Bangladesh University of Engineering and Technology (BUET)
Department of Computer Science and Engineering (CSE)
CSE310: Compiler Sessional
Session: July 2023

Assignment 2
Lexical Analysis

December 7, 2023

1 Introduction

In this assignment, we are going to construct a lexical analyzer, also known as a tokenizer. *Lexical Analysis* is the process of scanning the source program as a sequence of characters and converting them into a sequence of tokens. A program that performs this task is called a *Lexical Analyzer* or a *Tokenizer* or a *Lexer* or a *Scanner*. For example, if a portion of the source program contains a statement like, [redacted]he scanner will convert it into a sequence of tokens like [redacted]

After *successfully* completing the implementation of a simple symbol table, we will, now, construct a scanner for a subset of the C programming language. This task will be done using a tool named *Flex* (Fast Lexical Analyzer Generator) which is a widely used tool for conveniently generating scanners.

2 Tasks

You have to complete the following tasks in this assignment for the implementation of a working scanner.

2.1 Tokens Identification

2.1.1 Keywords

You have to identify the keywords listed in **Table 1** and print the corresponding [redacted]pairs in the output file. For example, [redacted] f[redacted]in the source program[redacted]
[redacted]

Keyword	Token	Keyword	Token
`if`	`IF`	`else`	`ELSE`
`for`	`FOR`	`while`	`WHILE`
`do`	`DO`	`break`	`BREAK`
`int`	`INT`	`char`	`CHAR`
`float`	`FLOAT`	`double`	`DOUBLE`
`void`	`VOID`	`return`	`RETURN`
`switch`	`SWITCH`	`case`	`CASE`
`default`	`DEFAULT`	`continue`	`CONTINUE`

Table 1: Keywords List

2.1.2 Constants

[redacted] the output file [redacted]

- **Integer Literals:** One or more consecutive digits form an integer literal. In this case, the [redacted] Note [redacted]
- **Floating-point Literals:** Numbers like `3.14159`, `3.14159E-10`, `.314159`, and `314159E10` will be considered as floating-point literals or constants. [redacted]
- **Character Literals:** C [redacted] There will be a single character within the single quotation enclosing with the exception of `'\''` (single quotation), `'\"'` (double quotation), `'\n'`, `'\t'`, `'\\'`, `'\''`, `'\a'`, `'\f'`, `'\r'`, `'\b'`, `'\v'`, and `'\0'`. For character literals, t [redacted] Note that you need to convert the detected *lexeme* into an actual character. For example, if you find `'a'` inside a source program, then you need to print `<CONST_CHAR, a>`. This means that we only need the actual character represented (ASCII code), not the quotation symbols around it. Similarly, you need a *newline* character (ASCII code of which is `10`) in your token if you detect `'\n'`.

2.1.3 Operators and Punctuators

The operators from the subset of the C programming language that we will be dealing with in this assignment are listed in **Table 2**. A token in the [redacted]hould be printed in the output file. [redacted]

Symbols	Type
+, -	ADDOP
*, /, %	MULOP
++, --	INCOP
<, <=, >, >=, ==, !=	RELOP
=	ASSIGNOP
&&, ||	LOGICOP
&, |, ^, <<, >>	BITOP
!	NOT
(	LPAREN
)	RPAREN
{	LCURL
}	RCURL
[	LSQUARE
]	RSQUARE
,	COMMA
;	SEMICOLON

Table 2: Operators and Punctuators List

2.1.4 Identifiers →(later)

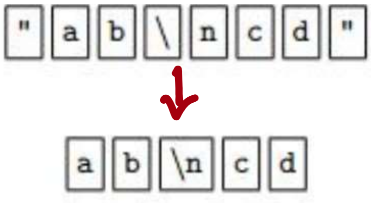
Identifiers are the names given to entities in the C programming language such as variables, functions, structures, etc. [redacted] a [redacted]he first character of an identifier can be either an alphabet (`A-z`, `a-z`) or an underscore. [redacted]

2.1.5 Strings →(State)

String literals are [redacted] String can [redacted] A m [redacted] line. Y[redacted]if you encounter a string “hello” in the input file. [redacted]

```
“This is an example of a single line string”;
“This is an example\
of a multiline\
string”;
```

Note that, just like character literals, you need to convert the special characters into their original ASCII values. [redacted] for the newline character. For example, if the source program contains the following eight (8) characters, then the scanner will convert them into the five (5) characters as depicted in the diagram below.



