

# **EMULATOR FOR BOTTOM-UP PARSING –CLR**

## **A MINI PROJECT REPORT**

***Submitted by***

**G V N S YASWANTH [RA2011003011160]  
K VAMSI KRISHNA[RA2011003011161]**

***Under the guidance of***

**Dr. Anitha K**

(Assistant Professor, Department of  
Computing Technologies)

***In partial satisfaction of the requirements for the degree of***

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE & ENGINEERING  
With specialization in Computing  
Technologies**



**SCHOOL OF COMPUTING  
COLLEGE OF ENGINEERING AND  
TECHNOLOGY SRM INSTITUTE OF SCIENCE  
AND TECHNOLOGY KATTANKULATHUR -  
603203  
APRIL 2023**



COLLEGE OF ENGINEERING & TECHNOLOGY  
SRM INSTITUTE OF SCIENCE & TECHNOLOGY  
S.R.M. NAGAR, KATTANKULATHUR – 603 203

## BONAFIDE CERTIFICATE

Certified that this project report "**EMULATOR FOR BOTTOM - UP PARASING – CLR**" is the bonafide work of "**G V N S YASWANTH [RA2011003011161], K VAMSI KRISHNA [RA2011003011160]**" of III Year/VI Sem B.Tech(CSE) who carried out the mini project work under my supervision for the course 18CSC304J – Compiler Design in SRM Institute of Science and Technology during the academic year 2022-2023(Even Sem)

### SIGNATURE

**Dr. Anitha K**  
Assistant Professor  
**Department of Computing**  
**Technologies**

### SIGNATURE

**Dr. M. PUSHPALATHA**  
HEAD OF THE DEPARTMENT  
**Department of Computing**  
**Technologies**

## ABSTRACT

---

The objective of this project is to implement an efficient and robust CLR parser using a high-level programming language, that is capable of correctly parsing any input fed to it. The LR(1) parsing is a technique of bottom-up parsing. ‘L’ says that the input string is scanned from left to right, ‘R’ says that the parsing technique uses rightmost derivations, and ‘1’ stands for the look-ahead. To avoid some of the invalid reductions, the states need to carry more information. Extra information is put into the state by adding a terminal symbol as the second component in the item.

Thus, the canonical-LR parser makes full use of look-ahead symbols. This method uses a large set of items, called LR(1) items.

The LR(1) parsing method consists of a parser stack, that holds non-terminals, grammar symbols and tokens; a parsing table that specifies parser actions, and a driver function that interacts with the parser stack, table, and scanner. The typical actions of a CLR parser include shift, reduce, and accept or error.

The project work would include a set of predefined grammar and an interface which would convert each phase of the parsing process into a visual representation and would display onto webpage. The implementation is straight forward and simple. Then it would take any input string belonging to the grammar language and would show the acceptance or rejection of that input string and the steps one by one.

## Introduction

---

The LR parser is a non-recursive, shift-reduce, bottom-up parser. It uses a wide class of context-free grammar which makes it the most efficient syntax analysis technique.

LR parsers are also known as LR(k) parsers, where L stands for left-to-right scanning of the input stream; R stands for the construction of right-most derivation in reverse, and k denotes the number of look ahead symbols to make decisions. LR parsing does a rightmost derivation in reverse.

LR(1) parser works on complete set of LR(1) grammar, which makes full use of the look-ahead symbols. It generates a large table with many states. An LR(1) item is a two-component element of the form where the first component is a marked production, called the core of the item and the second is a look ahead character that belongs to the super set.

## Algorithm

---

### Algorithm for constructing LR(1) sets of items

Input: An augmented grammar  $G^1$ .

Output: The sets of LR(1) items that are the set of items valid for one or more viable prefixes of  $G^1$ .

```
SetOfItems CLOSURE(I) { repeat for
    (each item  $[A \rightarrow \alpha.B\beta,a]$  in I )
        for (each production  $B \rightarrow \gamma$  in  $G^1$ )
            for (each terminal b in FIRST( $\beta a$ ) )
                add  $[B \rightarrow .\gamma, b]$  to set I;
    until no more items are added to I;
    return I;
}
```

```
SetOfItems GOTO(I,X) {
    initialize J to be the empty set; for (each
        item  $[A \rightarrow \alpha.X\beta,a]$  in I ) add item
         $[A \rightarrow \alpha X.\beta,a]$  to set J;
    return CLOSURE(J);
}
```

```
SetOfItems items( $G^1$ ) { initialize C to
    CLOSURE( $\{[S^1 \rightarrow .S, \$]\}$ ); repeat for
    (each set of items I in C )
        for (each grammar symbol X )
            if ( $GOTO(I,X)$  is not empty and not in C )
                add  $GOTO(I,X)$  to C;
    until no new set of items are added to C; }
```

### Algorithm for constructing Canonical LR(1) parsing table

Input: An augmented grammar  $G^1$ .

