4/9/19

CH-3 Stack

* Application Of Stack *

- Reversing a list
- Parenthesis checker
- Conversion of infix Exp. into postfix Exp.
- Conversion of infix to prefix "
- Evaluation of postfix Exp.
- Evaluation of prefix "
- Recursion
- Tower of Hanoi problem

* notation

(i) Infix Notation a+b

(ii) Prefix Notation +ab (Polish Notation)

(iii) Postfix Notation ab+ (Reverse Polish Notation)

* Algorithm for unparenthesis (Infix to postfix)

UNPARENTHESIZED_SUFFIX [INFIX, S, NEXTCHAR, POLISH, NEXT, TEMP]

- Infix represents infix exp.
- S vector S represents stack S
- nextchar returns next character of i/p string
- Polish postfix of given infix.
- next contain the symbol been examined.
- temp is temporary variable which contains unstack element.

S.1 [Initialized the Stack]
TOP <- 1
S[TOP] <- '#'

S.2 [Initialized O/P string & rank count]
POLISH <- ' '
RANK <- 0

* Priority Table

Precedence table

Symbol	Precedence (f)	Rank (r)
+, -	1	-1
*, /	2	-1
a, b, c, ...	3	1
#	0	0

	Operator	Precedence	Associativity
1.	^, $, ↑ (Exponential)	highest	Right associa. (Right to left)
2.	*, /	Second highest	left "
3.	+, -	lowest	left "

S.3 [Get first I/P Symbol]
NEXT <- NEXTCHAR(INFIX)

S.4 [Translate infix Expn]
Repeat through S.6 while NEXT ≠ '#'

S.5 [Remove Symbol with greater Or Equal Precedence from Stack]

Repeat while f(NEXT) ≤ f(S[TOP])
TEMP ← POP(S, TOP)
POLISH ← POLISH ○ TEMP
↓
Concatenation

RANK ← RANK + r[TEMP]
if RANK < 1
then write ('Invalid')
Exit

S.6 [Push current Symbol onto Stack & obtain next i/p Symbol]

Call PUSH (S, TOP, NEXT)
NEXT ← NEXTCHAR(INFIX)

S.7 [Remove remaining Element from Stack]

Repeat while S[TOP] ≠ '#'
TEMP ← POP(S, TOP)
POLISH ← POLISH ○ TEMP
RANK ← RANK + r[TEMP]

if RANK <- 1
then write ('Invalid')
Exit

5.8 [Is the Expr. valid ?]

If RANK = 1
then write ('valid')
else write ('Invalid')
Exit

Ex: a + b * c #

	next	STACK	POLISH	RANK
		#		0
3 ≤ 0	a	#a		0
1 ≤ 3	+	#+	a	1
3 ≤ 1	b	#+b	a	1
2 ≤ 3	*	#+*	ab	2
3 ≤ 2	c	#+*c	ab	2
0 ≤ 3	#	#+*	abc	3
0 ≤ 2		#+	abc*	2
0 ≤ 1		#	abc*+	1

1 → valid

Ex: a + b * c - d / e * h #

	Next	Stack	Polish	Rank
		#		0
3 ≤ 0	a	#		0
1 ≤ 3	+	#a	a	1
3 ≤ 1	b	#+	ab	1
2 ≤ 3	*	#+b*	ab	2
3 ≤ 2	c	#+*c	abc	3
1 ≤ 2	-	#-	abc*+	1
3 ≤ 1	d	#-d	abc*+	1
2 ≤ 3	/	#-/	abc*+d	2
3 ≤ 2	e	#-/e	abc*+d	2
2 ≤ 3	*	#-/*	abc*+de	3
3 ≤ 2	h	#-/*h	abc*+de	3
0 ≤ 3	#	#-/*	abc*+deh	4
			abc*+deh*	3
			abc*+deh*/	2
			abc*+deh*/-	1
			abc*+deh*/-	1

