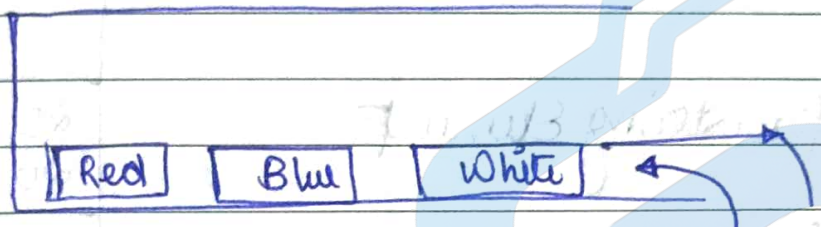
Section-13

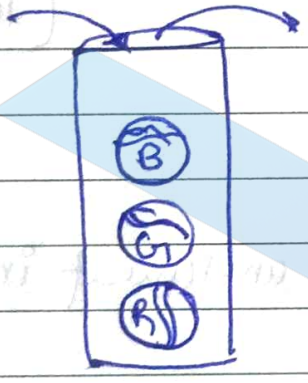
Stack

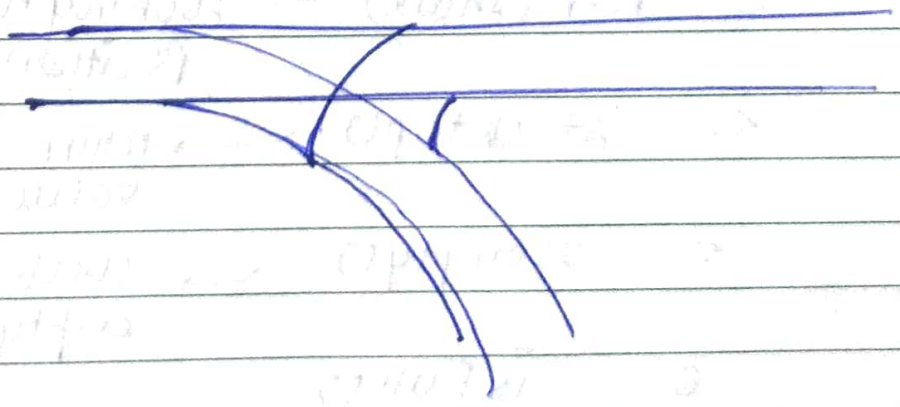
Introduction to stack :-

- Introduction

LIFO → Last in First Out







~~Stack~~ Stack is a collection of element which follows discipline
LIFO.
↳ array
↳ linked list

Recursion → use stack

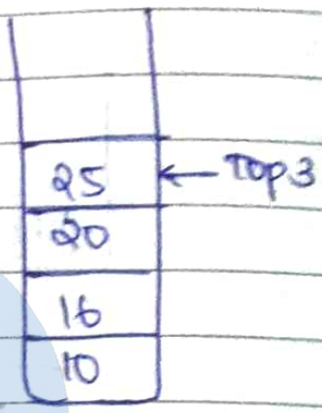
↓

Iterative (Sometime use stack)

~~Abstract~~ ADT stack:-

Data:-

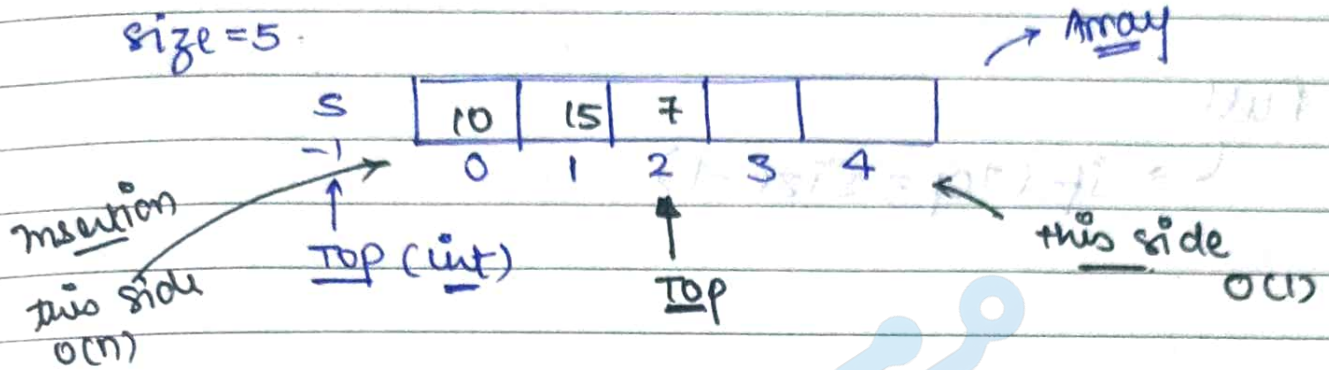
- Space for storing Element
- Top pointer.

