****

**Mini Project Report**

**of**

**Compiler Design Lab (CSE 3161)**

**PARSER FOR HTML USING FLEX AND BISON**

**SUBMITTED**

**BY**

**HARSHIT HANSRAJ (210905164)/27**

**LAXYA PAHUJA (210905166)/28**

**VISWATARA KALAMSETTY (210905170) / 29**

**TAKSH KOTHARI (210905338) / 54**

**CSE- ‘A’**

**Department of Computer Science and Engineering**

**Manipal Institute of Technology, Manipal.**

**November 2023**

**TABLE OF CONTENTS**

CHAPTER 1: ABSTRACT

CHAPTER 2: PROBLEM STATEMENT

CHAPTER 3: ALGORITHM

CHAPTER 4: METHODOLOGY

CHAPTER 5: IMPLEMENTATION

CHAPTER 6: REFERENCES

**ABSTRACT**

Our project focuses on creating a parser for the Hypertext Markup Language (HTML). HTML controls how information is presented and organized online in today's digital landscape. Therefore, strong tools that can parse, comprehend, and analyze HTML code are becoming more and more important as the demand for web applications rises. Our goal is to convert unstructured HTML code into a structured representation by using grammar definitions and parsing techniques. This will make it easier for browsers and other web-based apps to understand and render the code.

**PROBLEM STATEMENT**

Our project's main focus is on creating a compiler designed especially for HTML. We aim to construct a system that thoroughly analyzes HTML syntax, validates its structure, and generates a representation that is suitable for interpretation by web browsers or other applications.

The key challenges addressed are:

**Parsing Complexity:** HTML requires a strong parsing system in order to correctly break down its components because of its hierarchical structure, which includes nested tags, attributes, and content.

**Error Handling:** HTML code errors can result in rendering problems or functional inconsistencies. So, to find and fix problems quickly, developers must provide efficient error reporting and detection systems.

**Optimization and Efficiency:** Accurate translation is only possible with efficient code generation, which also helps web applications run more smoothly and make better use of their resources.

**ALGORITHM**

1. **Lexical Analysis:** Define tokens using regular expressions to recognize HTML elements, attributes, and content. Categorize tokens such as DOCTYPE, tags, text, etc., based on the matched patterns.
2. **Syntactic Parsing:** Use defined grammar rules to identify HTML elements, open/close tags, and their attributes. Construct a hierarchical representation of HTML elements using parsing logic.
3. **Semantic Processing:** Validate the structure and attributes of parsed HTML elements to ensure adherence to HTML standards. Detect and report errors related to incorrect syntax or semantic inconsistencies.
4. **Code Generation:** Display recognition messages to indicate successful identification of HTML components during parsing.

**METHODOLOGY**

**Lexer Implementation:** Utilize patterns defined in the code to identify and categorize tokens such as DOCTYPE, tags, text, etc.

**Parser Development:** Use Bison-like parsing rules to create grammar structures for HTML elements, tags, attributes, and content.

**Semantic Analysis:** Validate the syntax and structure of HTML elements, checking attributes for correct usage and adherence to standards.

**Code Generation:** Present recognition messages at various stages of parsing to indicate successful identification of HTML components.

**Grammar:**

document -> html\_element

html\_element -> open\_tag content close\_tag

| open\_tag close\_tag

open\_tag -> '<' tag\_name attribute\_list '>'

close\_tag -> '</' tag\_name '>'

tag\_name -> ID

attribute\_list -> attribute attribute\_list

| ε

attribute -> ID '=' STRING

content -> text content\_prime

| html\_element content\_prime

| ε

content\_prime -> content

| ε

text -> TEXT

ID -> [a-zA-Z][a-zA-Z0-9]\*

STRING -> "([^"])\*"

TEXT -> .+

**IMPLEMENTATION**

In this section, we will discuss the results obtained from the execution of the lexer and parser programs and provide an analysis of the program's behaviour.

**Sample Input:**

<!DOCTYPE html>

<html>

<head>

<title>

Hello

</title>

<meta >

</head>

<body>

<h1>

Welcome to my Sample HTML Page

</h1>

<p>

This is a paragraph of text.

