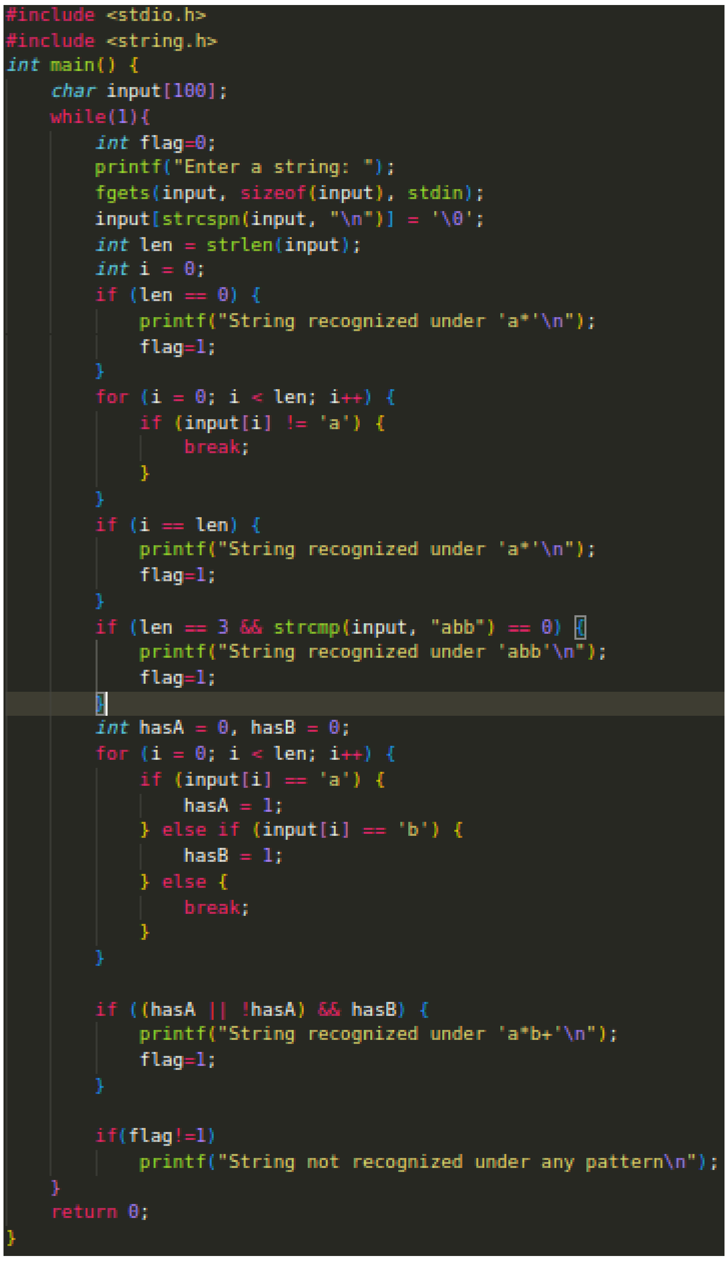
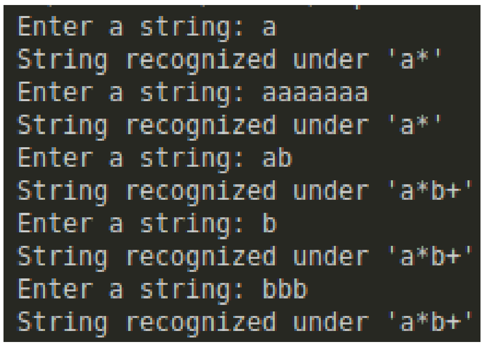
```
include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int states;
int inputs;
int transitions[10][10][10];
char nfa_table[10][10][10];
char final_dfa[10][10][10];

void main()
{
    printf("Enter no. of states: ");
    scanf("%d", &states);

    printf("Enter no. of input symbols: ");
    scanf("%d", &inputs);

    for (int i = 0; i < inputs; i++) {
        printf("Enter NFA matrix for input %d:\n", i + 1);
        for (int j = 0; j < states; j++)
        {
            for (int k = 0; k < states; k++)
            {
                scanf("%d", &transitions[i][j][k]);
            }
        }
    }
    char str[10];
    for (int i = 0; i < states; i++)
    {
        for (int j = 0; j < states; j++)
        {
            for (int k = 0; k < inputs; k++)
            {
                if (transitions[k][i][j] == 1)
                {
                    sprintf(str, "q%d", j);
                    if (strcmp(nfa_table[i][k], str) != 0)
                    {
                        strcat(nfa_table[i][k], str);
                    }
                }
            }
        }
    }
    char queue[20][10]; int front = 0;
    int rear = 0;
    int rows = 0;
    for (int i = 0; i < 20; i++)
        strcpy(queue[i], "");
    strcpy(queue[rear], "q0");
    rear++; strcpy(final_dfa[rows][0], "q0");
    while(strcmp(queue[front], "") != 0)
    {
        int temp_rows = rows;
        char new_states[20];
        for (int i = 0; i < 20; i++)
            strcpy(new_states, "");
        for (int j = 0; j < inputs; j++)
        {
            for (int i = 0; i < 20; i++)
                strcpy(new_states, "");
        for (int i = 1; i < strlen(queue[front]); i += 2)
        {
            if (isdigit(queue[front][i]))
            {
                int n = queue[front][i] - '0';
                for (int l = 1; l < strlen(nfa_table[n][j]); l +=
            2)
                {
                    int num1;
                    if (isdigit(nfa_table[n][j][l]))
                    {
                        num1 = nfa_table[n][j][l] - '0';
                    }