abc*+de/h*-

Ex: A/B$C+D*E/F-G+H#

	next	Stack	Polish	Rank
		#		0
3≤0	A	#a	a	1
2≤3	/	#/	a	1
3≤2	B	#/B	ab	2
	$	#/$	ab	2
	C	#/$C	abc	2
1≤3	+	#+	abc$/	1
3≤1	D	#+D	abc$/	1
2≤3	*	#+*	abc$/D	2
3≤2	E	#+*E	abc$/D	2
2≤3	/	#+/	abc$/DE*	2
3≤2	F	#+/F	abc$/DE*	2
1≤3	-	#-	abc$/DE*F/+	1
3≤1	G	#-G	"	1
1≤3	+	#+	abc$/DE*F/+G-	1
3≤1	H	#+H	"	1
	#			

abc$/DE*F/+G-H+ 1

Ex: a+b*c

= a+bc*

= abc*+

Ex: a+b*c - d/e * h

= a+bc*-d/e*h

= a+bc*-de/*h

= a+bc*-de/h*

= abc*+de/h*-

Ex: A$B-c * D+E$F/G

= A$B-c*D+EF$/G

= AB$-c*D+EF$/G

= AB$-CD*+EF$/G

= AB$-CD*+EF$G/

= AB$CD* - EF$G+/

Ex: A+B^C^D-E*F/G

= A+B^CD^-E*F/G Postfix

= A+BCD^^-E*F/G

= A+BCD^^-EF*/G

= A+BCD^^-EF*G/

= ABCD^^+EF*G/-

Prefix

A+B^^CD-E*F/G

= A+^B^CD-E*F/G

= A+^B^CD-*EF/G

= A+^B^CD-/*EFG

= -+A^B^CD/*EFG

7/9/19

* Expression Evaluation

Algorithm to Evaluate postfix Expr.

S.1 Read the postfix Expression

S.2 Initialize Stack S.

S.3 i = 0

S.4 If P[i] is an Operand then push P[i] into stack & go to S.5 else if P[i] is an Operator then

Operand 1 & 2 { Opd2 = Pop();
Opd1 = pop();

perform the actual Operation on Operand 1 & 2 & push the result into stack.

S.5 Get the nextchar from postfix Expression i.e.

i++

S.6 If i < length of postfix Exp. then go to S.4 else pop the result from the stack & print the result.

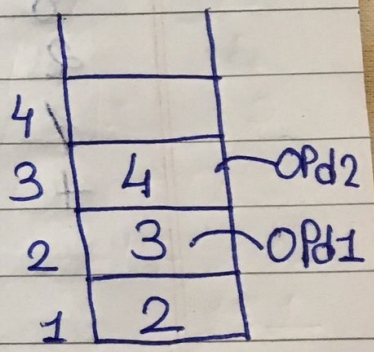
Ex: 234*+

P[i] = {2, 3, 4, *, +}

i = 0 1 2 3 4

Value = Opd1 * Opd2

P[0] = 2
P[1] = 3
P[2] = 4
P[3] = *
P[4] = +



Symbol	Opd1	Opd2	Value	Stack
2	–	–	–	2
3	–	–	–	2,3
4	–	–	–	2,3,4
*	3	4	12	2,12
+	2	12	14	14

Ans = 14

Ex: 62/102 - * 82/+

Symbol	Opd1	Opd2	value	Stack
6	-	-	-	6
2	-	-	-	6,2
/	6	2	3	3
10	-	-	-	3,10
2	-	-	-	3,10,2
-	10	2	8	3,8
*	3	8	24	24
8	-	-	-	24,8
2	-	-	-	24,8,2
/	8	2	4	24,4
+	24	4	28	28

