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

int main()
{
    char str[100];
    int i=2,a=0;
    printf("Enter the String: \n");
    // scanf("%s", &str);
    fgets(str, 100, stdin);

    printf("\nOutput:\n");

    if(str[0]=='/'){
        if(str[1]=='/')
            printf("It is a Single Line comment\n");
        else if(str[1]=='*'){
            for(i=2;i<100;i++){
                if(str[i]=='*' && str[i+1]=='/'){
                    printf("It is a Multi-Line comment\n");
                    a=1;
                    break;
                }


            }
            if(a==0){
                printf("It is not a comment 1\n");
            }
        }
        else
            printf("It is not a comment 2\n");
    }
    else
        printf("It is not a comment 3\n");
}
```

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

```c
#include<stdlib.h>

int main(){
    char s[100];
    printf("Enter the string\n");

    scanf("%s", &s);

    int count = 0;
    if(strlen(s)==1){
        if(s[0]=='a'){
            printf("Accepted");
        }
        else if(s[0]=='b'){
            printf("Accepted");
        }
        else{
            printf("Not Accepted");
        }
    }
    else if(strlen(s)==3 && s[0]=='a' && s[0]=='b' && s[0]=='b' ){
         printf("Accepted");
    }

    else{

        for(int i = 0; i < strlen(s);i++){
            if(i<strlen(s)-1 && s[i]=='b' && s[i+1]=='a'){
                 printf("Not Accepted");
                 exit(0);
                 i = strlen(s);
            }
            if(s[i]=='b'){
                count++;
            }
        }

        if(count >=1){
            printf("Accepted");
        }
        else{
             printf("Not Accepted");
```

```
        }
    }

}
```

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

int main(){
    char s[50];
    printf("Enter string: ");
    // fgets(s, sizeof(s), stdin);
     scanf("%[^\n]",s);
    FILE * fp;
    fp = fopen("habibi.txt","w");



     for (int i = 0; i<strlen(s); i++) {

            fputc(s[i], fp);
        }



    //   fputs(s,fp);


        puts(s); // output the entire string;
        fclose(fp);
        return 0;
}
```

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

int main(){
    FILE * fp;
    fp = fopen("habibi.txt","r");

    char s[100];

    if(fp==NULL){
        printf("Khali hai");
    }

    printf("content hai\n");

