Exp. No. 19
Write a C program to compute LEADING( ) – operator precedence
parser for the given grammar
Program:
E → E + T | T
T → T * F | F
F → (E) | id

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

// Array for parsing table
char arr[18][3] = {
    {'E', '+', 'T'}, {'E', '*', 'T'}, {'E', '(', 'T'}, {'E', ')', 'T'}, {'E', 'i', 'T'},
{'E', '$', 'T'},
    {'F', '+', 'T'}, {'F', '*', 'T'}, {'F', '(', 'T'}, {'F', ')', 'T'}, {'F', 'i', 'T'},
{'F', '$', 'T'},
    {'T', '+', 'F'}, {'T', '*', 'F'}, {'T', '(', 'F'}, {'T', ')', 'F'}, {'T', 'i', 'F'},
{'T', '$', 'F'}
};

// Production rules
char prod[] = "EETTFF";
char res[6][3] = {
    {'E', '+', 'T'}, {'T', '\0'}, {'T', '*', 'F'}, {'F', '\0'},
    {'(', 'E', ')'}, {'i', '\0'}
};

// Stack for tracking production rules
char stack[5][2];
int top = -1;

// Function to update parsing table
void install(char pro, char re) {
    for (int i = 0; i < 18; ++i) {
        if (arr[i][0] == pro && arr[i][1] == re) {
            arr[i][2] = 'T';
            break;
        }
    }
```

```
    ++top;
    stack[top][0] = pro;
    stack[top][1] = re;
}

int main() {
    int i, j;
    char pro, re, pri = ' ';

    // Filling parsing table
    for (i = 0; i < 6; ++i) {
        for (j = 0; j < 3 && res[i][j] != '\0'; ++j) {
            if (strchr("+-()*i$", res[i][j])) {
                install(prod[i], res[i][j]);
                break;
            }
        }
    }

    // Resolving production rules
    while (top >= 0) {
        pro = stack[top][0];
        re = stack[top][1];
        --top;
        for (i = 0; i < 6; ++i) {
            if (res[i][0] == pro && res[i][0] != prod[i]) {
                install(prod[i], re);
            }
        }
    }

    // Printing parsing table
    printf("\nParsing Table:");
    for (i = 0; i < 18; ++i) {
        printf("\n\t");
        for (j = 0; j < 3; ++j) {
            printf("%c\t", arr[i][j]);
        }
    }
```

```
    printf("\n\nProductions:\n");
    for (i = 0; i < 18; ++i) {
        if (pri != arr[i][0]) {
            pri = arr[i][0];
            printf("\n\t%c -> ", pri);
        }
        if (arr[i][2] == 'T') {
            printf("%c ", arr[i][1]);
        }
    }

    printf("\n");
    return 0;
}
```

Output

```
Parsing Table:
    E   +   T
    E   *   T
    E   (   T
    E   )   T
    E   i   T
    E   $   T
    F   +   T
    F   *   T
    F   (   T
    F   )   T
    F   i   T
    F   $   T
    T   +   F
    T   *   T
    T   (   T
    T   )   F
    T   i   T
    T   $   F

Productions:

    E -> + * ( ) i $
    F -> + * ( ) i $
    T -> * ( i


=== Code Execution Successful ===
```