Week-2 Assignment

**Q:** Implement lexical analyzer using C for recognizing the following tokens:

* A minimum of 10 keywords of your choice
* Identifiers with the regular expression : letter(letter | digit)\*
* Integers with the regular expression: digit+
* Relational operators: <, >, <=, >=, ==, !=
* Storing identifiers in symbol table.
* Using files for input and output.

**Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

enum Tokentype

{

    KEYWORD,

    IDENTIFIERS,

    INTEGERS,

    OPERATORS,

    SPECIAL\_SYMBOLS,

};

enum Tokentype getTokentype(char \*given\_lexeme)

{

    char \*keywords[11] = {"printf", "void", "return", "int", "while", "if", "else", "for", "break", "char", "main"};

    int keywords\_size = 11;

    char \*special\_symbols[9] = {"(", ")", "{", "}", "[", "]", ";", ",", "."};

    int special\_symbol\_size = 9;

    char \*operators[13] = {"+", "-", "/", "%%", "\*", "<", ">", "==", "=", "+=", "-=", "++", "--"};

    int operators\_size = 13;

    for (int i = 0; i < keywords\_size; i++)

    {

        if (strcmp(given\_lexeme, keywords[i]) == 0)

        {

            return KEYWORD;

        }

    }

    for (int i = 0; i < special\_symbol\_size; i++)

    {

        if (strcmp(given\_lexeme, special\_symbols[i]) == 0)

        {

            return SPECIAL\_SYMBOLS;

        }

    }

    for (int i = 0; i < operators\_size; i++)

    {

        if (strcmp(given\_lexeme, operators[i]) == 0)

        {

            return OPERATORS;

        }

    }

    int is\_a\_number = 1;

    for (int i = 0; i < strlen(given\_lexeme); i++)

    {

        if (!isdigit(given\_lexeme[i]))

        {

            is\_a\_number = 0;

            break;

        }

    }

    if (is\_a\_number && strlen(given\_lexeme) > 0)

    {

        return INTEGERS;

    }

    int is\_identifier = 1;

    for (int i = 0; i < strlen(given\_lexeme); i++)

    {

        if (i == 0)

        {

            if (isdigit(given\_lexeme[i]))

            {

                is\_identifier = 0;

                break;

            }

        }

        else

        {

            if (isalpha(given\_lexeme[i]) || isdigit(given\_lexeme[i]))

            {

                continue;

            }

            else

            {

                is\_identifier = 0;

                break;

            }

        }

    }

    if (is\_identifier)

    {

        return IDENTIFIERS;

    }

}

void main()

{

    FILE \*file = fopen("code.txt", "r");

    FILE \*file2 = fopen("output.txt", "w");

    if (file == NULL)

    {

        printf("Unable to open the file.\n");

        return;

    }

    if (file2 == NULL)

    {

        printf("Unable to opent the error.\n");

        return;

    }

    char lexeme[20];

    while (fscanf(file, "%s", lexeme) != EOF)

    {

        enum Tokentype t = getTokentype(lexeme);

        switch (t)

        {

        case KEYWORD:

            fprintf(file2, lexeme);

            fprintf(file2, " : keyword\n");

            break;

        case IDENTIFIERS:

            fprintf(file2, lexeme);

            fprintf(file2, " : identifers\n");

            break;

        case INTEGERS:

            fprintf(file2, lexeme);

            fprintf(file2, " : integers\n");

            break;

        case OPERATORS:

            fprintf(file2, lexeme);

            fprintf(file2, " : operators\n");

            break;

        case SPECIAL\_SYMBOLS:

            fprintf(file2, lexeme);

            fprintf(file2, " : special\_symbols\n");

            break;

        }

    }

    fclose(file);

    fclose(file2);

}

**Input:**

int main ( ) {

int a = 4 ;

int c = 5 ;

int b = a \* c ;

printf ( c ) ;

}

**Output:**

int : keyword

main : keyword

( : special\_symbols

) : special\_symbols

{ : special\_symbols

int : keyword

a : identifers

= : operators

4 : integers

; : special\_symbols

int : keyword

c : identifers

= : operators

5 : integers

; : special\_symbols

int : keyword

b : identifers

= : operators

a : identifers

\* : operators

c : identifers

; : special\_symbols

printf : keyword

( : special\_symbols

c : identifers

) : special\_symbols

; : special\_symbols

} : special\_symbols