STACK continued....

\* Infix: x+y --> We use them in mathematical operations daily.

\* Postfix: xy+

\* Prefix: +xy

--> Advantages of Prefix and Postfix:

- Donot require parenthesis, precedence rules and associativity rules.

- Can be evaluated by writing a program that traverse the expression only once.

-It’s very convenient for evaluating formulas on computers with stacks.

-->Precedence: (Top to bottom decreases)

^ -- Right to Left

\* / -- Left to Right

+ - -- Left to Right

(^ is exponential symbol not the XOR one)

\*\* INFIX TO POSTFIX:

1- = x + y \* z

= (x + (y \* z))

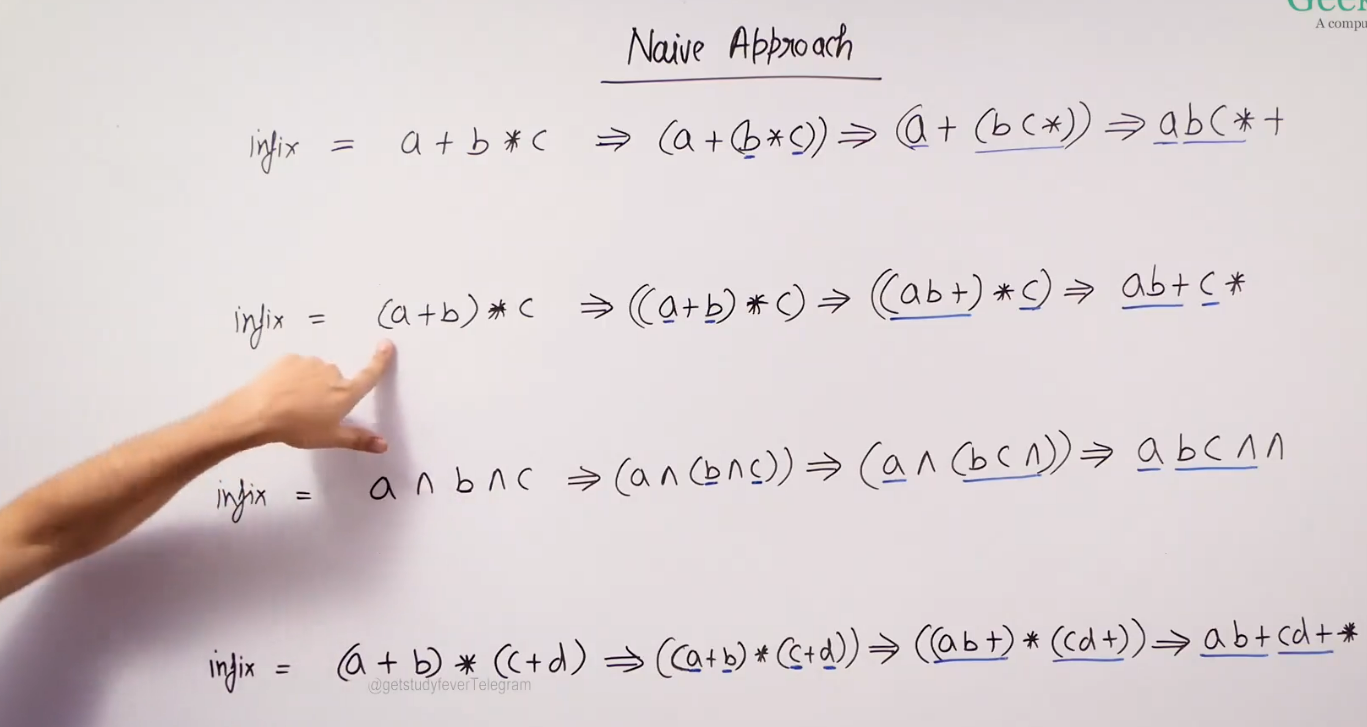
= (x + (yz\*))

