

## Minor Project- Report June-2022

Course Faculty: Dr. Vindhya P Malagi, Prof. Muquitha Almas, Prof. Shravya AR  
 Course Name: System Software and Operating System Laboratory  
 Course Code: 19CS6DLSSL  
 Semester: 6<sup>th</sup>  
 Date: 21-06-2022

TITLE OF THE PROJECT	Implementation of a Recursive Descent Parser			
STUDENT NAME	Kanishk Bhaskar	Karan V	Keshav Jhawar	Kiran N Deshmukh
USN	1DS19CS067	1DS19CS068	1DS19CS069	1DS19CS070
GUIDE	Dr. Vindhya P Malagi			
PROJECT ABSTRACT:	<p>We have implemented the Recursive Descent Parser as part of our mini project in C Programming Language.</p> <p>The algorithm for Recursive Descent Parser is as follows:</p> <ul style="list-style-type: none"> <li>➤ Consider the grammar:  <math>S \rightarrow cAd</math>  <math>A \rightarrow ab a</math></li> <li>➤ Let the input string be 'w' = cad.</li> <li>➤ Procedure S is written as follows:  <pre> procedure S () begin   if input symbol = 'c' then     ADVANCE ();     if A () then       if input symbol = 'd' then         begin ADVANCE ();         return true;       end     end;   return false; end </pre></li> </ul>			



	<p>➤ Procedure A is written as follows:</p> <pre>Procedure A () begin   isave := input-pointer;   if input symbol = 'a' then     begin       ADVANCE ();       if input symbol = 'b' then         begin ADVANCE (); return true;       end     end   input-pointer := isave;   /* failure to find ab */   if input symbol = 'a' then     Begin ADVANCE (); return true;   end   else return false; end</pre>
INTRODUCTION	<p>Parsing is the process to determine whether the start symbol of a specific grammar can derive the program.</p> <ul style="list-style-type: none"><li>➤ If successful, the program is a valid program.</li><li>➤ If failed, the program is invalid.</li></ul> <p>There are two approaches to parsing in general.</p> <ul style="list-style-type: none"><li>➤ Expanding from the start symbol to the whole program (Top Down). Ex. Recursive Descent Parser, LL Parser or Predictive Parser</li><li>➤ Reduction from the whole program to the start symbol (Bottom Up). Ex. LR (SLR, Canonical LR, LALR) Parser</li></ul> <p>Recursive Descent Parsing is a Top-Down method of syntax analysis in which a set of recursive procedures to process the input is executed. A procedure is associated with each non-terminal of a grammar. Top-Down parsing can be viewed as an attempt to find a leftmost derivation for an input string. It attempts to construct a parse tree for the input starting from the root and creating the nodes of the parse tree in preorder. Recursive Descent Parsing involves backtracking.</p>
DESIGN	<pre>graph LR     SP[Source Program] --&gt; LA[Lexical Analyzer]     LA -- token --&gt; P[Parser]     P -- "get next token" --&gt; LA     P -- "Parse tree" --&gt; RFE[Rest of Front End]     RFE --&gt; IR[Intermediate Representation]     LA &lt;--&gt; ST[Symbol Table]     P &lt;--&gt; ST     RFE &lt;--&gt; ST</pre> <p>The diagram illustrates the design of a parser. It shows a sequence of components: Source Program, Lexical Analyzer, Parser, Rest of Front End, and Symbol Table. The Lexical Analyzer takes the Source Program as input and outputs tokens to the Parser. The Parser sends 'get next token' requests back to the Lexical Analyzer. The Parser outputs a 'Parse tree' to the Rest of Front End, which then produces an 'Intermediate Representation'. A Symbol Table is shown at the bottom, with bidirectional arrows connecting it to the Lexical Analyzer, the Parser, and the Rest of Front End.</p>

	<p style="text-align: center;"> <span style="display: inline-block; width: 30%; text-align: center;">(a)</span> <span style="display: inline-block; width: 30%; text-align: center;">(b)</span> <span style="display: inline-block; width: 30%; text-align: center;">(c)</span> </p>
PLATFORM USED	C Programming Language, VS Code
PROJECT SOURCE CODE LINK	<a href="https://github.com/kirandeshmukh6421/recursive-descent-parser">https://github.com/kirandeshmukh6421/recursive-descent-parser</a>
CONCLUSION /FUTURE ENHANCEMENT	The parser could be improvised to use a non-recursive descent parsing algorithm which will not require any kind of backtracking to improve parse speeds and efficiency. These types of parsers are also called Predictive Parsers.

## SCREENSHOTS

```
Recursive Descent Parsing  
The given grammar is:
```

```
E->TE'  
E'->+TE'/@  
T->FT'  
T'->*FT'/@  
F->(E)/ID
```

```
Enter the string to be checked - a+(a*a)*a+a
```

```
String is accepted
```

```
Recursive Descent Parsing  
The given grammar is:
```

```
E->TE'  
E'->+TE'/@  
T->FT'  
T'->*FT'/@  
F->(E)/ID
```

```
Enter the string to be checked - a+(a
```

```
String not accepted
```