NO. Date: / /

Stack empty

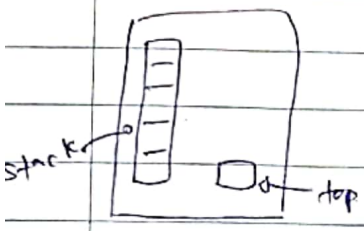
```
int stempty ( )
{
 if (st.top == -1)
    return 1;
 else
   return = 0
}
```

Stack full

```
int stfull ( )
{
int (st.top >= size-1)
   return 1;
   else
   return = 0
}
```

> Array have a Random access
> Stack haven't a Random acces — this the purpose

// Structure



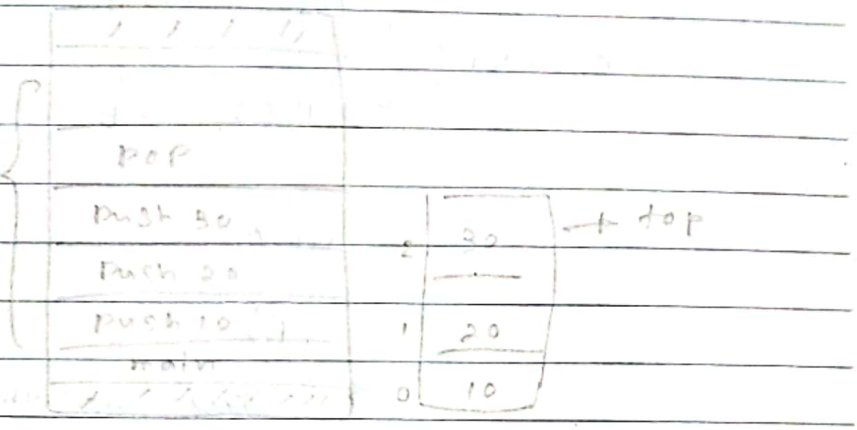

```
Struct stack {
int arry[size];
int top;
} st;
```

Find the erros

```
#include <stdio.h>
 Struct mystack {
    int my array [5];
    int top = -1;          → Int top;

} st;

 Void Push ( ) {           → Void Push (int ele)
 St. top++;
 St.myarry [top] = ele;    → St.myarray [St.top] = ele;
 Printf("item Added to index %d \n", St.top);
 }
```

NO. Date: / /

```
int pop(int ele) {          → int pop( )

int out = myarray[st.top];  → int out = st.myarray[st.top];
                            → st.top--;
printf("popped out item at index %d", st.top);

return out;
}
int main() {
push(10);
Push(20);
Push(30);
pop();
return;
```



is empty and it is full enter that code

Evaluate postfix notation

> infix notation

Ex:- x+y

[operator comes in between the operants]

> prefix notation

Ex:- ++x

[operator comes before the operants (+xy)]

> postfix notation

Ex:- xy+

[operator comes after the operants (xy+)]

* to infix ⟶ Postfix we can used stack

10 30 * 5 +

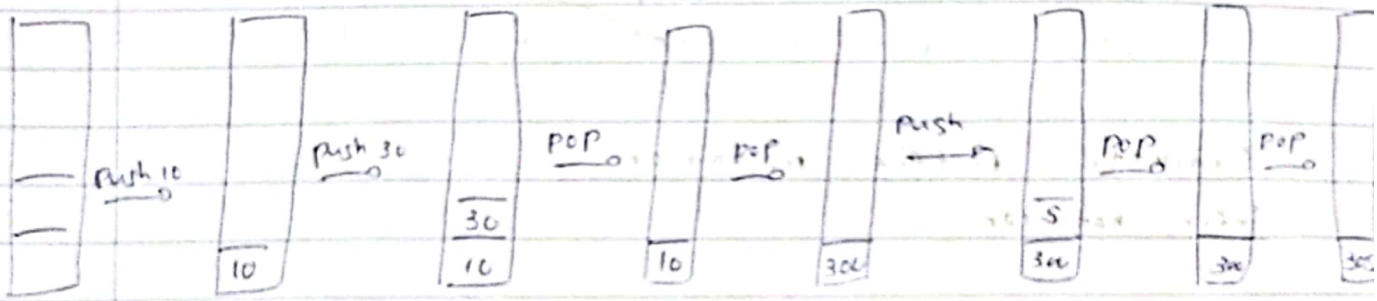
> If it is operand push into stack
> If it is operator can't push into stack