Ans = 28

Ex: 9 3 4 * 8 + 4 / -

Symbol	Opd1	Opd2	Value	Stack
9	—	—	—	9
3	—	—	—	9,3
4	—	—	—	9,3,4
*	3	4	12	9,12
8	—	—	—	9,12,8
+	12	8	20	9,20
4	—	—	—	9,20,4
/	20	4	5	9,5
-	9	5	4	4

Ans = 4

Ex: 5 6 2 + * 1 2 4 / - +

Symbol	opd1	opd2	value	Stack
5	-	-	-	5
6	-	-	-	5 6
2	-	-	-	5 6 2
+	6	2	8	5, 8
*	5	8	40	40
1	-	-	-	40, 1
2	-	-	-	40, 1, 2
4	-	-	-	40, 1, 2, 4
/	2	4	0.5	40, 1, 0.5
-	1	0.5	0.5	40, 0.5
+	40	0.5	40.5	40.5

Ans = 40.5

Ex: 4 8 * 12 12 8 4 * + − +

Symbol	opd1	opd2	value	Stack
4				4
8				4,8
12				4,8,12
12				4,8,12,12
8				4,8,12,12,8
4				4,8,12,12,8,4
*	8	4	32	4,8,12,12,32
+	12	32	44	4,8,12,44
−				

4	−	−	−	4
8	−	−	−	4,8
*	4	8	32	32
12	−	−	−	32,12
12	−	−	−	32,12,12
8	−	−	−	32,12,12,8
4	−	−	−	32,12,12,8,4
*	8	4	32	32,12,12,32
+	12	32	44	32,12,44
−	12	44	-32	32,-32
+	32	-32	0	0

Ans = 0

Ex: AB * CD$ - EF / G / +

A=5, B=2, C=3, D=2, E=8, F=2, G=2

Symbol	OPD1	Opd2	value	Stack
5	–	–	–	5
2	–	–	–	5, 2
*	5	2	10	10
3	–	–	–	10, 3
2	–	–	–	10, 3, 2
$	3	2	9	10, 9
-	10	9	1	1
8	–	–	–	1, 8
2	–	–	–	1, 8, 2
/	8	2	4	1, 4
2	–	–	–	1, 4, 2
/	4	2	2	1, 2
+	1	2	3	3

Ans = 3

Ex: A + B - C * D / E + F $ G / (I + J)

A = 1, B = 2, C = 3, D = 4, E = 6, F = 6, G = 1, I = 3, J = 3

= AB+ CD* E/- FG $IJ+ /+

= 12+ 34* 6/- 61$ 33+ /+

Symbol	Opd1	Opd2	Value	Stack
1	-	-	-	1
2	-	-	-	1, 2
+	1	2	3	3
3	-	-	-	3, 3
4	-	-	-	3, 3, 4
*	3	4	12	3, 12
6	-	-	-	3, 12, 6
/	12	6	2	3, 2
-	3	2	1	1
6	-	-	-	1, 6
1	-	-	-	1, 6, 1
$	6	1	6	1, 6
3	-	-	-	1, 6, 3
3	-	-	-	1, 6, 3, 3
+	3	3	6	1, 6, 6
/	6	6	1	1, 1
+	1	1	2	2

Ans = 2

Ex: 2$3+5*2$2-6/6

23$ + 5x22$ - 6/6

= 23$ + 522$ * - 66/

= 23$ 522$ * + 66/ -

Symbol	Opd1	Opd2	value	Stack
2	-	-	-	2
3	-	-	-	2,3
$	2	3	8	8
5	-	-	-	8,5
2	-	-	-	8,5,2
2	-	-	-	8,5,2,2
$	2	2	4	8,5,4
*	5	4	20	8,20
+	8	20	28	28
6	-	-	-	28,6
6	-	-	-	28,6,6
/	6	6	1	28,1
=	28	1	27	27

Ans = 27

Ex: 2$3+5*2$2-12$2

= 2$3+5*2$2-122$

= 2$3+5*22$-122$

= 23$+5*22$-122$

= 23$+522$*-122$

= 23$522$*+122$-

Symbol	Opd1	opd2	Value	Stack
2	-	-	-	2
3	-	-	-	2,3
$	2	3	8	8
5	-	-	-	8,5
2	-	-	-	8,5,2
2	-	-	-	8,5,2,2
$	2	2	4	8,5,4
*	5	4	20	8,20
+	8	20	28	28
12	-	-	-	28,12
2	-	-	-	28,12,2
$	12	2	144	28,144
-	28	144	116	-116

It is Prefix Exp
∴ we have to start from last.

