Data Structure

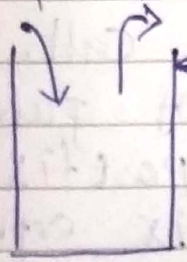
BTECCE 22302

Unit - 1 Stacks & Queues

Stacks :- Fundamentals of Stack.
- Representation & Implementation of Stack using Arrays.
- Application of stack:
  - Decimal to Binary
  - reversing a string
  - parsing : well-form parenthesis.
- Different expression conversion and evaluation.

* Stack : (LIFO) (FILO).
Linear Data Structure.

Rule:
(PUSh) Insertion } perform only
(Pop) Deletion } one end.

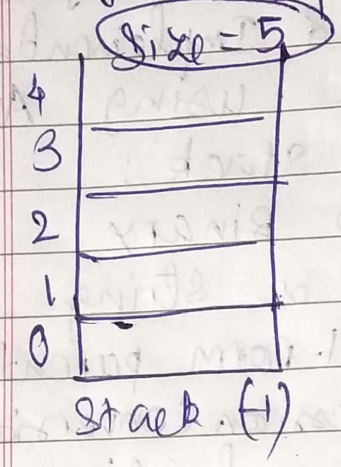


→ Operations on stack:

- push(x)
- pop()
- peek() / top()
- isEmpty()
- isFull()

- Overflow / underflow condition

- Logical representation of Stack




```
push(2)
  top++;
push(3)
  top++;
pop()
  top--;
```

if top → (max - 1) ⇒ overflow.

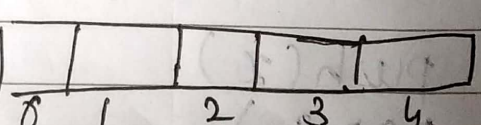
⇒ Application of ~~String~~ Stack

1. reverse a string.
2. Undo (ctrl+z)
3. Recursion / function call.
4. to check balance of parenthesis.
5. Infix to prefix / postfix
6. Evaluation of postfix expression.

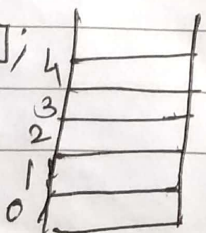
* Implementation of Stack using Array

Static

```
int a[5];
```




```
int stack[5];
```



```
#define N 5;
int stack [N];
int top = -1;

void push()
{
   int x;
   printf("Enter data");
   Scanf("%d", &x);
   if( top = N-1)
      printf(" overflow");
   else
      {
         top++;
         stack [top] = x;
      }
}
void pop()
{
   int item;
   if (top <= -1)
      printf ("underflow");
   else
   {  item = stack [top];
      top--;
      printf(" poped item is %d",
                        item);
   }
```

```
void peek()
{
    if( top == -1)
        printf(" stack is empty");
    else
    {
        printf(" top is %d ",
                stack[top]);
}   }
void display()
{
    int i;
    for(i= top ; i>=0 ; i--)
    {
        printf(" %d ", stack[i]);
    }
}
```

time complexity O(1)

(polish) (Reverse polish)

* Infix / prefix / postfix

precedance.

1	( ) { } [ ]	
2	^	R - L
3	* /	L - R
4	+ -	L - R

(5+1)*6 (Infix)

+51 (prefix)

51+ (postfix)

* Infix to postfix Conversion.

Example:

A - B / C * D + E

Stack	Postfix Exp.
	A
-	A B
- /	A B
- /	A B C
-	A B C /
- *	A B C /
- *	A B C / D
-	A B C / D *
-	A B C / D * -
+	A B C / D * -
+	A B C / D * - E
	A B C / D * - E +

- input element (operator) is having higher priority than the stack [Top] = push element on stack.
- input ope^r → lower priority → stack [Top] = pop → stack [Top]
- input ope^n → equal priority → stack [Top] = pop → stack [Top] (check assosiativity)
  - if L-R → pop
  - if R-L → push.

