LEXICAL ANALYSER

THIRD REVIEW REPORT

Submitted by

Harshit Kedia (15BCE0329)

Vinit Bodhwani (15BCE0719)

Ratnasambhav Priyadarshi (15BCE0646)

Prepared for

Theory of Computation and Compiler Design (CSE2002) – Project Component

Submitter to

Sendhil Kumar K.S

Assistant Professor

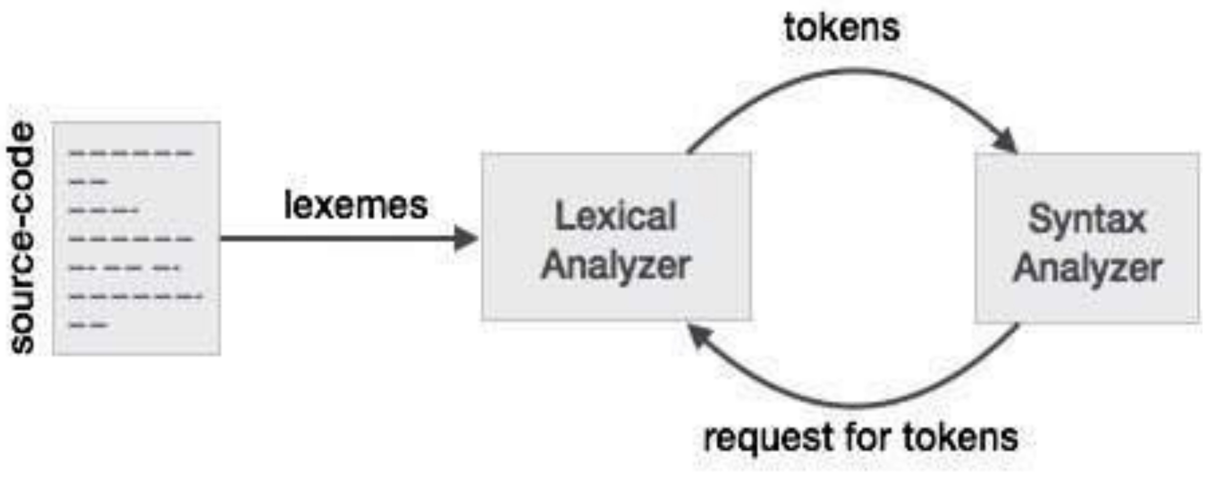
School of Computer Science and Engineering

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ▪ CHENNAI
www.vit.ac.in

Introduction

Lexical analysis is the first phase of a compiler. It takes the modified source code from language pre-processors that are written in the form of sentences. The lexical analyser breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyser finds a token invalid, it generates an error. The lexical analyser works closely with the syntax analyser. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyser when it demands.



Lexemes are said to be a sequence of characters (alphanumeric) in a token. There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions.

In programming language, keywords, constants, identifiers, strings, numbers, operators and punctuations symbols can be considered as tokens.

For example, in C language, the variable declaration line

```
int value = 100;
```

Contains the tokens:

```
int (keyword), value (identifier), = (punctuation), 100
(constant) and ;(punctuation)
```

Let’s look at another example. Consider this expression in the C programming language:

```
sum = 3 + 2;
```

Tokenized and represented by the following table:

Lexeme	Token category
sum	"Identifier"
=	" Punctuation "
3	"Number"
+	" Punctuation "
2	"Number"
;	"Punctuation"

Tokenization is the process of demarcating and possibly classifying sections of a string of input characters. The resulting tokens are then passed on to some other form of processing. The process can be considered a sub-task of parsing input.

'Tokenization' has a different meaning within the field of computer security.

Take, for example,

```
The quick brown fox jumps over the lazy dog
```

The string isn't implicitly segmented on spaces, as an English speaker would do. The raw input, the 43 characters, must be explicitly split into the 9 tokens with a given space delimiter (i.e. matching the string " " or regular expression /\s{1}/).

Project Scope:

Our lexical analyzer is designed to accept an input sting and tokenize it into the various kinds of tokens. It will be able to detect the following types of tokens:

1. Numbers
2. Escape Character
3. Whitespaces
4. Comments
5. Punctuations
6. End of File
7. Keywords

This will be done with the help of Regular Expressions. The output will include the position of each token, i.e. there line number and position in that line.