↓ to operators

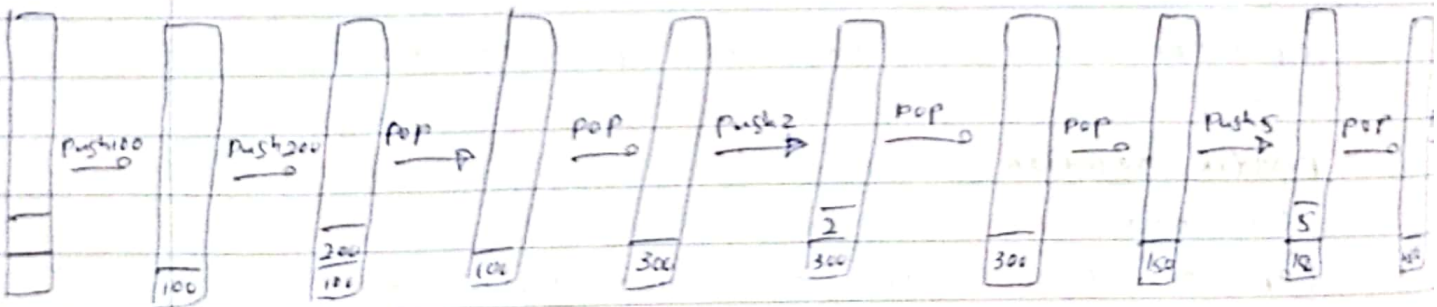
Operator → pop() → A
Operator → pop() → B

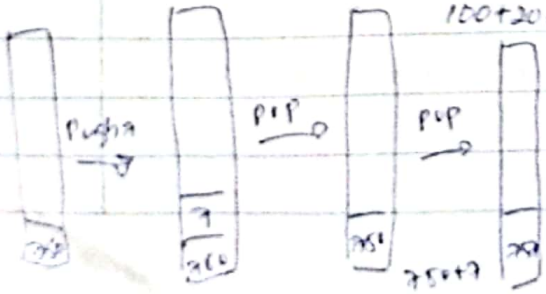
B O A

(R) add the ~~res~~ final result to the stack



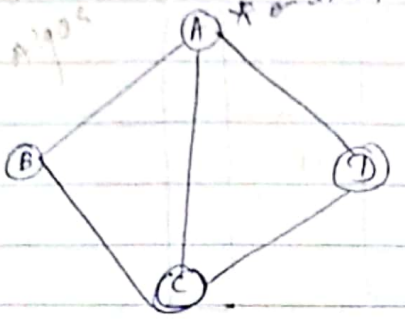
100 200 + 2 / 5 * 7 +



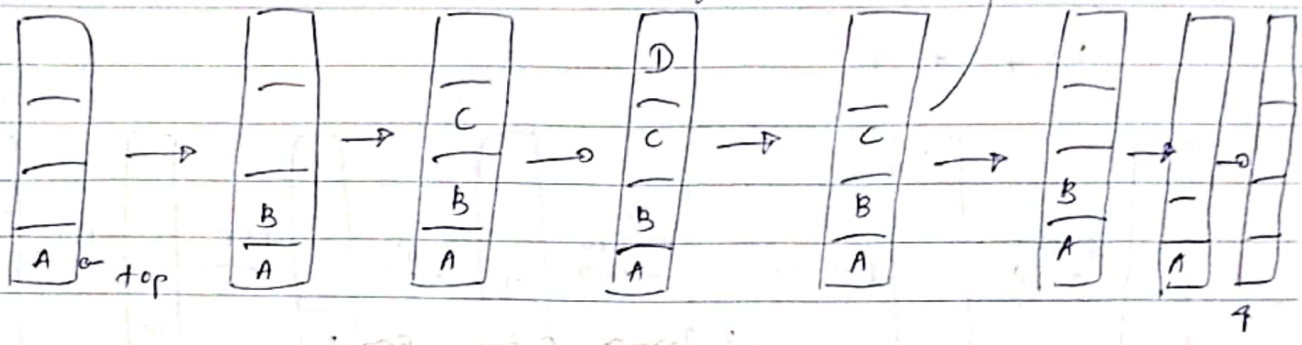


NO. Date: / /

DFS → Depth First Search [graphs traversing Algo]