                    int flag2 = 0;
                    int num2;
                    for (int m = 1; m <
                    strlen(new_states); m += 2)
                    {
                        if (isdigit(new_states[m]))
                        {
                            num2 = new_states[m] -
                        '0';
                            if (num1 == num2)
                                flag2 = 1;
                        }
                    }
                    if (flag2 == 0)
                    {
                        char temp[20];
                        sprintf(temp, "q%d", num1);
                        strcat(new_states, temp);
                    }
                }
            }
        }
        int temp_states[20];
        int temp_index = 0;
        for (int d = 0; d < strlen(new_states); d++)
        {
            if (isdigit(new_states[d]))
            {
                temp_states[temp_index++] = new_states[d] -
            '0';
            }
        }
        for (int q = 0; q < temp_index; q++)
        {
            for (int r = 0; r < temp_index - q - 1; r++)
            {
                if (temp_states[r] > temp_states[r + 1])
                {
                    int swap = temp_states[r];
                    temp_states[r] = temp_states[r + 1];
                    temp_states[r + 1] = swap;
                }
            }
        }
        char tempstr[20];
        strcpy(new_states, "");
        for (int q = 0; q < temp_index; q++)
        {
            sprintf(tempstr, "q%d", temp_states[q]);
            strcat(new_states, tempstr);
        }
        int flag = 0;
        for (int a = 0; a < rear; a++)
        {
            if (strcmp(queue[a], new_states) == 0)
            {
                flag = 1;
            }
        }
        if (flag == 0)
        {
            strcpy(queue[rear], new_states);
            rear++;
            strcpy(final_dfa[++temp_rows][0], new_states);
        }
        strcpy(final_dfa[rows][j + 1], new_states);
    }
    rows++;
    front++; }
printf("\nDFA:\n");
printf("%-10s|", " ");
for (int i = 0; i < inputs; i++)
    printf("Input %-4d|", i + 1);
printf("\n");