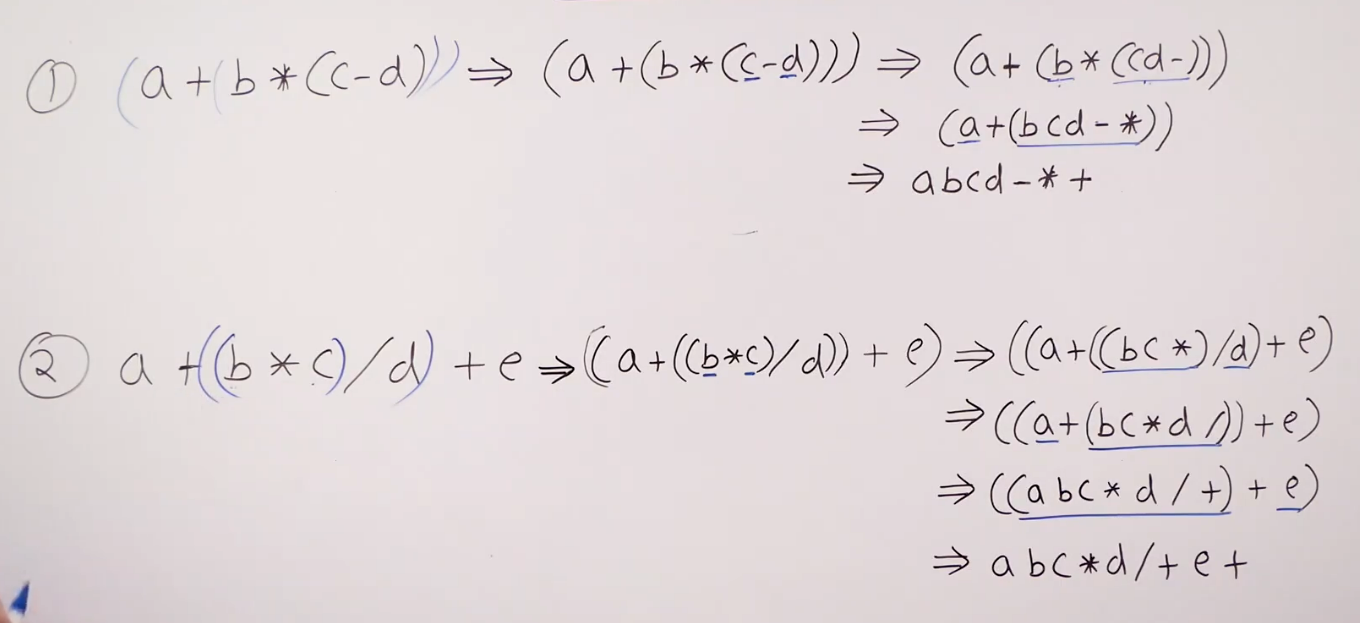
= xyz \* +

2- = (x + y) \* z

= ((xy +) \* z)

= xy + z\*





\*\* Conversion of Infix to postfix using Stack:

* Create an empty stack s.
* Do the following for every character of x from L to R.
* Operand – Output x
* Left Parenthesis – Push to s
* Right Parenthesis – Pop and print from s until left parenthesis is found.
* Operators:
  + If s is Empty push x to stack.
  + Else compare x with s.top.
    - Higher precedence wrt s.top() push to s.
    - Lower precedence, pop s.top() and print its value until higher precedence of operator is found.
    - Then push x to stack s.
    - Same precedence, use associativity.
* Pop and output everything from s.

- Code to convert infix to postfix:

int prec(char ch) {

if (ch == '^')

return 3;

else if (ch == '/' || ch == '\*')

return 2;

else if (ch == '+' || ch == '-')

return 1;

return -1;

}

string infixToPostfix(string s) {

stack<char> st;

string ans = "";

for (int i=0;i<s.length();i++)

{

char ch = s[i];

if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9'))

ans += ch;

// Append the current character of string in our answer

else if (ch == '(')

st.push('(');

else if (ch == ')') {

// pop that top character until '(' is encountered

while (st.top() != '(')

{

ans += st.top();

st.pop();

}

st.pop();

}

//If an operator is scanned

else {

while (!st.empty() && prec(s[i]) <= prec(st.top())) {

ans += st.top();

st.pop();

}

st.push(ch);

}

}

while (!st.empty()) {

ans += st.top();

