

SYSTEM SOFTWARES**ASSIGNMENT - 06**

Write a program to implement Lexical Analyzer(Lexer).

CODE=>

```
// U19CS009
// Brijesh Rohit

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

// STRUCTION TO STORE TOKENS
struct Token
{
    char name[1024];
    char type[128];
};

struct Token tokens[2048];
int token_count = 0;

// DELIMITERS FOR TOKENISING
int isDelimiter(char ch)
{
    if (ch == ' ')
        return 1;
    FILE *fd = fopen("delimiter.txt", "r");
    char c;
    while (fscanf(fd, "%c", &c) != EOF)
    {
        if (ch == c)
        {
            fclose(fd);
            return 1;
        }
    }
    fclose(fd);
    return 0;
}

// FUNCTION TO CHECK IF STRING IS OPERATOR OR NOT
int isOperator(char *ch)
{
    if (strlen(ch) == 0)
        return 0;
```

```
if (!strcmp(ch, "") || !strcmp(ch, "\n"))
    return 0;
```

```
char *c;
FILE *f = fopen("operator.txt", "r");
char tmp[10];
strcpy(tmp, ch);
while (fscanf(f, "%s", c) != EOF)
{
    if (!strcmp(tmp, c))
    {
        fclose(f);
        return 1;
    }
}
fclose(f);
return 0;
}
```

```
// FUNCTION TO CHECK IF THE STRING IS PUNCTUATOR OR NOT
```

```
int isPunctuator(char *ch)
{
```

```
    if (strlen(ch) == 0)
        return 0;
```

```
    if (!strcmp(ch, "") || !strcmp(ch, "\n"))
        return 0;
```

```
    char *c;
    FILE *f = fopen("punctuator.txt", "r");
    char tmp[10];
    strcpy(tmp, ch);
    while (fscanf(f, "%s", c) != EOF)
    {
        if (!strcmp(tmp, c))
        {
            fclose(f);
            return 1;
        }
    }
    fclose(f);
    return 0;
}
```

```
// FUNCTION TO CHECK IF THE STRING IS KEYWORD OR NOT
```

```
int iskeyword(char *str)
{
```

```
    FILE *fk = fopen("keywords.txt", "r");
    char s[20];
    while (fscanf(fk, "%s", s) != EOF)
```

```

{
    if (!strcmp(str, s))
    {
        fclose(fk);
        return 1;
    }
}
fclose(fk);
return 0;
}

```

// FUNCTION TO CHECK IF THE STRING IS VALID IDENTIFIER OR NOT

```

int isValidIdentifier(char *str)
{
    if (strlen(str) == 0)
        return -1;
    if ((str[0] >= '0' && str[0] <= '9') || isDelimiter(str[0]) == 1 ||
isKeyword(str))
        return 0;
    if (str[0] == '\\0' || str[0] == '\\n')
        return -1;
    return 1;
}

```

// FUNCTION TO CHECK IF THE STRING IS AN INTEGER OR NOT

```

int isInt(char *str)
{
    int i, len = strlen(str);

    if (len == 0)
        return (0);
    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 (0);
    }
    return (1);
}

```

// FUNCTION TO CHECK IF THE STRING IS DECIMAL OR NOT

```

int isDecimal(char *str)
{
    int i = 0;
    int flag = 0;
    while (str[i] != '\\0')
    {
        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 0;
        if (str[i] == '.')