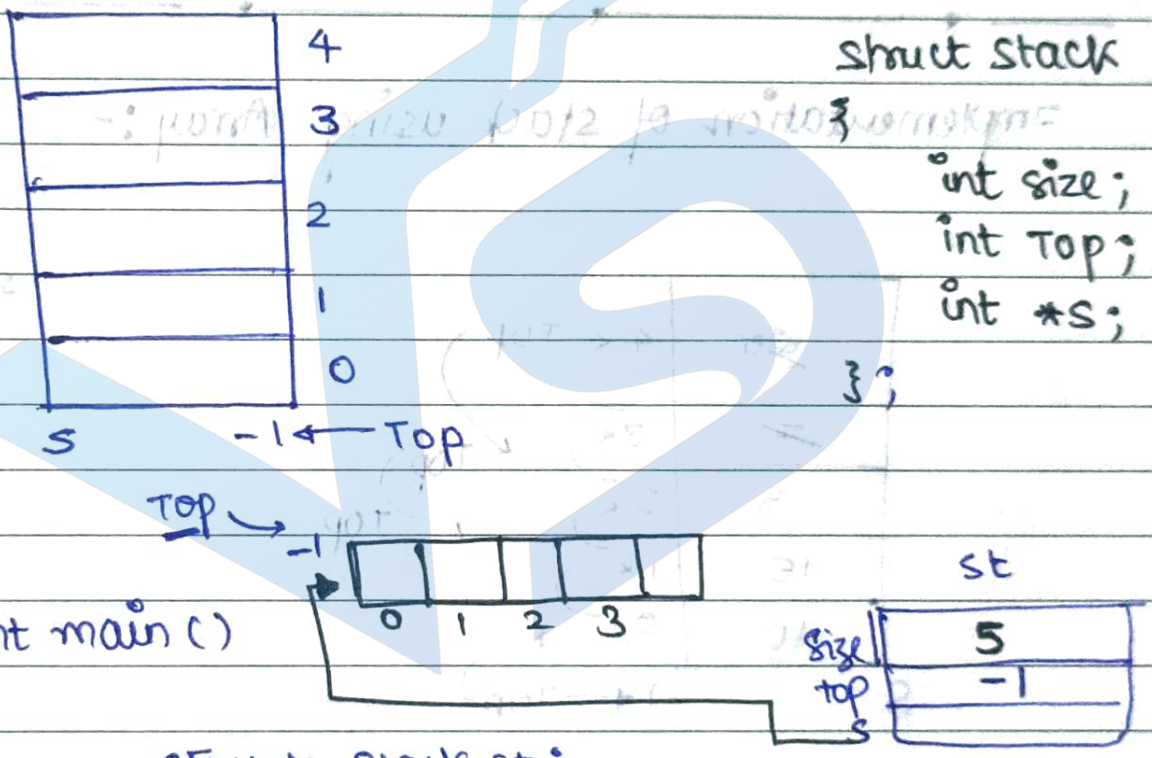


Operations

1. push(x) → Pushing an element into stack
2. pop(x) → Popping
3. Peek(Index) → Looking the value at given position
4. stacktop(): → return top of the stack value
5. IsEmpty() → check whether stack is empty or not.
6. isFull() → check whether stack is full or not.

