MCA SEM - IV

COMPILER DESIGN PROJECT

PARSING HTML TAGS FOR UNORDERED AND ORDERED LIST (WITH NESTING)

SUBMITTED BY :
CHAITALI SATIJA
RAVI BHUSHAN PRASAD
SANDEEP SAHU

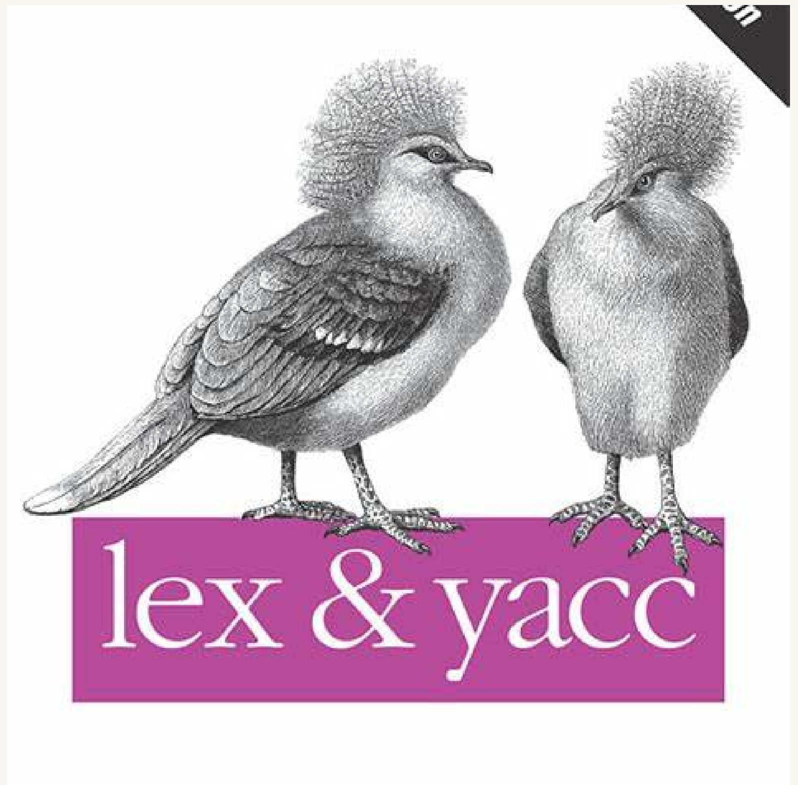
SUBMITTED TO :
DR. ANKIT RAJPAL

INTRODUCTION

BASIC OVERVIEW

Lex and **Yacc** are powerful tools that are widely used for parsing and analyzing computer programs. They are often used together to parse complex input languages, including HTML documents.

HTML documents often contain **lists**, which are used to organize content into ordered or unordered groups. It is important to ensure that these lists are properly formatted and validated, as invalid lists can cause issues with the display and functionality of the web page.



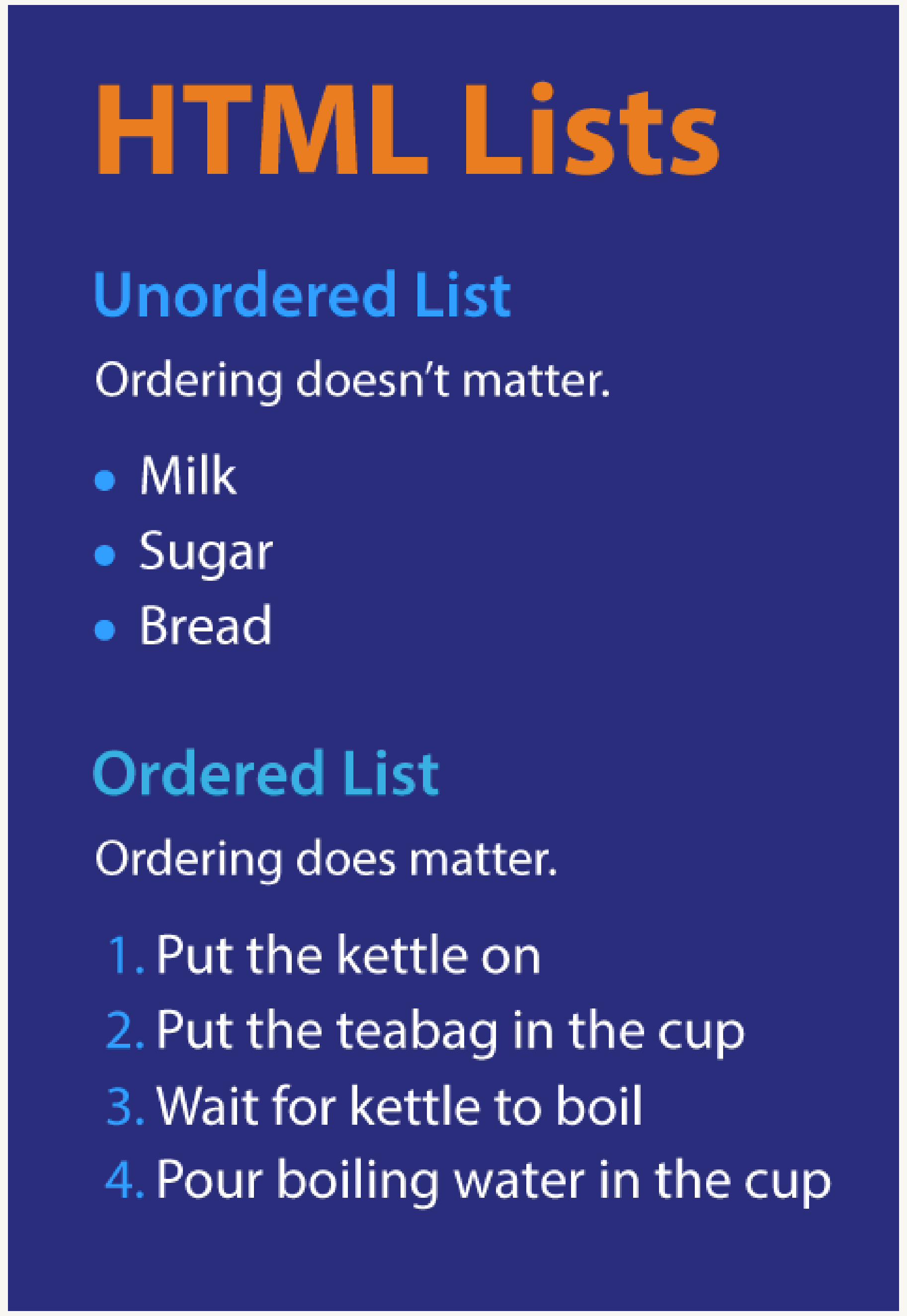
MCA SEM - IV

ABOUT THE PROJECT

AIM

TO PARSE HTML LIST TAGS

THROUGH THIS PROJECT, WE AIM TO VALIDATE / PARSE HTML LIST PROGRAMS CONSISTING OF ORDERED **(<OL> </OL>)** AND UNORDERED **(<UL> </UL>)** LIST TAGS BY DEVELOPING A PARSER USING LEX AND YACC TOOLS



HTML Lists

Unordered List

Ordering doesn't matter.

- Milk
- Sugar
- Bread

Ordered List

Ordering does matter.

