A Lexical and Syntax Checker Tool for the Hyper Text Markup Language

Maisaa Ibrahem Abdul-Hussain
Department of Computer Science, College of Science, University of Baghdad-Iraq.

Abstract

Hyper Text Markup Language (HTML) is one of the web sites design languages in the internet. HTML language differs from programming languages in that any editor (e.g. notepad, WordPad) can be used to write the code of the HTML language. On the other hand, and contrary to what one familiar with the role of the compiler while translating the high-level written code, this editing facility has a shortcoming of not being able to check the lexical and the syntax of the written HTML code. While any compiler takes into its responsibility to check for any lexical and syntax error as a part of its overall function, the interpreter of HTML provides the facility to translate the form of HTML codes to the target representation to be executed only. For this, the main aim of this paper is to propose and present an intermediate stage -*semi compiler*- to check the lexical and syntax of an HTML code before delivering it to the interpreter.

Keywords**:** Compiler, HTML code, Interpreter, Lexical analyzer, Parser.

Introduction

A website is a collection of related web pages, images, videos or other digital assets that are addressed relative to a common Uniform Resource Locator (URL), often consisting of only the domain name, or the IP address, and the root path ('/') in an Internet Protocol-based network. A web site is hosted on at least one web server, accessible via a network such as the Internet or a private local area network [1].

A web page is a document, typically written in plain text interspersed with formatting instructions of Hyper Text Markup Language (HTML). It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists etc. as well as for links, quotes, and other items. It allows images and objects to be embedded and can be used to create interactive forms. It is written in the form of HTML elements consisting of "tags" surrounded by angle brackets within the web page content. It can include or can load scripts in languages such as JavaScript, which affect the behaviour of HTML processors like Web browsers to define the appearance and layout of the text and other material [2].

A web page may incorporate elements from other websites with suitable markup anchors. This simplifies the situation and pretends that all web pages use only HTML file format with extensions ".html" or ".htm". A browser or similar software likes Windows internet explorer, is used to view HTML document. The browser opens the HTML document in the background and "decodes" it before showing it [1].

HTML documents are composed entirely of HTML elements**.** An HTML element is everything between and including the tags. A tag is a keyword enclosed in angle brackets. A common form of an HTML element is <tag>*content to be rendered*</tag>. In fact, many web designers prefer to use simple text editors but HTML language dose not contain a checker tool to check the lexical and the syntax of the HTML language. By this, the underling paper adopts an HTML *semi-compiler* tool composed of two phases: lexical checker and syntax checker to be as an intermediate stage between the web page itself and the interpreter. The reset of the paper is organized as follows. Section 2 briefly describes the background related to the lexical and syntax meaning common to any programming language with illustrated examples. Section 3 demonstrates the lexical and the syntax rules of the HTML together with the steps of the proposed HTML semi-compiler. In Section 4, experimental results of the HTML lexical and syntax checker tool are presented and the conclusion is drawn be in section 5.

Background

A compiler is a computer program (or set of programs) that transforms a source code written in a programming language (the *source language*) into another computer language (the target language, often having a binary form known as *object code*). The most common reason for transforming the source code is to create an executable program. A typical compiler consists of lexical analyzer, syntax analyzer, semantic analyzer and code generation [3] [4]. The following paragraph illustrates the general concepts of the lexical analyzer and the syntax analyzer according to the paper orientation.

A. Lexical analyzer

The lexical analysis or scanning process is where the stream of characters making up the source program is read from left-to-right and grouped into tokens. *Tokens* are sequences of characters with a collective meaning [3]. There are usually only a small number of tokens for a programming language: *constants* (integer, double, char, string, etc.), *operators* (arithmetic, relational, logical), *punctuation*, and *reserved words* [4]. Figure1 [2] depicts an example showing how the lexical analyzer takes a source program as input, and produces a stream of tokens as output. The scanner is tasked with determining that the input stream can be divided into valid symbols in the source language, but has no smarts about which token should come where [3] [4]. From the depicted figure, one can see that the lexical analyzer will produce twelve distinct tokens for the input source code example. The tokens varies from reserved words (e.g., while) to operators (e.g., LESSTHAN, and EQUEL) and constants (e.g., INTCONSTANT, and REALCONSTANT).