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

Example:

K+L-M*N+(O^P)*W/U/V
*.T+Q

Input exp.	stack	postfix exp.
K		K
+	+	K
L	+	KL
-	-	KL+
M	-	KL+M
*	-*	KL+M
N	-*	KL+MN
+	+	KL+MN*-
(	+(	KL+MN*-
O	+(	KL+MN*-O
^	+(^	KL+MN*-O
P	+(^	KL+MN*-OP
)	+	KL+MN*-OP^
*	+*	KL+MN*-OP^
W	+*	KL+MN*-OP^W
/	+/	KL+MN*-OP^W*
U	+/	KL+MN*-OP^W*U
/	+/	KL+MN*-OP^W*U/
V	+/	KL+MN*-OP^W*U/V
*	+*	KL+MN*-OP^W*U/V*
T	+*	KL+MN*-OP^W*U/V/T
+	+	KL+MN*-OP^W*U/V/T*+
Q	+	KL+MN*-OP^W*U/V/T*+Q

KL+MN*-OP^W*U/V/T*+Q

Infix to prefix using Stack.

Example:

K + L - M * N + (O ^ P) * W / U / V * T + Q

Reverse: Q + T * V / U / W *) P ^ O (+ N * M - L + K

Input Exp:	Stack	Prefix Exp.
Q		Q
+	+	Q
T	+	QT
*	+*	QT
V	+*	QTV
/	+*/	QTV
U	+*/	QTVU
/	+*//	QTVU
W	+*//	QTVUW
*	+*//*	QTVUW
)	+*//*)	QTVUW
P	+*//*)	QTVUWP
^	+*//*)^	QTVUWP
O	+*//*)^	QTVUWPO
(	+*//*	QTVUWPO^
+	++	QTVUWPO^*//*
N	++	QTVUWPO^*//*N
*	++*	QTVUWPO^*//*N
M	++*	QTVUWPO^*//*NM
-	++-	QTVUWPO^*//*NM*
L	++-	QTVUWPO^*//*NM*L
+	++-+	QTVUWPO^*//*NM*L
K		QTVUWPO^*//*NM*LK
		QTVUWPO^*//*NM*LK+-++

Rules: reverse.

