Department of computer science

University of Delhi

Compiler Design

(MCAC-404)

Parser Project-

Simple Html Table Tag

Submitted By:-

Sachin Pandey

Chandraveer Panwar

Saijal Sikka

Submitted to:-

Dr. Ankit Rajpal

Assistant prof.

Problem Statement

The task is to write a Lex and Yacc program that can handle simple HTML table tags. The program should be able to parse an HTML file containing table tags and identify all the tables and cells in the file.

Solution

The solution involves using Lex to tokenize the HTML input file and Yacc to parse the token stream using a set of grammar rules. The program needs to recognize HTML table tags, including **<table>**, **</table>**, **<tr>**, **</tr>**, **<td>**, and **</td>**, and ignore all other HTML tags and white space. When the program encounters a **<table>** tag, it needs to start looking for a table row (**<tr>** tag) and a list of cells (**<td>** tags), and when it encounters a **<td>** tag, it needs to start looking for the content of the cell, which can include nested tables.

Lexical Analysis

The lexical analyzer (implemented in **html.l**) uses regular expressions to match the HTML tags **<table>**, **</table>**, **<tr>**, **</tr>**, **<td>**, and **</td>**. It also ignores other HTML tags and white space characters. When the lexical analyzer encounters one of these tags, it returns a token indicating the type of tag.

Syntactic Analysis

The syntactic analyzer (implemented in **html.y**) defines the grammar rules for a document, a table, a list of cells, and a cell. When the program encounters a **<table>** tag, it starts looking for a table row (**<tr>** tag) and a list of cells (**<td>** tags), and when it encounters a **<td>** tag, it starts looking for the content of the cell, which can include nested tables. The grammar rules are used to build a syntax tree that represents the structure of the HTML file.

Assumptions:

1. The input file "html.txt" is assumed to exist and contain well-formed HTML table code.
2. The grammar rules assume that the HTML table structure follows the specified format, with a starting tag "<table>", followed by one or more rows of table data enclosed in "<tr>" tags, and ending with a closing "</table>" tag.
3. The tokens defined in the code assume that the input file contains only alphanumeric characters, digits, and the specific tags and markers used in the HTML table syntax.
4. The parser assumes that the input table is well-formed and does not contain any syntax errors or missing elements. Any missing or malformed tags could cause the parser to fail or produce incorrect results.
5. The code assumes that the user wants to parse an HTML table and does not provide any options for changing the input file or output format.

Lex Code(Html.l)

```
%{
    #include "html.tab.h"
    #undef yywrap
    #define yywrap() 1
%}
digit [0-9]+
alphanum [a-zA-Z0-9]+


%%
[\n\t\ ]   ;
"<"(t|T)(a|A)(b|B)(l|L)(e|E)">"  return START_TABLE;
"</"(t|T)(a|A)(b|B)(l|L)(e|E)">"  return END_TABLE;
"<"(T|t)(r|R)">"  return STR;
"<"(T|t)(d|D)">"  return STD;
"<"(T|t)(h|H)">"  return STH;
"</"(T|t)(r|R)">" return ETR;
"</"(T|t)(d|D)">" return ETD;
"</"(T|t)(h|H)">" return ETH;
{digit} return DIGIT;
{alphanum}  {   return ALNUM;}
.         return *yytext;
%%
```