1. Put the kettle on
2. Put the teabag in the cup
3. Wait for kettle to boil
4. Pour boiling water in the cup

TOOLS

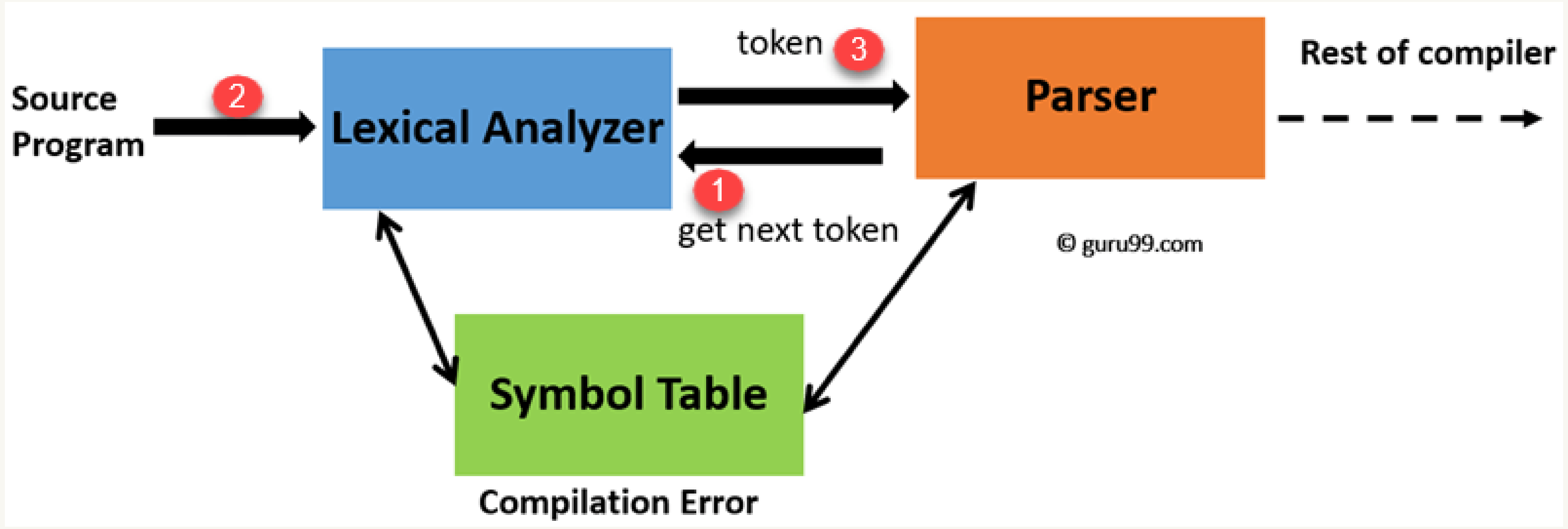
LEX
LEXICAL ANALYZER GENERATOR

Lex stands for "Lexical Analyzer Generator". It is a tool used to generate lexical analyzers, also known as **scanners** or **tokenizers.** It works in the **Lexical Analysis** phase of a Compiler. It reads an input stream and groups characters into meaningful characters called **Lexemes** based upon rules (a set of regular expressions and corresponding actions to be performed when lexemes match with any regex) specified in the lex file and produces **tokens** of the form <token-name, attribute-value>, which can then be parsed by a parser for further processing.

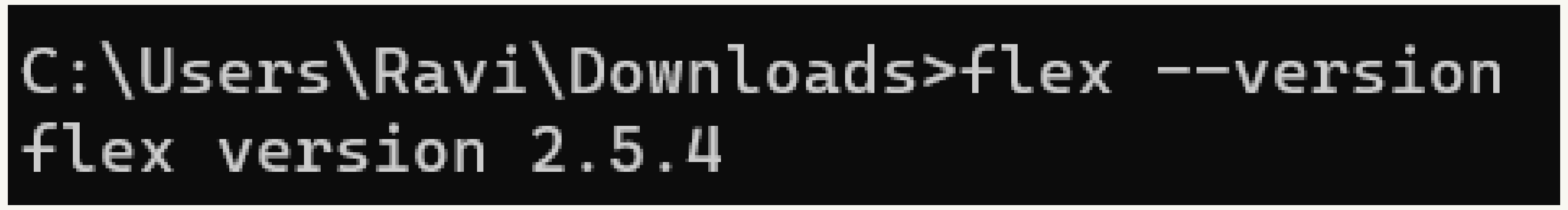
Flex stands for "Fast Lexical Analyzer Generator". It is an alternative to Lex, used in Windows, often used as a more modern and flexible replacement for Lex. Flex supports many of the same features as Lex, but also adds additional features such as support for backtracking and support for more advanced regular expression features.

Both Lex and Flex use a **.l file** to generate **lex.yy.c file** at the end of lexical analysis.

LEXICAL ANALYSIS



VERSION USED IN THE PROJECT



```
C:\Users\Ravi\Downloads>flex --version
flex version 2.5.4
```

COMMAND AND ITS INTERPRETATION

```
flex "file.l"        # generates a lex.yy.c file that specifies a set of regular expressions
                      and corresponding actions to be taken when each regular expression is matched.
```

LEX / FLEX

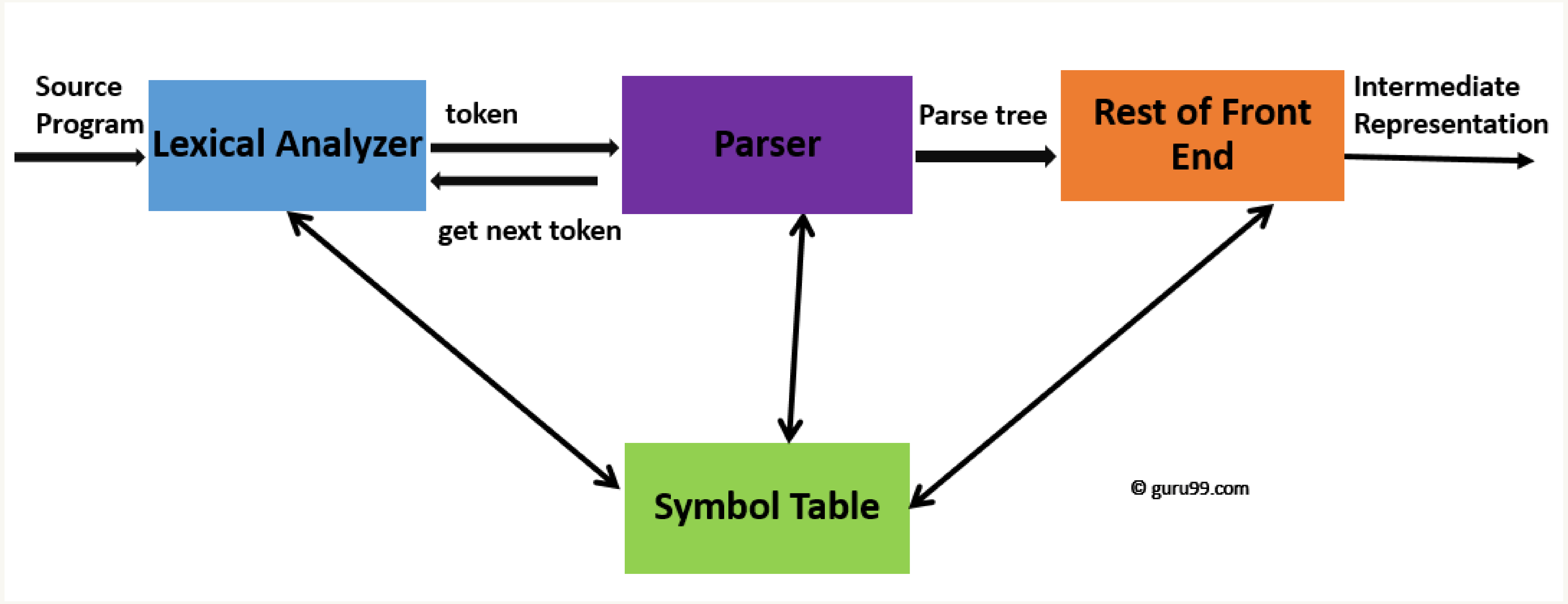
YACC
YET ANOTHER COMPILER COMPILER

Yacc stands for "Yet Another Compiler Compiler". It is a tool used to generate parsers based on a set of grammar rules. It works in the **Syntax Analysis** phase of a Compiler, which follows the Syntax Analysis phase. It takes input a set of grammar rules that define the structure of the language to be parsed and the tokens generated from Lexical Analysis and validates each token with respect to the supplied grammar and generates a **parse tree**.
There exists **2 types of parsers** - **Top-Down** (LL / predictive / recursive-descent) and **Bottom-Up** (LR - LR(0), SLR(1), CLR, LALR(1)) **parsers.** Yacc follows LR parser.

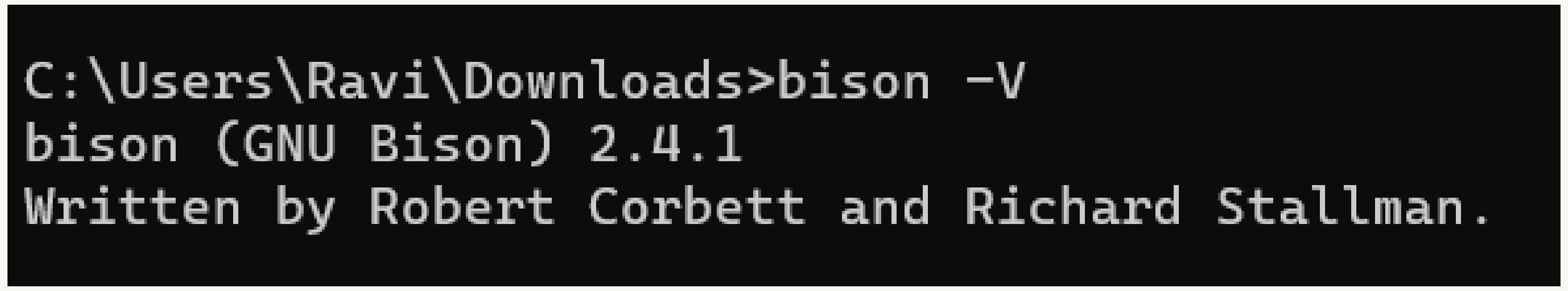
Bison is an updated version of Yacc, available on most Unix-like operating systems, as well as Windows and other platforms, that supports a wide range of parser types and features, including LALR(1) parsing, automatic conflict resolution, and support for semantic actions that can be used to construct parse trees or perform other tasks during parsing.

Both Yacc and Bison use a **.y file** to generate **y.tab.c file** at the end of syntax analysis which has the capability to parse input tokens with respect to supplied grammar rules.