Push

I/p (higher pre) →

I/p (lower pre) → pop

I/p (equal) → push

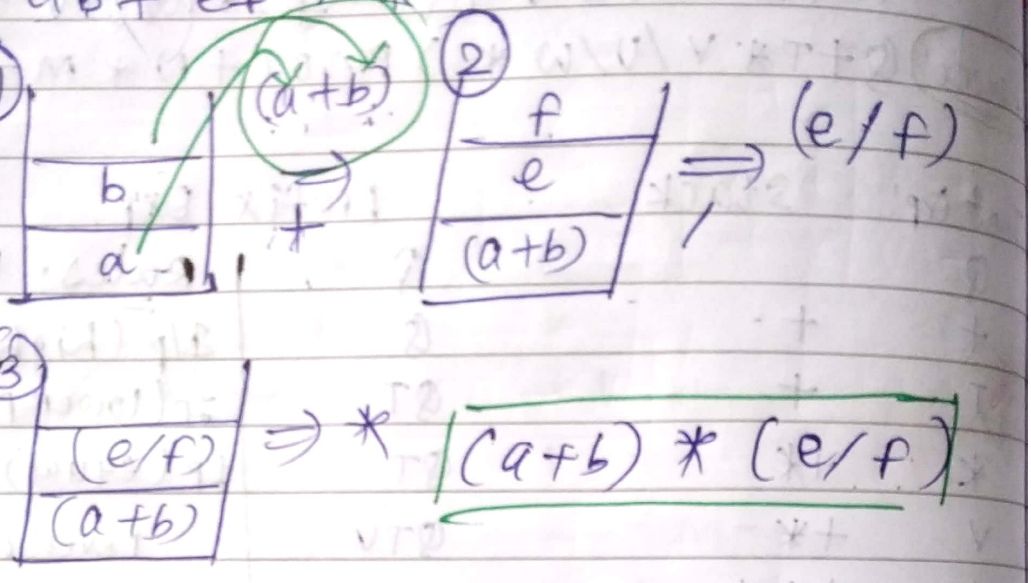
reverse at the end

⇓

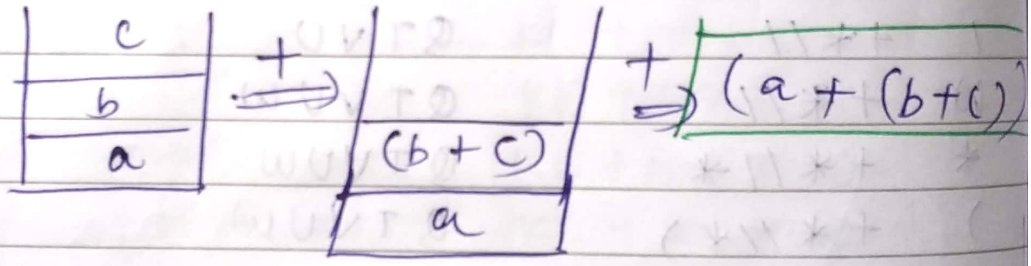
++-+KL*N*//*^OPWVUTQ

Postfix to Infix:

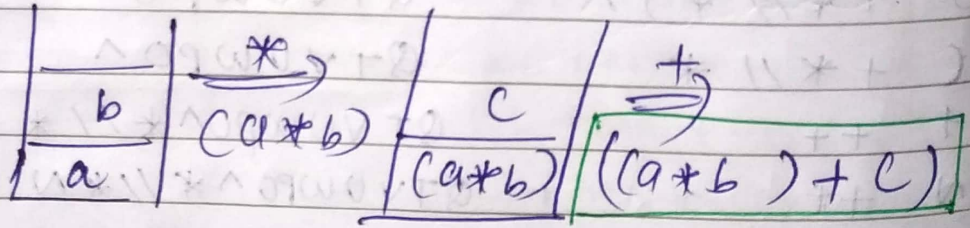
Example: ab+ef/*



Example: abc++



Example: ab*c+

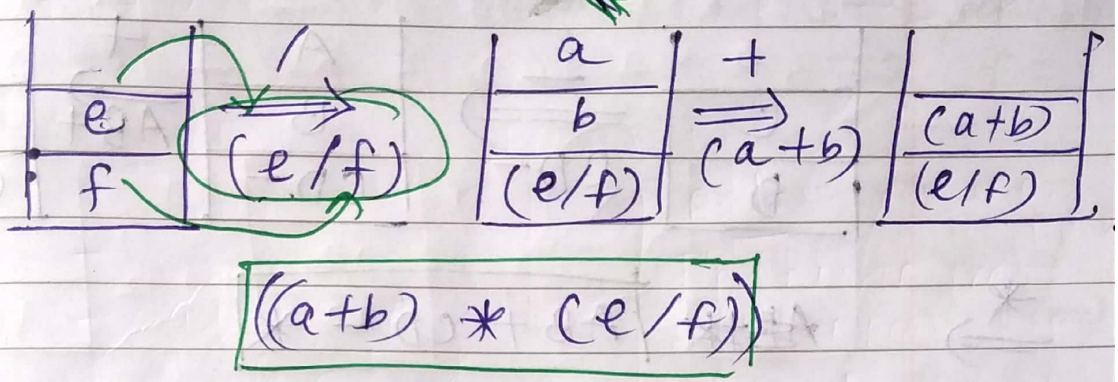


Example: AB+C*DE-FG+*-

Ans: [((A+B)*C))-((D-E)*(F+G))]