Stack using Array :-






```
struct stack st;
cout<<"Enter the size of stack";
cin>>st.size;
st.s = new int[st.size];
st.Top = -1;
}
```

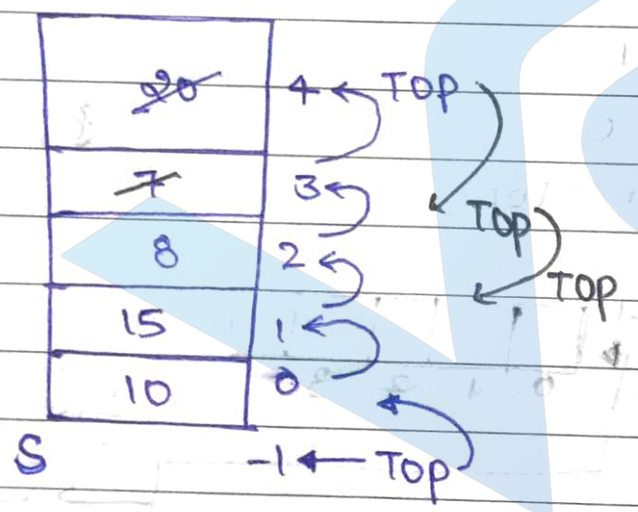
Empty

↳ ~~Top~~ → (Top == -1)

Full

↳ if (Top = size - 1)

Implementation of stack using Array :-




```
struct stack
{
    int size;
    int Top;
    int *s;
}
```

Push() → inserting an element into stack

```
Push(5);
Push(15);
Push(8);
Push(7);
Push(20);
```

```
void push(Struct *st, int x)
{

O(1)    if(st -> Top == st -> Size - 1)
        {
            cout<< "stack overflow";
        }
        else
        {
            st -> Top++;
            st -> S[st -> Top] = x;
        }
}

int pop(struct *st)
{      int x = -1;
       if (st -> Top == -1)
O(1)   {
           cout<< "stack Underflow";
       }
       else
       {
           x = st -> S[st -> Top];
           st -> Top--;
       }
       return x;
}
```

Top-pos+1

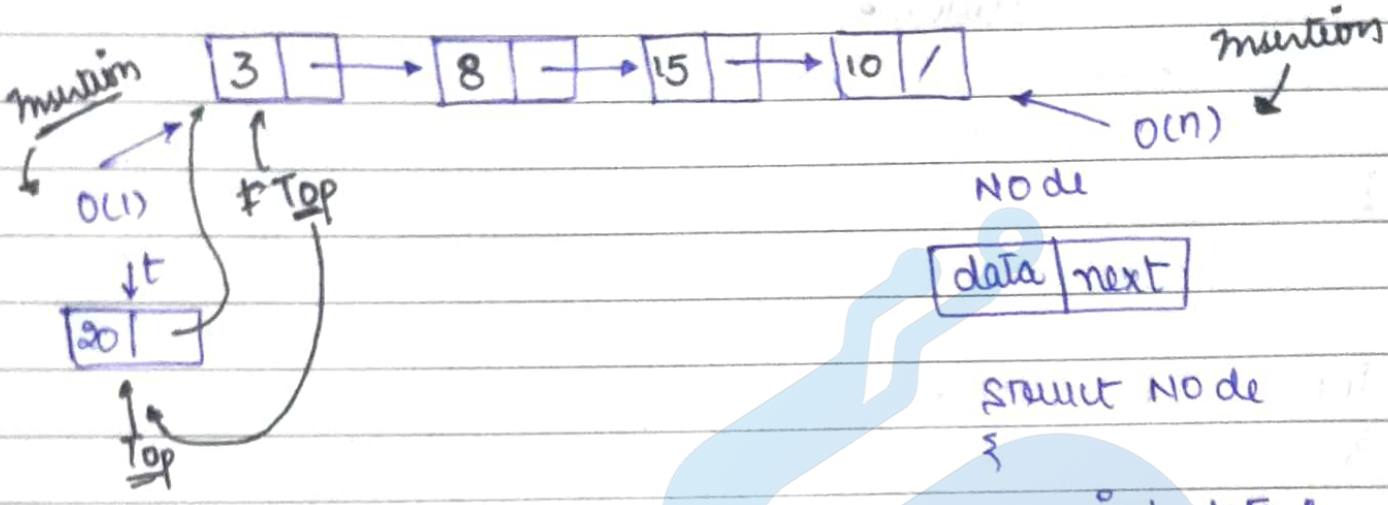
pos	Index
1	3 = 3-1+1
2	2 = 3-2+1
3	1 = 3-1+1
4	0 = 3-4+1

```
int peek(stack st, int pos)
{
    int x = -1;
    if (st.Top - pos + 1 < 0)
    {
        cout << "Invalid position";
    }
    else
    {
        x = st.S[st.Top - pos + 1];
    }
    return x;
}
int stackTop(stack st)
{
    if (st.Top == -1)
        return -1;
    else
        return st.S[st.Top];
}
```