Output: The canonical-LR parsing table functions ACTION and GOTO for  $G^1$ .

Method:

Construct  $C^l = \{ I_0, I_1, \dots, I_n \}$ , the collection of sets of LR(1) items for  $G^l$ .

State  $i$  of the parser is constructed from  $I_i$ . The parsing action for state  $i$  is determined as follows.

- A) If  $[A \rightarrow \alpha.a\beta, b]$  is in  $I_i$  and  $\text{GOTO}(I_i, a) = I_j$ , then set  $\text{ACTION}[i, a]$  to “shift  $j$ ”. Hence  $a$  must be a terminal.
- B) If  $[A \rightarrow \alpha.., a]$  is in  $I_i$ ,  $A \neq S^l$ , then set  $\text{ACTION}[i, a]$  to “reduce  $A \rightarrow \alpha$ .”
- C) If  $[S^l \rightarrow S.., \$]$  is in  $I_i$ , then set  $\text{ACTION}[i, \$]$  to “accept”.

If any conflicting actions result from the above rules, we say the grammar is not LR(1). The algorithm fails to produce a parser in this case.

The goto transitions for state  $i$  are constructed for all nonterminals  $A$  using the rule: If  $\text{GOTO}(I_i, a) = I_j$ , then  $\text{GOTO}[i, A] = j$ .

All entries not defined by rules (2) and (3) are made “error”.

The initial state of the parser is the one constructed from the set of items containing  $[S^l \rightarrow .S, \$]$ .



## Source Code

```
132 <?php
133 /* 1-states
134 2- terminals
135 3- parsing table
136 4-non terminals
137 5- input parsing
138 7- grammar
139 */
140 $choice=$_REQUEST['first'];
141
143 if($choice==1)
144 $res=shell_exec("C:/xampp/htdocs/CDMini/CD/CodeProject.Syntax.LALR/CD_grammar1/bin/Debug/CD_grammar1.exe 7 acede$");
145 else if($choice==3)
146 $res=shell_exec("C:/xampp/htdocs/CDMini/CD/CodeProject.Syntax.LALR2/CD_grammar1/bin/Debug/CD_grammar1.exe 7 abc$");
147 else if($choice==2)
148 $res=shell_exec("C:/xampp/htdocs/CDMini/CD/CodeProject.Syntax.LALR3/CD_grammar1/bin/Debug/CD_grammar1.exe 7 101$");
149
150
151 echo $res."<br><hr>";
152 $split = array();
153 $state = array();
154 $temp = array();
155 $split=explode("Grammar", $res);
156 print_r( $split);
157
158 $temp=explode("end", $split[1]);
159 print_r( $temp);
160 print_r( $state);
161
162 echo '
163 <div class="container">
164 | <h2>Grammar</h2>
165 | <table class="table table-hover">
166 |
167 | <tbody>';

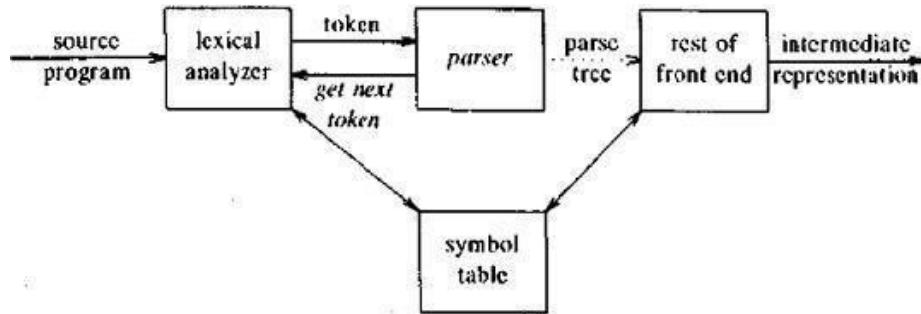
168
```

```
114 <div id="flip">Enter Input</div>
115 <div id="panel">
116
117 <?php
118 $choice=$_REQUEST['parser'];
119
120 ?>
121 <form action="parser.php" method="POST">
122 Enter the input string:<input type="text" name="string">
123
124 <?php
125 if($choice==1)
126 echo '<input type="hidden" name="parser" value="1" ><br><br>';
127 else if($choice==2)
128 echo '<input type="hidden" name="parser" value="2" ><br><br>';
129 else if($choice==3)
130 echo '<input type="hidden" name="parser" value="3" ><br><br>';
131
132
133 ?>
134 <button name="submit" >Submit</button>
135 </div></div>
136
137
138 </form>
139 </div>
```

