Expression Parsing Using Stack

Infix notation is easier for humans to read and understand whereas for electronic machines like computers, postfix is the best form of expression to parse. We shall see here a program to convert and evaluate **infix** notation to **postfix** notation −

Example

Live Demo

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

//char stack
char stack[25];
int top = -1;

void push(char item) {
   stack[++top] = item;
}

char pop() {
   return stack[top--];
}

//returns precedence of operators
int precedence(char symbol) {

   switch(symbol) {
      case '+':
      case '-':
         return 2;
         break;
      case '*':
      case '/':
         return 3;
         break;
      case '^':
         return 4;
         break;
```

```
      case '(':
      case ')':
      case '#':
         return 1;
         break;
   }
}

//check whether the symbol is operator?
int isOperator(char symbol) {

   switch(symbol) {
      case '+':
      case '-':
      case '*':
      case '/':
      case '^':
      case '(':
      case ')':
         return 1;
      break;
         default:
         return 0;
   }
}

//converts infix expression to postfix
void convert(char infix[],char postfix[]) {
   int i,symbol,j = 0;
   stack[++top] = '#';

   for(i = 0;i<strlen(infix);i++) {
      symbol = infix[i];

      if(isOperator(symbol) == 0) {
         postfix[j] = symbol;
         j++;
      } else {
         if(symbol == '(') {
            push(symbol);
         } else {
            if(symbol == ')') {

               while(stack[top] != '(') {
```

```
                postfix[j] = pop();
                j++;
             }

             pop();   //pop out (.
          } else {
             if(precedence(symbol)>precedence(stack[top])) {
                push(symbol);
             } else {

                while(precedence(symbol)<=precedence(stack[top])) {
                   postfix[j] = pop();
                   j++;
                }

                push(symbol);
             }
          }
       }
    }

    while(stack[top] != '#') {
       postfix[j] = pop();
       j++;
    }

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

//int stack
int stack_int[25];
int top_int = -1;

void push_int(int item) {
    stack_int[++top_int] = item;
}

char pop_int() {
    return stack_int[top_int--];
}

//evaluates postfix expression
int evaluate(char *postfix){
```

```
   char ch;
   int i = 0,operand1,operand2;

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

      if(isdigit(ch)) {
             push_int(ch-'0');  // Push the operand
      } else {
         //Operator,pop two  operands
         operand2 = pop_int();
         operand1 = pop_int();

         switch(ch) {
            case '+':
               push_int(operand1+operand2);
               break;
            case '-':
               push_int(operand1-operand2);
               break;
            case '*':
               push_int(operand1*operand2);
               break;
            case '/':
               push_int(operand1/operand2);
               break;
         }
      }
   }

   return stack_int[top_int];
}

void main() {
   char infix[25] = "1*(2+3)",postfix[25];
   convert(infix,postfix);

   printf("Infix expression is: %s\n" , infix);
   printf("Postfix expression is: %s\n" , postfix);
   printf("Evaluated expression is: %d\n" , evaluate(postfix));
}
```

If we compile and run the above program, it will produce the following result −

Output

```
Infix expression is: 1*(2+3)
Postfix expression is: 123+*
Result is: 5
```