a priority and there order in which elements are deleted and processed from the following rules -

An element of higher priority is processed according to the order in which they were added to the queue.

Date / /

★ Applications of priority queue –

1- Round Robin technique for processor scheduling is implemented using queue.
2- All types of costumer services (like – railway ticket reservation) Costumer services centre of software are designed using queues to store costumer information
3- Printer server routines are designed using queue.