Aim:

To implement the conversion of an **Infix expression to Postfix expression** using stack.

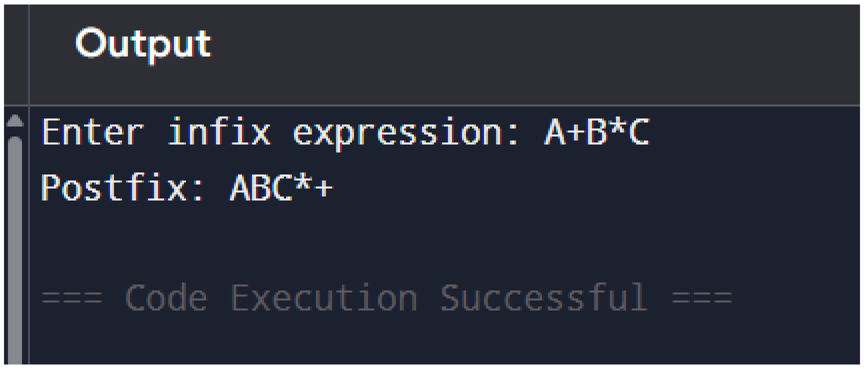
Algorithm:

1. Start
2. Initialize an empty stack.
3. Scan the infix expression from left to right:
    - If the symbol is an operand → output it.
    - If the symbol is (→ push it onto the stack.
    - If the symbol is) → pop and output until (is encountered.
    - If the symbol is an operator → pop operators with higher or equal precedence and output them, then push current operator.
4. Pop and output all remaining operators from the stack.
5. Stop.

CODE:

```
#include <stdio.h>
#include <ctype.h>
#define MAX 100

char stack[MAX];
int top = -1;

void push(char c) {
    stack[++top] = c;
```

```
}
char pop() {
    return stack[top--];
}
int precedence(char c) {
    if (c == '^') return 3;
    if (c == '*' || c == '/') return 2;
    if (c == '+' || c == '-') return 1;
    return -1;
}

void infixToPostfix(char* exp) {
    char *e, x;
    e = exp;
    while (*e != '\0') {
        if (isalnum(*e)) printf("%c", *e);
        else if (*e == '(') push(*e);
        else if (*e == ')') {
            while ((x = pop()) != '(') printf("%c", x);
        }
        else {
            while (top != -1 && precedence(stack[top]) >= precedence(*e))
                printf("%c", pop());
            push(*e);
        }
        e++;
    }
    while (top != -1) printf("%c", pop());
}

int main() {
    char exp[MAX];
    printf("Enter infix expression: ");
    scanf("%s", exp);
    printf("Postfix: ");
    infixToPostfix(exp);
    return 0;
}
```



```
Enter infix expression: A+B*C
Postfix: ABC*+

=== Code Execution Successful ===
```

RESULT:
The program successfully executed and displayed the implementation of stack notations.