    while(fgets(s,100,fp)!=NULL){
        printf("%s",s);
    }



    fclose(fp);
    return 0;
}
```

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

// Returns 'true' if the character is a DELIMITER.
bool isDelimiter(char ch)
{
    if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
        ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
```

```
        ch == '[' || ch == ']' || ch == '{' || ch == '}')
        return (true);
    return (false);
}

// Returns 'true' if the character is an OPERATOR.
bool isOperator(char ch)
{
    if (ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '=')
        return (true);
    return (false);
}

// Returns 'true' if the string is a VALID IDENTIFIER.
bool validIdentifier(char* str)
{
    if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
        str[0] == '3' || str[0] == '4' || str[0] == '5' ||
        str[0] == '6' || str[0] == '7' || str[0] == '8' ||
        str[0] == '9' || isDelimiter(str[0]) == true)
        return (false);
    return (true);
}

// Returns 'true' if the string is a KEYWORD.
bool isKeyword(char* str)
{
    if (!strcmp(str, "if") || !strcmp(str, "else") ||
        !strcmp(str, "while") || !strcmp(str, "do") ||
        !strcmp(str, "break") ||
         !strcmp(str, "continue") || !strcmp(str, "int")
        || !strcmp(str, "double") || !strcmp(str, "float")
        || !strcmp(str, "return") || !strcmp(str, "char")
        || !strcmp(str, "case") || !strcmp(str, "char")
        || !strcmp(str, "sizeof") || !strcmp(str, "long")
        || !strcmp(str, "short") || !strcmp(str, "typedef")
        || !strcmp(str, "switch") || !strcmp(str, "unsigned")
        || !strcmp(str, "void") || !strcmp(str, "static")
        || !strcmp(str, "struct") || !strcmp(str, "goto"))
        return (true);
```

```
    return (false);
}

// Returns 'true' if the string is an INTEGER.
bool isInteger(char* str)
{
    int i, len = strlen(str);

    if (len == 0)
        return (false);
    for (i = 0; i < len; i++) {
        if (str[i] != '0' && str[i] != '1' && str[i] != '2'
            && str[i] != '3' && str[i] != '4' && str[i] != '5'
            && str[i] != '6' && str[i] != '7' && str[i] != '8'
            && str[i] != '9' || (str[i] == '-' && i > 0))
            return (false);
    }
    return (true);
}

// Returns 'true' if the string is a REAL NUMBER.
bool isRealNumber(char* str)
{
    int i, len = strlen(str);
    bool hasDecimal = false;

    if (len == 0)
        return (false);
    for (i = 0; i < len; i++) {
        if (str[i] != '0' && str[i] != '1' && str[i] != '2'
            && str[i] != '3' && str[i] != '4' && str[i] != '5'
            && str[i] != '6' && str[i] != '7' && str[i] != '8'
            && str[i] != '9' && str[i] != '.' ||
            (str[i] == '-' && i > 0))
            return (false);
        if (str[i] == '.')
            hasDecimal = true;
    }
    return (hasDecimal);
}

// Extracts the SUBSTRING.
```

```
char* subString(char* str, int left, int right)
{
    int i;
    char* subStr = (char*)malloc(
                  sizeof(char) * (right - left + 2));

    for (i = left; i <= right; i++)
        subStr[i - left] = str[i];
    subStr[right - left + 1] = '\0';
    //printf("%s\n",subStr);
    return (subStr);
}

// Parsing the input STRING.
void parse(char* str)
{
    int left = 0, right = 0;
    int len = strlen(str);
    int count = 0;
    while (right <= len && left <= right) {
        if (isDelimiter(str[right]) == false)
            right++;

        if (isDelimiter(str[right]) == true && left == right) {
            if (isOperator(str[right]) == true)
            {
                printf("'%c' IS AN OPERATOR\n", str[right]);
                count++;

            }

            right++;
            left = right;
        }
         else if (isDelimiter(str[right]) == true && left != right
                  || (right == len && left != right)) {
            char* subStr = subString(str, left, right - 1);

            if (isKeyword(subStr) == true){
                printf("'%s' IS A KEYWORD\n", subStr);
                count++;
```

```
            }

            else if (isInteger(subStr) == true){

                printf("'%s' IS AN INTEGER\n", subStr);
                count++;
            }

            else if (isRealNumber(subStr) == true){
                printf("'%s' IS A REAL NUMBER\n", subStr);
                count++;

            }

            else if (validIdentifier(subStr) == true
                     && isDelimiter(str[right - 1]) == false){
                printf("'%s' IS A VALID IDENTIFIER\n", subStr);
                count++;

                     }

            else if (validIdentifier(subStr) == false
                     && isDelimiter(str[right - 1]) == false)
                printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);
            left = right;
        }
    }
    printf("You entered: %d", count);

    return;
}

// DRIVER FUNCTION
int main()
{
     // maximum length of string is 100 here
    char str[100] = "int a = 70 + 1c; ";
    // int count = 0;

    parse(str); // calling the parse function
```

```
    return (0);
}
```

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

int z = 0, i = 0, j = 0, c = 0;

char a[16], ac[20], stk[15], act[10];

void check()
{

    strcpy(ac, "REDUCE TO E -> ");

    for (z = 0; z < c; z++)
    {

        if (stk[z] == '4')
        {
            printf("%s4", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';

            printf("\n$%s\t%s$\t", stk, a);
        }
    }

    for (z = 0; z < c - 2; z++)
    {

        if (stk[z] == '2' && stk[z + 1] == 'E' &&
            stk[z + 2] == '2')
        {
```

```
            printf("%s2E2", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }

    for (z = 0; z < c - 2; z++)
    {

        if (stk[z] == '3' && stk[z + 1] == 'E' &&
            stk[z + 2] == '3')
        {
            printf("%s3E3", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 1] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    return;
}

int main()
{
    printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

    strcpy(a, "32423");

    c = strlen(a);

    strcpy(act, "SHIFT");

    printf("\nstack \t input \t action");

    printf("\n$\t%s$\t", a);

    for (i = 0; j < c; i++, j++)
    {
```

```
        printf("%s", act);

        stk[i] = a[j];
        stk[i + 1] = '\0';

        a[j] = ' ';

        printf("\n$%s\t%s$\t", stk, a);

        check();
    }

    check();

    if (stk[0] == 'E' && stk[1] == '\0')
        printf("Accept\n");
    else // else reject
        printf("Reject\n");
}
```

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

// function f to exit from the loop
// if given condition is not true
void f()
{
    printf("Not operator grammar");
    exit(0);
}

void main()
{
    char grm[20][20], c;

    // Here using flag variable,
    // considering grammar is not operator grammar
```

```
    int i, n, j = 2, flag = 0;

    // taking number of productions from user
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%s", grm[i]);

    for (i = 0; i < n; i++) {
        c = grm[i][2];

        while (c != '&#092;&#048;') {

            if (grm[i][3] == '+' || grm[i][3] == '-'
                || grm[i][3] == '*' || grm[i][3] == '/')

                flag = 1;

            else {

                flag = 0;
                f();
            }

            if (c == '$') {
                flag = 0;
                f();
            }

            c = grm[i][++j];
        }
    }

    if (flag == 1)
        printf("Operator grammar");
}
```