SYNTAX ANALYSIS



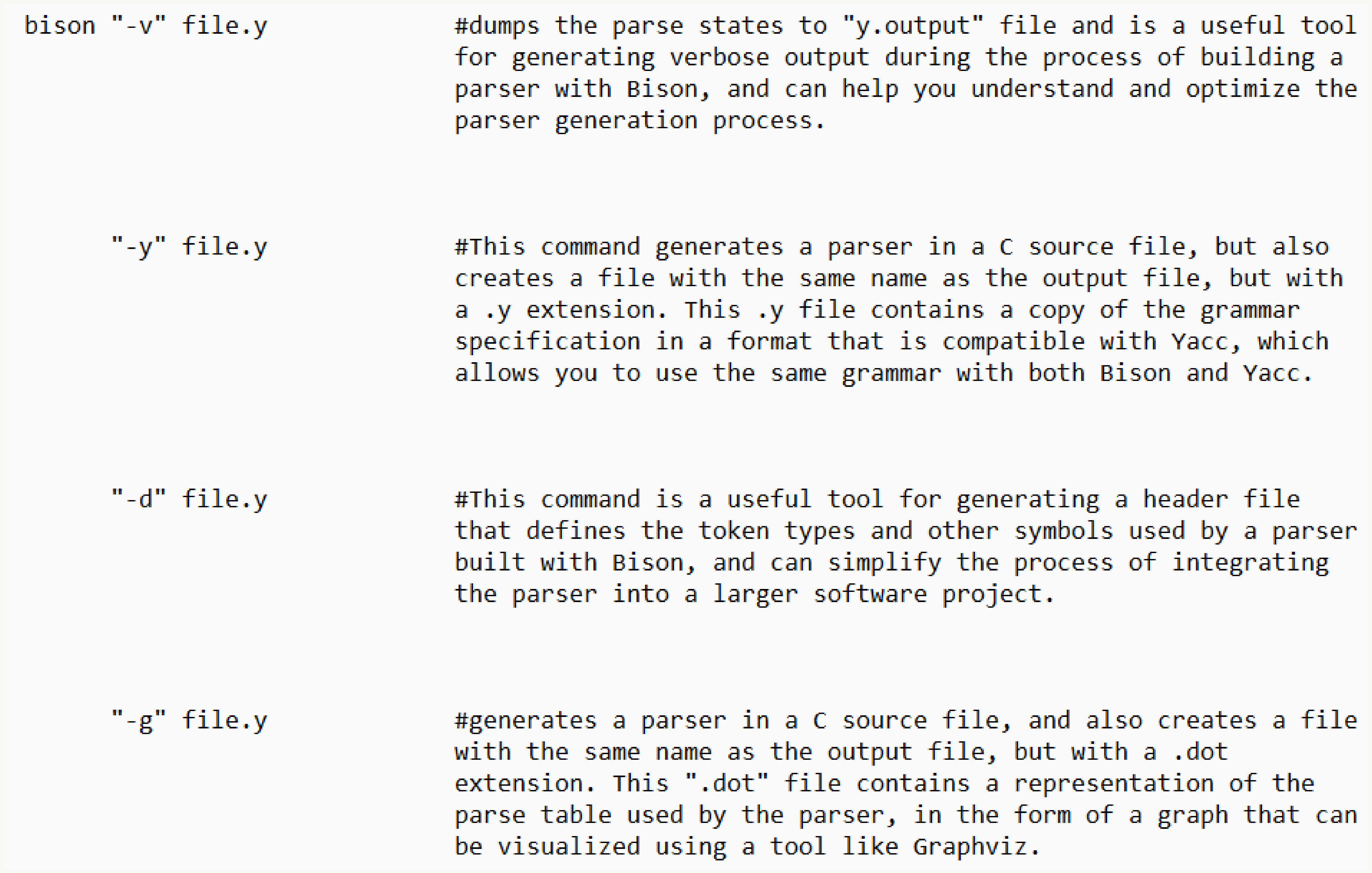
VERSION USED IN THE PROJECT



```
C:\Users\Ravi\Downloads>bison -V
bison (GNU Bison) 2.4.1
Written by Robert Corbett and Richard Stallman.
```

YACC / BISON

COMMAND AND ITS INTERPRETATION

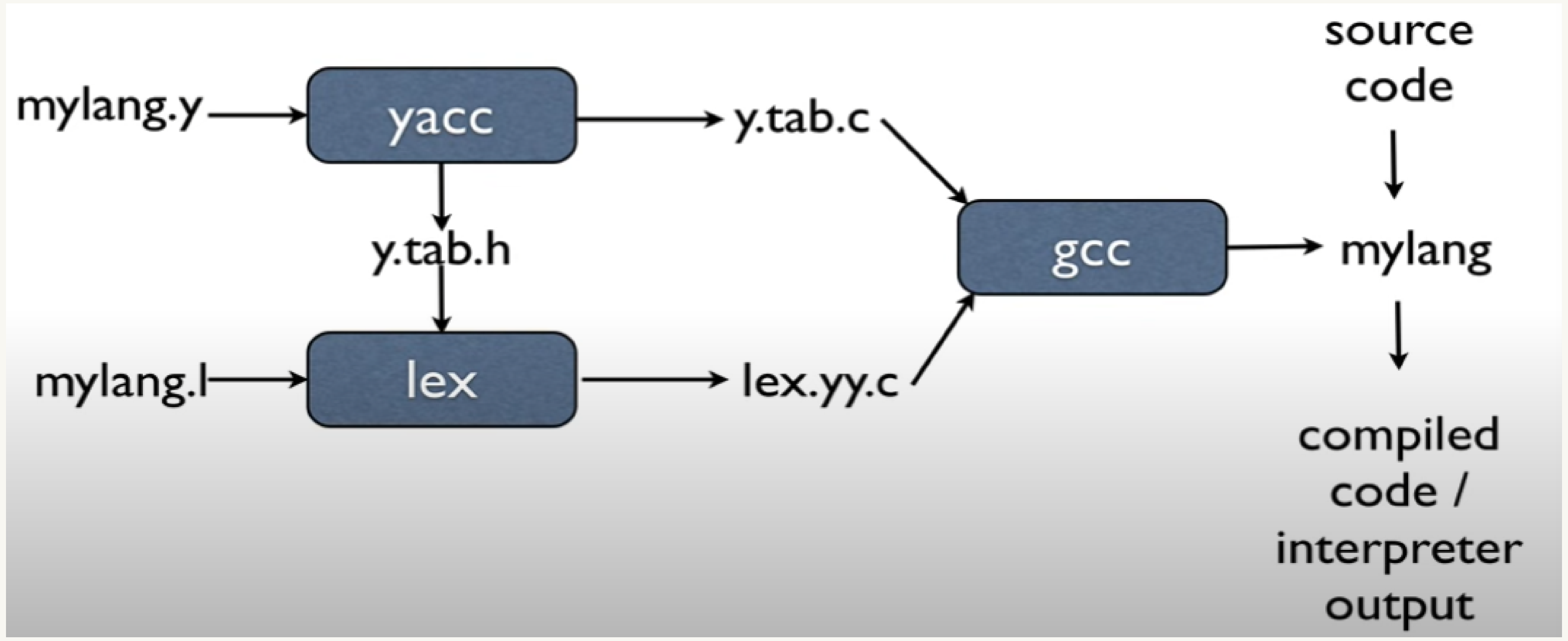


```
bison "-v" file.y            #dumps the parse states to "y.output" file and is a useful tool
                             for generating verbose output during the process of building a
                             parser with Bison, and can help you understand and optimize the
                             parser generation process.


      "-y" file.y            #This command generates a parser in a C source file, but also
                             creates a file with the same name as the output file, but with
                             a .y extension. This .y file contains a copy of the grammar
                             specification in a format that is compatible with Yacc, which
                             allows you to use the same grammar with both Bison and Yacc.


      "-d" file.y            #This command is a useful tool for generating a header file
                             that defines the token types and other symbols used by a parser
                             built with Bison, and can simplify the process of integrating
                             the parser into a larger software project.


      "-g" file.y            #generates a parser in a C source file, and also creates a file
                             with the same name as the output file, but with a .dot
                             extension. This ".dot" file contains a representation of the
                             parse table used by the parser, in the form of a graph that can
                             be visualized using a tool like Graphviz.
```

MCA SEM - IV

INTEGRATION BETWEEN LEX AND YACC

INTEGRATED WORKING OF LEX AND YACC



Step 01

Create .l and .y files. Lex file includes y.tab.h file verbatim before yacc executes.

Step 02

Linker links lex.yy.c generated by lex and y.tab.c generated by yacc

Step 03

Compile both the files using gcc compiler to produce an object file.

Step 04

Execute the object file with the desired input and the input is either declared valid / invalid.

