Exp. No. 15

Write a C Program to implement the operator precedence parsing.

Program:

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

char *input;
int i = 0;
char lasthandle[6], stack[50], handles[][5] = {")E(", "E*E", "E+E", "i",
"E^E"};
// "(E)" becomes ")E(" when pushed to stack
int top = 0, l;

// Operator precedence table
char prec[9][9] = {
    /* input */
    /* stack +  -  *  /  ^  i  (  )  $ */
    /* + */  '>', '>', '<', '<', '<', '<', '<', '>', '>',
    /* - */  '>', '>', '<', '<', '<', '<', '<', '>', '>',
    /* * */  '>', '>', '>', '>', '<', '<', '<', '>', '>',
    /* / */  '>', '>', '>', '>', '<', '<', '<', '>', '>',
    /* ^ */  '>', '>', '>', '>', '<', '<', '<', '>', '>',
    /* i */  '>', '>', '>', '>', '>', 'e', 'e', '>', '>',
    /* ( */  '<', '<', '<', '<', '<', '<', '<', '>', 'e',
    /* ) */  '>', '>', '>', '>', '>', 'e', 'e', '>', '>',
    /* $ */  '<', '<', '<', '<', '<', '<', '<', '<', '>'
};

int getindex(char c) {
    switch (c) {
        case '+': return 0;
        case '-': return 1;
        case '*': return 2;
        case '/': return 3;
        case '^': return 4;
        case 'i': return 5;
        case '(': return 6;
        case ')': return 7;
        case '$': return 8;
        default: return -1;
    }
```

```
}

void shift() {
   stack[++top] = input[i++];
   stack[top + 1] = '\0';
}

int reduce() {
   int len, found, t;
   for (int i = 0; i < 5; i++) { // Selecting handles
      len = strlen(handles[i]);
      if (stack[top] == handles[i][0] && top + 1 >= len) {
         found = 1;
         for (t = 0; t < len; t++) {
            if (stack[top - t] != handles[i][t]) {
               found = 0;
               break;
            }
         }
         if (found) {
            stack[top - t + 1] = 'E';
            top = top - t + 1;
            strcpy(lasthandle, handles[i]);
            stack[top + 1] = '\0';
            return 1; // Successful reduction
         }
      }
   }
   return 0;
}

void dispstack() {
   for (int j = 0; j <= top; j++)
      printf("%c", stack[j]);
}

void dispinput() {
   for (int j = i; j < l; j++)
      printf("%c", input[j]);
}

int main() {
   input = (char *)malloc(50 * sizeof(char));
```

```
    if (!input) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    printf("\nEnter the string: ");
    scanf("%s", input);
    input = strcat(input, "$");
    l = strlen(input);

    strcpy(stack, "$");
    printf("\nSTACK\t\tINPUT\t\tACTION");

    while (i < l) {
        shift();
        printf("\n");
        dispstack();
        printf("\t\t");
        dispinput();
        printf("\t\tShift");

        while (prec[getindex(stack[top])][getindex(input[i])] == '>') {
            if (reduce()) {
                printf("\n");
                dispstack();
                printf("\t\t");
                dispinput();
                printf("\t\tReduced: E->%s", lasthandle);
            } else {
                break;
            }
        }
    }

    if (strcmp(stack, "$E$") == 0)
        printf("\nAccepted.");
    else
        printf("\nNot Accepted.");

    free(input); // Free allocated memory
    return 0;
}
```

OUTPUT:

Enter the string
i+i*i

```
STACK        INPUT        ACTION
--------------------------------
$            i+i*i$       Shift
$E           +i*i$        Reduced: E->i
$E+          i*i$         Shift
$E+E         *i$          Reduced: E->i
$E+E*        i$           Shift
$E+E*E       $            Reduced: E->i
$E+E         $            Reduced: E->E*E
$E           $            Reduced: E->E+E
Accepted.
```