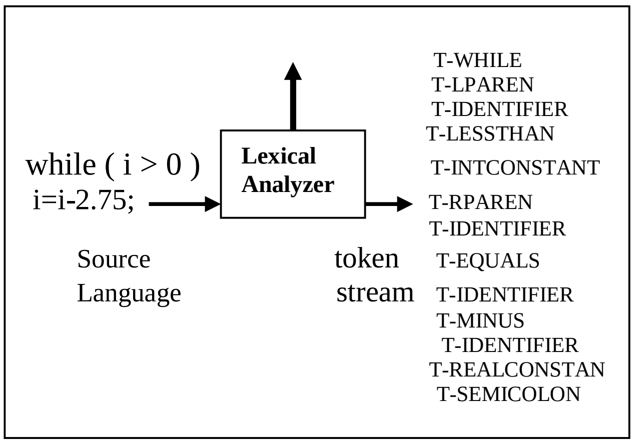


Fig.(1) Lexical Analyzer.

Few errors can be detected at the lexical level alone because the scanner has a much localized view of the source program without any context. The scanner can report about characters that are not valid tokens (e.g., an illegal or unrecognized symbol) and a few other malformed entities (illegal characters within a string constant, unterminated comments, etc.) [3] [4]. The lexical analyzer can be a convenient place to carry out some other chores like stripping out comments and white space between tokens and perhaps even some features like macros and conditional compilation (although often these are handled by some sort of pre-processor which filters the input before the compiler runs) [3]. The lexical analyzer can store all the recognized tokens in an intermediate file and give it to the parser as an input. However it is more convenient to have the lexical analyzer as a subroutine which the parser calls whenever it requires a token [4].

There are two primary methods for implementing a scanner, the first is a program that is hard-coded to perform the scanning tasks the second uses regular expression and finite automata theory to model the scanning process [4] [5].

B. Syntax Analyzer

A term parsing comes from Latin *pars* (*ōrātiōnis*), meaning part (of speech) the syntax analyzer or parser is a program, usually part of a compiler, that receives input in the form of sequential source program instructions, interactive online commands, mark-up tags, or some other defined interface and breaks them up into parts (for example, the nouns (objects), verbs (methods), and their attributes or options) [5]. Parsing is also an earlier term for the diagramming of sentences of natural languages, and is still used for the diagramming of inflected languages, such as the Romance languages or Latin [6] [7]. The parser will check whether the tokens produced by the lexical analyzer form an allowable expression. This is usually done with reference to a context-free grammar which recursively defines components that can make up an expression and the order in which they must appear. However, not all rules defining programming languages can be expressed by context-free grammars alone (for e.g., type validity and proper declaration of identifiers). These rules were can be formally expressed with attribute grammars [8 [9]. In parsing, which is working out the implications of the expression just validated and taking the appropriate action In the case of a calculator or interpreter, the action is to evaluate the expression or program; a compiler, on the other hand, would generate some kind of code. Attribute grammars can also be used to define these actions. The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar. This can be done in essentially two ways: [5] [10]

- Top-down parsing -Top-down parsing can be viewed as an attempt to find *left-most derivations* of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules. Tokens are consumed from left to right. Inclusive choice is used to accommodate ambiguity by expanding all alternative right-hand-sides of grammar rules.
- Bottom-up parsing - A parser can start with the input and attempt to rewrite it to the start symbol. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers.

The proposed HTML Semi-compiler Tool

This section presents the components of the proposed HTML semi-compiler tool. It mainly consists of two parts: lexical analyzer and syntax analyzer, together used to check the lexical and syntax of an HTML code before delivering it to the interpreter (e.g. the internet explorer, Netscape). Fig.(2) depicts the lexical and syntax checker tool for HTML.