O(1)

```
int isEmpty (stack st)
{
        if (st.Top == -1)
                return 1;
        else
                return 0;
}

int isFull (stack st)
{
        if (st.Top == st.size - 1)
        {
                return 1;
        }
        else
                return 0;
}
```

Stack Using Linked List



Node

data	next

```
struct Node
{
    int data;
    struct Node *nxt;
};
```

Top

=

```
[ Empty
  if(Top == NULL) ]
```

```
[ Full
  Node * t = new Node;
  if(t == NULL ]
```

stack operations using linked list :-

```
void push(int x)
{
        Node *t = new Node;
        if (t==NULL)
        {
                cout << "Stack Overflow";
        }
        else
        {
                t->data = x;
                t->next = Top;
                Top = t;
        }
}

int pop()
{   Node *p;
    int x=-1;
    if (Top==NULL)
    {
            cout << "Stack is Empty";
    }
    else
    }
            p=Top;
            Top=Top->next;
```

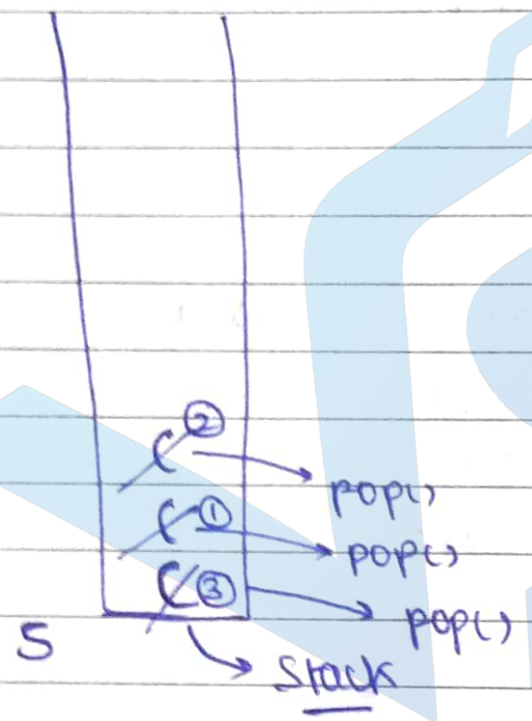
```
        x=p->data;
        delete(p)
    }
    return x;
}

int Peek(int pos)
{
    int i;
    Node *p=Top;

    for(i=0; p!=NULL && i<pos-1 ; i++)
    {
        p=p->next;
    }
    if(p!=NULL)
        return p->data;
    else
        return -1;
}
```

```
int stackTop()
{
    if(Top!=NULL)
        return pTop->data;
    return -1;
}

int isEmpty()
{
    return Top? 0:1;
}

int isFull()
{
    Node *t=new Node;
    int r=t?1:0;
    delete(t);
    return r;
}
```

Parenthesis Matching

((a+b) * (c-d))

↳ check whether it is balanced or not.



reach end of the stack

Parenthesis is matching

Program for Parenthesis Matching :-

exp

(	(	a	+	b	)	*	(	(	-	d	)	)	\0
0	1	2	3	4	5	6	7	8	9	10	11	12	13

```
struct stack
{
    int size;
    int Top;
    char *S;
}
```

```
int isBalance (char *exp)
{
```