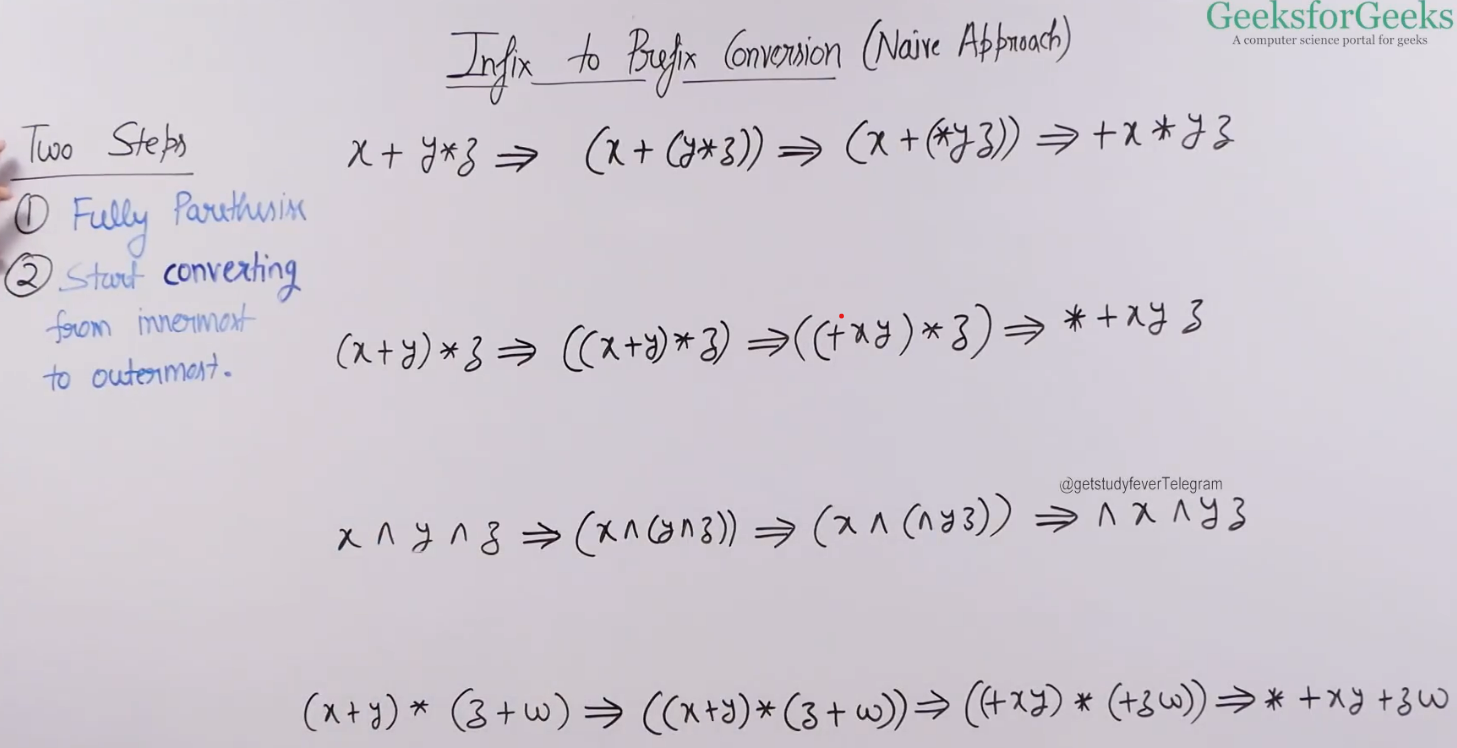
st.pop();

}

return ans;

}

\*\* INFIX TO PREFIX



\*\* Conversion of Infix to prefix using Stack:

* Create an empty stack s.
* Create an empty string prefix.
* Do the following for every character of x from L to R.
* Operand – Push it to prefix
* Left Parenthesis – Push to s
* Right Parenthesis – Pop and print from s until left parenthesis is found. Append the popped character to prefix.
* Operators:
  + If s is Empty push x to stack.
  + Else compare x with s.top.
    - Higher precedence wrt s.top() push to s.
    - Lower precedence, pop s.top() and print its value until higher precedence of operator is found.
    - Then push x to stack s.
    - Same precedence, use associativity.
* Pop everything and append to prefix.
* Print the reverse of prefix.