2.1.6 Comments → (State)

Comments can consist of a single line or multiple lines. A s[redacted]. However, a[redacted]s w[redacted]. A multiline comment s[redacted] If there is any comment in the input file, then you have to recognize it, [redacted]

```
// This is a single line comment
// This is a multiline comment\
starting with double slash
/* This is another multiple line comment
starting and ending with slash and asterisk */
```

2.1.7 Whitespaces

[redacted] You may want to check the [redacted]

2.2 Line Count

[redacted]

2.3 Lexical Error Detection

[redacted] You will detect the following types of lexical errors.

- Appearance of redundant decimal points in a number like `3.1.4159`.
- Ill-formed numbers such as `1E10.7`.
- Invalid suffixes in numeric constants or invalid prefixes in identifiers like `12abcd`.
- Appearance of multiple characters in a character literal like `'ab'`.
- Unfinished character literal such as `'a` or `'\'`. You may explore the sample files to see what to do with empty characters `''`.
- Unfinished string. You may explore different cases where a string remains unfinished.
- Unfinished comment.
- Unrecognized character.

Also, [redacted]

2.4 Wrong Indentation Detection → later

The lexical analyzer, being the primary component scanning user-provided code in a compiler, can potentially assess code quality aspects such as formatting and indentation. For this task, [redacted] that notifies users of [redacted].

```
1 #include <stdio.h>

2 int main() {}
3     printf("Welcome, ");
4         printf("to, "); if(1) {printf("flex!");}
5   return 0;

6 }
```

The above-mentioned code should give a warning - “Line no 4: warning, 1 of tabs needed but got 2 tabs.” and “Line no 5: Warning, tab required but got space.”

3 Input — Done

The input to the lexical analyzer will be a text file containing a source program written in C programming language. The input file name will have to be provided from the terminal/command line.

4 Output

There will be t[redacted]This file should be named as `<your_student_id>_token.txt`. For example, a student with ID `2005123` will name the output file for tokens as `2005123_token.txt`. You will output all the corresponding tokens in this file.

T[redacted]named as `<your_student_id>_log.txt`. In this file, you will log all the actions performed inside the source program. [redacted].

```
Line# <line_count>: Token <TOKEN> Lexeme <LEXEME> found
```

For example, if you encounter a comment such as `//hello` at line `5` in the source program, then you will print as follows.

```
Line# 5: Token <COMMENT> Lexeme <hello> found
```

[redacted]ou will pr[redacted]. For any insertion into thesymbol table, you have to print the current symbol table in the `log.txt` file.[redacted]f a s[redacted]For any detected error, you have to print the following message in the `log.txt` file.

```
Error at line# <line_count>: <corresponding_error_message>
```

Y[redacted]

For further clarification about input and output, kindly refer to the provided sample input and output files. **You are highly encouraged to produce the output exactly like the sample output.**

5 Submission Guidelines

All submissions will be taken only via Moodle. Please, follow the steps listed below to submit your assignment.

1. On your local machine, create a new folder with your 7 digit Student ID as its name.
2. Put inside the newly created folder the *lex* file named as `<your_student_id>.l` which contains your implementation of the scanner. Also, put additional `<your_student_id>.cpp` file or `<your_student_id>.h` header file necessary to compile your *lex* file. Do not put the generated `lex.yy.c` file or executable file inside this folder.
3. Compress the folder in a *zip* file which should be named after your 7 digit Student ID.
4. Submit the *zip* file.

You are strongly encouraged to follow this submission guideline and naming convention for the files.

6 Warning

If you adopt any unfair means or get yourselves involved in acts of plagiarism, **then you will be penalized with -100% marks for this assignment regardless of your role in the incident.**

7 Submission Deadline

The submission deadline for this assignment is set for **Saturday, December 17, 2023 at 09:59 PM** for all the lab groups.