Ex: + * 4 8 - 12 + 12 * 8 4

Here, top value will be in Opd 1 & another in opd 2

Symbol	Opd 1	Opd 2	value	Stack
4	-	-	-	4
8	-	-	-	4, 8
*	8	4	32	32
12	-	-	-	32, 12
+	12	32	44	44
12	-	+	-	44, 12
-	12	44	-32	-32
8	-	-	-	-32, 8
4	-	-	-	-32, 8, 4
*	4	8	32	-32, 32
+	+32	-32	0	0

Ans = 0

Ex: $+*AB-C+C*BA$

$= +(AB*) - C+C(BA*)$

$= +(AB*) - C(CBA*+)$

$= +(AB*)CCBA*+-$

$= AB*CCBA*+-+$

Ex: $ABC*+DE/F*-$

$= A*BC+DE/F*-$

$= A*BC+/DEF*-$

$= A(*BC)+(*/DEF)-$

$= +(A*BC)(*/DEF)-$

$= -+A*BC*/DEF$

* Algorithm for converting infix notation to postfix notation with paranthize

Symbol	Input Precedence (f)	Stack Precedence (g)	Rank function
+, −	1	2	−1
*, /	3	4	−1
↑	6	5	−1
abc (var.)	7	8	1
(	9	0	—
)	0	—	—

REVPOL

5.1 (Initialize Stack)
TOP ← 1
S[TOP] ← 'C'

5.2 (Initialize Output String & Rank count)
POLISH ← ''
RANK ← 0

5.3 [Get first input Symbol]
NEXT ← NextChar (INFIX)

5.4 [Translate the infix Exp.]
Repeat thru 5.7 while NEXT ≠ ''

5.5 [Remove Symbol with greater precedence from Stack]
if TOP < 1
then
write ('Invalid')
Exit

Repeate while f(Next) < g(S[TOP])
TEMP ← POP(S, TOP)
POLISH ← POLISH ○ TEMP
RANK ← RANK + r(TEMP)
if RANK < 1
then
write ("Invalid")
Exit

5.6 [Are there matching parenthize]

if f(Next) ≠ g(S[TOP])
then
    Call push(S, TOP, Next)
else
    POP (S, TOP)

S.7 [Get next input symbol]

Next ← NextChar(Infix)

S.8 [Is the Exp. valid?]

If TOP≠0 or Rank≠1
then
    Write('Invalid')
else
    Write('Valid')
Exit

11/9/19

Ex: (a+b↑c↑d)*(e+f/d))

Next	Stack	Polish	rank
	(		0
(	((		0
a	((a		0
+	((+	a	1
b	((+b	a	1
↑	((+↑	ab	2
c	((+↑c	ab	2
↑	((+↑↑	abc	3
d	((+↑↑d	abc	3
)	(	abcd↑↑+	1
*	(*	abcd↑↑+	1
(	(*(	abcd↑↑+	1
e	(*(e	abcd↑↑+	1
+	(*(+	abcd↑↑+e	2
f	(*(+f	abcd↑↑+e	2
/	(*(+/	abcd↑↑+ef	3
d	(*(+/d	abcd↑↑+ef	3
)	(*	abcd↑↑+efd/+	2
)		abcd↑↑+efd/+*	1

Ex: (A+B)* C+D/(CB+A*C)+D)