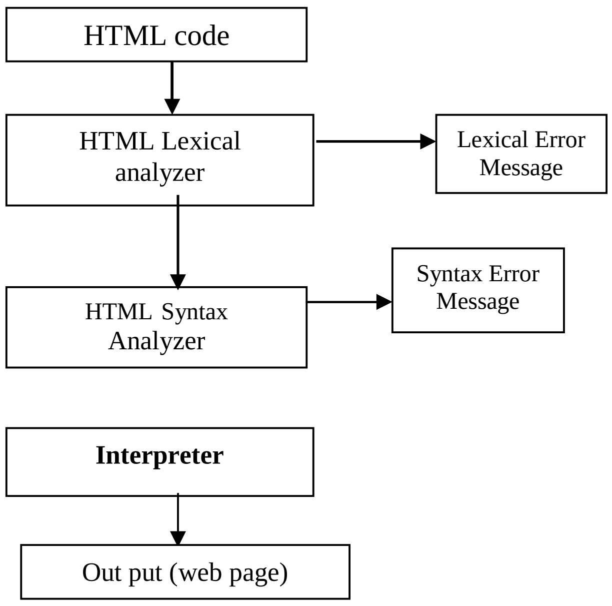


Fig.(2) HTML Checker Tool.

A. Lexical Analyzer

The proposed HTML lexical analyzer is similar to most of the scanners exist in the compilers of other programming languages (e.g., C, Java, and Perl).

The HTML lexical analyzer is used to create tokens from the sequence of input characters and can also be used to carry out some other chores like removing comments, white space and all string between tokens. Each token in HTML is enclosed with two tags, a start-tag begins with "<" followed by a name, and an end with ">" a end-tag is similar, but begins with "</"and ends with "> "such as <tag> or </tag>. Regular expression for the various tokens exist in the HTML (lexeme file) .Thus the various tokens constitute the whole HTML lexem can be specified using the following regular expressions shown in Table (1).

For example the regular expression of HTML tag <font size=6 style="comic sans ms"> any string</font>is <\w\s\w.*?=\d\s\w.*?="\w">.*?</\w>

Table (1)
(HTML Regular Expression).

	Symbol	*Meaning*
1.	<	Match open angle bracket
2.	\s	Whitespace
3.	\r	carriage return
4.	\w	Reserve words between 2 angle <> [a..z][A..Z][0..9] like(html,head,body,h2,h6,font,…)
5.	\d	will match a single digit from[0..9]
6.	.*?	And anything up to
7.	=	Equal
8.	>	close angle brackets - with attributes
9.	/	close tag symbol (/)
10.	!	Comment
11.	>	first closing tag

The lexical analyzer will then compare each token with original correct token that save in file name (lexeme file) contains all HTML correct tags to check if its correct or not, error messages appear if token not correct Fig.(3) shows the diagram of HTML lexical analyzer

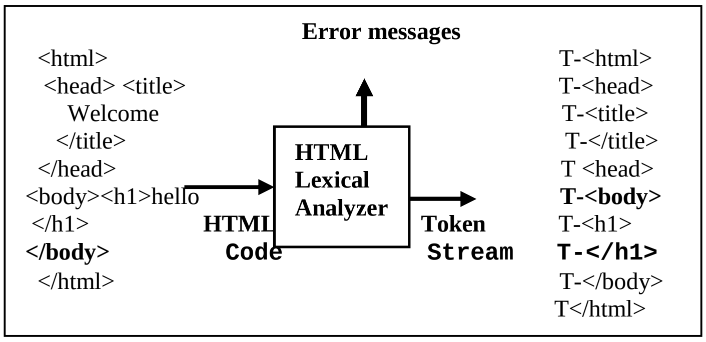


Fig.(3) HTML Lexical Analyzer.

