Write a C program to compute TRAILING( ) – operator precedence parser for the given grammar

E → E + T | T

T → T \* F | F

F → ( E ) | id

Program:

#include<conio.h>

#include<stdio.h>

char arr[18][3] ={{'E', '+', 'F'}, {'E', '\*', 'F'}, {'E', '(', 'F'}, {'E', ')', 'F'}, {'E', 'i', 'F'},

{'E', '$', 'F'}, {'F', '+', 'F'}, {'F', '\*', 'F'}, {'F', '(', 'F'}, {'F', ')', 'F'}, {'F', 'i', 'F'},

{'F', '$', 'F'}, {'T', '+', 'F'}, {'T', '\*', 'F'}, {'T', '(', 'F'}, {'T', ')', 'F'}, {'T', 'i', 'F'},

{'T', '$', 'F'},

};

char prod[6] = "EETTFF";

char res[6][3] ={ {'E', '+', 'T'}, {'T', '\0', '\0'}, {'T', '\*', 'F'}, {'F', '\0', '\0'}, {'(', 'E', ')'}, {'i', '\0', '\0'},};

char stack [5][2];

int top = -1;

void install(char pro, char re) {

int i;

for (i = 0; i < 18; ++i) {

if (arr[i][0] == pro && arr[i][1] == re) {

}

}

++top;

arr[i][2] = 'T';

stack[top][0] = pro;

stack[top][1] = re;

}

int main() {

int i = 0, j;

char pro, re, pri = ' ';

for (i = 0; i < 6; ++i) {

for (j = 2; j >= 0; --j) {

if (res[i][j] == '+' || res[i][j] == '\*' || res[i][j] == '(' || res[i][j] == ')' || res[i][j] == 'i' || res[i][j] == '$') {

install(prod[i], res[i][j]);

break;

} else if (res[i][j] == 'E' || res[i][j] == 'F' || res[i][j] == 'T') {

if (res[i][j - 1] == '+' || res[i][j - 1] == '\*' || res[i][j - 1] == '(' || res[i][j -

1] == ')' || res[i][j - 1] == 'i' || res[i][j - 1] == '$') {

install(prod[i], res[i][j - 1]);

break;

}

}

}

}

while (top >= 0) {

pro = stack[top][0];

re = stack[top][1];

--top;

for (i = 0; i < 6; ++i) {

for (j = 2; j >= 0; --j) {

if (res[i][0] == pro && res[i][0] != prod[i]) {

install(prod[i], re);

break;

} else if (res[i][0] != '\0') break;

}

}

}

for (i = 0; i < 18; ++i) {

printf("\n\t");

for (j = 0; j < 3; ++j)

printf("%c\t", arr[i][j]);

}

printf("\n\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]);}

}

Output:

E + F

E \* F

E ( F

E ) F

E i F

E $ F

F + F

F \* F

F ( F

F ) F

F i F

F $ F

T + F

T \* F

T ( F

T ) F

T i F

T $ F

E ->

F ->

T ->