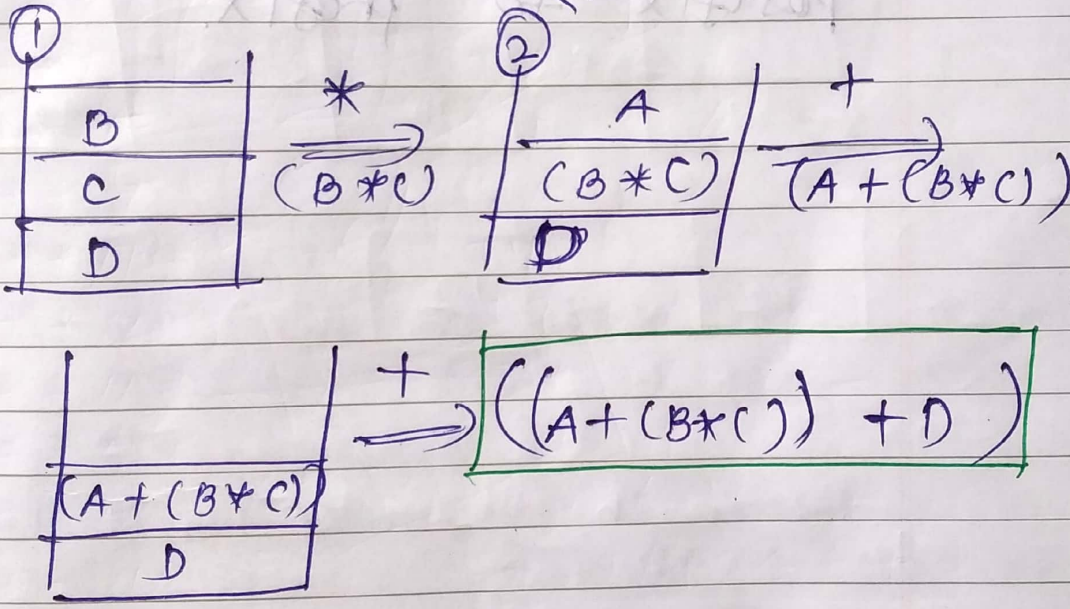
prefix to Infix.

Example: * + a b / e f



((a+b) * (e/f))

Example: + + A * B C D

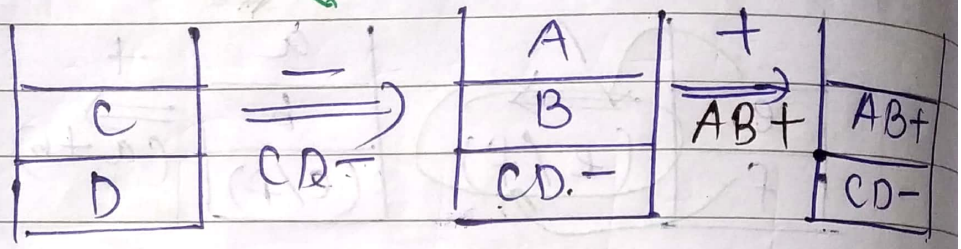


Example: - * + A BC * - DE + FG

Ans: (((A+B) * C) - ((D-E) * (F+G)))

Prefix to postfix Using Stack

* + A B - C D



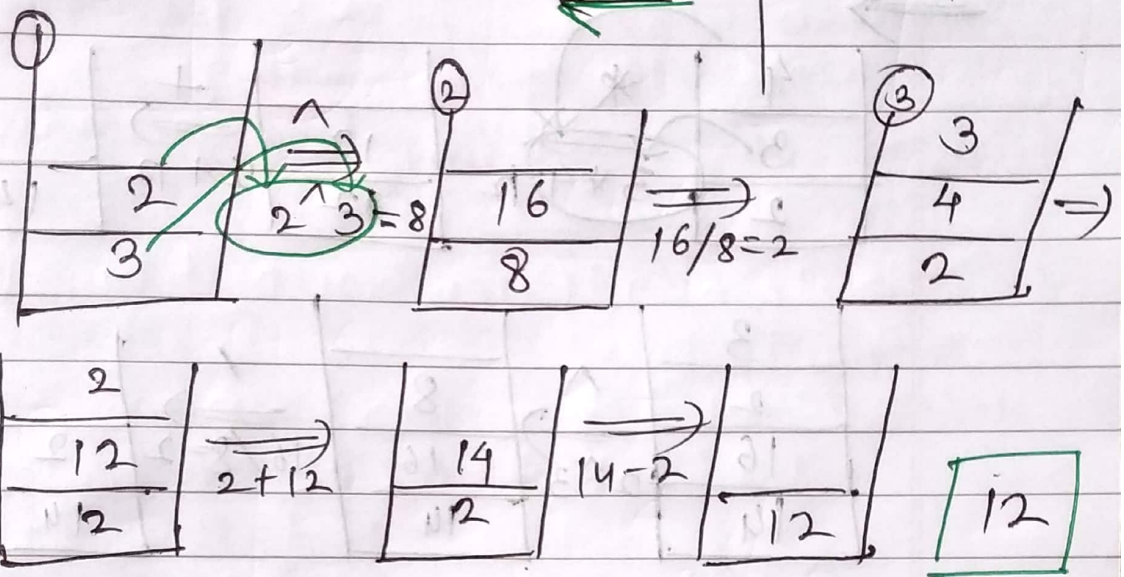
* ⟹ ~~AB+*~~ AB + CD - *

Postfix to prefix.