B. Syntax Analyzer

The proposed HTML syntax analyzer (i.e., Parser) is a program used to parse HTML code written in both a linear and nested fashion (see Fig.(4).The parser attempts to balance opening tags with ending tags to present a correct structure of the written HTML code. If the application requires knowledge of the nested structure of the page, for example processing tables, will probably want to use the full parser. The output from the parser would nest the tags as children of the <html>, <head> and other nodes here represented by Fig.(3) shows how the parser balances opening tags with ending tags of HTML code.

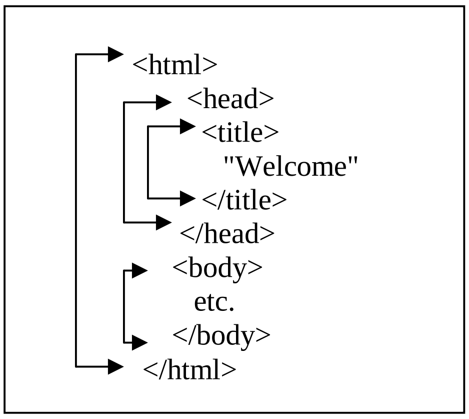


Fig.(4) Syntax Analyzer Balance.

The proposed HTML checker tool uses top-down parsing where tokens are consumed from left to right. The HTML parser will check whether the tokens produced by the lexical analyzer form an allowable expression. This is usually done with reference to a context-free grammar which recursively defines components that can make up an expression and the order in which they must appear. The context-free grammar of an HTML code represent in the following.

T $\longrightarrow$ **tat*|bat**
a $\longrightarrow$ **R|RQw**
Q $\longrightarrow$ **=**
W $\longrightarrow$ **n|s**
R $\longrightarrow$ **r**
t $\longrightarrow$ **<|>**
b $\longrightarrow$ **</**

where(r(reserve tag),n([0..9]),s([a..z])

Experimental Results

This section presents the experimental results using the proposed HTML checker tool. First, an HTML code written in a text editor of this tool or load any HTML code then click on the button (checker) to check this code from error. The following results are obtained depend upon the HTML code file content.

1. Figs.(5, 6 and 7) specify how the proposed tool checks the errors and displays error messages. Any error, warning or proposed

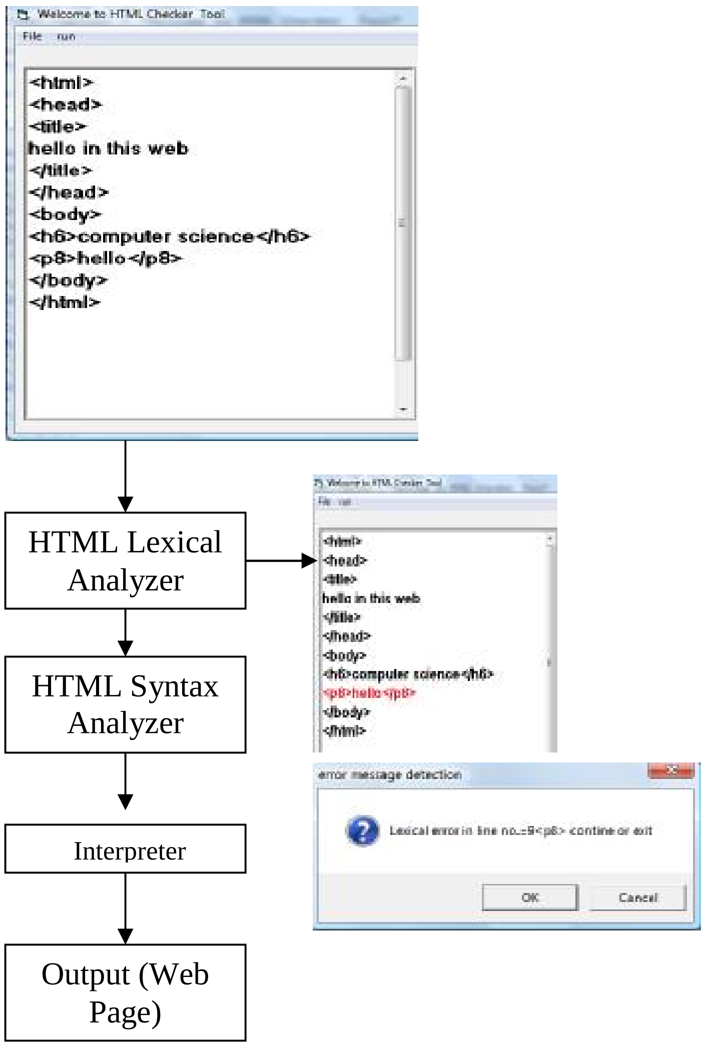
produces will be displayed in the "Error Messages Window". Some errors, marked with a red icon, may prevent "HTML" from continuing to check the rest of the document or producing a corrected version of the HTML code. If this happens, examine the "Messages Window", correct the errors and invoke "HTML" again.

