Mihir sahu

1nt21is093

8/12/22

1.Infix to prefix algorithm -

Algorithm for the Evaluation of the Prefix Expression:

Step 1:We use for loop that runs as many times as the length of the string

Step 2:Inside the loop, we first check whether the scanned character is between 0 and 9.

Step 3:If the condition is valid, we convert that character into an integer and push it into the stack.

Step 4:If the scanned character is an operator, we pop out two-element from the stack.

Step 5:With the help of the switch case, we performed the arithmetic operation on the pop-out elements.

Step 6:After performing the Arithmetic operation, we pushed the result back into the stack.

Step 7:At the end of the function, we pop out the result, the last value stored in the stack.

Code:-

#include <stdio.h>

 #include <stdlib.h>

 #include <math.h>

 #include <string.h>

 #define MAX 100

 #define TAB '\t'

 #define BLANK ' '

 int top = -1;

 void push(long int symbol);  //defining push func

 long int pop();

 void infix\_to\_postfix();

 long int eval\_post();

 int priority(char symbol);

 int isEmpty();

 int white\_space(char symbol);

 char infix[MAX], postfix[MAX];

 long int stack[MAX];  // assigning array stack

 char \*reverses(char str[]);

 char str\_tmp[100]; //temp array

 long long int count = 0;

 int main()

 {

   long int value;

   top = -1;

   printf("Enter the infix:"); //enter the whole expression here

   gets(infix);

   strcpy(infix, reverses(infix));//Here we reverse the expression

   infix\_to\_postfix(); //calling function

   printf("Prefix:%s\n", reverses(postfix));

   value = eval\_post();

   printf("Value of expression: %ld\n", value);

 }

 void infix\_to\_postfix()

 {

   unsigned int i, p = 0;

   char next;

   char symbol;

   for (i = 0; i < strlen(infix); i++)

   {

     symbol = infix[i]; //assigning each value of infix expression to symbol

     if (!white\_space(symbol))

     {

       switch (symbol)

       {

       case '(':  // checking the operator precedence

         push(symbol);

         break;

         //PEMDAS = Parenthesis > expo > \* / > + -

       case ')':

         while ((next = pop()) != '(')

         {

           postfix[p++] = next;

         }

         break;

       case '+':

       case '-':  // check for all cases

       case '\*':

       case '/':

       case '%':

       case '^':

         while ((!isEmpty()) && (priority(stack[top]) > priority(symbol)))

         {

           postfix[p++] = pop();  // calling pop

         }

         push(symbol);

         break;

       default:

         postfix[p++] = symbol;  // ++

       }

     }

   }

   while (!isEmpty()) //if some symbols remains left they are pop out here

   {

     postfix[p++] = pop();

   }

   postfix[p] = '\0'; //the expression in string format

 }

 char \*reverses(char str[])//it reverse the string

 {

   int len = strlen(str);

   int j = 0;

   for (int i = len - 1; i >= 0; i--)

   {

     if (str[i] == '(')

     {

       str\_tmp[j] = ')';

       j++;

     }

     else if (str[i] == ')')

     {

       str\_tmp[j] = '(';

       j++;

     }

     else

     {

       str\_tmp[j] = str[i];

       j++;

     }

   }

   str\_tmp[j++] = '\0';

   return str\_tmp;

 }

 int priority(char symbol)//it assign the priority to the operators

 {

   switch (symbol)

   {

   case '(':

     return 0;

     break;

   case '+':

   case '-':

     return 1;

     break;

   case '\*':

   case '/':

   case '%':

     return 2;

     break;

   case '^':

     return 3;

     break;

   default:

     return 0;

   }

 }

 void push(long int symbol)

 {

   if (top > MAX)

   {

     printf("Stack overflow");

     return;

   }

   top = top + 1;

   stack[top] = symbol;

   count++;

 }

 long int pop()

 {

   if (isEmpty())

   {

     printf("Stack underflow\n");

     return 0;

   }

   return stack[top--];

 }

 int isEmpty()

 {

   if (top == -1)

   {

     return 1;

   }

   else

   {

     return 0;

   }

 }

 int white\_space(char symbol)

 {

   if (symbol == BLANK || symbol == TAB)

   {

     return 1;

   }

   else

     return 0;

 }

 long int eval\_post()//this function is for the evaluation of the expression

 {

   long int a, b, temp, result;

   unsigned int i;

   for (i = 0; i < strlen(postfix); i++)

   {

     if (postfix[i] <= '9' && postfix[i] >= '0')

     {

       push(postfix[i] - '0'); //conversion of string format to numeric for calcultion

     }

     else

     {

       a = pop();

       b = pop();

       switch (postfix[i])

       {

       case '+':

         temp = a + b;

         break;

       case '-':

         temp = a-b;

         break;

       case '\*':

         temp = b \* a;

         break;

       case '/':

         temp = a/b;

         break;

       case '%':

         temp = b% a;

         break;

       case '^':

         temp = pow(a, b);

         break;

       }

       push(temp);

     }

   }

   result = pop();

   return result;

 }

O/P-