Nesct	Stack	Polish	Rank
	(	''	0
C	((	''	0
A	((A	''	0
+	((+	A	1
B	((+B	A	1
)	(	AB+	1
*	(*	AB+	1
C	(*C	AB+	1
+	(+	AB+C*	1
D	(+D	AB+C*	1
/	(+/	AB+C*D	2
C	(+/C	''	2
B	(+/CB	''	2
+	(+/(+	AB+C*DB	3
A	(+/(+A	''	3
*	(+/(+*	AB+C*DBA	4
C	(+/(+*C	''	4
)	(+/	AB+C*DBAC*+	3
+	(+	AB+C*DBAC*+/+	1
D	(+D	''	1
)		AB+C*DBAC*+/+D+	1

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

Next	Stack	Polish	Rank
	(	''	0
A	(A	''	0
+	(+	A	1
(	(+(	A	1
B	(+(B	A	1
*	(+(*	AB	2
C	(+(*C	AB	2
-	(+(-	ABC*	2
(	(+(-(	''	2
D	(+(-(D	''	2
/	(+(-(/	ABC*D	3
E	(+(-(/E	''	3
^	(+(-(/^	ABC*DE	4
F	(+(-(/^F	''	4
)	(+(-	ABC*DEF^/	3
*	(+(-*	''	3
G	(+(-*G	''	3
)	(+	ABC*DEF^/G*-	2
)		ABC*DEF^/G*-+	1

Ex:- ((A-(B+C))*D)/(E+F))

Next	Stack	Polish	Rank
	(		0
(	((		0
(	A((		0
A	(((A		0
-	(((-		1
(	(((-(		1
B	(((-(B		1
+	(((-(+		2
C	(((-(+C		"
)	(((-	ABC+	2
)	((	ABC+-	1
*	((*		"
D	((*D		"
)	(	ABC+-D*	"
/	(/		"
(	(/(		"
E	(/(E		"
+	(/(+	ABC+-D*E	2
F	(/(+F		2
)	(/	ABC+-D*EF+	2
)		ABC+-D*EF+/	1

[Prefix using Tabular Method]

[In this condition will be F(next) < F(S[TOP])]

Ex: A+B-C*D↑F↑G

↓

G↑F↑D*C-B+A#

Next	Stack	Polish	Rank
	#		0
3≤0 G	#G		0
↑	#↑	G	1
F	#↑F	G	1
↑	#↑	GF↑	1
D	#↑D	"	"
*	#*	GF↑D↑	1
C	#*C	"	"
-	#-	GF↑D↑C*	1
B	#-B	"	"
+	#+	GF↑D↑C*B-	1
A	#+A	GF↑D↑C*B-A+	1
#			
		-+A-B*C↑D↑FG	

(Polish to reverse Polish)
(Prefix to Postfix)

Ex: ++abc

= +(a+b)c (Choose operator which has two following operand)

= a+b+c = ab+c+

Ex: +a+bc

= +abc+

= abc++

Ex: +a*bc

= +a(bc*)

= abc*+

Ex: *a+bc

= *ab+c

= ab(c+)*

* Push, Pop and Display

```
#include <stdio.h>
#include <conio.h>
#define MAX 5
int top = -1, stack[MAX];
void push();
void pop();
void display();
void main()
{
    int ch;
    while(1)
    {
        printf("\n 1.Push \n 2.Pop \n 3.Dis
                \n 4.Exit");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
```

```
        case 3 : display();
                 break;
        case 4 : exit(0);
        default : printf("\n invalid");
    }
  }
}
void push()
{
    int val;
    if (top == MAX-1)
    {
        printf("overflow");
    }
    else
    {
        scanf("%d", &val);
        top = top+1;
        stack[top] = val;
    }
}
```

```
void pop()
{
    if (top == -1)
    {
        Printf("\n underflow");
    }
    else
    {
        top = top - 1;
    }
}
void display()
{
    int i;
    if (top == -1)
    {
        Printf("\n empty");
    }
    else
    {
        for(i = top; i >= 0; --i)
        {
            Printf("%d \n", Stack[i]);
        }
    }
}
```