for (int i = 0; i < 11 * (inputs + 1); i++)
    printf("%s", "=");
printf("\n");
for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < inputs + 1; j++)
    {
    printf("%-10s|", final_dfa[i][j]);
    }
    printf("\n");
    }
}
```

CODE

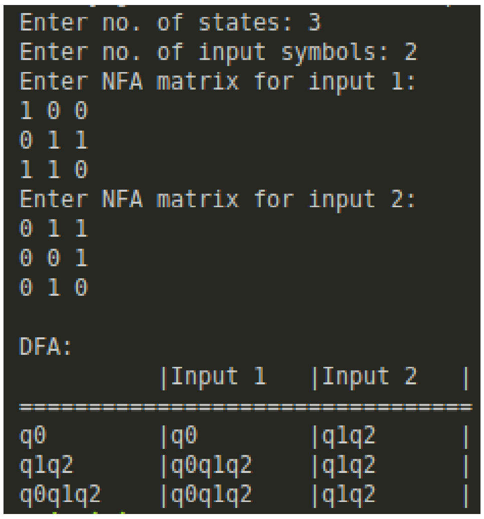


```c
#include <stdio.h>
#include <string.h>
int main() {
    char input[100];
    while(1){
        int flag=0;
        printf("Enter a string: ");
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = '\0';
        int len = strlen(input);
        int i = 0;
        if (len == 0) {
            printf("String recognized under 'a*'\n");
            flag=1;
        }
        for (i = 0; i < len; i++) {
            if (input[i] != 'a') {
                break;
            }
        }
        if (i == len) {
            printf("String recognized under 'a*'\n");
            flag=1;
        }
        if (len == 3 && strcmp(input, "abb") == 0) {
            printf("String recognized under 'abb'\n");
            flag=1;
        }
        int hasA = 0, hasB = 0;
        for (i = 0; i < len; i++) {
            if (input[i] == 'a') {
                hasA = 1;
            } else if (input[i] == 'b') {
                hasB = 1;
            } else {
                break;
            }
        }

        if ((hasA || !hasA) && hasB) {
            printf("String recognized under 'a*b+'\n");
            flag=1;
        }

        if(flag!=1)
            printf("String not recognized under any pattern\n");
    }
    return 0;
}
```

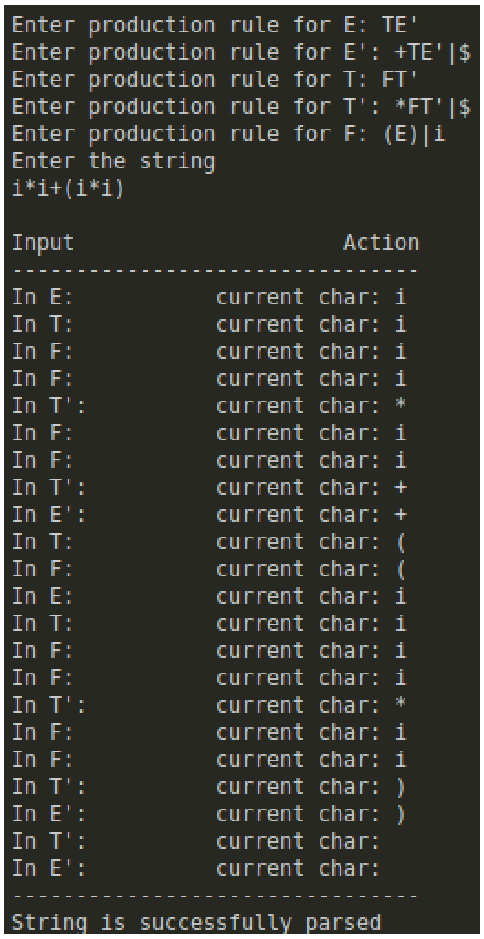
OUTPUT



```
Enter a string: a
String recognized under 'a*'
Enter a string: aaaaaaa
String recognized under 'a*'
Enter a string: ab
String recognized under 'a*b+'
Enter a string: b
String recognized under 'a*b+'
Enter a string: bbb
String recognized under 'a*b+'
```

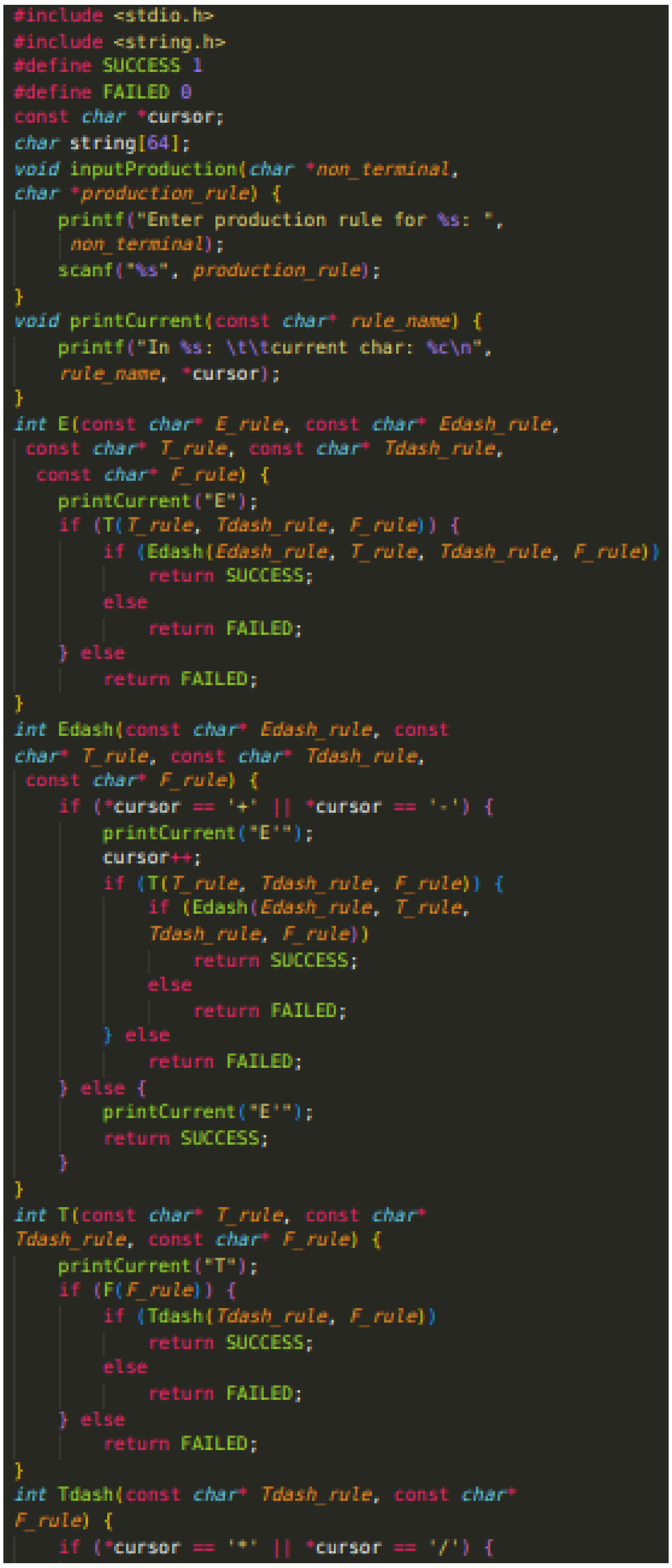


```
Enter no. of states: 3
Enter no. of input symbols: 2
Enter NFA matrix for input 1:
1 0 0
0 1 1
1 1 0
Enter NFA matrix for input 2:
0 1 1
0 0 1
0 1 0

DFA:
          |Input 1   |Input 2   |
=================================
q0        |q0        |q1q2      |
q1q2      |q0q1q2    |q1q2      |