SYNTAX

SYNTAX OF HTML LISTS

<ol> </ol> -> ORDERED LIST TAGS
<ul> </ul> -> UN-ORDERED LIST TAGS
<li> </li> -> LIST ITEM TAGS

NESTED LISTS

```
<ol><li></li><ul></ul></ol>
<ul><ol><li></li></ol></ul>
<ul><ol></ol></ul>
<ol><ol><li></li></ol><ul></ul></ol>
<ul><ol><ul><li></li></ul></ol></ul>
<li></li><ul><li></li><li><ol></ol><li></ul>
```

Note

HTML is a **case-insensitive** language. <Ol>, <uL>, <LI> tags are also valid.

ASSUMPTIONS

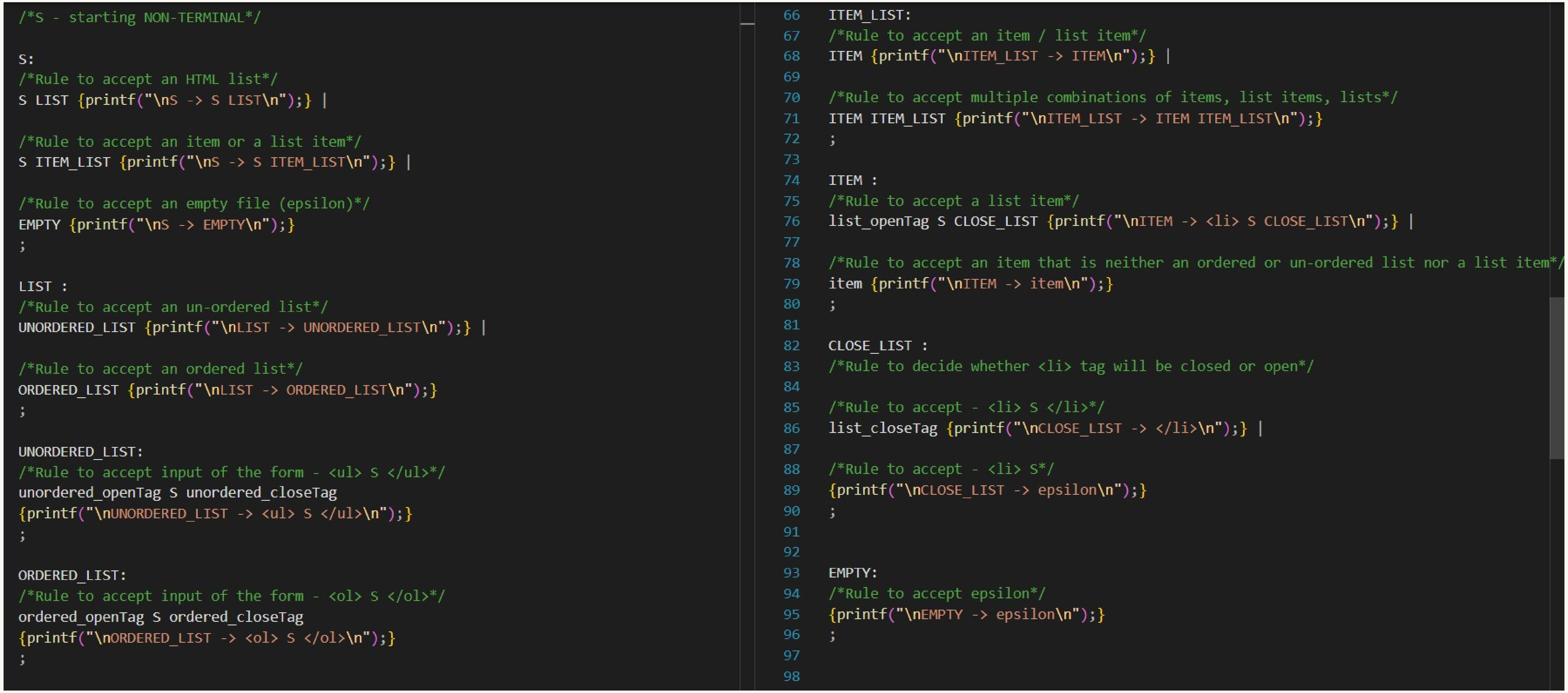
ASSUMPTIONS IN THE PROJECT

- An HTML list tag should occur alone in a single line.
- An HTML list tag should be exactly at the start of a line.
- Our program only parses the list tags and data inside, no other HTML tags other than List Tags can be parsed.
- Attributes of lists are not handled by our parser e.g. style, color, etc.
- Empty file is treated as a valid HTML file.
- Completely written code works correctly, copied code from the Internet might produce unexpected results.

MCA SEM - IV

GRAMMAR

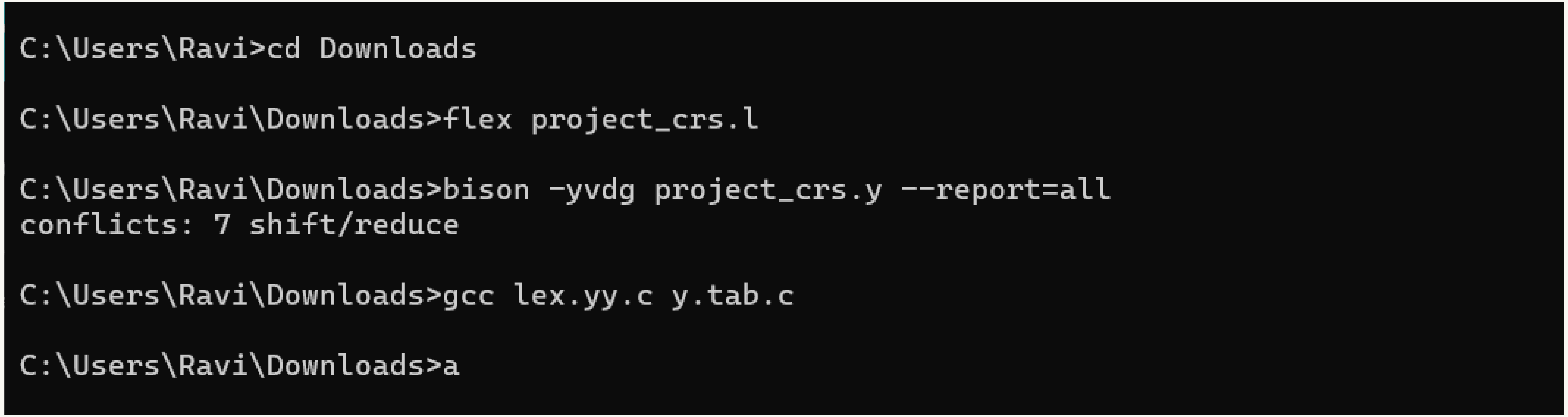
CONTEXT FREE GRAMMAR



```
/*S - starting NON-TERMINAL*/

S:
/*Rule to accept an HTML list*/
S LIST {printf("\nS -> S LIST\n");} |

/*Rule to accept an item or a list item*/
S ITEM_LIST {printf("\nS -> S ITEM_LIST\n");} |

/*Rule to accept an empty file (epsilon)*/
EMPTY {printf("\nS -> EMPTY\n");}
;

LIST :
/*Rule to accept an un-ordered list*/
UNORDERED_LIST {printf("\nLIST -> UNORDERED_LIST\n");} |

/*Rule to accept an ordered list*/
ORDERED_LIST {printf("\nLIST -> ORDERED_LIST\n");}
;

UNORDERED_LIST:
/*Rule to accept input of the form - <ul> S </ul>*/
unordered_openTag S unordered_closeTag
{printf("\nUNORDERED_LIST -> <ul> S </ul>\n");}
;

ORDERED_LIST:
/*Rule to accept input of the form - <ol> S </ol>*/
ordered_openTag S ordered_closeTag
{printf("\nORDERED_LIST -> <ol> S </ol>\n");}
;
```

```
ITEM_LIST:
/*Rule to accept an item / list item*/
ITEM {printf("\nITEM_LIST -> ITEM\n");} |

/*Rule to accept multiple combinations of items, list items, lists*/
ITEM ITEM_LIST {printf("\nITEM_LIST -> ITEM ITEM_LIST\n");}
;

ITEM :
/*Rule to accept a list item*/
list_openTag S CLOSE_LIST {printf("\nITEM -> <li> S CLOSE_LIST\n");} |

/*Rule to accept an item that is neither an ordered or un-ordered list nor a list item*/
item {printf("\nITEM -> item\n");}
;

CLOSE_LIST :
/*Rule to decide whether <li> tag will be closed or open*/

/*Rule to accept - <li> S </li>*/
list_closeTag {printf("\nCLOSE_LIST -> </li>\n");} |

/*Rule to accept - <li> S*/
{printf("\nCLOSE_LIST -> epsilon\n");}
;


EMPTY:
/*Rule to accept epsilon*/
{printf("\nEMPTY -> epsilon\n");}
;
```

EXECUTING THE PROJECT



```
C:\Users\Ravi>cd Downloads

C:\Users\Ravi\Downloads>flex project_crs.l

C:\Users\Ravi\Downloads>bison -yvdg project_crs.y --report=all
conflicts: 7 shift/reduce

C:\Users\Ravi\Downloads>gcc lex.yy.c y.tab.c

C:\Users\Ravi\Downloads>a
```