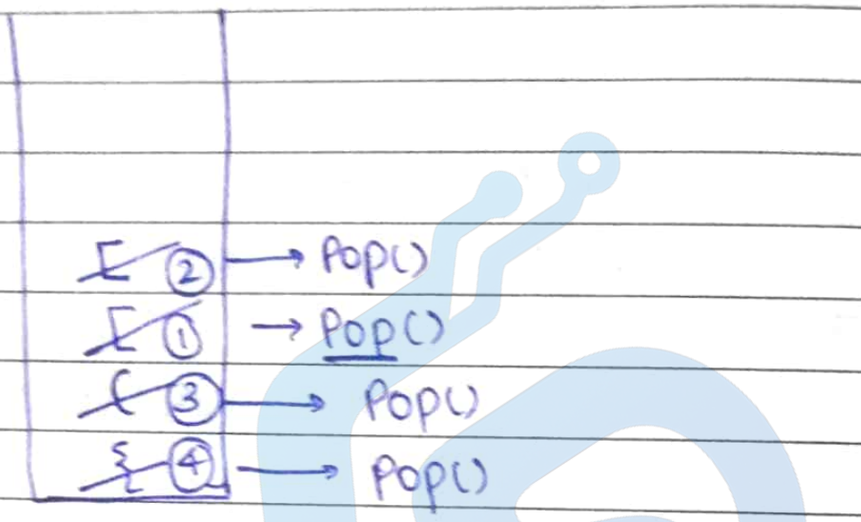
st

size	13
Top	-1
S	→ [ ][ ][ ]

```
    struct stack st;
    st.size = strlen(exp);
    st.Top = -1;
    st.S = new char [st.size];

    for(int i=0; exp[i] != '\0'; i++)
    {
        if (exp[i] == '(')
        {
            push(&st, exp[i]);
        }
        else if ( exp[i] == ')')
        {
            if( isEmpty(st) )
            {
                return false;
            }
            pop(&st);
        }
    }
    return isEmpty(st) ? true : false;
}
```

More on Parenthesis Matching:-

{ ([a+b] * [c-d]) / e }



We have to check whether the bracket is matching or not.

```
if(exp[i]=='}' || exp[i]=='(' || exp[i]=='[')
        push(&st, exp[i]);

else if( st.top()=='}' || st.top()==')' || st.top()==']')

        x = pop(&st);

else if ( exp[i]=='}' || exp[i]==')' ...
        x = pop(&st);
```

ASCII

40	(
41	)

91	[
93	]

123	{
125	}

Infix to Postfix conversion :-

1. What is Postfix
2. Why Postfix
3. Precedence
4. Manual conversion

1. Infix : operand operator operand

   eg: a + b

2. Prefix : operator operand operand

   eg: + ab

3. Postfix :- operand operand operator

   eg :- ab +

$$\underset{6}{8} + \underset{5}{3} * \underset{1}{(9-6)} / \underset{3}{2^2} + \underset{7}{6} / \underset{4}{2}$$

Postfix → ~~8 3 9 6~~

8 3 9 6 − 2 ^ / * + 6 2 / +

Purpose of writing expression in Postfix form is that we can scan the expression only once and perform all the operations.

Precedence

symbol	Precedence
+, −	1
~~* /~~ *, /	2
( )	3

- Infix to Postfix

① a + b * c

(a + (b * c))

Prefix

(a + * b c)

+ a * b c

Postfix

(a + b c *)

a b c * +

left to right

② a+b+c*d

Prefix

a+b+c*d

a+b+*cd

+ab+*cd

++ab*cd

a+b+c*d

a+b+cd*

ab++cd*

ab+cd*+

③ (a+b)*(c-d)

Prefix

(a+b)*(c-d)

(+ab)*(c-d)

[+ab)*[cd]

*+ab-cd

Postfix

(a+b)*(c-d)

[ab+]*(c-d)

[ab+]*[cd-]

ab+cd-*

Associativity and Unary operators

1. Associativity

sym	Precedence	Associa.
+, -	1	L-R
*, /	2	L-R
^	3	R-L
-	4	R-L
( )	5	L-R

$a+b+c-d$

L-R $(((a+b)+c)-d)$

$ab+c+d-$

$a=b=c=5$

Asso → R-L

R-L $(a=(b=(c=5)))$

$abc5===$ (Postfix)