## Relevance with respect to other compiler phases

In a typical compiler model, the parser obtains a string of tokens from the lexical analyzer, as shown in Fig. 4.1, and verifies that the string can be

generated by the grammar for the source language. We expect the parser to report any syntax errors in an intelligible fashion. It should also recover from commonly occurring errors so that it can continue processing the remainder of its input.



**Fig. 4.1.** Position of parser in compiler model.

The parsing takes place in the syntax analysis phase. The job of this phase is to build a relationship between the lexeme values and generate a syntax tree. One more job of the phase to handle errors. Application programmers frequently write incorrect programs, and a good compiler should assist the programmer in identifying and locating errors.

The generated tree is then passed onto the intermediate code representation. Output of the parser is some representation of the parse tree for the stream of tokens produced by the lexical analyzer. In practice, there are a number of tasks that might be conducted during parsing, such as collecting information about various tokens into the symbol table, performing type checking and other kinds of semantic analysis, and generating intermediate code.

## **Code:**

### **Items.php:-**

```
<!DOCTYPE html>

<html>

<head>

<title>CLR Parser</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<link href="css/style3.css" rel="stylesheet" type="text/css" media="all" />

<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">

<link rel="css/bootstrap.min.css">

<link href="css/bootstrap.css" rel='stylesheet' type='text/css' />

<link rel="stylesheet" type="text/css" href="css/animate.css"/>

<script src="js/jquery.min.js"></script>

<script type="text/javascript" src="js/move-top.js">
</script>

<script src="js/bootstrap.min.js"></script>

<script type="text/javascript" src="js/wow.js"></script>

<script src="js/wow.min.js"></script>
```

```
<script type="text/javascript" src="js/easing.js"></script>
<script type="text/javascript" src="js/main.js"></script>
<script type="text/javascript">
jQuery(document). ready(function ($){
  $(". scroll"). click(function(event){
    event.preventDefault();
    $('html,body'). animate({scroll Top:$ (this.hash). offset(). top},900);
  });
});
</script>
```

```
<script> new
WOW().init();
</script>
```

```
<script>
$(document).ready(function(){
  $("#flip").click(function(){
    $("#panel").slideToggle("slow");
  });
});
$(document). ready (function (){
  $("#flip2"). click(function (){
    $("#panel2").slideToggle("slow");
  });
});
```

```
    });
}

</script>

<style>
.carousel-inner > .item > img,
.carousel-inner > .item > a > img { width:
100%;
margin: auto;

}

#panel, #flip ,#panel2,#flip2{ padding:
30px;
text-align: center; background-color:#484848;
font-style: bold; font-size: 14pt; letter-
spacing: 6px;
color: Coral;

border: solid 4px NavajoWhite;
}

#panel, #panel2{
padding: 60px;
display: none;
}
```

```
</style>

</head>

<body>

<!-- Header Section --&gt;

&lt;header&gt;

&lt;div class="container"&gt;

&lt;div class="logo pull-left animated wow fadeInLeft"&gt;
<img alt="CLR PARSER logo" data-bbox="111 360 883 380"/>CLR PARSER

</div>

<nav class="pull-right">
<ul class="list-unstyled">
<li class="animated wow fadeInLeft" data-wow-delay=".1s"><a href="about.html">About</a></li>
<li class="animated wow fadeInLeft" data-wow-delay=".1s"><a href="algo.html">Algorithm</a></li>
<li class="animated wow fadeInLeft" data-wow-delay=".1s"><a href="http://github.com/deepakjayaprakash/Emulator-for-CLR-Parser">GitHub</a></li>
<li class="animated wow fadeInLeft" data-wow-delay=".1s"><a href="start.html">Start</a></li>
</ul>
</nav>

<span class="burger_icon">menu</span>

</div>
```

```

</header>
<!-- End Header Section -->

<div class="banner2">
<div class="container">

<div id="flip">LR(1) item sets generated</div>
<div id="panel">

<?php

/* 1-states
2- terminals
3- parsing table
4-non terminals
5- input parsing
7- grammar
*/
$choice=$_REQUEST['items'];

if($choice==1)
$res=shell_exec("C:/xampp/htdocs/CodeProject.Syntax.LALR/CD_grammar1/bin/Debug/CD_gr
ammar1.exe 1 acede$");
else if($choice==3)
$res=shell_exec("C:/xampp/htdocs/CodeProject.Syntax.LALR2/CD_grammar1/bin/Debug/CD
_g
rammar1.exe      1      abc$");
else
if($choice==2)