13/9/19

* Recursion :

=> Function calling itself again & again.

```
fact = 1 ;

for (i = 1; i <= n , i++)
{
        fact = fact * i ;
        fact++ ;
}
```

Factorial (using recursion)

```
fact(n)
{
    if (n == 1)
    {
        return 1 ;
    }
    else
    {
        return fact(n-1) * n ;
    }
}
```

* Fibonacci

```
fibo (n)
{
    if (n < 2)
    {
        return n;
    }
    else
    {
        return fibo(n-1) + fibo(n-2);
    }
}
```

* Tower Of Hanoi :[TOH]

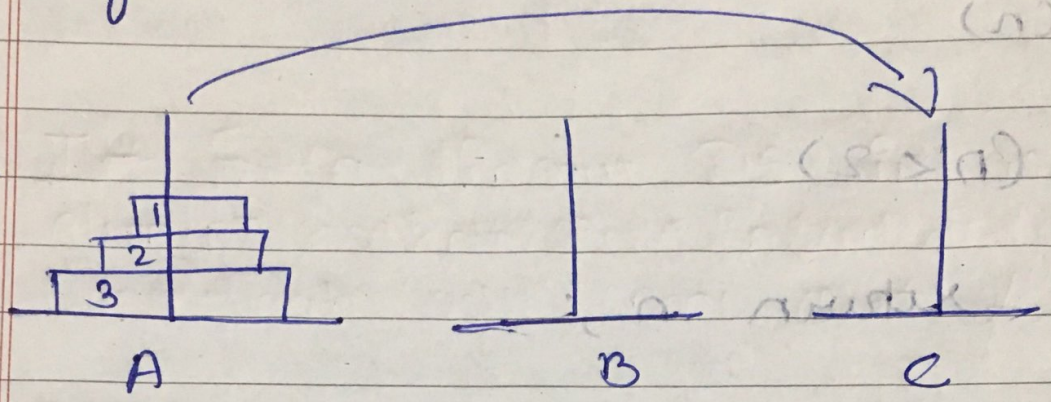
* Rules

-> Only 1 disk can be moved at a time.

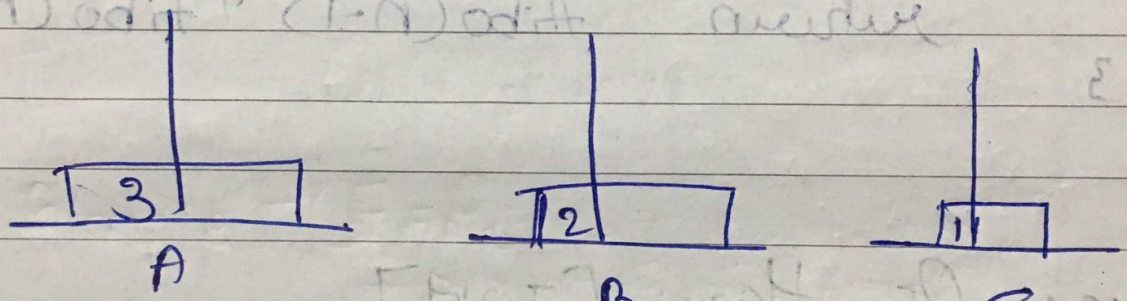
-> Each move consist of taking upper disk from one of the stack. & place it on top of the another stack.