</p>

<ul>

<li>

hi

</li>

<li>

hi

</li>

<li>

hi

</li>

</ul>

</body>

</html>

**Code:**

**parser.y:**

%{

#include <stdio.h>

#include <stdlib.h>

int yylex();

int yyerror(char\* msg);

extern FILE\* yyin;

%}

%token DOCTYPE HTML\_BODY\_OPEN HTML\_BODY\_CLOSE

%token HEAD\_OPEN HEAD\_CLOSE TITLE\_OPEN TITLE\_CLOSE META

%token BODY\_OPEN BODY\_CLOSED OPEN\_TAG CLOSE\_TAG TEXT

%token DIGIT EQUAL QUOTED\_STRING UNQUOTED\_STRING CB

%%

html\_doc: DOCTYPE html\_body { printf("HTML Document recognized!\n"); }

;

html\_body: HTML\_BODY\_OPEN head body HTML\_BODY\_CLOSE { printf("HTML Body recognized!\n"); }

;

head: HEAD\_OPEN title meta HEAD\_CLOSE { printf("Head recognized!\n"); }

| /\* ε \*/ { printf("Empty Head recognized!\n"); }

;

title: TITLE\_OPEN TEXT TITLE\_CLOSE { printf("Title recognized!\n"); }

| TITLE\_OPEN TITLE\_CLOSE

;

meta: META attributes CB { printf("Meta recognized!\n"); }

|

;

body: BODY\_OPEN content BODY\_CLOSED { printf("Body recognized!\n"); }

;

content: element content { printf("Element content recognized!\n"); }

| TEXT content { printf("Text content recognized!\n"); }

| /\* ε \*/ { printf("Empty content recognized!\n"); }

;

element: open\_tag content close\_tag { printf("Element recognized: %s\n", yylex); }

;

open\_tag: OPEN\_TAG { printf("Open tag recognized!\n"); }

;

close\_tag: CLOSE\_TAG { printf("Close tag recognized!\n"); }

;

attributes: attribute attributes { printf("Attributes recognized!\n"); }

| /\* ε \*/ { printf("Empty attributes recognized!\n"); }

;

attribute: attr\_name EQUAL attr\_value { printf("Attribute recognized!\n"); }

;

attr\_name: TEXT { printf("Attribute name: %s\n", yylex); }

;

attr\_value: QUOTED\_STRING { printf("Quoted attribute value: %s\n", yylex); }

| UNQUOTED\_STRING { printf("Unquoted attribute value: %s\n", yylex); }

;

%%

int yyerror(char\* msg)

{

printf("Error: %s\n", msg);

return 1;

}

int main(int argc, char\*\* argv)

{

yyin = fopen(argv[1], "r");

if (!yyin)

{

yyerror("File Error\n");

return 1;

}

yyparse();

fclose(yyin);

return 0;

}

**lexer.l:**

%{

#include "parse.tab.h"

%}

%%

"<!DOCTYPE html>" { return DOCTYPE; }

"<html>" { return HTML\_BODY\_OPEN; }

"</html>" { return HTML\_BODY\_CLOSE; }

"<head>" { return HEAD\_OPEN; }

"</head>" { return HEAD\_CLOSE; }

"<title>" { return TITLE\_OPEN; }

"</title>" { return TITLE\_CLOSE; }

"<body>" { return BODY\_OPEN; }

"</body>" { return BODY\_CLOSED; }

"<meta" { return META; }

">" { return CB; }

"<"([a-zA-Z][a-zA-Z0-9\_]\*)">" { return OPEN\_TAG; }

"</"([a-zA-Z][a-zA-Z0-9\_]\*)">" { return CLOSE\_TAG; }

[ \t\n] ; // ignore whitespaces

[a-zA-Z][a-zA-Z0-9\_ .]\* { return TEXT; }

[0-9]+ { return DIGIT; }

"=" { return EQUAL; }

\"[^\"] { return QUOTED\_STRING; }

[^ \t\n]+ { return UNQUOTED\_STRING; }

%%

int yywrap() {

return 1;

}

**Execution of Program:**  


**REFERENCES**

1. Compilers – Principles, Techniques & Tools (Second Edition) – Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffery D. Ulmann
2. Compiler Design Lab Manual, Manipal Institute of Technology