DATE : 20.12.2022

USN : 1NT21IS117

Q.Write a Program to convert infix expression to postfix expression.

AIM : Convert Infix to Postfix.

1.If the scanned character is an operand, put it into postfix expression.

2.If the scanned character is an operator and operator's stack is empty, push operator into operators' stack.

3.If the operator's stack is not empty, there may be following possibilities.

4.If the precedence of scanned operator is greater than the top most operator of operator's stack, push this operator into operator's stack.

5.If the precedence of scanned operator is less than the top most operator of operator's stack, pop the operators from operator's stack until we find a low precedence operator than the scanned character.

6.If the precedence of scanned operator is equal then check the associativity of the opertor, if assciavity is left to right then pop the operators from stack until we find a low operator precedence.If associativity right to left then simply put into stack.

7.If the scanned character is opening round bracket ( '(' ), push it into operator's stack.

8.If the scanned character is closing round bracket (')'), pop out operators from operator's stack until we find a opening bracket.

9.Repeat steps and pop out all at end.

10.Now pop expression.

ALGORITHM : Step 1 − scan the expression from left to right

Step 2 − if it is an operand push it to stack

Step 3 − if it is an operator pull operand from stack and perform operation

Step 4 − store the output of step 3, back to stack

Step 5 − scan the expression until all operands are consumed

Step 6 − pop the stack and perform operation

CODE :

1. #include<stdio.h>

2. #include<string.h>

3.

4. //char stack

5. char stack[25];

6. int top = -1;

7.

8. void push(char item) {

9.    stack[++top] = item;

10. }

11.

12. char pop() {

13.    return stack[top--];

14. }

15.

16. //returns precedence of operators

17. int precedence(char symbol) {

18.

19.    switch(symbol) {

20.       case '+':

21.       case '-':

22.          return 2;

23.          break;

24.       case '\*':

25.       case '/':

26.          return 3;

27.          break;

28.       case '^':

29.          return 4;

30.          break;

31.       case '(':

32.       case ')':

33.       case '#':

34.          return 1;

35.          break;

36.    }

37. }

38.

39. //check whether the symbol is operator?

40. int isOperator(char symbol) {

41.

42.    switch(symbol) {

43.       case '+':

44.       case '-':

45.       case '\*':

46.       case '/':

47.       case '^':

48.       case '(':

49.       case ')':

50.          return 1;

51.       break;

52.          default:

53.          return 0;

54.    }

55. }

56.

57. //converts infix expression to postfix

58. void convert(char infix[],char postfix[]) {

59.    int i,symbol,j = 0;

60.    stack[++top] = '#';

61.

62.    for(i = 0;i<strlen(infix);i++) {

63.       symbol = infix[i];

64.

65.       if(isOperator(symbol) == 0) {

66.          postfix[j] = symbol;

67.          j++;

68.       } else {

69.          if(symbol == '(') {

70.             push(symbol);

71.          } else {

72.             if(symbol == ')') {

73.

74.                while(stack[top] != '(') {

75.                   postfix[j] = pop();

76.                   j++;

77.                }

78.

79.                pop();   //pop out (.

80.             } else {

81.                if(precedence(symbol)>precedence(stack[top])) {

82.                   push(symbol);

83.                } else {

84.

85.                   while(precedence(symbol)<=precedence(stack[top])) {

86.                      postfix[j] = pop();

87.                      j++;

88.                   }

89.

90.                   push(symbol);

91.                }

92.             }

93.          }

94.       }

95.    }

96.

97.    while(stack[top] != '#') {

98.       postfix[j] = pop();

99.       j++;

100.    }

101.

102.    postfix[j]='\0';  //null terminate string.

103. }

104.

105. //int stack

106. int stack\_int[25];

107. int top\_int = -1;

108.

109. void push\_int(int item) {

110.    stack\_int[++top\_int] = item;

111. }

112.

113. char pop\_int() {

114.    return stack\_int[top\_int--];

115. }

116.

117. //evaluates postfix expression

118. int evaluate(char \*postfix){

119.

120.    char ch;

121.    int i = 0,operand1,operand2;

122.

123.    while( (ch = postfix[i++]) != '\0') {

124.

125.       if(isdigit(ch)) {

126.       push\_int(ch-'0');  // Push the operand

127.       } else {

128.          //Operator,pop two  operands

129.          operand2 = pop\_int();

130.          operand1 = pop\_int();

131.

132.          switch(ch) {

133.             case '+':

134.                push\_int(operand1+operand2);

135.                break;

136.             case '-':

137.                push\_int(operand1-operand2);

138.                break;

139.             case '\*':

140.                push\_int(operand1\*operand2);

141.                break;

142.             case '/':

143.                push\_int(operand1/operand2);

144.                break;

145.          }

146.       }

147.    }

148.

149.    return stack\_int[top\_int];

150. }

151.

152. void main() {

153.    char infix[25] = "1\*(2+3)",postfix[25];

154.    convert(infix,postfix);

155.

156.    printf("Infix expression is: %s\n" , infix);

157.    printf("Postfix expression is: %s\n" , postfix);

158.    printf("Result is: %d\n" , evaluate(postfix));

159. }

OUTPUT :