```

```

        flag = 1;
        i++;
    }
    return flag;
}

// FUNCTION TO CHECK IF THE STRING IS CONSTANT OR NOT
int isConst(char *str)
{
    if (isInt(str))
        return 1;
    if (isDecimal(str))
        return 1;
    if (strlen(str) < 3)
        return 0;
    if (str[0] == '\\' && str[strlen(str) - 1] == '\\')
    {
        return 1;
    }
}

```

```

// FUNCTION TO CHECK IF THE STRING IS STRING OR NOT
int isString(char *str)
{
    if (strlen(str) < 3)
        return 0;
    // printf("%d\t%s",strlen(str),str);
    if (str[0] == '"' && str[strlen(str) - 1] == '"')
    {
        return 1;
    }
    return 0;
}

```

```

// FUNCTION TO EXTRACT SUBSTRING
char *sbstr(char *str, int l, int r)
{
    int i;
    char *str1 = (char *)malloc(sizeof(char) * (r - l + 2));

    for (i = l; i <= r; i++)
        str1[i - l] = str[i];
    str1[r - l + 1] = '\0';
    return (str1);
}

```

```

// TOKENIZING FUNCTION
void tokenise(char *str)
{
    int l = 0, r = 0, len = strlen(str);
    int flag = 0;
    while (l <= r && r <= len)
    {

```

```

// CHECKING FOR STRINGS
// STRINGS ARE BOUNDED BY DOUBLE QUOTES
if (str[r] == '"' && flag == 0)
{
    flag = 1;
    r++;
}
if (flag)
{
    if (str[r] == '"')
    {
        flag = 0;
        char *str1 = sbstr(str, l, r);
        if (isString(str1) == 1)
            strcpy(tokens[token_count].name, str1);
        strcpy(tokens[token_count++].type, "String");
    }
    r++;
    continue;
}

```

```

// IF NOT DELIMITER THEN CHECK FOR NEXT CHARACTER
if (isDelimiter(str[r]) == 0)
    r++;
// IF DELIMITER AND SINGLE CHARACTER
if (isDelimiter(str[r]) == 1 && l == r)
{
    // CHECKING IF THE CHARACTER IS PUNCTUATOR OR NOT
    if (str[r] != '.')
    {
        char *st = sbstr(str, r, r);
        if (isPunctuator(st))
        {
            strcpy(tokens[token_count].name, st);
            strcpy(tokens[token_count++].type, "Punctuator");
            r++;
            l = r;
            continue;
        }
    }
    // IF STRING IS '...' THEN ADD IT TO PUNCTUATOR
    else if (str[r + 1] == '.' && str[r + 2] == '.')
    {

```

```

        strcpy(tokens[token_count].name, "...");
        strcpy(tokens[token_count++].type, "Punctuator");
        r += 3;
        l = r;
        continue;
    }
    r++;
    // EXTRACT STRING OF CONTINUOUS OPERATORS
    while (isDelimiter(str[r]) && str[r] != ' ')

```

```

    {
        r++;
    }
    char *str1 = sbstr(str, l, r - 1);
    // CHECK IF IT IS AN OPERATOR OR NOT
    if (isOperator(str1) == 1)
    {
        strcpy(tokens[token_count].name, str1);
        strcpy(tokens[token_count++].type, "Operator");
    }

    l = r;
}
else if (isDelimiter(str[r]) == 1 && l != r || (r == len && l != r))
{
    char *str1 = sbstr(str, l, r - 1);

    // CHECK FOR KEYWORD
    if (isKeyword(str1) == 1)
    {
        strcpy(tokens[token_count].name, str1);
        strcpy(tokens[token_count++].type, "keyword");
    }
    // CHECK FOR CONSTANTS
    else if (isConst(str1) == 1)
    {
        strcpy(tokens[token_count].name, str1);
        strcpy(tokens[token_count++].type, "Constant");
    }
    // CONTINUE IF STRING AS ALREADY FOUND ABOVE
    else if (isString(str1) == 1)
    {
        l = r;
        continue;
    }
    // CHECK FOR VALID IDENTIFIER
    else if (isValidIdentifier(str1) == 1 && isDelimiter(str[r - 1]) ==
0)
    {
        strcpy(tokens[token_count].name, str1);
        strcpy(tokens[token_count++].type, "Identifier");
    }
    l = r;
}
}

```

```

return;

```

```

int main()

