EMERTXE TRAINING PROJECT DOCUMENTATION FRAMEWORK

REQUIREMENTS & DESIGN DOCUMENT

Module – Advanced C

Lexical Analyzer

Contents

1 Abstract

In computer science, lexical analysis is the process of converting a sequence of characters into a sequence of tokens. A program or function which performs lexical analysis is called a lexical analyzer, lexer, or scanner. A lexer often exists as a single function which is called by a parser or another function.

Lexical analyzers are designed to recognize keywords , operators , and identifiers , as well as integers, floating point numbers , character strings , and other similar items that are written as part of the source program.

Token:

A token is a string of characters, categorized according to the rules as a symbol (e.g., IDENTIFIER, NUMBER, COMMA). The process of forming tokens from an input stream of characters is called tokenization, and the lexer categorizes them according to a symbol type. A token can look like anything that is useful for processing an input text stream or text file.

A lexical analyzer generally does nothing with combinations of tokens, a task left for a parser. For example, a typical lexical analyzer recognizes parentheses as tokens, but does nothing to ensure that each "(" is matched with a ")".

Consider this expression in the C programming language:

sum=3+2;

Tokenized in the following table:

Lexeme	Token type
sum	Identifier
=	Assignment operator
3	Integer literal
+	Addition operator
2	Integer literal
;	End of statement

Tokens are frequently defined by regular expressions, which are understood by a lexical analyzer generator such as lex. The lexical analyzer (either generated automatically by a tool like lex, or hand-crafted) reads in a stream of characters, identifies the lexemes in the stream, and categorizes them into tokens. This is called "tokenizing." If the lexer finds an invalid token, it will report an error.c

Following tokenizing is parsing. From there, the interpreted data may be loaded into data structures for general use, interpretation, or compiling.