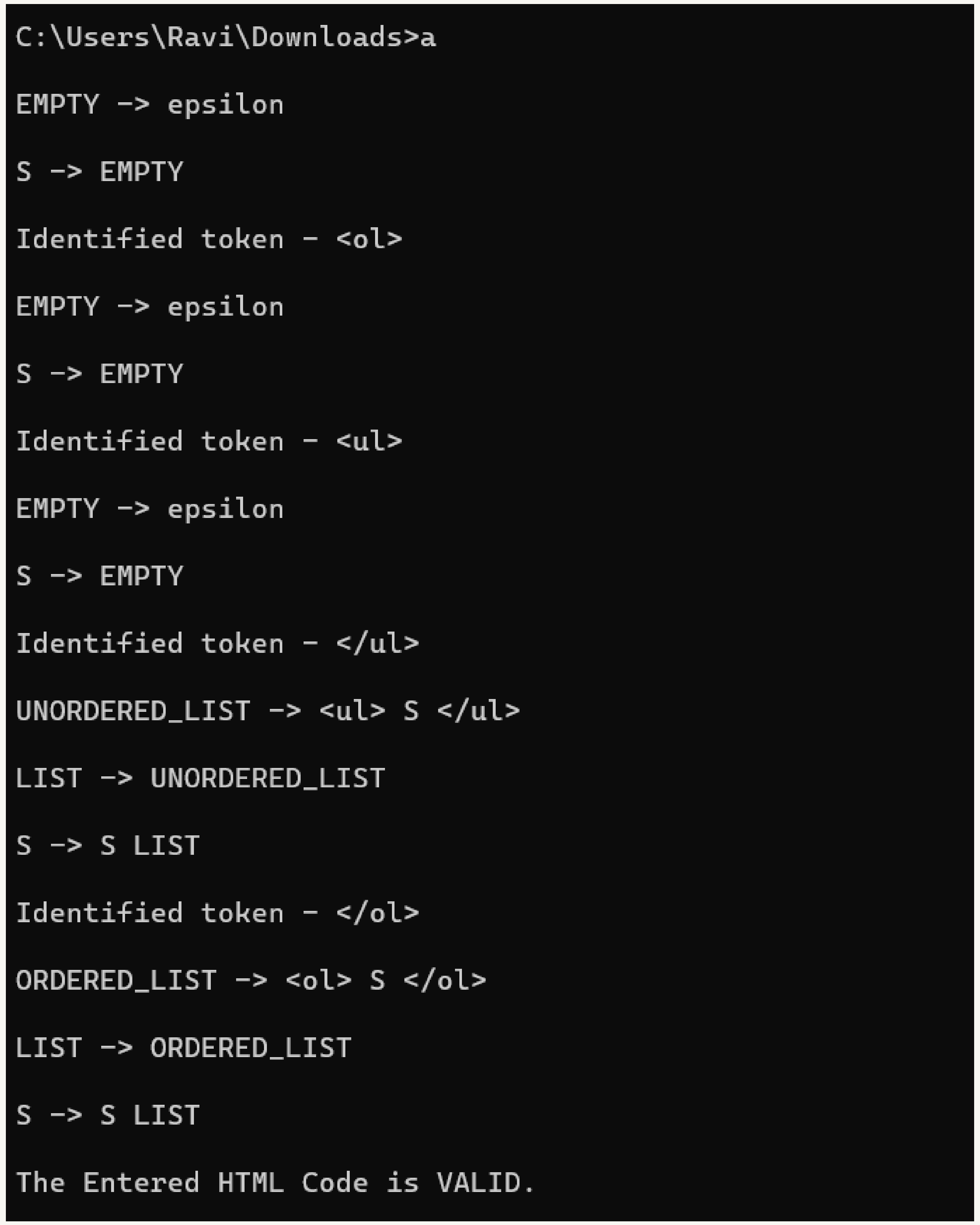
TESTCASES

A VALID TEST CASE (in file html.txt)

```
<ol>
<ul>
</ul>
</ol>
```

Actual Result - **VALID**

Predicted Result - **VALID**

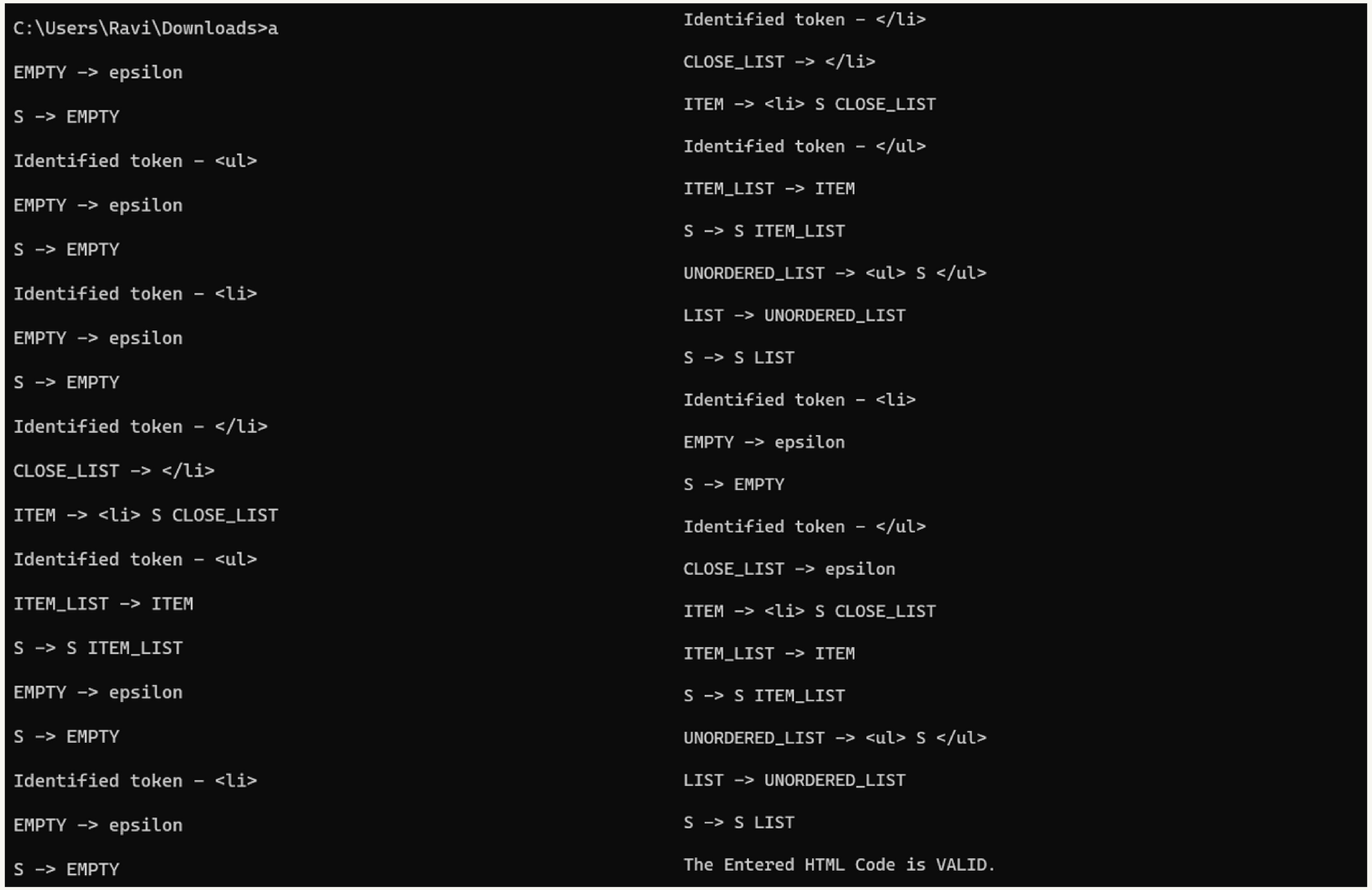


```
C:\Users\Ravi\Downloads>a
EMPTY -> epsilon
S -> EMPTY
Identified token - <ol>
EMPTY -> epsilon
S -> EMPTY
Identified token - <ul>
EMPTY -> epsilon
S -> EMPTY
Identified token - </ul>
UNORDERED_LIST -> <ul> S </ul>
LIST -> UNORDERED_LIST
S -> S LIST
Identified token - </ol>
ORDERED_LIST -> <ol> S </ol>
LIST -> ORDERED_LIST
S -> S LIST
The Entered HTML Code is VALID.
```

