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:

```c
#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