Department of Computer Science and Engineering

Course Code: CSE 420	**Credits: 1.5**
Course Name: Compiler Design	**Semester: Summer 2023**

1 Introduction

In this assignment we are going to construct a lexical analyzer. Lexical analysis is the process of scanning the source program as a sequence of characters and converting them into sequences of tokens. A program that performs this task is called a lexical analyzer or a lexer or a scanner. For example, if a portion of the source program contains int x=5; the scanner would convert in a sequence of tokens like <INT><ID><ASSIGNOP><COST_NUM><SEMICOLON>. You will also mention the associated symbol where applicable.

We will construct a scanner for a subset of Python language (a python function). The task will be performed using a tool named flex (Fast Lexical Analyzer Generator) which is a popular tool for generating scanners.

2 Tasks

You have to perform the following tasks in this assignment

2.1 Identifying Tokens

2.1.1 Keywords

You have to identify the keywords given in Table 1 and print the token in the output file. For example, you will have to print **<IF>** in case you find the keyword **"if"** in the source program.

Keyword	Token	Keyword	Token	Keyword	Token
if	IF	try	TRY	and	AND
else	ELSE	except	EXCEPT	or	OR
for	FOR	True	TRUE	print	PRINT
range	RANGE	False	FALSE	in	IN
break	BREAK	def	DEF	continue	CONTINUE
not	NOT	return	RETURN	pass	PASS
catch	CATCH	finally	FINALLY	None	NONE
elif	ELIF	WHILE	while	do	DO

Table 1 : Keyword List

2.1.2 Constants

You have to identify constants using regular expressions.

a. **Integer Literals:** One or more consecutive digits form an integer literal. Type of token will be **CONST_INT**. Note that + or - will not be the part of an integer.

   i. **Binary Numbers:** You need to identify binary numbers in your code. Examples: 0b0110, 0b1011 etc. Type of token will be **CONST_BIN**
   ii. **Octal Numbers:** You need to identify octal numbers in your code. Examples: 0o01234567, 0o626103 etc. Type of token will be **CONST_OCT**
   iii. **Hexadecimal Numbers:** You need to identify hexadecimal numbers in your code. Examples: 0x0123456789ABCDEF, 0x5265AbD2653F, 0x0123456789abcdef, 0x5265Abd2653f etc. Type of token will be **CONST_HEX**

b. **Floating Point Literals:**
   i. Numbers like 3.147 and .3147 will be considered ad floating point constants. In this case, token type will be **CONST_FLOAT**.
   ii. You also need to capture the scientific notations of floating point numbers, e.g. 3.14159E-10, 3.14159E10 etc. Type of token will be **CONST_FLOAT_S**

2.1.3 Operators and Punctuators

The operator list for the subset of the python language we are dealing with is given in Table 2. A token in the form of <Type> along with the particular symbol should be printed in the output log file.

Symbols	**Type**
+, -	ADDOP
++, - -	INCOP
<, >, ==, <=, >=, !=	RELOP
=	ASSIGNOP
&&, ||	LOGICOP
!	NOT
(	LPAREN
)	RPAREN
{	LCURL
}	RCURL
[	LTHIRD
]	RTHIRD
,	COMMA
:	COLON

Table 2: Operators and Punctuators List

2.1.4 Identifiers

Identifiers are names given to entities, such as variables, functions, structures etc. An identifier can only have alphanumeric characters (a-z, A-Z, 0-9) and underscore (_). The first character of an identifier can only contain alphabet (a-z, A-Z) or underscore (_). For any identifier encountered in the input file you have to print the token <ID> along with the symbol.

2.1.5 White Space and Newlines

You have to capture the white spaces and newlines in the input file, but no actions needed regarding this.

2.1.6 Scope count

You have to count the number of tabs in each line and number of tabs will indicate a scope number. Refer to the sample input output for better understanding.

3 Input

The input will be a text file containing a python source program. File name will be given from the command line.

4 Output

In this assignment, there will be one output file. The output file should be named as <Your_student_ID>_log.txt. Here, you will output all the tokens as well the line number where it was found.

For example, after detecting any lexeme except one representing white spaces you will print a line containing **Scope No. <scope_count>: Token <Token> Lexeme <Lexeme> found.** For example, if you find an identifier **abcd** at line no 5 of your source code, you will print
Scope No. 2: Token <ID> Lexeme abcd found.

For more clarification about input output please refer to the sample input output file given in the lab folder. You are highly encouraged to produce output exactly like the sample one.

5 Submission

1. In your local machine create a new folder whose **name is your student id.**
2. Put the lex file named as **<your_student_id>.l** containing your code. **DO NOT** put the generated lex.yy.c file or any executable file in this folder.
3. Compress the folder in a **.zip file** which should be **named as your student id**.
4. Submit the .zip file.

Failure to follow these instructions will result in penalty.