→ No disk may be placed on top of smaller disk

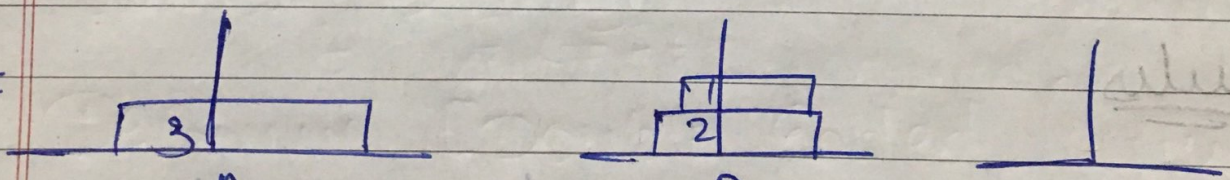
S.1



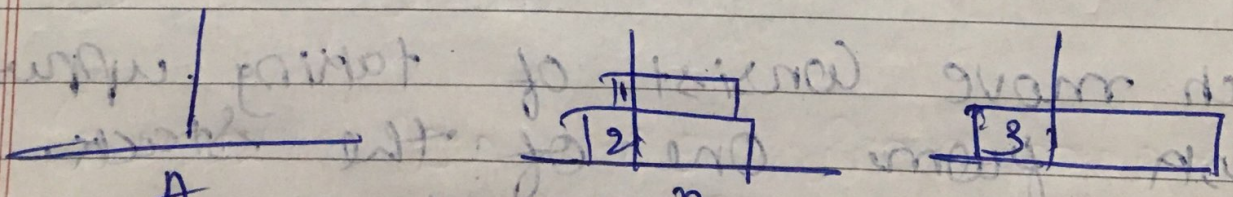
S.2



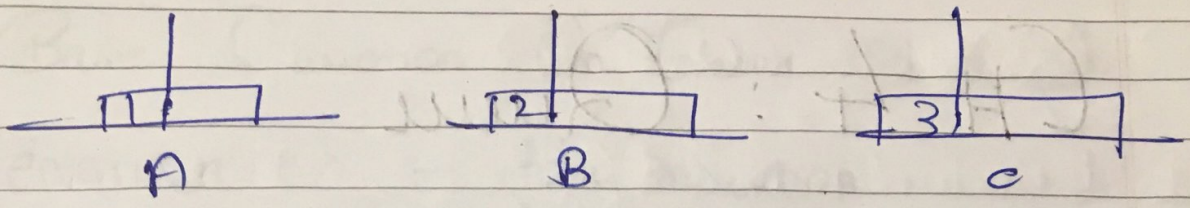
S.3



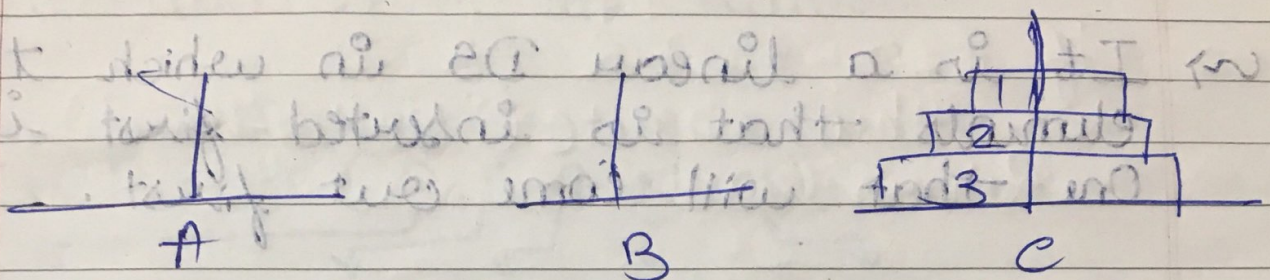
S.4



S.5



S.6



Algorithm TOH (n, A, C, B)
↳ transfer disks from A to C using B

```
{
    if (n=1) then
    {
        write (" Peg moved from A to C").
        return
    }
    else
    {
        TOH (n-1, A, B, C) // move n-1 from A to B using C
        TOH (n-1, B, C, A) // move n-1 from B to C, using A
    }
```