- If HTML code contains error then the error message is appeared "Lexical error in line number 9 <p8> hello </p8>" as shown in Fig.(5).

- If HTML code contains error, then the error message is appeared "Syntax error in line number 13 tag </td> end without begins" as shown in Fig.(6).

- If HTML code contains error, then the error message is appeared "Lexical error in line number 10 no (tag <picture src="ff.jpg"> in this name)" as shown figure 7 and after the error is corrected another error message appears "Syntax error in line number 11 (tag <table> begin without end tag)".

2. If HTML code contains no errors then click on the display button. The page will be interpreted by the internet explorer which were impeded as an object in this semi-compiler tool and will display the web page as shown in Fig.(8).

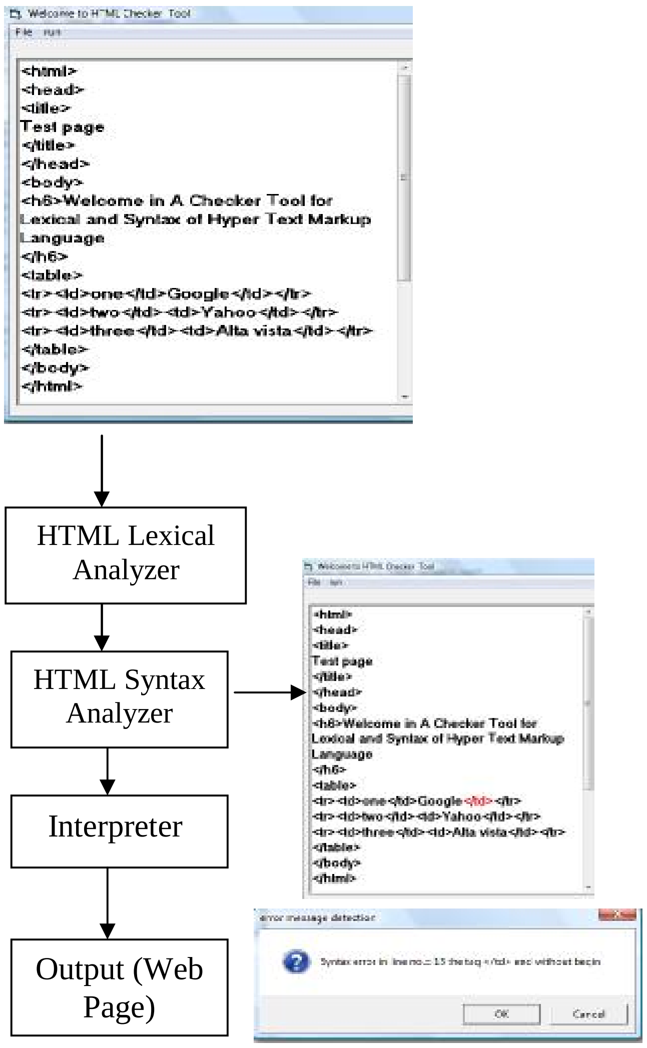


```
<html>
<head>
<title>
hello in this web
</title>
</head>
<body>
<h6>computer science</h6>
<p8>hello</p8>
</body>
</html>
```

```
<html>
<head>
<title>
hello in this web
</title>
</head>
<body>
<h6>computer science</h6>
<p8>hello</p8>
</body>
</html>
```

Lexical error in line no.:=9<p8> contine or exit

Fig.(5) Experimental Result 1.