Components Required:

- PHP
- HTML, CSS, JS
- Apache Web Server

Code:

We have written the backend and Lexical Analyzer program in PHP.

The tokens are detected using the following regular expressions:

```
public $tokenTypes = array(

     "T_LATEX_COMMAND" => array(

        "regex" => '/^\\\([^_$#%&][a-zA-Z0-9\._-]+)(({[^}]*})|(\[[^\]]*\]))*/s',

        "store" => true

        ),


      "T_KEYWORD" => array(

        "regex" =>
'/^(int)|(double)|(return)|(goto)|(while)|(if)|(break)|(continue)|(public)|(private)|(float)|(class)/',

        "store" => true

         ),
     "T_INDENTIFIER" => array(
        "regex" => '/^[a-zA-ZÀ-ÿ][a-zA-Z0-9À-ÿ]*/',
        "store" => true
        ),

     "T_NUMBER" => array(
        "regex" => '/^[0-9]([0-9\.,]*[0-9])?/',
        "store" => true
        ),
     "T_ESCAPED" => array(
        "regex" => '/^\\\[_$#%&]/',
        "store" => true
        ),
     "T_WHITESPACE" => array(
        "regex" => '/^\s+/',
        "store" => true
        ),
     "T_COMMENT" => array(
        "regex" => '/^%[^\n]*\n/',
        "store" => true
        ),
     "T_PUNCTUATION" => array(
        "regex" => '/^[,\._!\?\(\)-:;\"'–<>|@\*\{\}$#%&=+\[\]`]+/',
        "store"=> true
        ),
     "T_EOF"=> array(
        "regex" => '/^$/',
        "store"=> true
        ),
     );
```

Analyzer.php

```php
<?php

namespace LexicalAnalyzer\Analyzers;

abstract class Analyzer
{
    public $tokenTypes = array();

    public function parse($code)
    {
        if (is_string($code)){
            return $this->analyze(new \LexicalAnalyzer\Handlers\StringHandle($code));
        } elseif(is_resource($code)) {
            return $this->analyze(new \LexicalAnalyzer\Handlers\StreamHandle($code));
        }
    }

    public function analyze($handle)
    {
        $line = 1;
        $column = 1;
        $latex = $handle->getData();
        $tokens = array();

        while($handle->isEndded() || !empty($latex)) {
            if ($handle->hasMoreData()) {
                $latex .= $handle->getData();
            }

            $token = null;
```

```
foreach ($this->tokenTypes as $tokenType => $property) {
  if (preg_match($property['regex'], $latex, $group)) {
    $pointer = strlen($group[0]);
    $latex = substr($latex, $pointer);

    $token = new \LexicalAnalyzer\Tokens\LatexToken;
    $token->type = $tokenType;
    $token->value = $group[0];
    $token->line = $line;
    $token->column = $column;

    if ($property['store']) {
      $tokens[] = $token;
    }

    if (preg_match_all('/\n/', $group[0], $linefeeds)) {
      $line += count($linefeeds[0]);
      $column = 1;
    }

    if (preg_match('/(.*\n)?([^\n]*)$/', $group[0], $columnfeeds)) {
      $sizeof = count($columnfeeds) - 1;
      $column += mb_strlen($columnfeeds[$sizeof], 'UTF-8');
    }

    break;
  }
}

if ($token->type == 'T_EOF') {
```

```
                break;
            }
        }
        return $tokens;
    }
}
```

Index.php file:

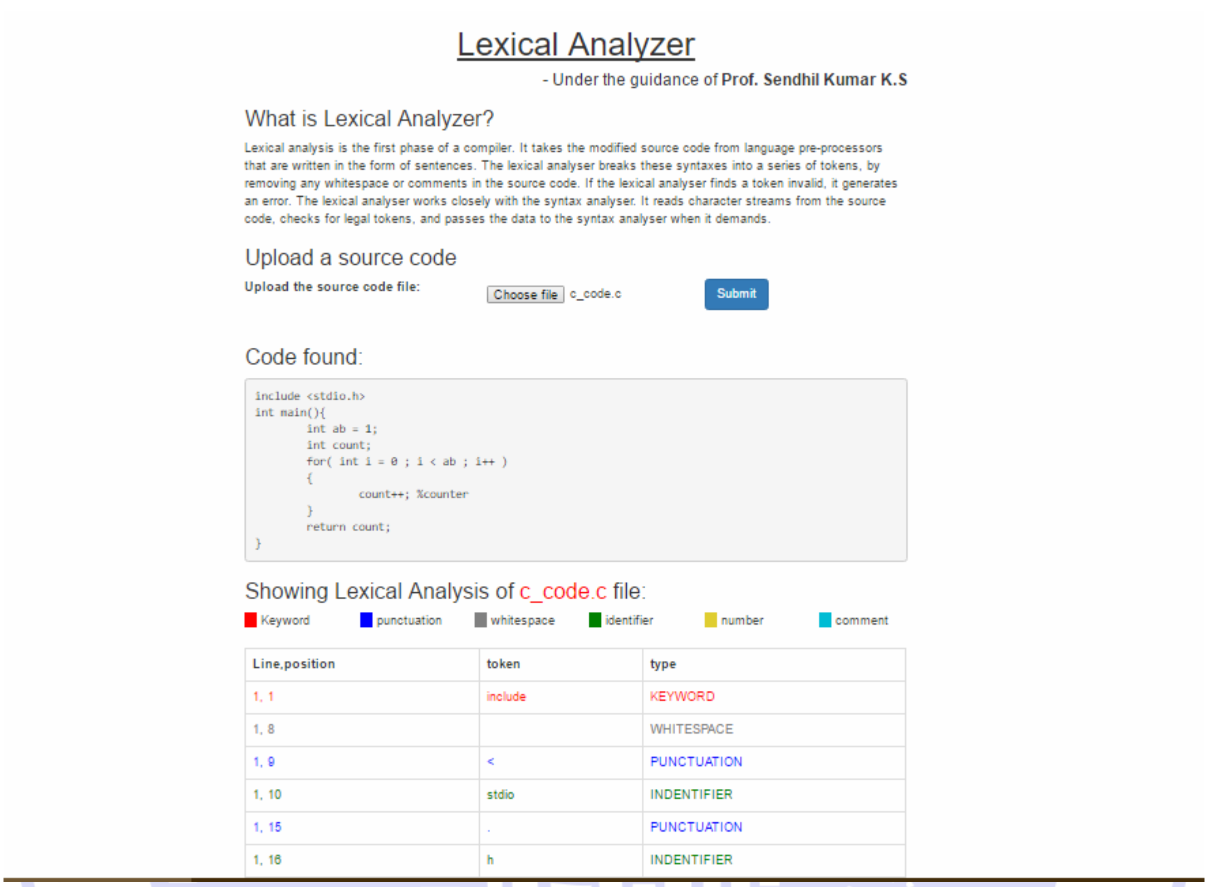
```
<?php
if(isset($_POST['submit'])){
            $fileName=$_FILES["fileToUpload"]["name"];
            function __autoload($class)
            {
                        $class = str_replace('\\', DIRECTORY_SEPARATOR, $class);
                        require_once(__DIR__ . "/lib/$class.php");
            }
            $file = file_get_contents($fileName);
/**
 * We have a new instance of the LaTeX lexical analyzer, that will parse
 * the input string when we call the parse method, returning an array
 * of token objects
 */
$latex = new LexicalAnalyzer\Analyzers\LatexAnalyzer;
$tokens = $latex->parse($file);
}
?>
<!DOCTYPE html>
<html>
<head>
            <meta charset="utf-8">
            <title>Lexical Analyzer - Theory of Computation</title>
            <link rel="stylesheet" type="text/css" href="bootstrap/css/bootstrap.min.css">
```

```
        <link rel="stylesheet" type="text/css" href="css/style.css">
</head>
<body>
        <div class="container">
                <div class="col-md-8 col-md-offset-2 table-div table-responsive">
                        <h1 class="text-center"><u>Lexical Analyzer</u></h1>
                        <h4 class="text-right"> - Under the guidance of <b>Prof. Sendhil Kumar
K.S</b></h4>
                        <h3 align="left">What is Lexical Analyzer?</h3>
                        <p>Lexical analysis is the first phase of a compiler. It takes the modified
source code from language pre-processors that are written in the form of sentences. The lexical
analyser breaks these syntaxes into a series of tokens, by removing any whitespace or comments in
the source code.
                                If the lexical analyser finds a token invalid, it generates an error. The
lexical analyser works closely with the syntax analyser. It reads character streams from the source
code, checks for legal tokens, and passes the data to the syntax analyser when it demands.
                        </p>
                        <h3>Upload a source code</h3>
                        <form action="#" method="POST" enctype="multipart/form-data"">
                                <div class="row">
                                        <div class="form-group">
                                                <label class="control-label col-sm-4">Upload the
source code file:</label>
                                                <div class="col-sm-4">
                                                        <input type="file" class="btn"
name="fileToUpload" id="fileToUpload">
                                                </div>
                                                <div class="col-sm-4">
                                                        <input type="submit" class="btn btn-
primary" name="submit">
                                                </div>
                                        </div>
                                </div>
                        </form>
                        <br>
                        <?php
                        if(isset($_POST['submit'])){
                                echo "<h3>Code found:</h3>";
                                echo "<pre>";
                                print_r(htmlspecialchars($file));
                                echo "</pre>";
                                echo "<h3>Showing Lexical Analysis of <span
class='red'>".$fileName."</span> file:</h3>";
                        }
                        ?>
                        <div class="row spanrow">
                                <div class="key-wrap col-xs-6 col-sm-2">