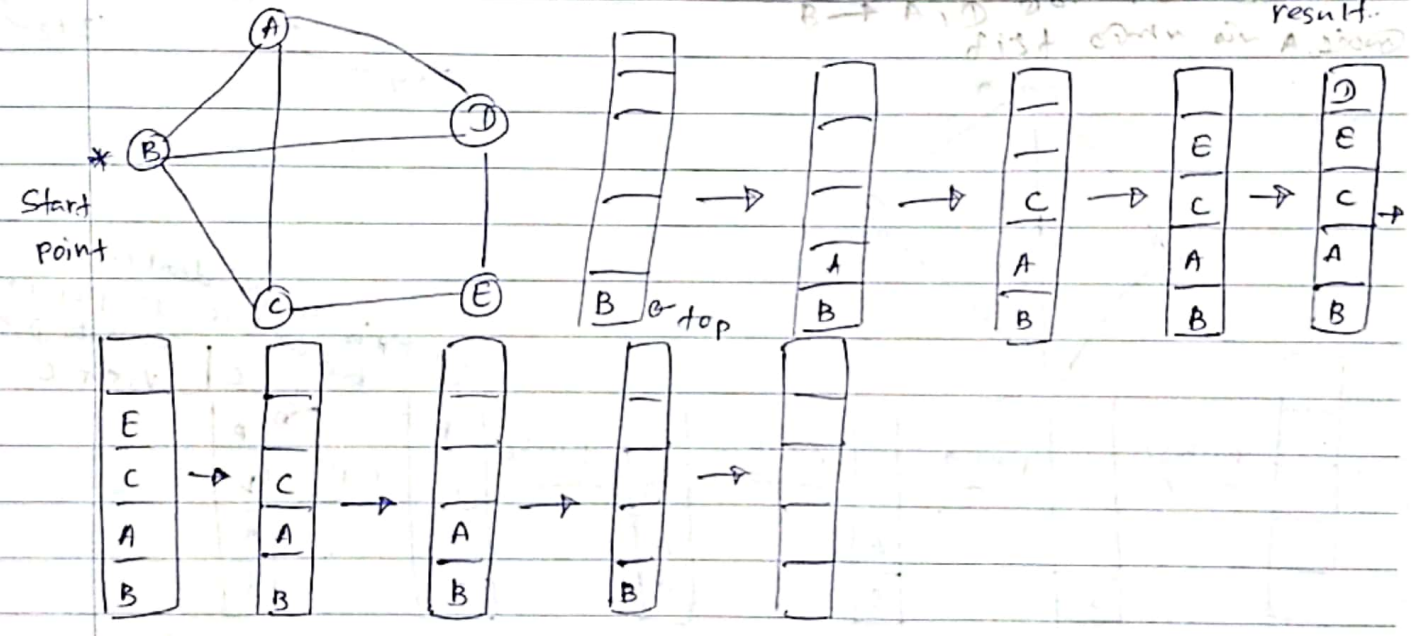


* → start point.



