Aim:

Write a program to convert infix expression to postfix using Stack.

Algorithm:

**Step 1** – START

**Step 2** - declare an array 'stack' of size 100 of character data type and initialize an integer variable 'top' = -1.

**Step 3** - define functions isEmpty(), peek(), push(), pop(), checkIfAlnum(), and getPriority().

**Step 4** - under the function isEmpty(), return top is equal to -1.

**Step 5** - under the function peek(), return the element from the stack at the index top.

**Step 6** - under the function push(), get a character 'x' from the expression to be added to the stack; then ~~goto Step 7~~ continue.

**Step 7 -** increment the top by 1 and push the character 'x' into the address pointed by the top.

**Step 8** - under the function pop(), check if the stack is empty [goto Step 4]. If True, return -1; else, decrement the top by 1 and return the top element from the stack.

**Step 9** - under the function checkIfAlnum(), get a character 'x' from the expression to evaluate whether it is alphanumeric(letter or number) or not; then ~~goto Step 10~~ continue.

**Step 10** -check if x is between 'a' and 'z', x is between 'A' and 'Z', or x is between '0' and '9'. if True, return True; else, return False.

**Step 11** - under the function getPriority(), get a character 'x' from the expression to be evaluated and introduce a Switch case statement having the variable 'x' as its condition; then ~~goto Step 12~~ continue.

**Step 12** - if x is equal to '-', return 1; if x is equal to '+', return 2; if x is equal to ‘%’, return 3;if x is equal to ‘/’, return 4;if x is equal to ‘\*’, return 5;if x is equal to ‘^’, return 6;else, return -1.

**Step 13** - declare a pointer variable \*i, variable x, array 'infix' of size 100 of character data type, and 'choice' of integer data type.

**Step 14** – display 'INFIX TO POSTFIX CONVERTER' and Operations Start and Stop.

**Step 15** - introduce a do-while loop having the condition, 'choice equals to 1'. If the condition is True, ~~goto Step 16~~ continue; else, exit the loop and goto step 30.

**Step 16** - Under the do-while loop's body, get the operation to perform in the program from the user and store the value in the variable 'choice'; then continue.

**Step 17** - introduce a switch case statement having the variable 'choice' as its condition.

**Step 18** – if the choice equals 1, continue; if the choice equals 2, goto Step 15; default, display 'Invalid Input! – Enter 1 or 2'.

**Step 19** - get the infix expression and initialize i equals to infix.

**Step 20** – introduce a while loop that runs until the pointer variable \*i meets the end of the expression. If True, continue; else, goto Step 29.

**Step 21** – Check if a character from the expression is alphanumeric or not using the function checkIfAlnum() [goto Step 9]. If True, display the character; else, continue.

**Step 22** – check if a character is equal to open parenthesis '('. If True, append the character to the stack using the function push() [goto Step 6]; else, continue.

**Step 23** - check if the character is equal to close parenthesis ')'. If True, continue; else, goto Step 25.

**Step 24** -introduce a while loop that runs until the stack's popped element [goto Step 8] from the stack is not equal to '('. Store the popped element in variable 'x'. If True, display the variable 'x'; else, continue.

**Step 25** – introduce a while loop that runs until the priority [goto Step 11] of the stack's top element [goto Step 5] is greater than the priority [goto Step 11] of the current pointed character from the expression; Then continue.

**Step 26** - If True, display the popped element [goto Step 8] from the stack; else, continue.

**Step 27** – append character from the expression to the stack.

**Step 28** – increment 'i' by 1.

**Step 29** – introduce a while loop that runs until the stack is not empty [goto Step 4]. If True, display the popped element [goto Step 8] from the stack; else, goto Step 16.

**Step 30** - STOP

Result:

The program was executed successfully and achieved its aim of the program.