**Experiment 3**

**AIM:** Write a program to convert an infix expression to a postfix expression.

**THEORY:** There are majorly two types of expressions

* Infix expression: The expression of the form a op b. When an operator is in-between every pair of operands.
* Postfix expression: The expression of the form a b op. When an operator is followed for every pair of operands.

The repeated scanning makes it very in-efficient. It is better to convert the expression to postfix(or prefix) form before evaluation.

**Code:**

1. #include < iostream >
2. #include < stack >
3. #include < cstring >
4. #include < unordered\_map >
5. using namespace std;
6. unordered\_map < char, int > prec;
7. void set\_prec() {
8. prec['('] = 4;
9. prec[')'] = 4;
10. prec['/'] = 3;
11. prec['\*'] = 2;
12. prec['+'] = 1;
13. prec['-'] = 0;
14. }
15. bool is\_op(char c) {
16. return prec.count(c);
17. }
18. char \* to\_postfix(char \* infix) {
19. stack < char > s;
20. int i = 0, k = 0, n = strlen(infix);
21. char \* postfix = new char[n];
22. while (i < n) {
23. char c = infix[i];
24. if (!is\_op(c))
25. postfix[k++] = c;
26. else if (c == ')') {
27. while (s.top() != '(') {
28. postfix[k++] = s.top();
29. s.pop();
30. }
31. s.pop();
32. }
33. else if (c == '(') s.push(c);
34. else if (!s.empty() && prec[s.top()] > prec[c]) {
35. while (!s.empty() && prec[s.top()] > prec[c]) {
36. if (s.top() == '(')
37. break;
38. postfix[k++] = s.top();
39. s.pop();
40. }
41. s.push(c);
42. }
43. else
44. s.push(c);
45. i++;
46. }
47. while (!s.empty()) {
48. postfix[k++] = s.top();
49. s.pop();
50. }
51. return postfix;
52. }
53. int main() {
54. set\_prec();
55. char infix[100];
56. cin >> infix;
57. char \* postfix = to\_postfix(infix);
58. cout << postfix << endl;
59. return 0;
60. }

**Output:**