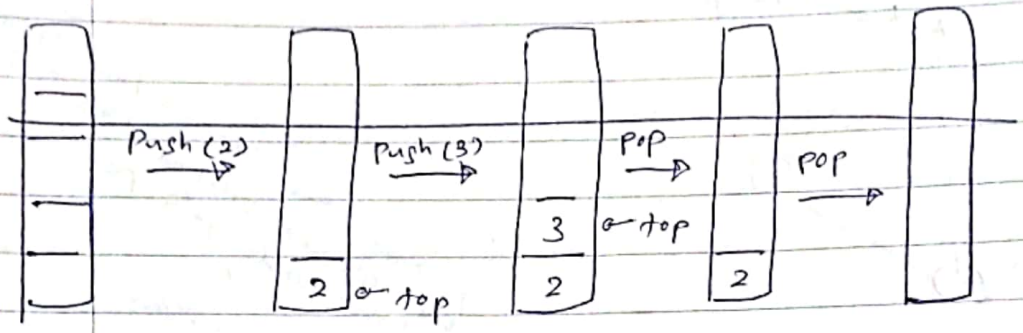
Alphabet order

final result.

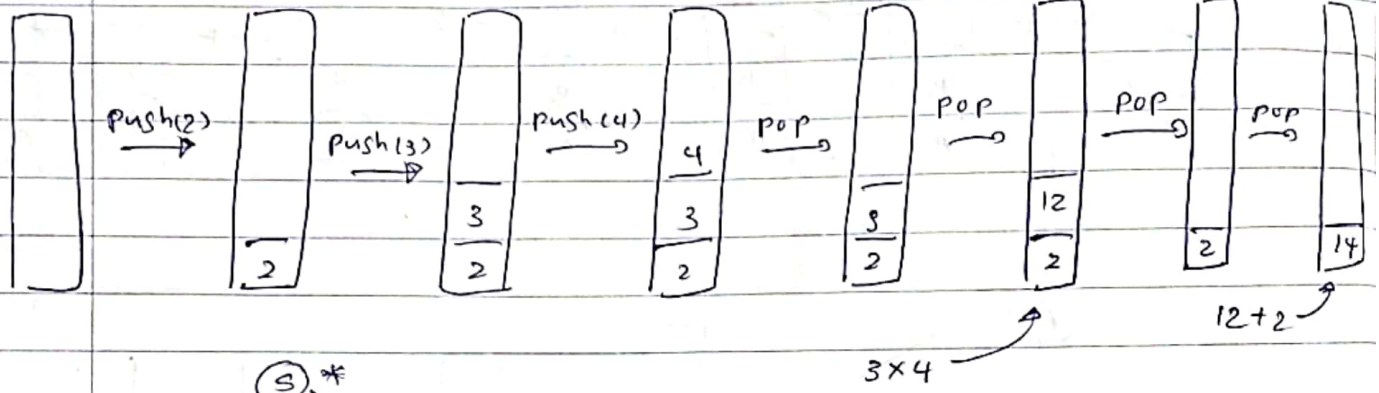


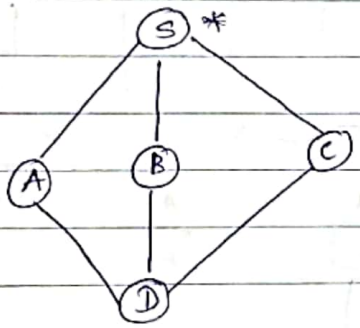
NO. Date: 25/10/2022

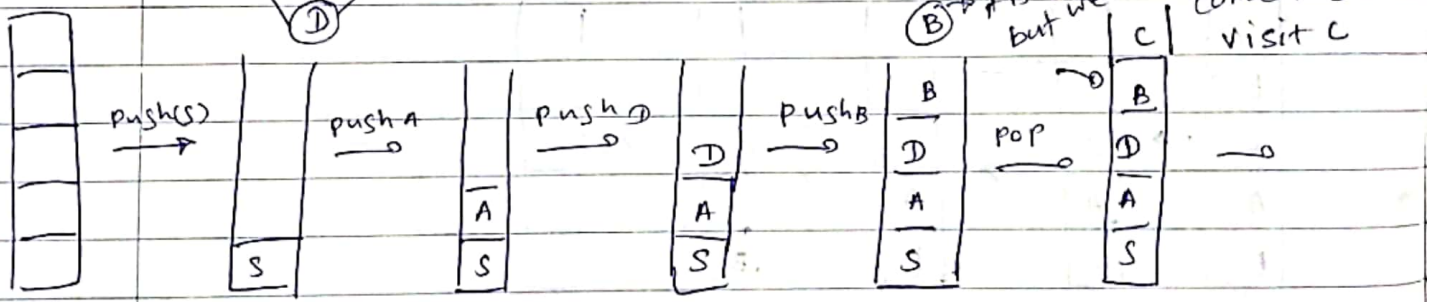
~~2 3 4 * +~~



2 3 4 * +







B → it is already dead end but we have to pop it and come to D and visit C

NO. Date: / /

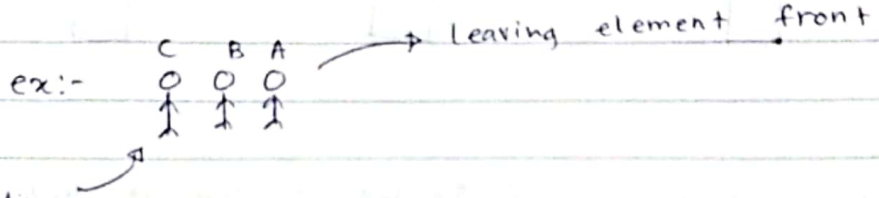
- Queue Data Structure -