TESTCASES

A VALID TEST CASE (in file html.txt)

```
<ul>
<li>
</li>
<ul>
<li>
</li>
</ul>
<li>
</ul>
```



```
C:\Users\Ravi\Downloads>a
EMPTY -> epsilon
S -> EMPTY
Identified token - <ul>
EMPTY -> epsilon
S -> EMPTY
Identified token - <li>
EMPTY -> epsilon
S -> EMPTY
Identified token - </li>
CLOSE_LIST -> </li>
ITEM -> <li> S CLOSE_LIST
Identified token - <ul>
ITEM_LIST -> ITEM
S -> S ITEM_LIST
EMPTY -> epsilon
S -> EMPTY
Identified token - <li>
EMPTY -> epsilon
S -> EMPTY
Identified token - </li>
CLOSE_LIST -> </li>
ITEM -> <li> S CLOSE_LIST
Identified token - </ul>
ITEM_LIST -> ITEM
S -> S ITEM_LIST
UNORDERED_LIST -> <ul> S </ul>
LIST -> UNORDERED_LIST
S -> S LIST
Identified token - <li>
EMPTY -> epsilon
S -> EMPTY
Identified token - </ul>
CLOSE_LIST -> epsilon
ITEM -> <li> S CLOSE_LIST
ITEM_LIST -> ITEM
S -> S ITEM_LIST
UNORDERED_LIST -> <ul> S </ul>
LIST -> UNORDERED_LIST
S -> S LIST
The Entered HTML Code is VALID.
```

Actual Result - **VALID**

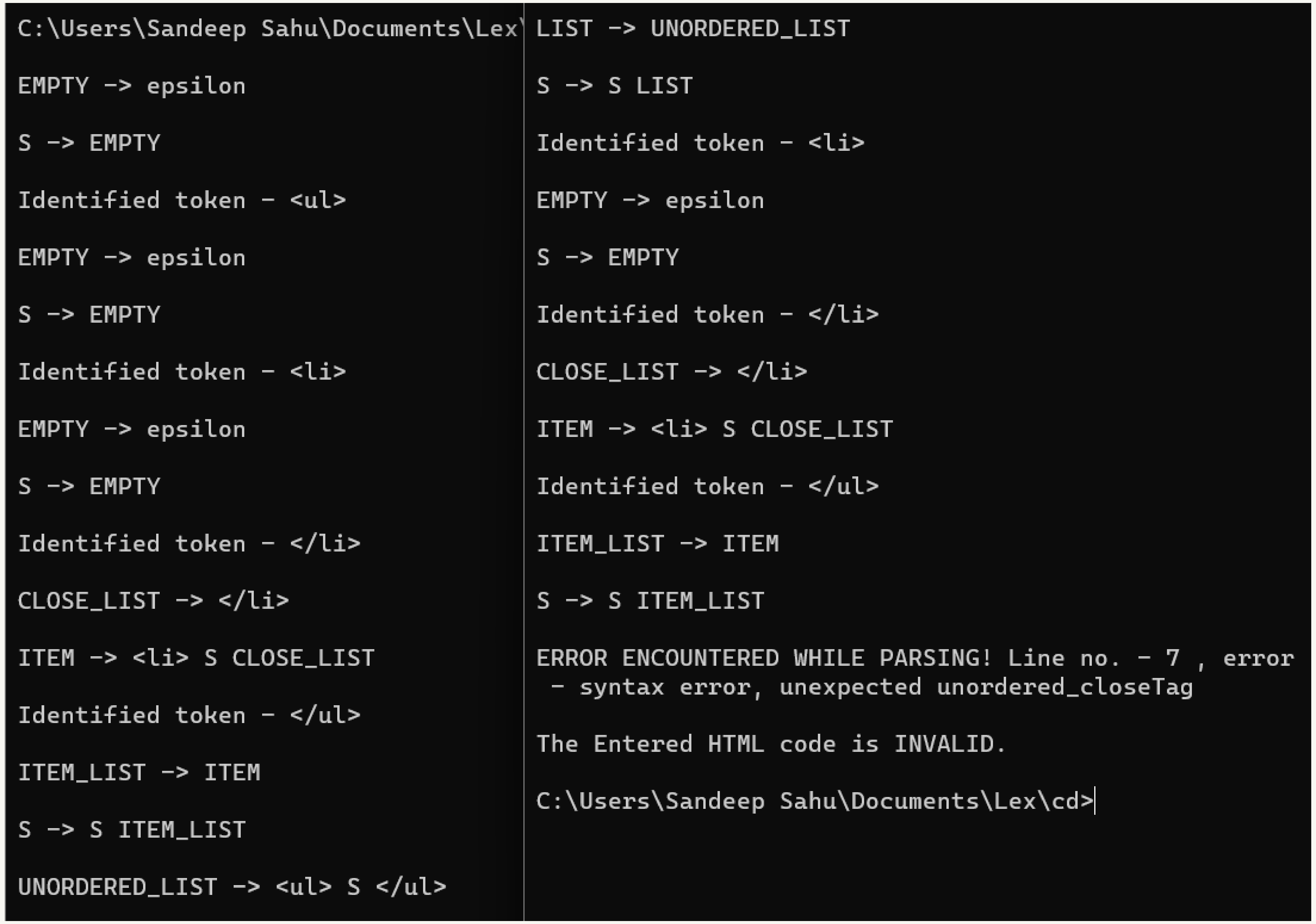
Predicted Result - **VALID**

MCA SEM - IV

TESTCASES

AN INVALID TEST CASE (in file html.txt)

```
<ul>
<li>
</li>
</ul>
<li>
</li>
</ul>
<li>
</ul>
```



```
C:\Users\Sandeep Sahu\Documents\Lex\
EMPTY -> epsilon
S -> EMPTY
Identified token - <ul>
EMPTY -> epsilon
S -> EMPTY
Identified token - <li>
EMPTY -> epsilon
S -> EMPTY
Identified token - </li>
CLOSE_LIST -> </li>
ITEM -> <li> S CLOSE_LIST
Identified token - </ul>
ITEM_LIST -> ITEM
S -> S ITEM_LIST
UNORDERED_LIST -> <ul> S </ul>
LIST -> UNORDERED_LIST
S -> S LIST
Identified token - <li>
EMPTY -> epsilon
S -> EMPTY
Identified token - </li>
CLOSE_LIST -> </li>
ITEM -> <li> S CLOSE_LIST
Identified token - </ul>
ITEM_LIST -> ITEM
S -> S ITEM_LIST
ERROR ENCOUNTERED WHILE PARSING! Line no. - 7 , error
 - syntax error, unexpected unordered_closeTag
The Entered HTML code is INVALID.
C:\Users\Sandeep Sahu\Documents\Lex\cd>
```

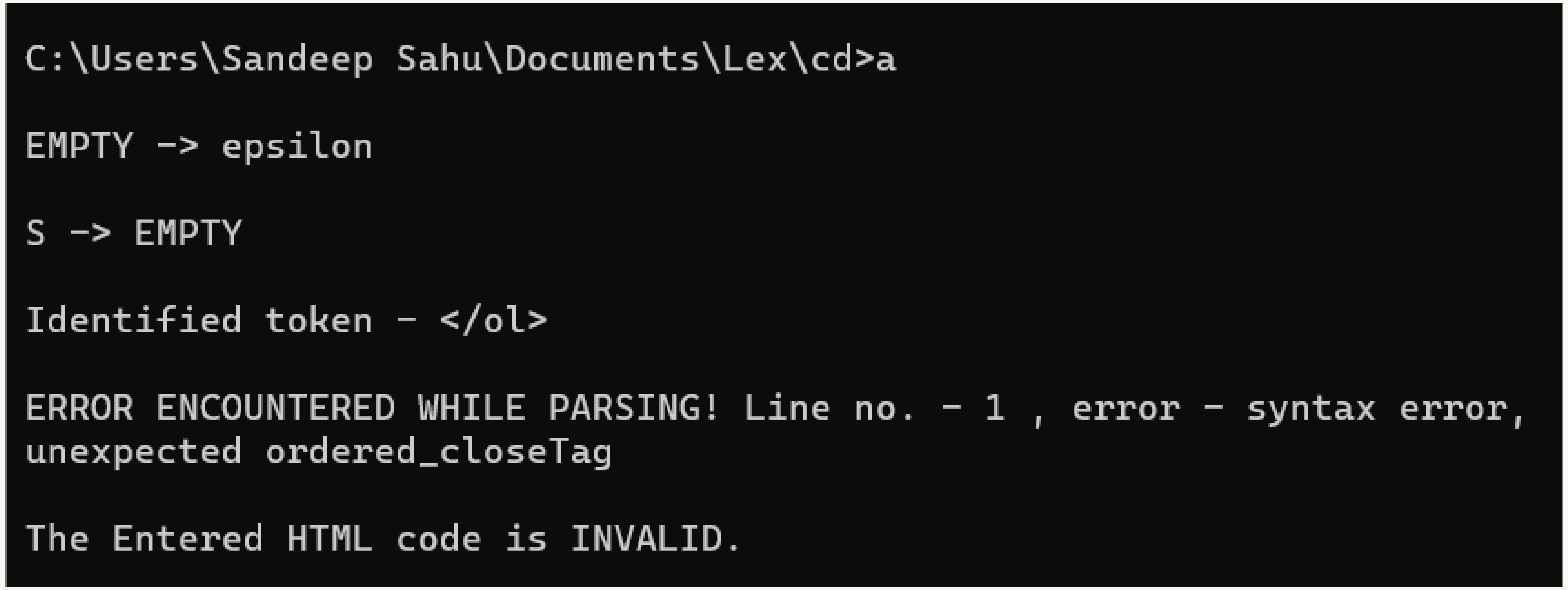
Actual Result - **INVALID**

Predicted Result - **INVALID**

TESTCASES

AN INVALID TEST CASE (in file html.txt)

```
</ol>
<ol>
```



```
C:\Users\Sandeep Sahu\Documents\Lex\cd>a

EMPTY -> epsilon

S -> EMPTY

Identified token - </ol>

ERROR ENCOUNTERED WHILE PARSING! Line no. - 1 , error - syntax error,
unexpected ordered_closeTag

The Entered HTML code is INVALID.
```