q0q1q2    |q0q1q2    |q1q2      |
```

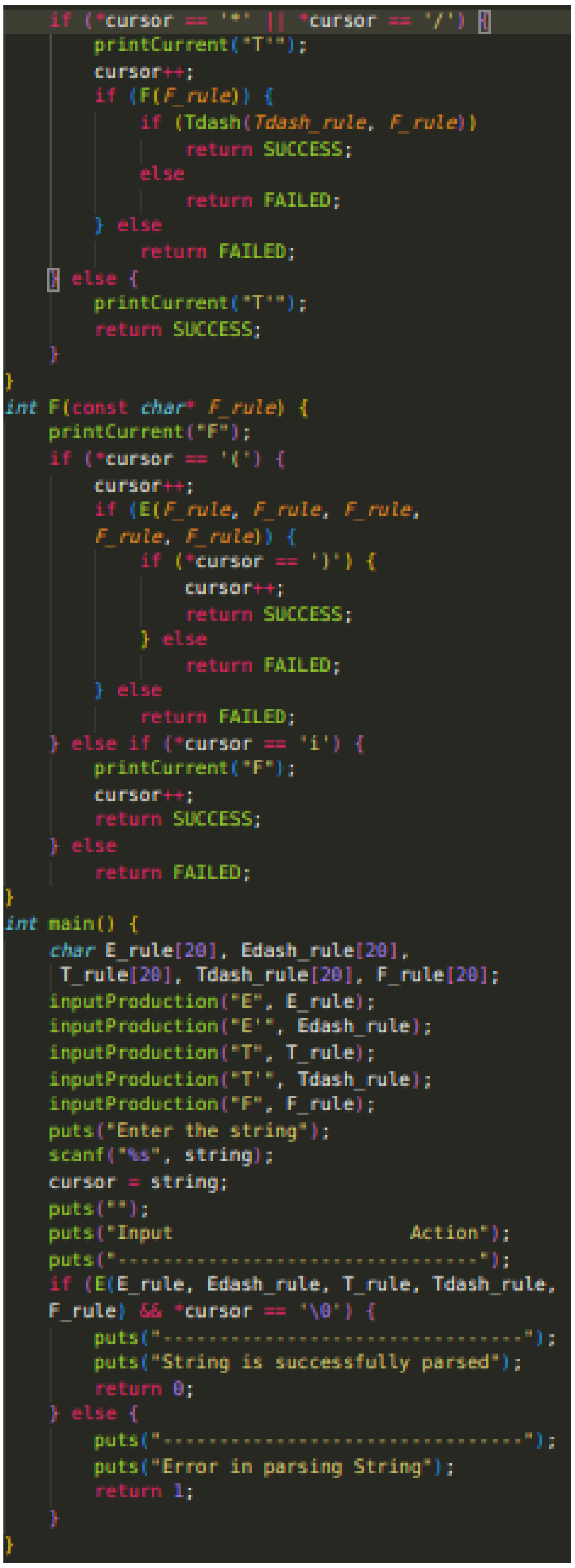


```
Enter production rule for E: TE'
Enter production rule for E': +TE'|$
Enter production rule for T: FT'
Enter production rule for T': *FT'|$
Enter production rule for F: (E)|i
Enter the string
i*i+(i*i)

Input                        Action
--------------------------------
In E:          current char: i
In T:          current char: i
In F:          current char: i
In F:          current char: i
In T':         current char: *
In F:          current char: i
In F:          current char: i
In T':         current char: +
In E':         current char: +
In T:          current char: (
In F:          current char: (
In E:          current char: i
In T:          current char: i
In F:          current char: i
In F:          current char: i
In T':         current char: *
In F:          current char: i
In F:          current char: i
In T':         current char: )
In E':         current char: )
In T':         current char:
In E':         current char:
--------------------------------
String is successfully parsed
```

CODE



```
#include <stdio.h>
#include <string.h>
#define SUCCESS 1
#define FAILED 0
const char *cursor;
char string[64];
void inputProduction(char *non_terminal,
char *production_rule) {
    printf("Enter production rule for %s: ",
     non_terminal);
    scanf("%s", production_rule);
}
void printCurrent(const char* rule_name) {
    printf("In %s: \t\tcurrent char: %c\n",
    rule_name, *cursor);
}
int E(const char* E_rule, const char* Edash_rule,
 const char* T_rule, const char* Tdash_rule,
  const char* F_rule) {
    printCurrent("E");
    if (T(T_rule, Tdash_rule, F_rule)) {
        if (Edash(Edash_rule, T_rule, Tdash_rule, F_rule))
            return SUCCESS;
        else
            return FAILED;
    } else
        return FAILED;
}
int Edash(const char* Edash_rule, const
char* T_rule, const char* Tdash_rule,
 const char* F_rule) {
    if (*cursor == '+' || *cursor == '-') {
        printCurrent("E'");
        cursor++;
        if (T(T_rule, Tdash_rule, F_rule)) {
            if (Edash(Edash_rule, T_rule,
            Tdash_rule, F_rule))