Yacc Code(Html.y)

```
%{
  #include <stdio.h>
  #include <stdlib.h>
  extern FILE *yyin;           // Declare yyin as a pointer to FILE for input
file stream

  int yylex();                // Declare yylex function for tokenizing input
  int yyerror(char const *s)
  {
      printf("%s \n",s);    // Print error message s
      exit(0);             // Exit program with status code 0
  }

%}
%token START_TABLE STR STH STD ALNUM DIGIT END_TABLE ETR ETD ETH // Define the
tokens used in the grammar


%%
//Start of grammar rules
T        : STAG { printf("input accepted \n"); exit(0); }      // Top-level rule
that matches STAG and outputs a message before exiting
         ;

STAG       : START_TABLE BODY END_TABLE      // Rule that matches a table
definition
         ;

BODY     : STR BODY1 ETR      // Rule that matches a table row
         | STR BODY1 ETR BODY // Rule that matches multiple table rows
         ;

BODY1    : STD DATA ETD // Rule that matches a table cell with standard alignment
and data
         | STH DATA ETH  // Rule that matches a table cell with header alignment
and data
         | BODY1 STD DATA ETD // Rule that matches multiple table cells with
standard alignment and data
         | BODY1 STH DATA ETH // Rule that matches multiple table cells with
header alignment and data
         ;
```

```
DATA    : DIGIT   // Rule that matches a numeric data item
        | ALNUM   // Rule that matches an alphanumeric data item
        | DATA STAG // Rule that matches data followed by a subtable definition
        | STAG     // Rule that matches a subtable definition
        ;
%%  // End of grammar rules
int main()
{
    yyin = fopen("html.txt", "r"); // Open input file "table.html"
    yyparse();                     // Parse input using the defined grammar
rules
    return 0;                      // Exit program with status code 0
}
```

Test Cases:-

Pass Cases:

A simple table with one row and one column:

```
<table>
<tr>
<td>1</td>
</tr>
</table>
```

A table with multiple rows and columns:

```
<table>
<tr>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
</tr>
<tr>
<td>John</td>
<td>25</td>
<td>Male</td>
</tr>
<tr>
<td>Jane</td>
<td>30</td>
<td>Female</td>
</tr>
</table>
```

A table with nested tables:

```
<table>
<tr>
<th>Outer Table</th>
<th>Inner Table</th>
</tr>
<tr>
<td>
<table>
<tr>
<td>A</td>
<td>B</td>
</tr>
</table>
</td>
<td>
<table>
<tr>
<td>X</td>
<td>Y</td>
</tr>
</table>
</td>
</tr>
</table>
```

Fail Cases

Missing start tag:

```

<tr>
<td>1</td>
</tr>
</table>
```

Missing end tag:

```
<table>
<tr>
<td>1</td>
</tr>
```

Incorrectly nested tags:

```
<table>
<tr>
<td>
<table>
<td>A</td>
</table>
</td>
</tr>
</table>
```

Invalid characters in table data:

```
<table>
<tr>
<td>1@</td>
</tr>
</table>
```

Missing closing tag for table row:

```
<table>
<tr>
<td>1</td>
```

Extra closing tag for table row:

```
<table>
<tr>
<td>1</td>
</tr>
</tr>
</table
```

Explaination:-

This is a flex and bison (or yacc) code for a simple HTML table parser. The flex code defines the tokens and the bison code defines the grammar rules.

The tokens defined in the flex code are:

- START_TABLE
- STR (string)
- STH (table header string)
- STD (table data string)
- ALNUM (alphanumeric characters)
- DIGIT (digits)
- END_TABLE
- ETR (end of table row)
- ETD (end of table data)
- ETH (end of table header)

The grammar rules defined in the bison code are:

- T: This is the start symbol. It consists of a single STAG rule.
- STAG: This rule matches the start and end tags of a table and a BODY in between.
- BODY: This rule matches the table rows and consists of STR and DATA tokens. It can have multiple rows (BODY) separated by ETR tokens.
- BODY1: This rule matches the table data or header in each row. It can have multiple STD or STH tokens separated by DATA tokens. It can also have multiple rows (BODY1) separated by STD, STH, or DATA tokens.
- DATA: This rule matches any alphanumeric character or digit. It can also match another table (STAG) or a table row (BODY).

The main function sets the input file for the parser (yyin) and calls yyparse(), which is a function generated by bison that starts parsing according to the grammar rules.

If the input is successfully parsed according to the grammar rules, the T rule will be matched and the program will print "input accepted" and exit with code 0. Otherwise, if there is a syntax error, the yyerror() function is called, which prints an error message and exits with code 0

Goto Graph:-

graphlink