                                        <span class="key-color-all spanall" href="">&nbsp;</span>
                                        <span class="key-word spanall">showall</span>
                                </div>
                        </div>
                        <div class="row">
                                <div class="key-wrap col-xs-6 col-sm-2">
                                        <span class="key-color-keyword spankeyword"
href="">&nbsp;</span>
                                        <span class="key-word spankeyword">Keyword</span>
                                </div>
                                <div class="key-wrap col-xs-6 col-sm-2">
                                        <span class="key-color-punctuation
spanpunctuation">&nbsp;</span>
                                        <span class="key-word
spanpunctuation">punctuation</span>
                                </div>
                                <div class="key-wrap col-xs-6 col-sm-2">
                                        <span class="key-color-whitespace
spanwhitespace">&nbsp;</span>
                                        <span class="key-word spanwhitespace">whitespace</span>
                                </div>
                        <div class="key-wrap col-xs-6 col-sm-2">
                            <span class="key-color-identifier
spanidentifier">&nbsp;</span>
                            <span class="key-word spanidentifier">identifier</span>
                        </div>
                        <div class="key-wrap col-xs-6 col-sm-2">
                            <span class="key-color-number
spannumber">&nbsp;</span>
                            <span class="key-word spannumber">number</span>
                        </div>
                        <div class="key-wrap col-xs-6 col-sm-2">
                            <span class="key-color-comment
spancomment">&nbsp;</span>
                            <span class="key-word spancomment">comment</span>
                        </div>
                    </div>
                    <br>
                    <table class="table table-bordered table-hover" id="table">
                        <thead>
                            <tr>
                                <th>Line,position</th>
                                <th>token</th>
                                <th>type</th>
                            </tr>
                        </thead>
                        <tbody>
                            <?php
                            if(isset($_POST['submit'])){
                                foreach ($tokens as $token) {
                                    if(!isset($token->type) or empty($token-
>type))
                                        $totype="None";
                                    else
```

```
                                                        $totype=$token->type;

                                                $totype=substr($totype,2);

                                                if($token->value=='include'||$token-
>value=='for'||$token->value=='return'||$token->value=='int'||$token->value=='double'||$token-
>value=='float'||$token->value=='while'||$token->value=='do'||$token->value=='continue'||$token-
>value=='break'||$token->value=='goto'||$token->value=='class'||$token->value=='public'||$token-
>value=='private')

                                                        $totype="KEYWORD";

                                                echo "<tr class='{$totype} trow'>";

                                                echo "<td>{$token->line}, {$token-