> Stack have top for add and remove

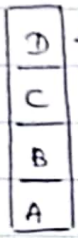
> Queue have two options

leave elements - front

add elements - back



In that case there B is moving to front it's not easly then we can change the acces point

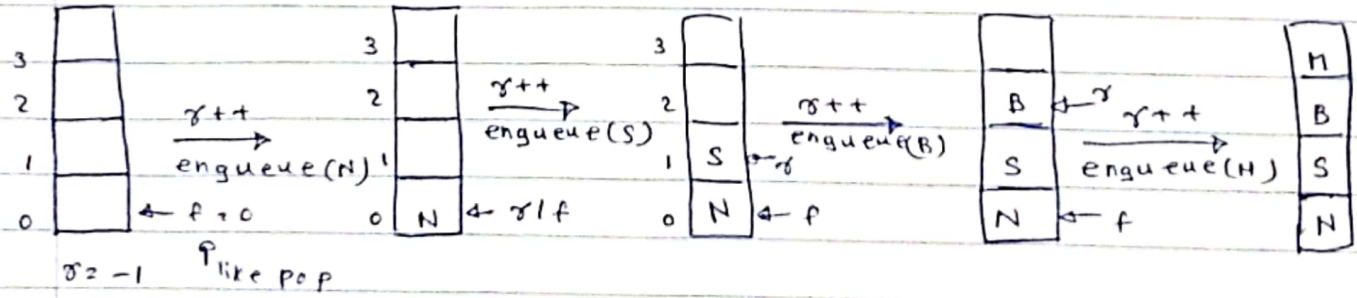


When we are adding element it will Change

if we remove the A, then the acces point is going to B

Enqueue (Insert)

