Experiment 3
Date 20-10-2022

INFIX TO POSTFIX WITHOUT USING STACK

Aim

Without using stack, convert an infix expression to a postfix expression and evaluate the postfix expression.

1 Infix to Postfix Algorithm

1.1 Algorithm

```
In_to_Post_without_Paranthesis(expr, converted, start, end)
Step 1: for i = end downto start
              if (expr[i] = '^' and converted[i] != 0)
                  left = GetLeftOperand(expr, converted, i, start)
                  right = GetRightOperand(expr, converted, i, end)
                  WritePostfix(expr, converted, i-length(left), left, right, expr[i])
              end if
          end for

Step 2: for i = start to end
              if ((expr[i] = '*' or expr[i] = '/') and converted[i] != 0)
                  left = GetLeftOperand(expr, converted, i, start)
                  right = GetRightOperand(expr, converted, i, end)
                  WritePostfix(expr, converted, i-length(left), left, right, expr[i])
              end if
          end for


Step 3: for i = start to end
              if ((expr[i] = '+' or expr[i] = '-') and converted[i] != 0)
                  left = GetLeftOperand(expr, converted, i, start)
                  right = GetRightOperand(expr, converted, i, end)
                  WritePostfix(expr, converted, i-length(left), left, right, expr[i])
              end if
          end for
```

```
GetLeftOperand(expr, converted, pos, start)

Step 1: left = []
Step 2: i = pos-1
Step 3: while ( i>= start and converted[i] != 0)
             left = concat(expr[i], left)
             i = i-1
         end while
Step 4: return left
```

```
GetRightOperand(expr, converted, pos, end)

Step 1: right = []
Step 2: i = pos+1
Step 3: while (i <= end and converted[i] != 0)
             right = concat(right, expr[i])
             i = i+1
         end while
Step 4: return right
```

```
WritePostfix(expr, converted, startIndex, left, right, op)

Step 1:  i = 0
Step 2:  while (i < length(left))
             expr[startIndex+i] = left[i]
             converted[startIndex+i] = 1
             i = i+1
         end while
Step 3: j = 0
Step 4: while (j < length(right))
             expr[i+j] = right[j]
             converted[i+j] = 1
             j = j+1
         end while

Step 5: expr[i+j] = op
Step 6: converted[i+j] = op
```

```
In_to_Post(expr)

Step 1:  for i = 0 to length(expr)-1
             converted[i] = 0
         end for
Step 2:  do
             i = 0
             startIndex = 0
             endIndex = length(expr)
             while (i < length(expr) and expr[i]!= ')')
                 if (expr[i] = '(')
                     startIndex = i+1
                     i = i+1
                 else if (expr[i] = ')')
                     endIndex = i-1
                     In_to_Post_without_Paranthesis(expr, converted, startIndex, endIndex)
                     expr[startIndex-1] = expr[endIndex+1] = '#'
                     converted[startIndex-1] = converted[endIndex+1] = 1
                     break;
                 else
                     i = i+1
                 end if
             end while
          while (startIndex != 0)
Step 3: In_to_Post_without_Paranthesis(expr, converted, 0, length(expr)-1)
Step 4: newExpr = []
Step 5: for i = 0 to length(expr)
             if (expr[i] != '#')
                 concat(newExpr, expr[i])
             end if
         end for
Step 6: return newExpr
```