>column}</td>";

                                                echo "<td>".str_replace(PHP_EOL, '',
$token->value) . PHP_EOL."</td>";

                                                echo "<td>{$totype}</td>";

                        //echo "{$token->type} at {$token->line}, {$token->column}:" .
str_replace(PHP_EOL, '', $token->value) . PHP_EOL;

                                                echo "</tr>";

                                        }

                                }

                                else{

                                        echo "<h3 class='red'>No file is choosen</h3>";

                                }

                                ?>

                        </tbody>

                </table>

                        <p class="count"></p>

        </div>

    </div>

    <footer>

        <hr>

        <div class="container">

                <div class="pull-left">Theory of Computation<br>CSE2002<br>slot-
F1</div>

                <div class="pull-right">Project by:
                        <ul>
                                <li><b>Harshit Kedia</b> (15BCE0329)</li>
                                <li><b>Vinit Bodhwani</b> (15BCE0719)</li>
                                <li><b>Ratnasambhav Priyadarshi</b> (15BCE0646)</li>
                        </ul>
                </div>
        </div>
</footer>
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="bootstrap/js/bootstrap.min.js"></script>
<script type="text/javascript">
        function update_count(){
                var trs=$(".show-tr");
                var show_len=trs.length;
                $(".count").html("No. of tokens:"+show_len);
        }
        $(".spanrow").hide();
        $(".trow").show();
        $(".trow").addClass('show-tr');
        update_count();
        $(".spankeyword").click(function(){
                $(".trow").removeClass('show-tr');
                $(".trow").hide();
                $(".KEYWORD").addClass('show-tr');
                $(".KEYWORD").show();
                $(".spanrow").show();
                update_count();
        });
        $(".spanpunctuation").click(function(){
                $(".trow").removeClass('show-tr');
                $(".trow").hide();
        $(".PUNCTUATION").addClass('show-tr');
        $(".PUNCTUATION").show();
        $(".spanrow").show();
        update_count();
});
$(".spanwhitespace").click(function(){
        $(".trow").removeClass('show-tr');
        $(".trow").hide();
        $(".WHITESPACE").addClass('show-tr');
        $(".WHITESPACE").show();
        $(".spanrow").show();
        update_count();
});
$(".spanidentifier").click(function(){
        $(".trow").removeClass('show-tr');
        $(".trow").hide();
        $(".INDENTIFIER").addClass('show-tr');
        $(".INDENTIFIER").show();
        $(".spanrow").show();
        update_count();
});
$(".spannumber").click(function(){
        $(".trow").removeClass('show-tr');
        $(".trow").hide();
        $(".NUMBER").addClass('show-tr');
        $(".NUMBER").show();
        $(".spanrow").show();
        update_count();
});
$(".spancomment").click(function(){
        $(".trow").removeClass('show-tr');
```

```
            $(".trow").hide();

            $(".COMMENT").addClass('show-tr');

            $(".COMMENT").show();

            $(".spanrow").show();

            update_count();

        });

        $(".spanall").click(function(){

            $(".trow").show();

            $(".spanrow").removeClass('show-tr');

        });


    </script>

</body>

</html>
```