Evaluation of prefix Expression

Infix: a+b*c-d/e^f
prefix: -+a*bc/d^ef
⇒ -+2*34/16^23

a=2, b=3
c=4, d=16
e=2, f=3



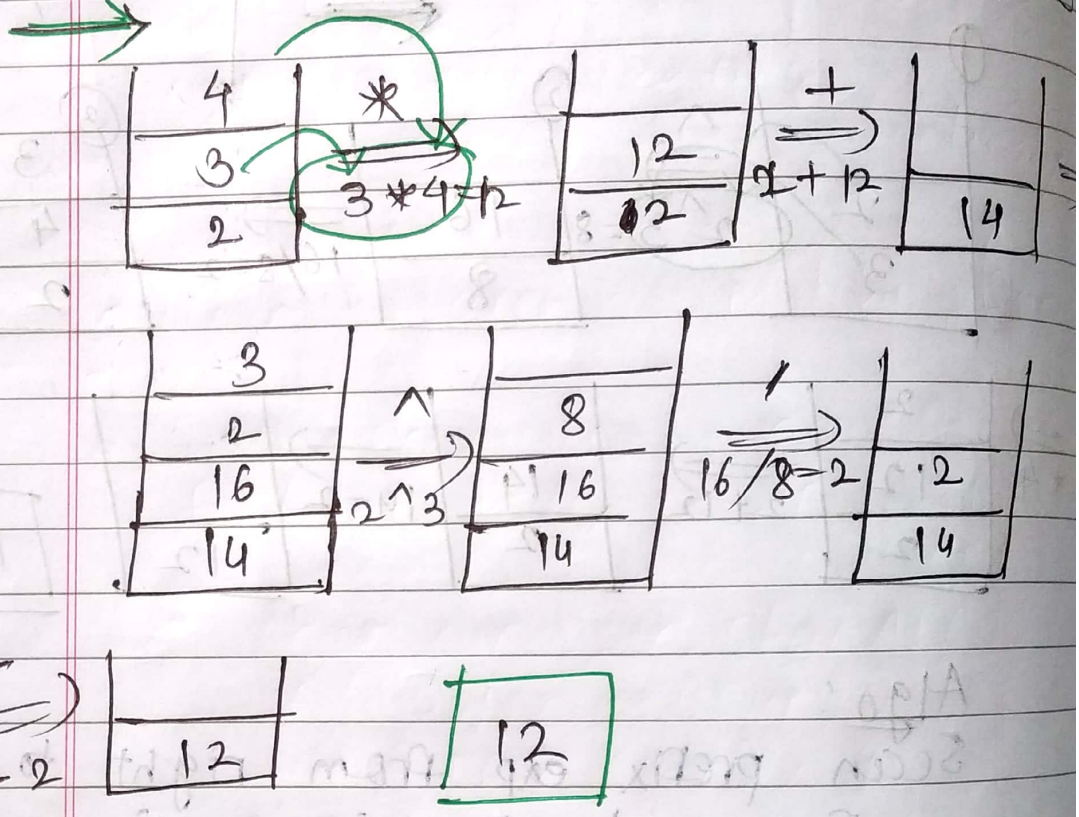
Algo:

Scan prefix exp. from right to left
for each char in prefix expr.
do
if ~~operator~~ operand is there,
push it onto stack
else if operator is there,
pop 2 elements
op1 = top element
op2 = next to top
result = op1 operator op2
push result onto stack.
return stack [top]

Evaluation of postfix Exp.