- $a \wedge b \wedge c$

$(a \wedge (b \wedge c)) = (a)^{(b^c)}$

$a \wedge bc\wedge$

$abc\wedge\wedge$ $abc^{\wedge\wedge}$

① -a

Pre -a Post a-

(a(a))

③ *p

Pre *p Post p*

n! log x

Pre → !n Post → n! Pre - log x Post x log

$-a+b*\log n!$

$-a+b*\log[n!]$

$-a+b*[n!\log]$

$[a-]+b*[n!\log]$

$[a-]+[bn!\log]$

$a-bn!\log**+$

Infix to Postfix using stack

Method 1 :-

a+b*c-d/e

Postfix: abC*+de/-

Pos expression

Symbol	stack	Postfix
a	~~+~~	a
+	+	a
b	+	ab
*	+*	ab
c	+*	abc
-	-	abc*+
d		abc*+d
/	-/	abc*+d
e	-/	abc*+d
		abc*+d/-

Method 2:-

sym	Prec	Asso
+, -	1	L-R
*, /	2	L-R
a, b, c	3	L-R

a+b*c-d/e

Postfix : abc*+de/-

St

Program for Infix to Postfix Conversion

Infix

a	+	b	*	c	-	d	/	e	\0

Sym	Prec	Ass
+,-	1	L-R
/,*	2	L-R

```
int isOperand(char x)
{
    if(x=='+' || x=='-' || x=='*' || x=='/')
        return 0;

    else
        return 1;

}

int precedence(char x)
{
        if(x=='+' || x=='-')
            return 1;

        else if (x=='*' || x=='/')
            return 2;

        return 0;

}
```

```
char * convert(char *infix)
{
    struct stack st; // initialized
    char * postfix = new char[strlen(infix)+1];
    int i=0, j=0;
    while(infix[i]!='\0')
    {
        if(isOperand(infix[i]))
            postfix[j++] = infix[i++];
        else
        {
            if(precedence(infix[i]) > Precendence
                                    (stackTop(st)))
                push(&st,infix[i++]);
            else
                postfix[j++]= pop(&st)
        }
    }
    while(!isEmpty(st))
    {
        Postfix[j++] = pop(&st);
    }
    postfix[j]='\0';
    return postfix;
}
```

Student Challenge: infix to postfix with Associativity and Parenthesis

Symbol	Out stack precedence	Instack precedence
+, -	1	2
*, /	3	4
^ (right to left Associativity)	6	5
(	7	0
)	0	?