                return SUCCESS;
            else
                return FAILED;
        } else
            return FAILED;
    } else {
        printCurrent("E'");
        return SUCCESS;
    }
}
int T(const char* T_rule, const char*
Tdash_rule, const char* F_rule) {
    printCurrent("T");
    if (F(F_rule)) {
        if (Tdash(Tdash_rule, F_rule))
            return SUCCESS;
        else
            return FAILED;
    } else
        return FAILED;
}
int Tdash(const char* Tdash_rule, const char*
F_rule) {
    if (*cursor == '*' || *cursor == '/') {
```



```
    if (*cursor == '*' || *cursor == '/') {
        printCurrent("T'");
        cursor++;
        if (F(F_rule)) {
            if (Tdash(Tdash_rule, F_rule))
                return SUCCESS;
            else
                return FAILED;
        } else
            return FAILED;
    } else {
        printCurrent("T'");
        return SUCCESS;
    }
}
int F(const char* F_rule) {
    printCurrent("F");
    if (*cursor == '(') {
        cursor++;
        if (E(F_rule, F_rule, F_rule,
        F_rule, F_rule)) {
            if (*cursor == ')') {
                cursor++;
                return SUCCESS;
            } else
                return FAILED;
        } else
            return FAILED;
    } else if (*cursor == 'i') {
        printCurrent("F");
        cursor++;
        return SUCCESS;
    } else
        return FAILED;
}
int main() {
    char E_rule[20], Edash_rule[20],
     T_rule[20], Tdash_rule[20], F_rule[20];
    inputProduction("E", E_rule);
    inputProduction("E'", Edash_rule);
    inputProduction("T", T_rule);
    inputProduction("T'", Tdash_rule);
    inputProduction("F", F_rule);
    puts("Enter the string");
    scanf("%s", string);
    cursor = string;
    puts("");
    puts("Input                     Action");
    puts("--------------------------------");
    if (E(E_rule, Edash_rule, T_rule, Tdash_rule,
    F_rule) && *cursor == '\0') {
        puts("--------------------------------");
        puts("String is successfully parsed");
        return 0;
    } else {
        puts("--------------------------------");
        puts("Error in parsing String");
        return 1;
    }
}
```