**1. Topic – Converting a Valid Infix Expression to Postfix Expression**

* **Problem Statement**

**Given a valid infix expression ( each operand is one digit), write a C program to convert the infix expression into a postfix expression.**

**Input example:**

Enter a valid infix expression

8/4-(9\*(6/3))+(5\*(3+2))

**Output example:**

The required postfix expression of this infix expression is 84/963/\*-532+\*+

* **Proposed C Code**

**/\* ---------- infix\_to\_postfix.c--------------- \*/**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**#define SIZE 100**

**/\* Stack size is taken as 100 \*/**

**static int top=-1; /\* global top variable which is used for push and pop \*/**

**void stackempty()**

**{**

**printf("Stack is empty\n");**

**exit(0);**

**}**

**void stackfull()**

**{**

**printf("Stack is full\n");**

**exit(0);**

**}**

**/\* function to push \*/**

**void push(char\* stack,char ch)**

**{**

**if(top>=SIZE-1)**

**stackfull();**

**else**

**stack[++top] = ch;**

**}**

**/\* function to pop \*/**

**char pop(char \*stack)**

**{**

**if(top==-1)**

**stackempty();**

**else**

**return stack[top--];**

**}**

**/\* function to check precedence \*/**

**int precendence\_check(char ch)**

**{**

**/\* precedence order '+','-','\*','/' is set \*/**

**if ((ch == '/') || (ch == '\*'))**

**{**

**return 2;**

**}**

**else if ((ch == '+') || (ch == '-'))**

**{**

**return 1;**

**}**

**else**

**{**

**return 0;**

**}**

**}**

**/\* function to check if a character is parentheses or operator or not \*/**

**int check\_operator(char ch)**

**{**

**if ((ch == '(') || (ch == ')') || (ch == '+') || (ch == '-') || (ch == '\*') || (ch == '/'))**

**{**

**return 1;**

**}**

**else**

**{**

**return 0;**

**}**

**}**

**int main()**

**{**

**char \*str = (char\*)malloc(100\*sizeof(char));/\* string for input infix expression \*/**

**printf("Enter a valid infix expression\n");**

**scanf("%s",str);**

**int infixlength = strlen(str);**

**char \*output = (char\*)malloc((infixlength+1)\*sizeof(char)); /\* output string which will give postfix expression \*/**

**char \*stack = (char\*)malloc(SIZE\*sizeof(char));**

**int j =0; /\* variable to insert character in postfix output string \*/**

**for ( int i = 0 ; i < infixlength ; i++ )**

**{**

**if ( check\_operator(str[i]) == 0 )/\* if no operator is found then just inserted \*/**

**{**

**output[j] = str[i];**

**j++;**

**}**

**else**

**{**

**if ( str[i] == '(') /\* whenever '(' is found it is only pushed into the stack \*/**

**{**

**push(stack,str[i]);**

**}**

**else if ( str[i] == ')')**

**/\* whenever ')' is found it pops until the corresponding '(' is found and that '(' is also poped \*/**

**{**

**while ( stack[top] != '(')**

**{**

**output[j] = pop(stack);**

**j++;**

**}**

**top--;**

**}**

**else**

**{**

**while ( top != -1 )**

**{**

**/\* an operator is pushed if it has higher precedence than the operator on stack \*/**

**if ( precendence\_check(str[i]) > precendence\_check(stack[top]) )**

**{**

**push(stack,str[i]);**

**break;**

**/\* if operator is pushed then we will iterate for next character so we just break the loop \*/**

**}**

**/\* if operator has less or same precedence than stack[top] \*/**

**/\* then we will pop until it has higher presedence than stack[top]\*/**

**else**

**{**

**output[j] = pop(stack);**

**j++;**

**}**

**}**

**/\* we will push the current operator if all the previous operators are poped \*/**

**if ( top == -1)**

**{**

**push(stack,str[i]);**

**}**

**}**

**}**

**}**

**/\* when the input string is scanned totally then we will pop the rest operators in stack \*/**

**while( top != -1 )**

**{**

**output[j] = pop(stack);**

**j++;**

**}**

**output[j] = '\0'; /\* terminating the output string \*/**

**printf("The required postfix expression of this infix expression is %s\n",output);**

**free(str); /\* freeing the memory after desired output is generated \*/**

**free(output);**

**free(stack);**

**return 0;**

**}**

**/\*------------------------------------------------------------------------------------------------------------------------- \*/**

* **Conclusion**

**The proposed algorithm has overall runtime of O(n) where n is size of the string.**

* **Limitations : This algorithm will work for the expression for which each operand is of single digit.**
* **Assumptions: Here we are considering the infix expression is valid and there is one operator between each operand such that total operands are total operator + 1(excluding the parentheses) .**