```

```
$res=shell_exec("C:/xampp/htdocs/CodeProject.Syntax.LALR3/CD_grammar1/bin/Debug/CD
_g rammar1.exe 1 101$");

$split = array();
$state = array();
$temp = array();
$split=explode("States", $res);
//print_r( $split);

$temp=explode("InputParsing", $split[1]);
//print_r( $temp);

$state=explode("State", $temp[0]);
// print_r( $state);

echo '
<div class="container">
<h2>Items</h2>
<table class="table table-hover">

<tbody>';
foreach ($state as $key) { print("<tr><td>".$key."</tr></td>");}
}

echo ('</tbody>
</table>
</div>');
```

```
?>

</div>
</div>
</div>
</div>
</body>
</html>
```

### **Menu.php:**

```
<!DOCTYPE html>

<html>
<head>

<title>CLR Parser</title>

<meta name="viewport" content="width=device-width, initial-scale=1">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> <link href="css/style3.css" rel="stylesheet" type="text/css" media="all" />
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
<link rel="css/bootstrap.min.css">
<link href="css/bootstrap.css" rel='stylesheet' type='text/css' />

<link rel="stylesheet" type="text/css" href="css/animate.css"/>
```

```
<script src="js/jquery.min.js"></script>

<script type="text/javascript" src="js/move-top.js">
</script>

<script src="js/bootstrap.min.js"></script>

<script type="text/javascript" src="js/wow.js"></script>
<script src="js/wow.min.js"></script>
<script type="text/javascript" src="js/easing.js"></script>
<script type="text/javascript" src="js/main.js"></script>
<script type="text/javascript">
jQuery(document).ready(function($){
  $(".scroll").click(function(event){
    event.preventDefault();
    $('html,body').animate({scrollTop:$('this.hash').offset().top},900);
  });
});
</script>
```

```
<script>      new
WOW().init();

</script>
<script>
$(document).ready(function(){
    $("#flip").click(function(){
        $("#panel").slideToggle("slow");

    });
});

$(document).ready(function(){
    $("#flip2").click(function(){
        $("#panel2").slideToggle("slow");

    });
}

</script>

<style>
.carousel-inner > .item > img,
.carousel-inner > .item > a > img {
    width: 100%; margin: auto;
}

}
```

```
#panel, #flip ,#panel2,#flip2{ padding: 30px; text-align:center; background-color:#484848; font-style:bold; font-size: 14pt; letter-spacing: 6px; color: Coral;

border: solid 4px NavajoWhite;
}

#panel, #panel2{
padding: 60px;
display: none;
}

</style>

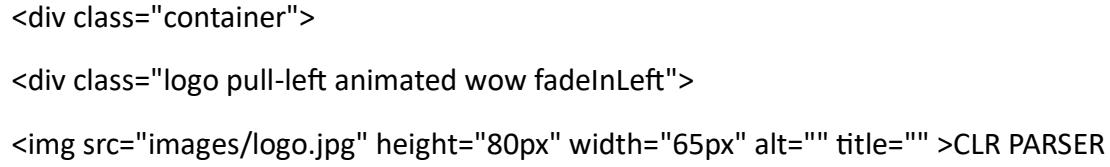
</head>

<body>

<!-- Header Section -->

<header>

<div class="container">

<div class="logo pull-left animated wow fadeInLeft">
The logo consists of the text "CLR PARSER" in a bold, sans-serif font, with a small orange square icon preceding the text.
```

```
</div>

<nav class="pull-right">
  <ul class="list-unstyled">
    <li class="animated wow fadeInLeft" data-wow-delay=".1s"><a href="about.html">About</a></li>
    <li class="animated wow fadeInLeft" data-wow-delay=".1s"><a href="algo.html">Algorithm</a></li>
    <li class="animated wow fadeInLeft" data-wow-delay=".1s"><a href="http://github.com/deepakjayaprakash/Emulator-for-CLR-Parser">GitHub</a></li>
    <li class="animated wow fadeInLeft" data-wow-delay=".1s"><a href="start.html">Start</a></li>
  </ul>
</nav>

<span class="burger_icon">menu</span>
