Rules for the conversion of infix to prefix expression:

- First, reverse the infix expression given in the problem.
- Scan the expression from left to right.
- Whenever the operands arrive, print them.
- If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
- If the incoming operator has higher precedence than the TOP of the stack, push the incoming operator into the stack.
- If the incoming operator has the same precedence with a TOP of the stack, push the incoming operator into the stack.
- If the incoming operator has lower precedence than the TOP of the stack, pop, and print the top of the stack. Test the incoming operator against the top of the stack again and pop the operator from the stack till it finds the operator of a lower precedence or same precedence.
- If the incoming operator has the same precedence with the top of the stack and the incoming operator is ^, then pop the top of the stack till the condition is true. If the condition is not true, push the ^ operator.
- When we reach the end of the expression, pop, and print all the operators from the top of the stack.
- If the operator is ')', then push it into the stack.
- If the operator is '(', then pop all the operators from the stack till it finds) opening bracket in the stack.
- If the top of the stack is ')', push the operator on the stack.
- At the end, reverse the output.

ALGORITHM:

Function InfixtoPrefix(stack, infix)
infix = reverse(infix)
loop i = 0 to infix.length
if infix[i] is operand → prefix+= infix[i]
else if infix[i] is '(' → stack.push(infix[i])
else if infix[i] is ')' → pop and print the values of stack till the symbol ')' is not found
else if infix[i] is an operator(+, -, *, /, ^) →

if the stack is empty then push infix[i] on the top of the stack.
Else →
If precedence(infix[i] > precedence(stack.top))
→ Push infix[i] on the top of the stack
else if(infix[i] == precedence(stack.top) && infix[i] == '^')
→ Pop and print the top values of the stack till the condition is **true**
→ Push infix[i] into the stack
else if(infix[i] == precedence(stack.top))
→ Push infix[i] on to the stack
Else **if**(infix[i] < precedence(stack.top))
→ Pop the stack values and print them till the stack is not empty and infix[i] < preceden
ce(stack.top)
→ Push infix[i] on to the stack
End loop
Pop and print the remaining elements of the stack
Prefix = reverse(prefix)
return