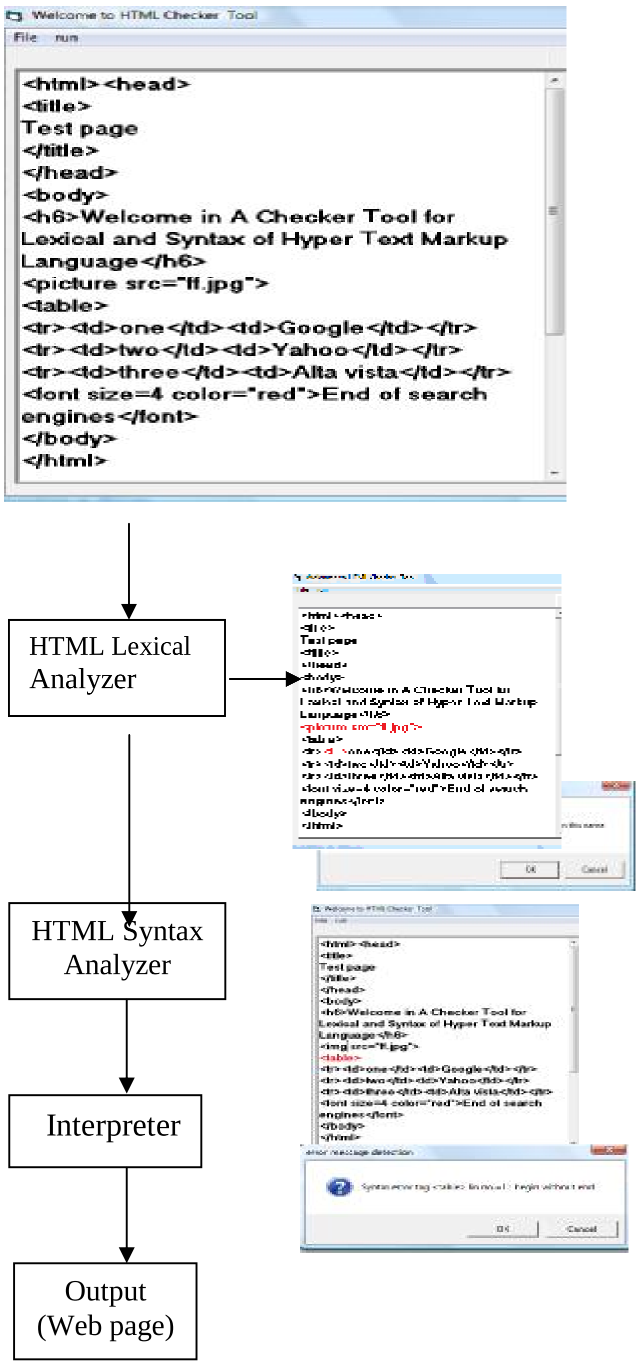


```
<html>
<head>
<title>
Test page
</title>
</head>
<body>
<h6>Welcome in A Checker Tool for
Lexical and Syntax of Hyper Text Markup
Language
</h6>
<table>
<tr><td>one</td>Google</td></tr>
<tr><td>two</td><td>Yahoo</td></tr>
<tr><td>three</td><td>Alta vista</td></tr>
</table>
</body>
</html>
```

```
<html>
<head>
<title>
Test page
</title>
</head>
<body>
<h6>Welcome in A Checker Tool for
Lexical and Syntax of Hyper Text Markup
Language
</h6>
<table>
<tr><td>one</td>Google</td></tr>
<tr><td>two</td><td>Yahoo</td></tr>
<tr><td>three</td><td>Alta vista</td></tr>
</table>
</body>
</html>
```

Syntax error in line no.:= 13 the tag </td> end without begin

Fig.(6) Experimental Result 2.