Actual Result - **INVALID**

Predicted Result - **INVALID**

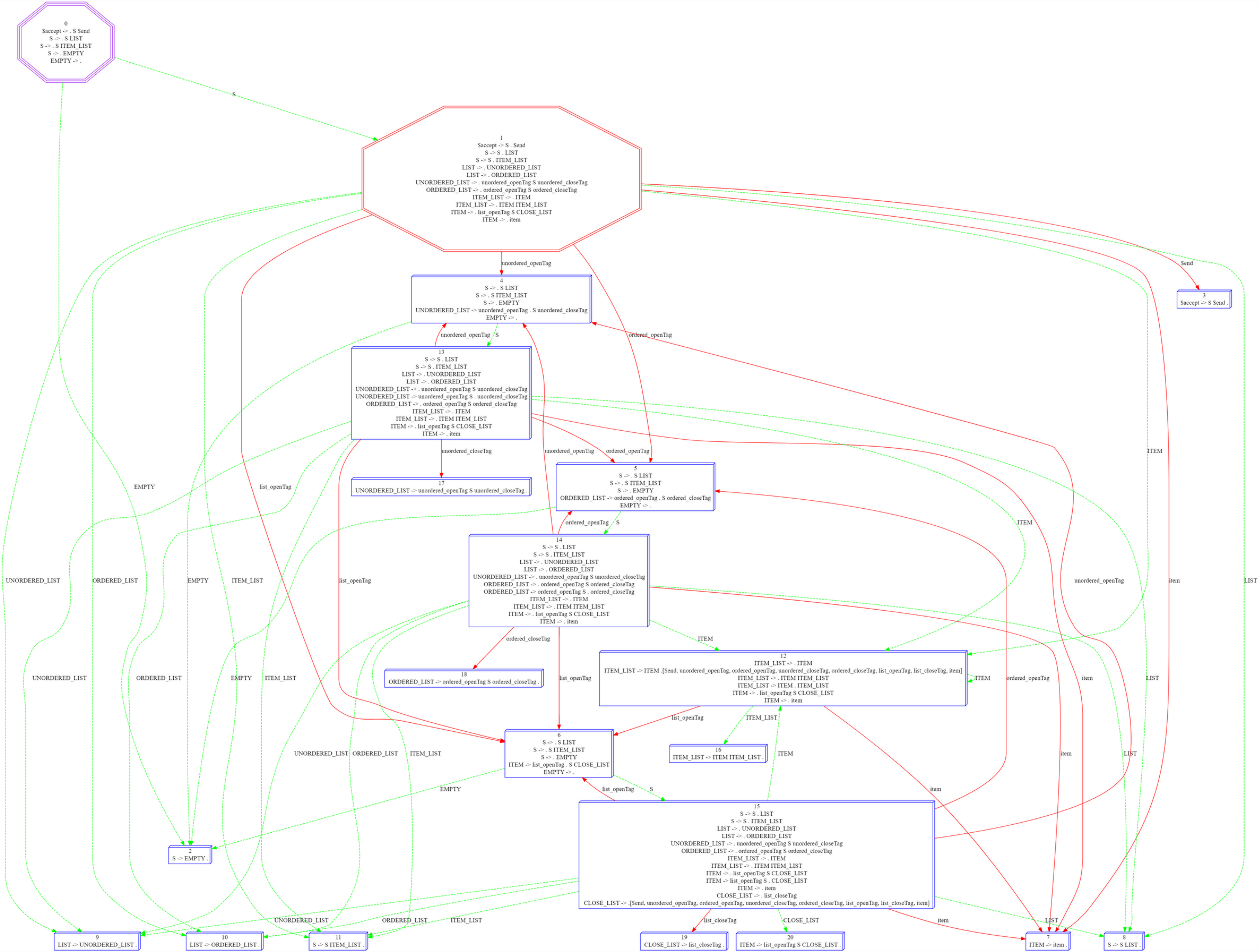
Note : The remaining test cases are mentioned in the testcases_crs.pdf document supplied alongwith the other files.

MCA SEM - IV

GO-TO GRAPH (AUTOMATON)