$((a+b)*c)-d\wedge e\wedge f$

$(ab+*c)-d\wedge e\wedge f$

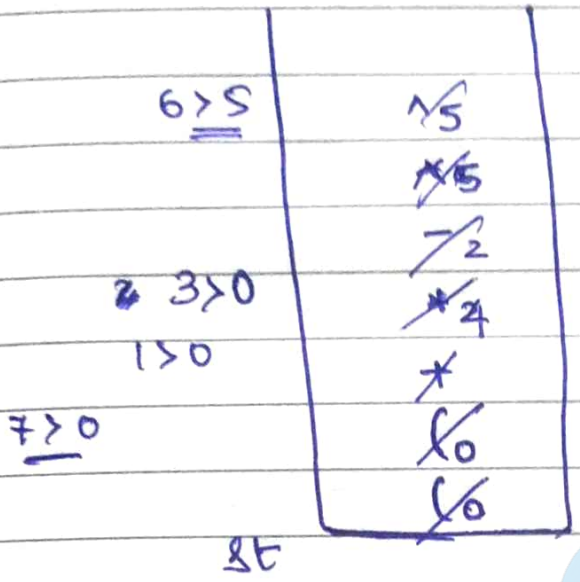
$[ab+c*]-d\wedge e\wedge f$

$[ab+c*]-d\wedge[ef\wedge]$

$[ab+c*]-[def\wedge\wedge]$

$ab+c*def\wedge\wedge-$

→ Postfix expression



Postfix

ab+c*df^^-

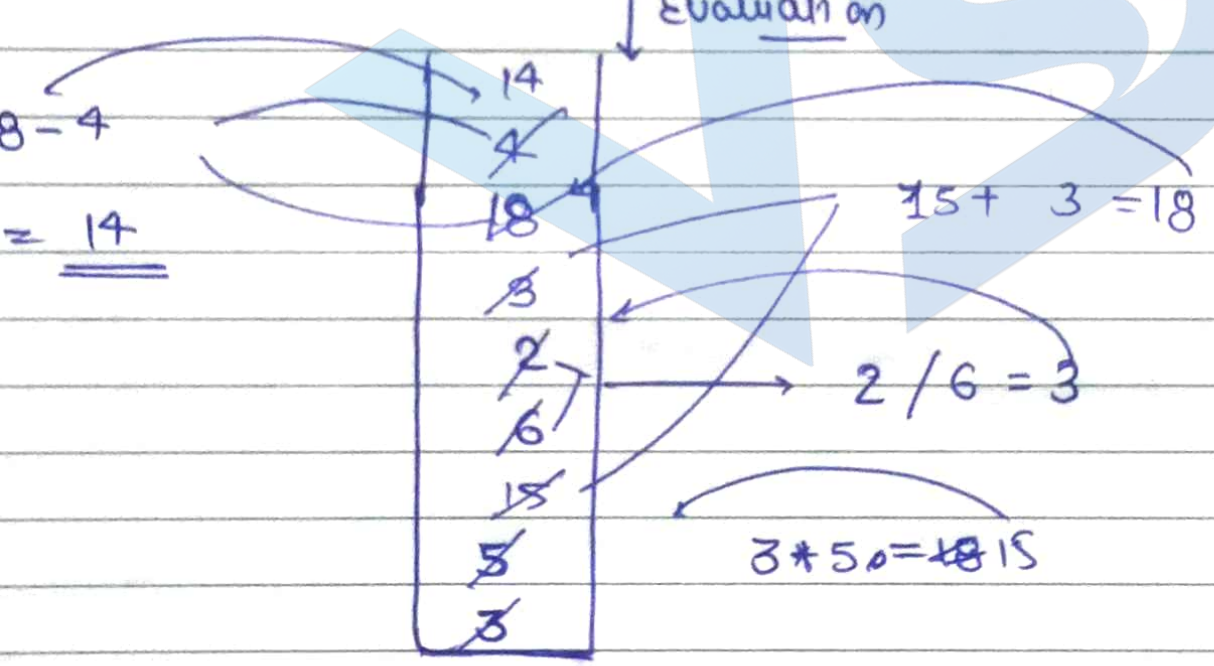
Evaluation of Postfix Expression

Exp:- 3*5+6/2-4

15+3-4

18-4 = 14

35*62/+4-

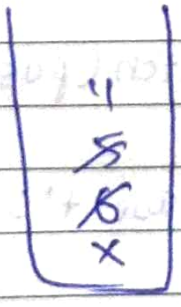


Symbol	stack	operation
3	3	
5	5, 3	
*	15	3 * 5 = 15
6	6, 15	
2	2, 6, 15	
/	3, 15	6 / 2 = 3
+	18	15 + 3 = 18
4	4, 18	
−	14	18 − 4 = 14

Precedence and Associativity is meant for parenthesized and not for execution.

$x = ((6 + 5) + (3 * 4))$

x 6 5 + 3 4 * + =



6 + 5 = 11

• Program for Evaluation of Postfix

Postfix

3	5	*	6	2	/	+	4	-	\0
0	1	2	3	4	5	6	7	8	9

```
int Eval(char * postfix)
{
    struct stack st;
    int i, x1, x2, r;
    for(i=0; postfix[i]!='\0'; i++)
    {
        if(isOperand(postfix[i])
            push(&st, postfix[i]-'0');
        else
        {
            x2 = pop(&st);
            x1 = pop(&st);
            switch(postfix[i])
            {
                case '+': r = x1 + x2;
                          push(&st, r);
                          break;
                case '-': r = x1 - x2;
                          push(&st, r)
                          break;
```

```
case '*':   r = x1 * x2
            push(&st, r)
            break;

case '/':   r = x1 / x2
            push(&st, r)
            break;
        }
    }
}
return pop(&st);
}
```