Infix : a+b*c-d/e^f
postfix : abc*+def^/-
2 3 4 *.+ 16 2 3 ^./-

a=2, b=3
c=4, d=16
e=2, f=3



Algo:

for each character in postfix exp
do
if operand is there,
push it onto stack
else if operator is there
pop 2 elements
A → top element
B → Next to top
result = B operator A
push result onto stack
return Stack[top]

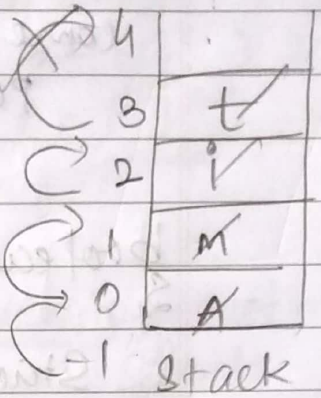
* Reverse a String using Stack.

```
#include <string.h>
Char stack [20];
int top = -1;

void push (char);
char pop();

Main()
{
   char str[20], ch;
   int l, i;
   printf("Enter string");
   gets (str);
   l = strlen(str);
   for (i = 0; i<l; i++)
      push (str[i]);
   printf ("reversed string: ");
   for (i = 0; i<l; i++)
   {
      ch = pop();
      printf("%c", ch);
   }
}
void push (char c)
{   top++;
    stack [top] = c;
}
char pop()
{ char c; c=stack[top]; top--;
return c;}
```

str

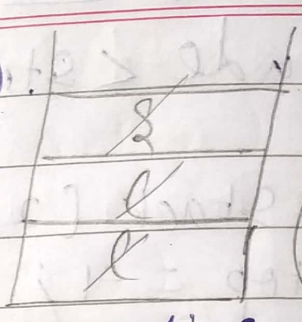
A	m	i	t	.
0	1	2	3	4

tiMA



* Balanced parenthesis using stack

s = (({ })).



push opening.
when closing
come to corresponding
pop opening brq. accordingly.

```
boolean isparenthesismatching (String s
{
    Stack <character> s = new Stack<>();
    for( int i=0; i< str.length();
                                i++)
    { char cur = str.charAt();
      if (isopening(cur))
        { s.push(cur); }
      else {
        if (s.isEmpty()) {
            return false;
        }else if (!isMatching (s.peek(), cur))
        { return false };
        else {
            s.pop();
    }}}
    return s.isEmpty();
boolean isopening (char c)
{   return c = '(' ||
             c = '{' ||
             c = '[';
}
```

```
boolean isMatching (char a, char b)
{
    return (a='(' && b=')') ||
           (a='{' && b='}') ||
           (a='[' && b=']');
}
```

* Decimal to Binary conversion using Stack.

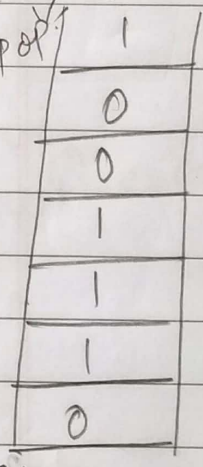
```
#define MAX 100;
int stack[MAX];
int top = -1;

int main()
{
    int dec;
    printf("Enter Decimal Number");
    scanf("%d", &dec);
    dectobin(dec);
    print();
    return 0;
}
```

Example:

2	78	0
2	39	1
2	19	1
2	9	1
2	4	0
2	2	0
	1	

100111 0

pop



Stack

Algo:
1. Divide the Number by 2.
2. Store the remainder inside stack.
3. Repeat step 1 & 2 until quotient becomes 0.
4. pop all the elements out of stack & print.

```
- void dectobin(int n)
{
    while (n != 0)
    { push (n % 2);
      n = n / 2;
    }
}
- void print()
{
    if (isEmpty())
    { printf("Stack underflow");
      exit(1);
    }
    while (!isEmpty())
    {
      print ("%d", pop());
    }
}
- int isFull()
- int isEmpty()
- void push(int data)
{
  if(isFull())
  { printf("Overflow");
    exit(1);
  }
  top++;
  Stack[top] = data;
}
void pop()
{ int val;
  if (isEmpty()
  { printf("underflow
  exit(1);
  val = Stack[top];
  top--;
  return val;
}
```