</div>
</header>
<!-- End Header Section -->

<div class="banner2">
  <div class="container">
    <div id="flip">Show Grammar</div>
    <div id="panel">
```

```
<?php

/* 1-states
2- terminals
3- parsing table
4-non terminals
5- input parsing
7- grammar
*/
$choice=$_REQUEST['first'];

if($choice==1)
$res=shell_exec("C:/xampp/htdocs/CodeProject.Syntax.LALR/CD_grammar1/bin/Debug/CD_gr
ammar1.exe 7 acede$");
else if($choice==3)
$res=shell_exec("C:/xampp/htdocs/CodeProject.Syntax.LALR2/CD_grammar1/bin/Debug/CD
_g rammar1.exe 7 abc$"); else if($choice==2)
$res=shell_exec("C:/xampp/htdocs/CodeProject.Syntax.LALR3/CD_grammar1/bin/Debug/CD_g
rammar1.exe 7 101$");

//echo $res."</br><hr>";
$split = array();
$state = array();
$temp = array();
$split=explode("Grammar", $res);
//print_r( $split);
```

```
$temp=explode("end", $split[1]);  
//print_r( $temp);  
// print_r( $state);  
  
echo '  
<div class="container">  
    <h2>Grammar</h2>  
    <table class="table table-hover">  
  
        <tbody>;  
        foreach ($temp as $key) { print("<tr><td>".$key."</tr></td>");  
    }  
  
    echo ('</tbody>  
        </table>  
</div></div></div> <div id="flip2">Options</div>  
<div id="panel2">      '>;  
if($choice=='1'){ echo '  <form  
action="items.php">  
    <button name="items" value="1">ItemSets</button></form><br><br>  
  <form action="table.php">  
    <button name="table" value="1">ParsingTable</button></form><br><br>      <form  
action="parser2.php">  
    <button name="parser" value="1">InputParser</button></form><br><br>
```

```
</form>';

}

else if($choice==2)
{
echo '<form action="items.php">
<button name="items" value="2">ItemSets</button></form><br><br>
<form action="table.php">
<button name="table" value="2">ParsingTable</button></form><br><br>
<form action="parser2.php">
<button name="parser" value="2">InputParser</button></form><br><br>
</form>';

}

else if($choice==3)
{
echo '<form action="items.php">
<button name="items" value="3">ItemSets</button></form><br><br>
<form action="table.php">
<button name="table" value="3">ParsingTable</button></form><br><br>
<form action="parser2.php">
<button name="parser" value="3">InputParser</button></form><br><br>
</form></div></div>';

}

?>

</div>
```

```
</div>  
</body>  
</html>
```

## Conclusion

---

The final application interface would represent a pictorial and a visual figure of each step of the process of CLR parsing. The grammar for the parser is assumed i.e. fixed in the initial phase itself and the user is given the freedom of entering any input string possible. The parser would generate LR(1) set of items and display that. Then it would generate a parsing table given when the generated item set is fed on to it.

Then the parsing of any user input is shown step by step. But the only static thing in the project is the set of pre-defined grammar. The add-on to the project would be to generalize any grammar that is accepted by the user and then generate LR(1) set of tokens on it and then parse the input string.

We could add the other parsing techniques to the implementation too and generate an illustrative case study on which parser would be the best fit for a given grammar and for a given set of input strings.