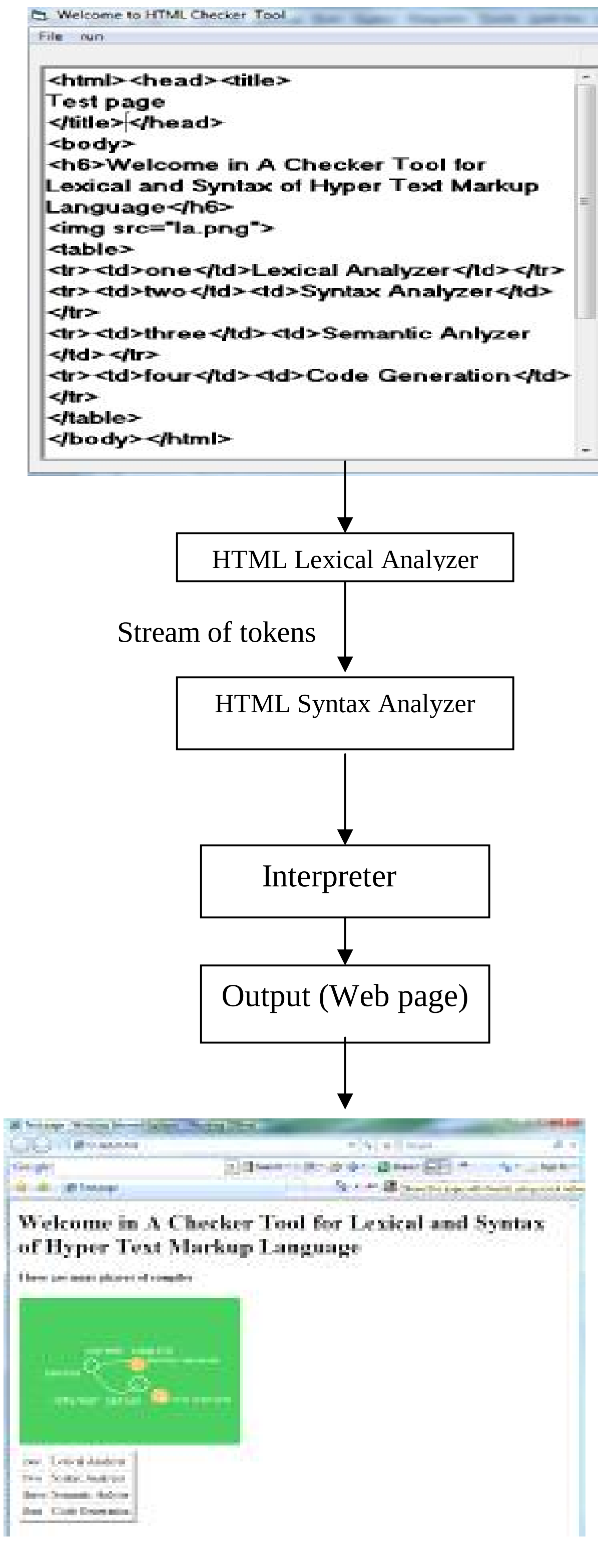
```
<html><head>
<title>
Test page
</title>
</head>
<body>
<h6>Welcome in A Checker Tool for
Lexical and Syntax of Hyper Text Markup
Language</h6>
<picture src="ff.jpg">
<table>
<tr><td>one</td><td>Google</td></tr>
<tr><td>two</td><td>Yahoo</td></tr>
<tr><td>three</td><td>Alta vista</td></tr>
<font size=4 color="red">End of search
engines</font>
</body>
</html>
```

```
<html><head>
<title>
Test page
</title>
</head>
<body>
<h6>Welcome in A Checker Tool for
Lexical and Syntax of Hyper Text Markup
Language</h6>
<picture src="ff.jpg">
<table>
<tr><td>one</td><td>Google</td></tr>
<tr><td>two</td><td>Yahoo</td></tr>
<tr><td>three</td><td>Alta vista</td></tr>
<font size=4 color="red">End of search
engines</font>
</body>
</html>
```

```
<html><head>
<title>
Test page
</title>
</head>
<body>
<h6>Welcome in A Checker Tool for
Lexical and Syntax of Hyper Text Markup
Language</h6>
<img src="ff.jpg">
<table>
<tr><td>one</td><td>Google</td></tr>
<tr><td>two</td><td>Yahoo</td></tr>
<tr><td>three</td><td>Alta vista</td></tr>
<font size=4 color="red">End of search
engines</font>
</body>
</html>
```

