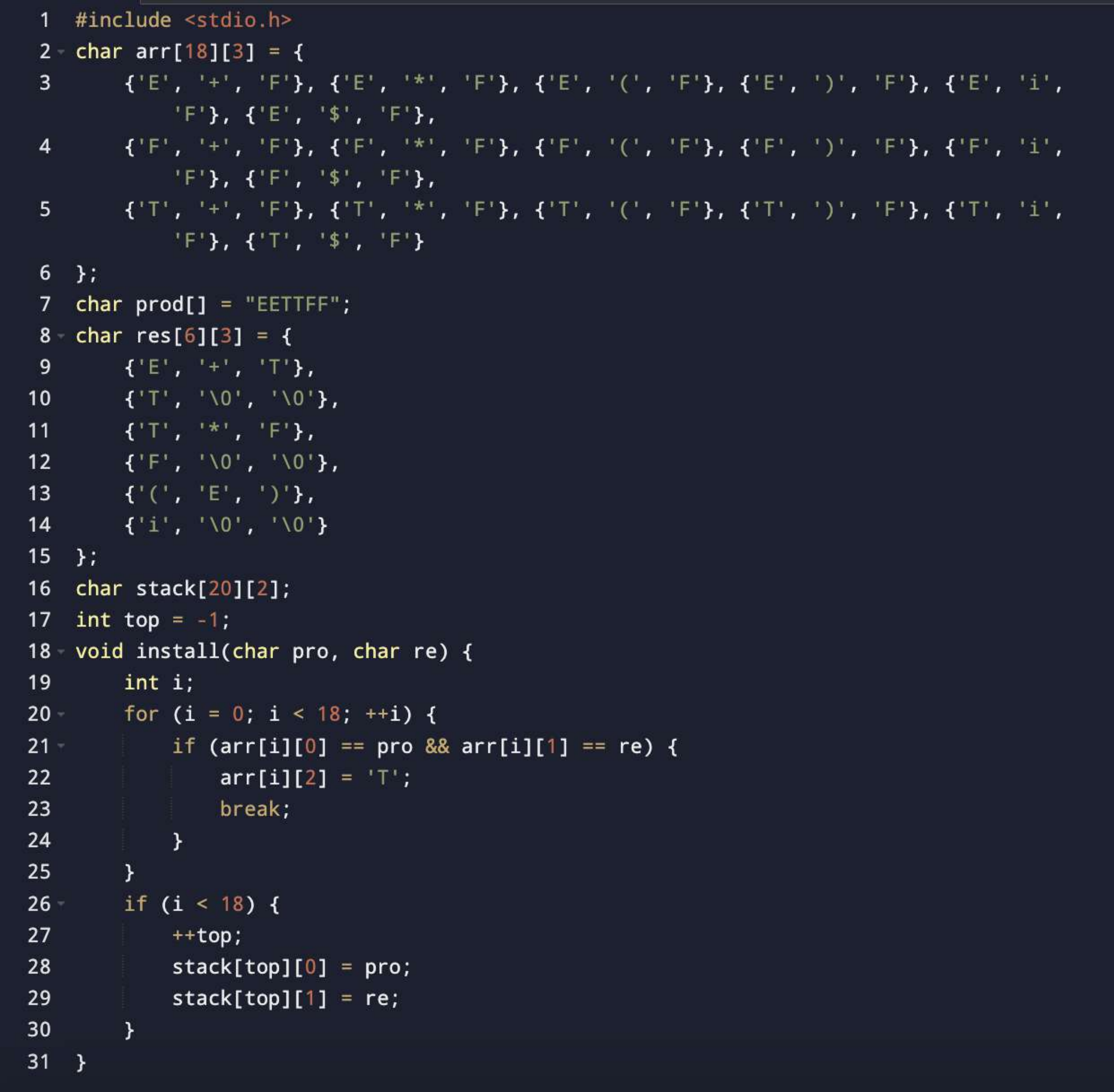
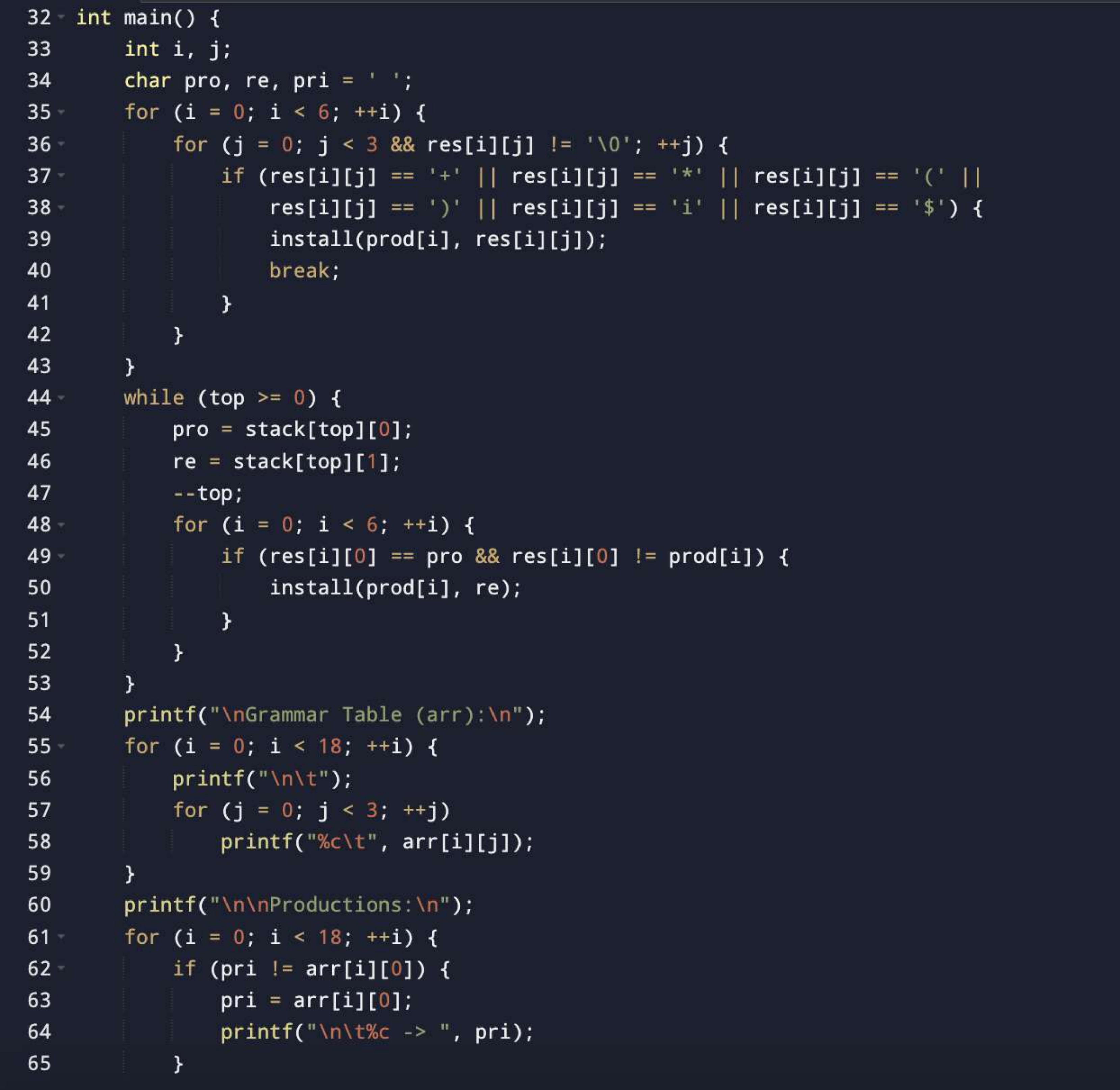
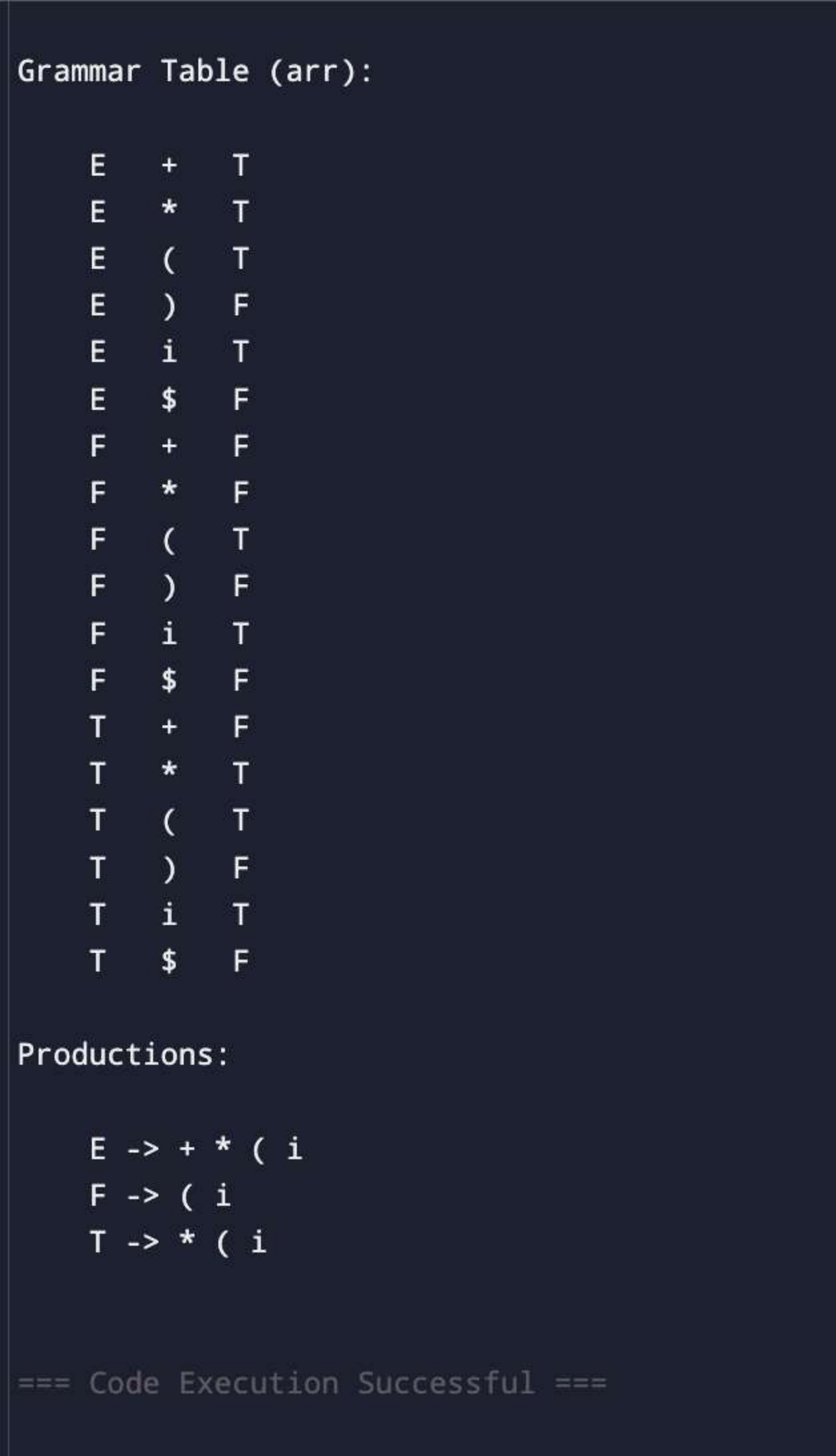


```c
#include <stdio.h>
char arr[18][3] = {
    {'E', '+', 'F'}, {'E', '*', 'F'}, {'E', '(', 'F'}, {'E', ')', 'F'}, {'E', 'i', 'F'}, {'E', '$', 'F'},
    {'F', '+', 'F'}, {'F', '*', 'F'}, {'F', '(', 'F'}, {'F', ')', 'F'}, {'F', 'i', 'F'}, {'F', '$', 'F'},
    {'T', '+', 'F'}, {'T', '*', 'F'}, {'T', '(', 'F'}, {'T', ')', 'F'}, {'T', 'i', 'F'}, {'T', '$', 'F'}
};
char prod[] = "EETTFF";
char res[6][3] = {
    {'E', '+', 'T'},
    {'T', '\0', '\0'},
    {'T', '*', 'F'},
    {'F', '\0', '\0'},
    {'(', 'E', ')'},
    {'i', '\0', '\0'}
};
char stack[20][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) {
            arr[i][2] = 'T';
            break;
        }
    }
    if (i < 18) {
        ++top;
        stack[top][0] = pro;
        stack[top][1] = re;
    }
}
```



```c
int main() {
    int i, j;
    char pro, re, pri = ' ';
    for (i = 0; i < 6; ++i) {
        for (j = 0; j < 3 && res[i][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;
            }
        }
    }
    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);
            }
        }
    }
    printf("\nGrammar Table (arr):\n");
    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;
}
```



```
Grammar Table (arr):

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

Productions:

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


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