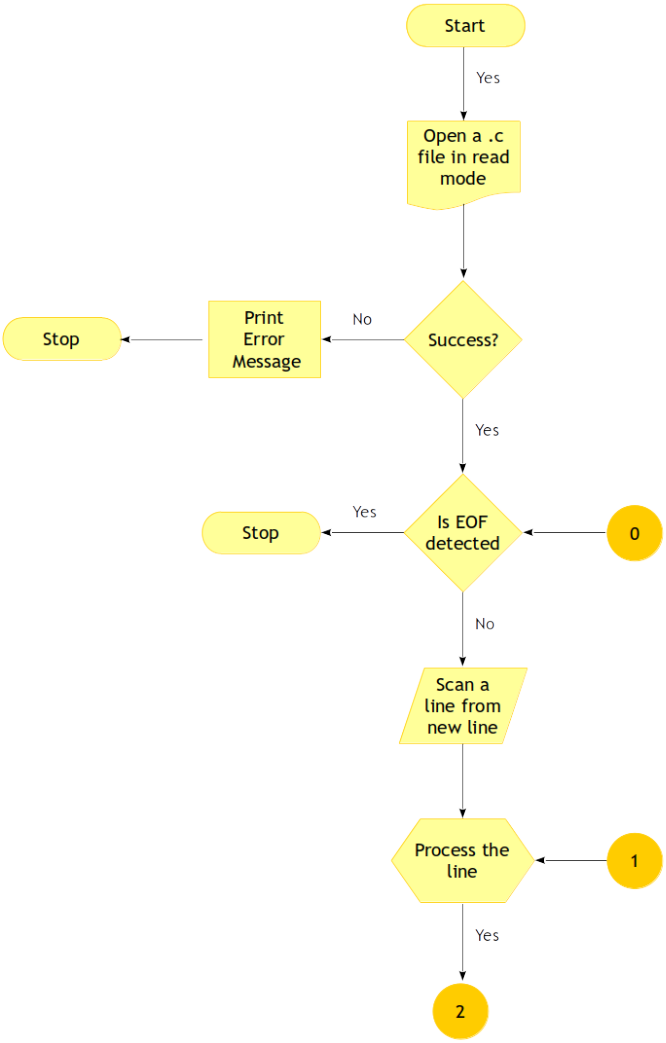
2 Requirements

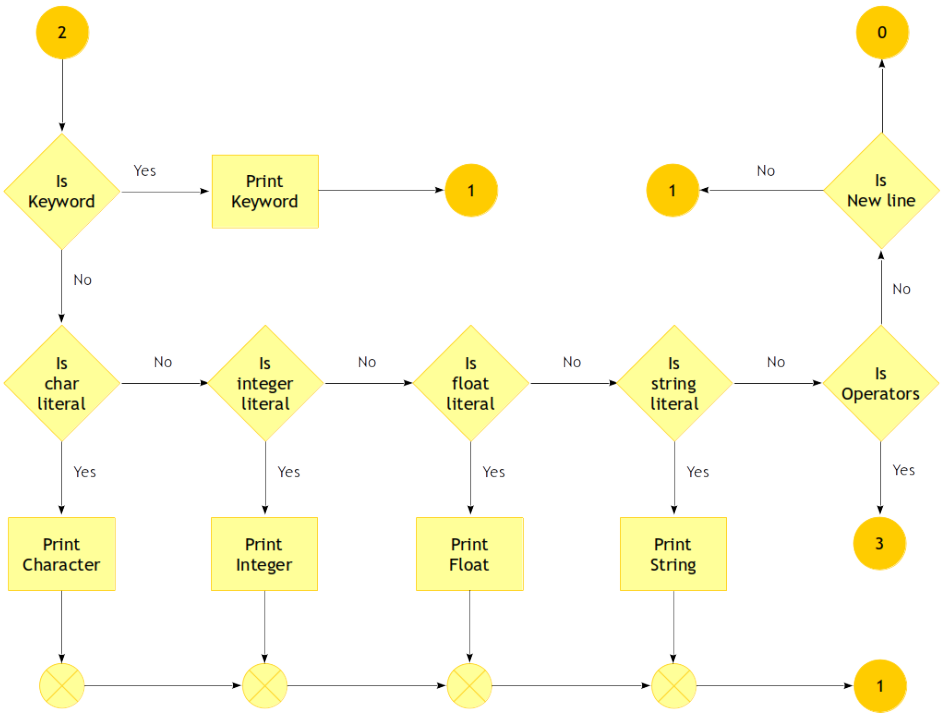
- This project has to be designed only for the sub-set of C.
- It should identify all the keywords
- It should identify identify all the Identifiers.
- It should identify the literals, such as float , characters, string literals, decimals.
- It should identify the arrays

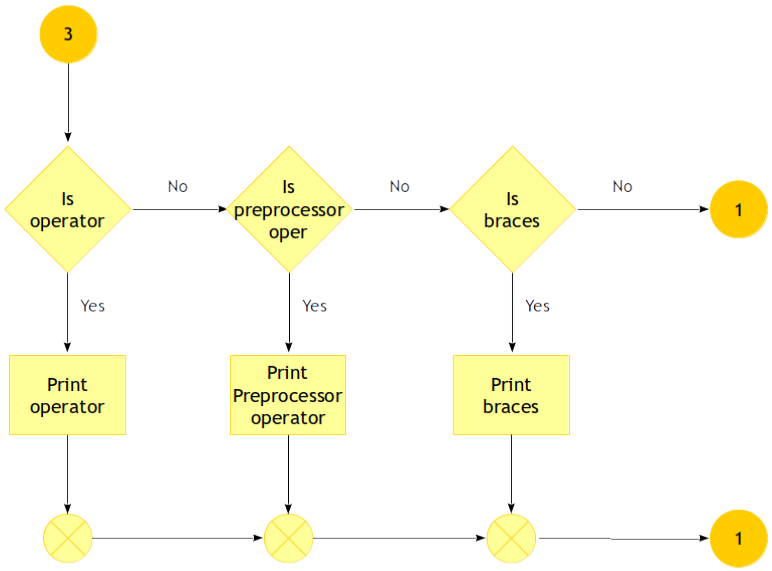
3 Prerequisites

- Basic knowledge of compiler design
- File I/O
- String Handling
- Pointers

4 Design


Start
Yes
Open a .c file in read mode
Success?
No
Print Error Message
Stop
Yes
Is EOF detected
0
Yes
Stop
No
Scan a line from new line
Process the line
1
Yes
2


2
Is Keyword
Yes
Print Keyword
1
No
Is char literal
No
Is integer literal
No
Is float literal
No
Is string literal
No
Is Operators
Yes
Print Character
Yes
Print Integer
Yes
Print Float
Yes
Print String
Yes
3
1
No
Is New line
No
0
1


3
Is operator
No
Is preprocessor oper
No
Is braces
No
1
Yes
Print operator
Yes
Print Preprocessor operator
Yes
Print braces
1

5 Sample Output

```
user@emertxe] ./lexical_analyser
Usage: ./lexical_analyser <.c file>
user@emertxe]
```

Fig 5 1: Usage

```
user@emertxe] cat test.c
int main()
{
     printf("Hello World\n");

     return 0;
}
user@emertxe]
```

Fig 5 2: Sample C File to be parsed

```
user@emertxe] ./lexical_analyser text.c
Open     : text.c : Success
Parsing  : text.c : Started

Keyword      : int
Identifer    : main
Operator     : (
Operator     : )
Operator     : {
Identifer    : printf
Operator     : (
Literal      : "Hello World0
"
Operator     : )
Operator     : ;
Keyword      : return
Literal      : 0
Operator     : }
Operator     : ;

Parsing  : text.c : Done
user@emertxe]
```

Fig 5 3: Expected Output

6 Artifacts

6.1 Skeleton Code

- TBD

6.2 References

- https://en.wikipedia.org/wiki/Lexical_analysis