Syntax error tag <table> is found : begin without end

Fig.(7) Experimental Result 3.



```
<html><head><title>
Test page
</title></head>
<body>
<h6>Welcome in A Checker Tool for
Lexical and Syntax of Hyper Text Markup
Language</h6>
<img src="la.png">
<table>
<tr><td>one</td>Lexical Analyzer</td></tr>
<tr> <td>two</td><td>Syntax Analyzer</td>
</tr>
<tr><td>three</td><td>Semantic Anlyzer
</td></tr>
<tr><td>four</td><td>Code Generation</td>
</tr>
</table>
</body></html>
```

Welcome in A Checker Tool for Lexical and Syntax of Hyper Text Markup Language

Fig.(8) Experimental Result 4.

Conclusion

This paper proposed and presented an intermediate stage -semi-compiler- to check the lexical and syntax of an HTML code error before delivering it to the interpreter. The proposed tool makes the design of HTML simple and helps the user to discover the lexical and syntax errors as early as possible. A future extension to this work can be done for adopting an error recovery technique with that checker to automatic recovery from the lexical and syntax errors.

References

[1] Deborah H.Ray and Eric J.Ray 2000 “*HTML Complete*” 2nd Edition sybex, San Francisco

[2] Pual Hain Dec 2006 “*HTML Mastery*” 2nd Edition Amazon

[3] Aho, A.V., Sethi, R. and Ullman, J.D. (1986) " *Compilers: principles, techniques, and tools*." Addison-*Wesley Longman Publishing Co., Inc. Boston, MA, USA.*

[4] McGettrick, The Definition of Programming Languages. Cambridge: Cambridge University Press, 1980.

[5] Dick Grune and Ceriel J.H. Jacobs , 1990 “*Parsing Techniques*” published by Ellis Horwood, Chichester, England ISBN 0 13 651431 6.

[6] L.Wexelblat, “*History of Programming Languages*” London: Academic Press, 1981.

[7] J.P. Bennett,” *Introduction to Compiling Techniques*”. Berkshire, England: McGraw-Hill, 1990.

[8] D. Cohen, “*Introduction to Computer Theory*”, New York: Wiley, 1986.

[9] Frost, R., Hafiz, R. and Callaghan, P. (2007) “Modular *and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars*." *10th* International Workshop on Parsing Technologies (IWPT), ACL-SIGPARSE , Pages: 109 - 120, June 2007, Prague.

[10] Tim Berners-Lee April 2009 “*Syntax and semantic*”

الخلاصة

لغة النص التشعبية واحده من لغات تصميم المواقع في الانترنيت .لغة النص التشعبية تختلف عن اللغات البرمجيه بأنها تكتب في أي محرر نصوص مثل (notpad,wordpad). وعلى نقيض ماهو مألوف في المترجم(compiler) الذي يقوم بترجمة الشفرة المكتوبة بالمستوى العالي (high-level) يقوم المترجم بفحص الاخطاء اللغوية والقواعدية كجزء من وظيفتة بينما المفسر في اللغة التشعبية يترجم من شفرة اللغة التشعبية الى الهدف لغرض التنفيذ فقط لذا فأن الهدف الاساسي لهذا البحث أقتراح وتقديم مرحله وسطية جزء من مترجم (semi-compiler) لفحص الشفرة المكتوبة بلغة النص التشعبية قبل ان تصل الى المفسر (interpreter).