NSBH

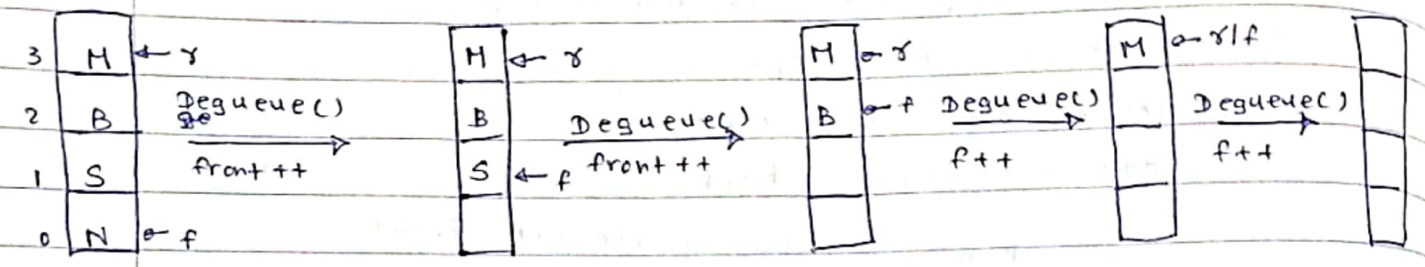


Same as top

because we are adding element ~~always~~ using rear

NO. Date: / /

Dequeue → remove the element at the front



Queue

> output order → NSBH
> Insert order → NSBH

} In Queue output and input same

FIFO

first in first out

> In stack, Input and output is not name

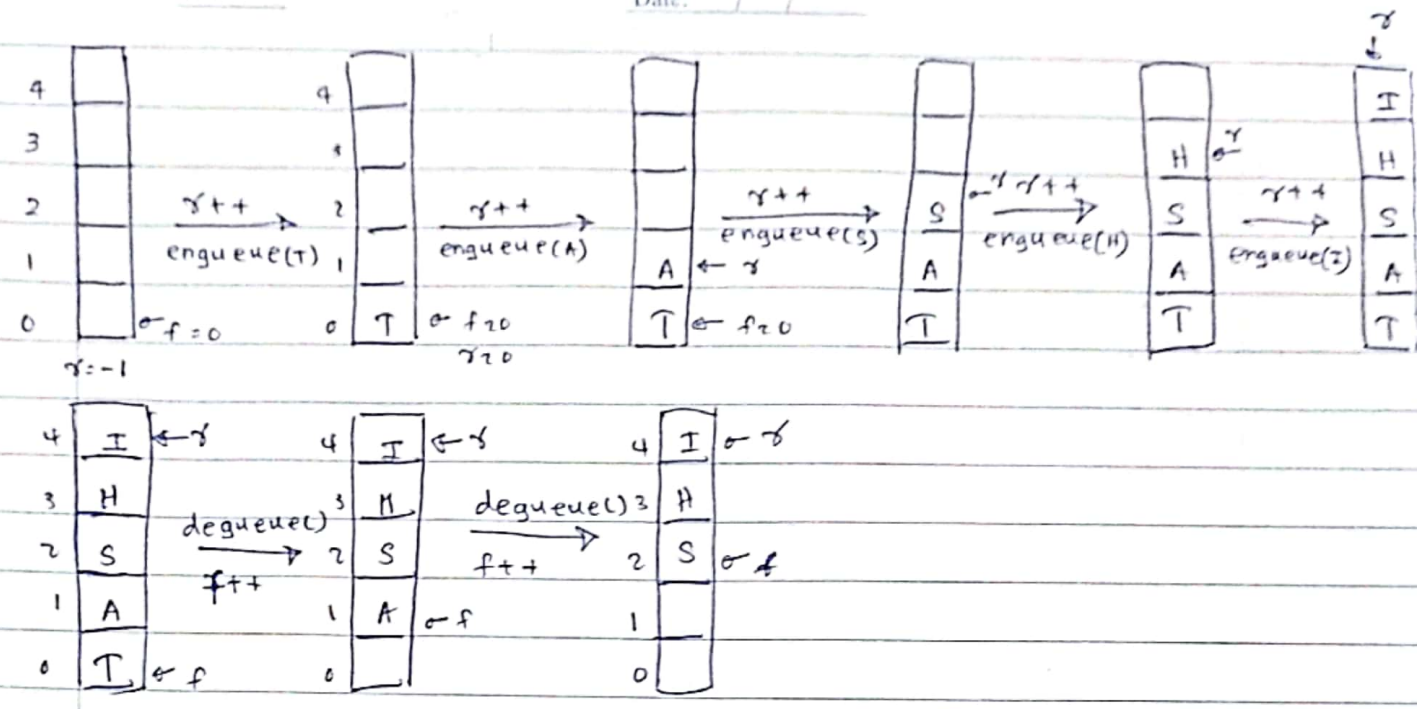
Stack

input → NSBH
output → HBSN

} LIFO

Tapping

NO. Date: / /



- write a code declare ~~two~~ the create a queue -

```
# define size
  struct Queue {
  int arr [size];
  int front;
  int rear;
  } q;
```

```
int front = 0
int rear = -1
```

int f, r;

Engueue