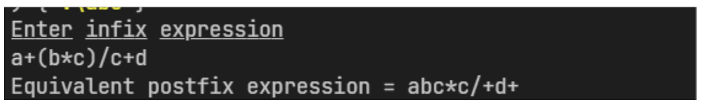
1.2 Program

```
#include <stdio.h>
#include<string.h>

void WritePostfixBrackets(char expr[]);
void PostNoBrackets(char expr[], int start, int end);
void GetLeftOperand(char expr[], char left[], int pos, int start);
void GetRightOperand(char expr[], char right[], int pos, int end);
void WritePostfix(char expr[], int startIndex, char left[], char right[], char op);
int converted[50];
```

```
void main()
{
    char expr[50];
    printf("Enter infix expression \n");
    scanf("%s",expr);
    char st[52];
    st[0]='(';
    st[1]='\0';
    strcat(st,expr);
    strcpy(expr,st);
    st[0]=')';
    st[1]='\0';
    strcat(expr,st);
    int l =strlen(expr);
    int i;
    for(i=0; i<l; i++)
    {
        converted[i]=0;
    }
    WritePostfixBrackets(expr);
}


void WritePostfixBrackets(char expr[])
{

    int l = strlen(expr),i,startIndex,endIndex,k=0;
    do
    {
        i=0;
        int c=0;
        startIndex=0;
        endIndex = l;
        while(i<l && expr[i]!=')')
        {
            if(expr[i]=='(')
            {
                c=1;
                startIndex = i+1;
                i++;
            }

            else
                ++i;
        }
        if(c==1)
        {
            i=0;
            while(i<l)
            {
                if(expr[i]==')')
                {
                  endIndex=i-1;
                PostNoBrackets(expr, startIndex, endIndex);
                expr[startIndex-1]=expr[endIndex+1]='#';
                converted[startIndex-1]=converted[endIndex+1]=1;
                c=0;
                break;
                }
                i++;
            }
        }
    }while(startIndex!=0);

    char newExp[50];
    for(i=0; i<l; i++)
    {
        if(expr[i]!='#')
        {
            newExp[k++]=expr[i];
        }
    }
    newExp[k]='\0';

    printf("Equivalent postfix expression = %s\n",newExp);
}

void PostNoBrackets(char expr[], int start, int end)
{
    int i,l;
    char left[30];
    char right[30];

    for(i=end; i>=start; i--)
    {
        if (expr[i] == '^' && converted[i] != 1)
{
                converted[i]=1;
                converted[i-1]=converted[i+1]=1;
                GetLeftOperand(expr, left, i, start);
                GetRightOperand(expr, right, i, end);
                l=strlen(left);
                WritePostfix(expr, i-l, left, right, expr[i]);
            }
    }

    for(i=start; i<=end; i++)
    {
        if ((expr[i] == '*' || expr[i] == '/') && converted[i] != 1)
            {
                converted[i]=1;
                converted[i-1]=converted[i+1]=1;
                GetLeftOperand(expr, left, i, start);
                GetRightOperand(expr, right, i, end);
                l=strlen(left);
                WritePostfix(expr, i-l, left, right, expr[i]);
            }
    }

    for(i=start; i<=end; i++)
    {
        if ((expr[i] == '+' || expr[i] == '-') && converted[i] != 1)
            {
                converted[i]=1;
                converted[i-1]=converted[i+1]=1;
                GetLeftOperand(expr, left, i, start);
                GetRightOperand(expr, right, i, end);
                l=strlen(left);
                WritePostfix(expr, i-l, left, right, expr[i]);
            }

    }

}


void GetLeftOperand(char expr[], char left[], int pos, int start)
{
    left[0]='\0';
    int i = pos-1;int j,d;
while ( i>= start && converted[i] != 0)
{   char st[20];
            st[0]=expr[i];
            st[1]='\0';
            strcat(st,left);
            strcpy(left,st);
            i = i-1;
        }
```

```
}

void GetRightOperand(char expr[], char right[], int pos, int end)
{
    right[0]='\0';
    int i = pos+1;
while ( i<= end && converted[i] != 0)
{
            char st[2];
            st[0]=expr[i];
            st[1]='\0';
            strcat(right, st);
    i = i+1;
        }
}

void WritePostfix(char expr[], int startIndex, char left[], char right[], char op)
{
    int i = 0;
while (i < strlen(left))
{
     expr[startIndex+i] = left[i];
i = i+1;
    }

    int j = 0;
while (j < strlen(right))
{
        expr[startIndex+i+j] = right[j];
//converted[i+j] = 1;
j = j+1;
    }

    expr[startIndex+i+j] = op;

}
```

1.3 Sample Input and Output



```
Enter infix expression
a+(b*c)/c+d
Equivalent postfix expression = abc*c/+d+
```

Figure 1: Output

1.4 Result

Successfully executed program to convert Infix expression to postfix expression without using stack.