Generated using the command -

yacc -yvd --graph project_crs.y

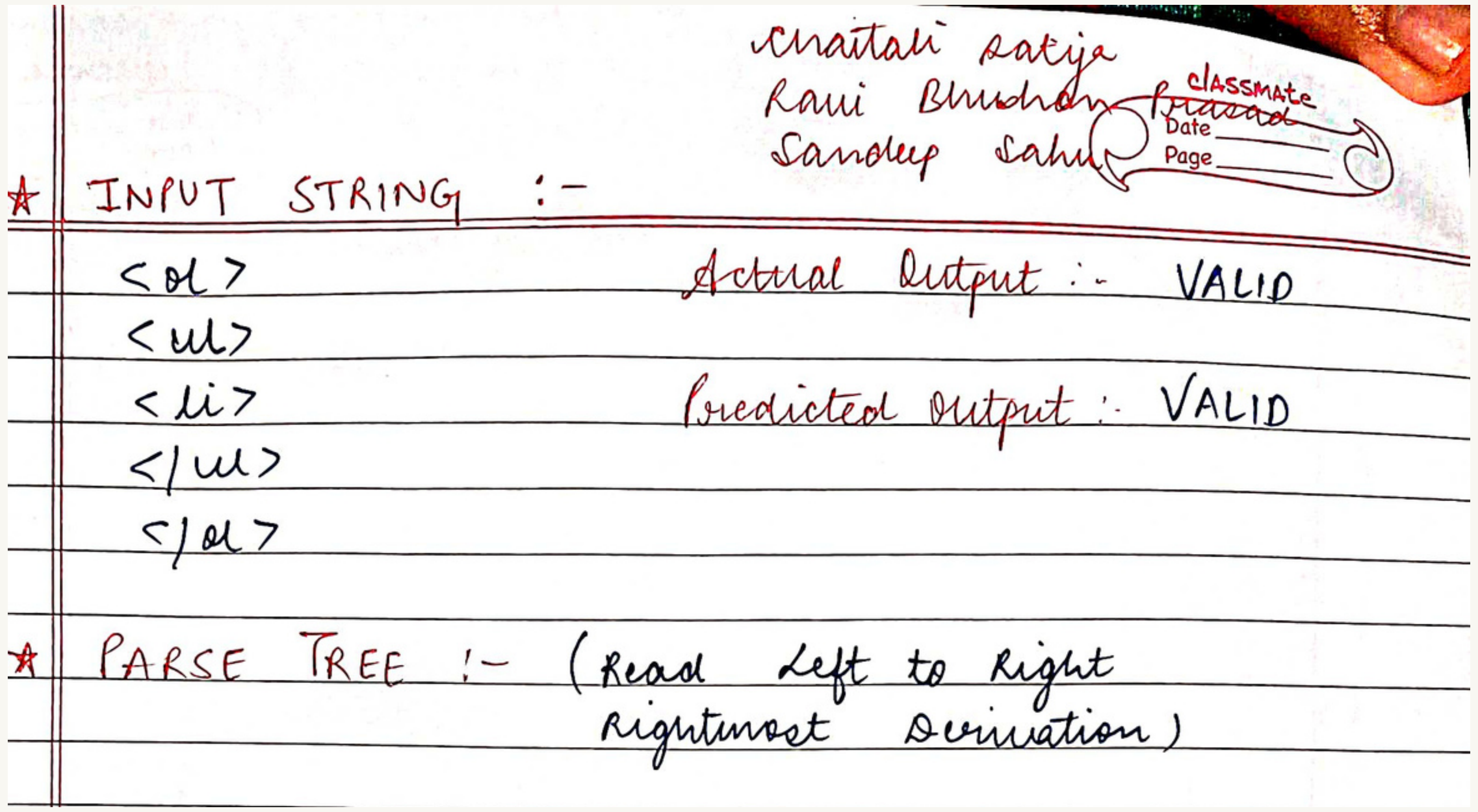


0 $accept -> . S $end S -> . S LIST S -> . S ITEM_LIST S -> . EMPTY EMPTY -> .
1 $accept -> S . $end S -> S . LIST S -> S . ITEM_LIST LIST -> . UNORDERED_LIST LIST -> . ORDERED_LIST UNORDERED_LIST -> . unordered_openTag S unordered_closeTag ORDERED_LIST -> . ordered_openTag S ordered_closeTag ITEM_LIST -> . ITEM ITEM_LIST -> . ITEM ITEM_LIST ITEM -> . list_openTag S CLOSE_LIST ITEM -> . item
2 S -> EMPTY .
3 $accept -> S $end .
4 S -> . S LIST S -> . S ITEM_LIST S -> . EMPTY UNORDERED_LIST -> unordered_openTag . S unordered_closeTag EMPTY -> .
5 S -> . S LIST S -> . S ITEM_LIST S -> . EMPTY ORDERED_LIST -> ordered_openTag . S ordered_closeTag EMPTY -> .
6 S -> . S LIST S -> . S ITEM_LIST S -> . EMPTY ITEM -> list_openTag . S CLOSE_LIST EMPTY -> .
7 ITEM -> item .
8 S -> S LIST .
9 LIST -> UNORDERED_LIST .
10 LIST -> ORDERED_LIST .
11 S -> S ITEM_LIST .
12 ITEM_LIST -> ITEM . ITEM_LIST -> ITEM .[$end, unordered_openTag, ordered_openTag, unordered_closeTag, ordered_closeTag, list_openTag, list_closeTag, item] ITEM_LIST -> . ITEM ITEM_LIST ITEM_LIST -> ITEM . ITEM_LIST ITEM -> . list_openTag S CLOSE_LIST ITEM -> . item
13 S -> S . LIST S -> S . ITEM_LIST LIST -> . UNORDERED_LIST LIST -> . ORDERED_LIST UNORDERED_LIST -> . unordered_openTag S unordered_closeTag UNORDERED_LIST -> unordered_openTag S . unordered_closeTag ORDERED_LIST -> . ordered_openTag S ordered_closeTag ITEM_LIST -> . ITEM ITEM_LIST -> . ITEM ITEM_LIST ITEM -> . list_openTag S CLOSE_LIST ITEM -> . item
14 S -> S . LIST S -> S . ITEM_LIST LIST -> . UNORDERED_LIST LIST -> . ORDERED_LIST UNORDERED_LIST -> . unordered_openTag S unordered_closeTag ORDERED_LIST -> . ordered_openTag S ordered_closeTag ORDERED_LIST -> ordered_openTag S . ordered_closeTag ITEM_LIST -> . ITEM ITEM_LIST -> . ITEM ITEM_LIST ITEM -> . list_openTag S CLOSE_LIST ITEM -> . item
15 S -> S . LIST S -> S . ITEM_LIST LIST -> . UNORDERED_LIST LIST -> . ORDERED_LIST UNORDERED_LIST -> . unordered_openTag S unordered_closeTag ORDERED_LIST -> . ordered_openTag S ordered_closeTag ITEM_LIST -> . ITEM ITEM_LIST -> . ITEM ITEM_LIST ITEM -> . list_openTag S CLOSE_LIST ITEM -> list_openTag S . CLOSE_LIST ITEM -> . item CLOSE_LIST -> . list_closeTag CLOSE_LIST -> .[$end, unordered_openTag, ordered_openTag, unordered_closeTag, ordered_closeTag, list_openTag, list_closeTag, item]
16 ITEM_LIST -> ITEM ITEM_LIST .
17 UNORDERED_LIST -> unordered_openTag S unordered_closeTag .
18 ORDERED_LIST -> ordered_openTag S ordered_closeTag .
19 CLOSE_LIST -> list_closeTag .
20 ITEM -> list_openTag S CLOSE_LIST .
S
$end
unordered_openTag
ordered_openTag
list_openTag
unordered_closeTag
ordered_closeTag
list_closeTag
EMPTY
UNORDERED_LIST
ORDERED_LIST
ITEM_LIST
ITEM
LIST
item
CLOSE_LIST

AUTOMATON

MCA SEM - IV

PARSE TREE

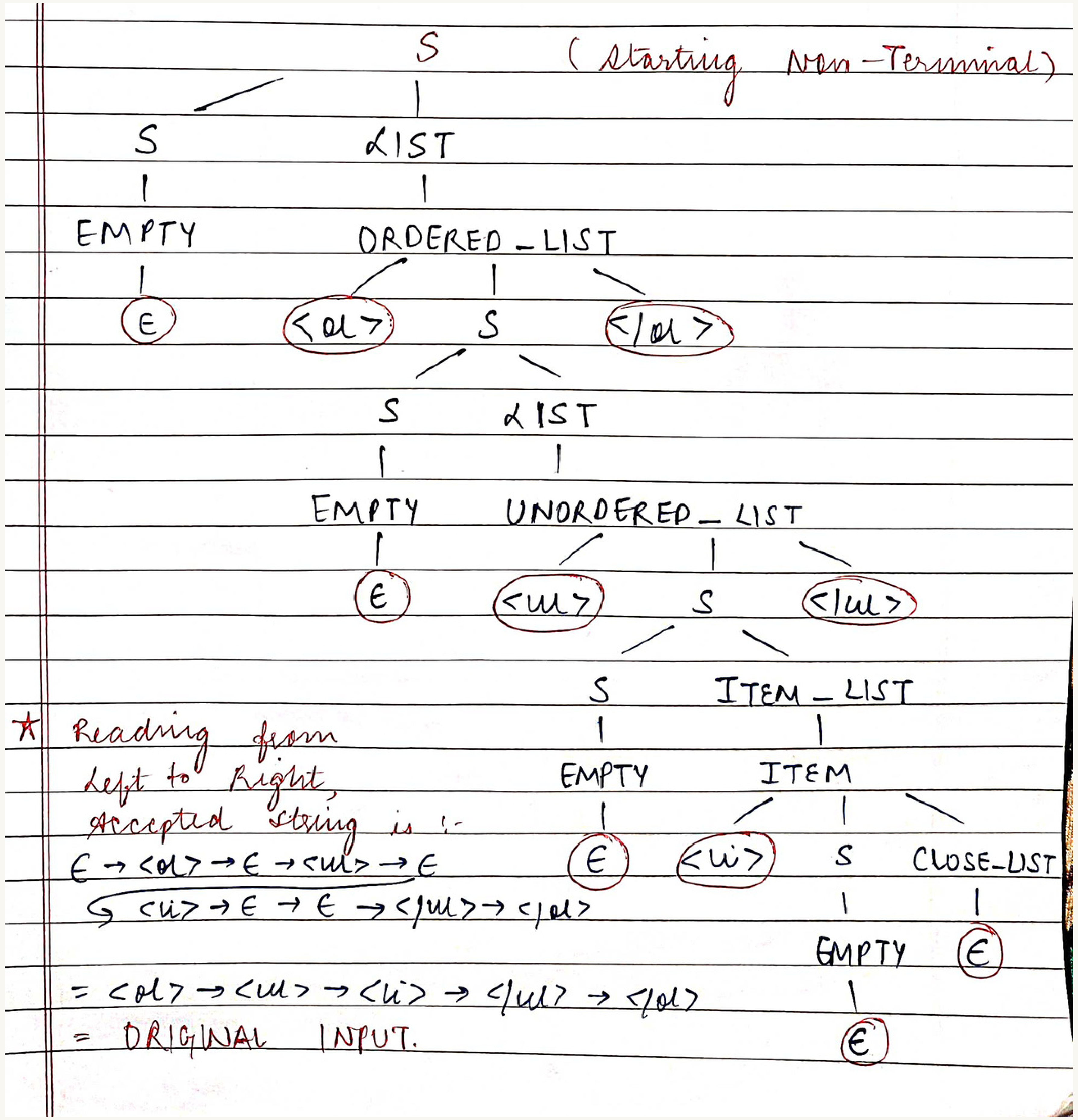


Chaitali Satija
Ravi Bhushan Prasad
Sandeep Sahu

★ INPUT STRING :-

<ol>	Actual Output :- VALID
<ul>	
<li>	Predicted Output :- VALID
</ul>	
</ol>	

★ PARSE TREE :- (Read Left to Right Rightmost Derivation)



★ Reading from Left to Right, Accepted string is :-
$\epsilon \rightarrow$ <ol> $\rightarrow \epsilon \rightarrow$ <ul> $\rightarrow \epsilon$
$\hookrightarrow$ <li> $\rightarrow \epsilon \rightarrow \epsilon \rightarrow$ </ul> $\rightarrow$ </ol>

= <ol> $\rightarrow$ <ul> $\rightarrow$ <li> $\rightarrow$ </ul> $\rightarrow$ </ol>
= ORIGINAL INPUT.

PARSE TREE

PROJECT FILES

Files supplied for the project

1. project_crs.l
2. project_crs.y
3. y.output
4. html.txt
5. testcases_crs.pdf
6. automaton_crs.png
7. Compiler Design Project Report - CRS

MCA SEM - IV

THANK YOU

SUBMITTED BY :
CHAITALI SATIJA
RAVI BHUSHAN PRASAD
SANDEEP SAHU

SUBMITTED TO :
DR. ANKIT RAJPAL