Compiler design lab - Assignment 1

Lexical analysis is the first phase of the compilation process. In this phase, the source code of a language is accepted as input. It is the process where the stream of characters that make up the source program is read from left to right and grouped into tokens, by removing any whitespace or comments, resulting in a token stream.

Similar to C, In this course we will be designing constructs and subcontractors of a language compiler, we start with the first of compilers, i.e designing a tokenizer.

There are five classes of tokens in MiniC: Operators, Separators, Identifiers, Keywords and Constants. The MiniC compiler forms the longest tokens possible when it collects characters in left-to right order. Adjacent tokens may be separated by whitespace characters or comments.

1. Operators
    a. Simple operators: ! (not operator) + (Plus Sign), ∗ (multiply), − (Minus), = (Equal Sign), | (or operator), < (Less-than Sign), > (Greater-than Sign), / (divide)
    b. Compound assignment operators: += , −= , *= , /= , >= , <=
    c. Other compound operators: ++, --, ==, !=
    d. Regular expression : "<=" | "==" | "=" | "++" | "-" | "+"

2. Separators
    a. Separator characters are: ( ) [ ] ; :
    b. Regular expression : [() {} | , ;]

3. Identifiers
    a. An identifier, or name, is a sequence of latin capital and small letters, digits, and underscore character. An identifier must not begin with a digit, and it must not have the same spelling as a keyword. Identifiers are case sensitive and the maximum length of an identifier is 255 characters.
    b. Regular expression : [a-z A-Z _][a-z A-Z 0-9 _]*

4. Keywords
    a. They must not be used as ordinary identifiers. E.g. char, else, if, int, return, void, and while.
    b. Regular expression : "int" | "float"

5. Constants
    a. There are three different kinds of constants in MiniC: integers, characters, and strings.
    b. Regular Expression for integers : [0-9]* "." [0-9]$^{+}$
    c. Regular expression for float : [0-9]$^{+}$

Note: Both Input and Output should be taken in files only.

Sample Input file:

int a = 0;

Sample Output file:

- **Abbreviations to follow :**
    - Keyword - KY
    - Identifier - ID
    - Operator - OP
    - Constant - CT
    - Separator - SP

- Sample Outputted file for the above program-
    - **Format : <Token, Lexim>**
    - <KY, int>
    - <ID, a>
    - <OP, =>
    - <CT, 0>
    - <SP, ;>
    - Total number of tokens in the above program is 5.

Sample Lex Program: To identify and count total number of tokens.

```
%{
int n = 0 ;
%}

%%
"while" | "if" | "else" {n++; printf("\t keywords : %s", yytext);}
"Int" | "float" {n++; printf("\t keywords : %s", yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {n++; printf("\t identifier : %s", yytext);}
"<=" | "==" | "=" | "++" | "-" | "*" | "+" {n++; printf("\t operator : %s", yytext);}
[() {} | , ;] {n++; printf("\t separator : %s", yytext);}
[0-9]* "." [0-9]+ {n++; printf("\t float : %s", yytext);}
[0-9]+ {n++; printf("\t integer : %s", yytext);}
```

```
. ;
%%
int main()
{
        yylex();
        printf("\n total no. of token = %d\n", n);
}
```