```

```

{
    char str[100];
    // OPENING INPUT FILE

```

```

FILE *f = fopen("input.txt", "r");
int flag = 0, i;
// READING LINE BY LINE
while (fgets(str, 100, f))
{
    // TO REMOVE COMMENTS
    flag = 0;
    int end = strlen(str);
    for (i = 0; i < strlen(str) - 1; i++)
    {
        if (str[i] == '/' && str[i + 1] == '/')
        {
            flag = 1;
            end = i - 1;
            break;
        }
    }
    if (flag)
    {
        char *tmp;
        tmp = sbstr(str, 0, end);
        strcpy(str, tmp);
    }
    // TOKENISING AFTER REMOVING COMMENTS
    tokenise(str);
}
// CLOSING FILE
fclose(f);

```

```

// PRINTING TOKENS
printf("\nToken Type          Token Name \n");
char type[6][15] = {"Operator", "Punctuator", "Keyword", "Identifier",
"Constant", "String"};
for (i = 0; i < 6; i++)
{
    int j = strlen(type[i]);
    printf("\n%s", type[i]);
    while (j < 15)
    {
        printf(" ");
        j++;
    }
    printf(":  ");
    for (j = 0; j < token_count; j++)
    {
        if (!strcmp(tokens[j].type, type[i]))
            printf(" [%s]", tokens[j].name);
    }
}

```

```

printf("\n\nTotal Number of Tokens : %d\n", token_count);
return 0;

```

```

}

```

INPUT FILE=>

```
input.txt
1 //U19CS009
2 //Brijesh Rohit
3
4 #include<stdio.h>
5
6 int main()
7 {
8     printf("Hello World!");
9     printf("\n");
10    int a = 10;
11    float b = 1.2;
12    char name[] = {'b','r','i','j','e','s','h','\0'};
13    return 0;
14 }
```

OUTPUT=>

Delimiters :	Keywords :	Operators :	Punctuations :
<pre>delimiter.txt 1 2 - 3 * 4 / 5 , 6 ; 7 > 8 < 9 = 10 (11) 12 [13] 14 { 15 }</pre>	<pre>keywords.txt 1 auto 2 break 3 case 4 char 5 const 6 continue 7 default 8 do 9 double 10 else 11 enum 12 extern 13 float 14 for 15 goto 16 if 17 int 18 long 19 register 20 return 21 short 22 signed 23 sizeof 24 static 25 struct 26 switch 27 typedef 28 union 29 unsigned 30 void 31 volatile 32 while</pre>	<pre>operator.txt 1 2 - 3 ! 4 ~ 5 ++ 6 -- 7 * 8 / 9 % 10 << 11 >> 12 < 13 <= 14 > 15 >= 16 == 17 != 18 & 19 ^ 20 21 && 22 23 ?: 24 = 25 += 26 -= 27 *= 28 /= 29 %= 30 >>= 31 <<= 32 &= 33 ^= 34 = 35 sizeof</pre>	<pre>punctuator.txt 1 [2] 3 (4) 5 { 6 } 7 , 8 : 9 ; 10 ... 11 #</pre>

OUTPUT :

```
C:\Users\brije\Desktop\ss-assign06>gcc -o make u19cs009-ss-assign06-lex.c
C:\Users\brije\Desktop\ss-assign06>make

Token Type      Token Name
Operator        :    [<],
Punctuator      :    [(), []], [{}, [(), []], [;], [(), []], [;], [;], [;], [[]], [[]], [,], [,],
, [,], [,], [,], [,], [}], [;], [;], [}],
Keyword         :    [int], [int], [float], [char],
Identifier      :    [#include], [stdio.h], [main], [ printf], [printf], [a], [b], [name],
[ return],
Constant        :    [10], [1.2], ['b'], ['r'], ['i'.'j'], ['e'], ['s'], ['h'], ['\0'], [0],
String          :    ["Hello World!"], ["\n"],

Total Number of Tokens : 49
```