ScreenShots:



Lexical Analyzer

- Under the guidance of **Prof. Sendhil Kumar K.S**

What is Lexical Analyzer?

Lexical analysis is the first phase of a compiler. It takes the modified source code from language pre-processors that are written in the form of sentences. The lexical analyser breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. If the lexical analyser finds a token invalid, it generates an error. The lexical analyser works closely with the syntax analyser. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyser when it demands.

Upload a source code

Upload the source code file: Choose file c_code.c Submit

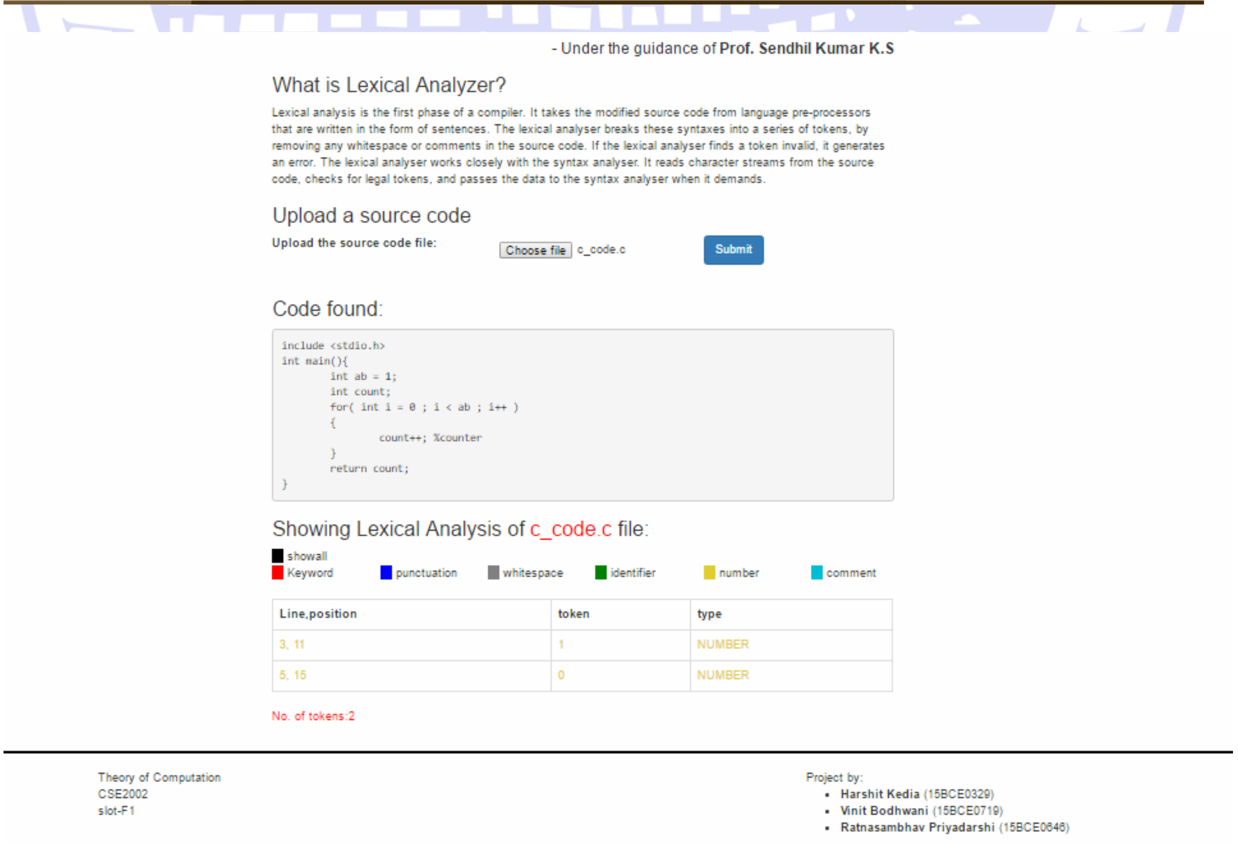
Code found:

```
include <stdio.h>
int main(){
        int ab = 1;
        int count;
        for( int i = 0 ; i < ab ; i++ )
        {
                count++; %counter
        }
        return count;
}
```

Showing Lexical Analysis of c_code.c file:

🟥 Keyword 🟦 punctuation ⬛ whitespace 🟩 identifier 🟨 number 🟦 comment

Line,position	token	type
1, 1	include	KEYWORD
1, 8		WHITESPACE
1, 9	<	PUNCTUATION
1, 10	stdio	INDENTIFIER
1, 15	.	PUNCTUATION
1, 16	h	INDENTIFIER



- Under the guidance of **Prof. Sendhil Kumar K.S**

What is Lexical Analyzer?

Lexical analysis is the first phase of a compiler. It takes the modified source code from language pre-processors that are written in the form of sentences. The lexical analyser breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. If the lexical analyser finds a token invalid, it generates an error. The lexical analyser works closely with the syntax analyser. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyser when it demands.

Upload a source code

Upload the source code file: Choose file c_code.c Submit

Code found:

```
include <stdio.h>
int main(){
        int ab = 1;
        int count;
        for( int i = 0 ; i < ab ; i++ )
        {
                count++; %counter
        }
        return count;
}
```

Showing Lexical Analysis of c_code.c file:

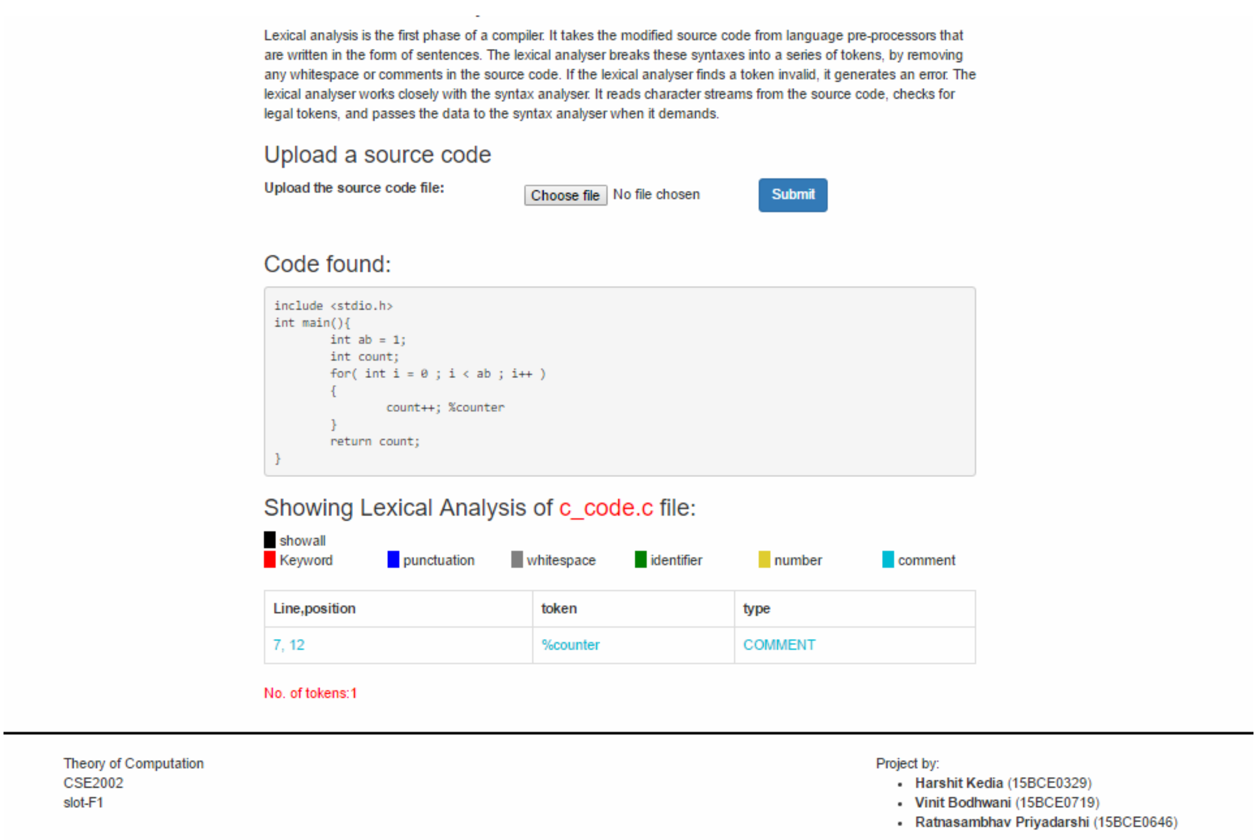
⬛ showall
🟥 Keyword 🟦 punctuation ⬛ whitespace 🟩 identifier 🟨 number 🟦 comment

Line,position	token	type
3, 11	1	NUMBER
5, 15	0	NUMBER

No. of tokens:2

Theory of Computation
CSE2002
slot-F1

Project by:
- **Harshit Kedia** (15BCE0329)
- **Vinit Bodhwani** (15BCE0719)
- **Ratnasambhav Priyadarshi** (15BCE0646)



Lexical analysis is the first phase of a compiler. It takes the modified source code from language pre-processors that are written in the form of sentences. The lexical analyser breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. If the lexical analyser finds a token invalid, it generates an error. The lexical analyser works closely with the syntax analyser. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyser when it demands.

Upload a source code

Upload the source code file:

Code found:

```
include <stdio.h>
int main(){
        int ab = 1;
        int count;
        for( int i = 0 ; i < ab ; i++ )
        {
                count++; %counter
        }
        return count;
}
```

Showing Lexical Analysis of c_code.c file:

■ showall
🟥 Keyword 🟦 punctuation ▨ whitespace 🟩 identifier 🟨 number 🟦 comment

Line,position	token	type
7, 12	%counter	COMMENT

No. of tokens:1

Theory of Computation
CSE2002
slot-F1

Project by:
- **Harshit Kedia** (15BCE0329)
- **Vinit Bodhwani** (15BCE0719)
- **Ratnasambhav Priyadarshi** (15BCE0646)

Conclusion:

Therefore, Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. We have successfully build a GUI(Graphic User Interface) in the form of a website using PHP and front-end laguages. The GUI successfully took a file as an input from the user, and converted it into tokens.