**LAB SESSION 6: PREDICTIVE PARSER USING PYTHON**

**AIM**: To implement a Predictive parser using Python.

**PROBLEM DEFINITION:** Develop a python program to implement predictive parser for a given grammar.

**THEORY:** Parser for any grammar is program that takes as input string w (obtain set of strings tokens from the lexical analyzer) and produces as output either a parse tree for w , if w is a valid sentences of grammar or error message indicating that w is not a valid sentences of given grammar.

The goal of the parser is to determine the syntactic validity of a source string is valid; a tree is built for use by the subsequent phases of the computer. The tree reflects the sequence of derivations or reduction used during the parser. Hence, it is called parse tree. If string is invalid, the parse has to issue a diagnostic message identifying the nature and cause of the errors in the string. Every elementary subtree in the parse tree corresponds to a production of the grammar.

There are two ways of identifying an elementary subtree:

1. By deriving a string from a non-terminal or

2. By reducing a string of symbol to a non-terminal

Predictive Parser is also another method that implements the technique of Top- Down parsing without Backtracking. A predictive parser is an effective technique of executing recursive-descent parsing by managing the stack of activation records, particularly.

How predictive parsers work

**LL(1) Parsing**: The parser reads the input from left to right, a leftmost derivation is used to construct the parse tree, and the decision for the next step is based on a single look-ahead token.

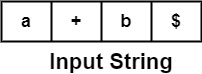
**Stack**: A stack is used to store grammar symbols. It is initialized with the grammar's start symbol and a special end-of-input marker.

**Parsing Table**: A 2D table is used to guide the parser's decisions. The table is indexed by a non-terminal (from the top of the stack) and the current input token.

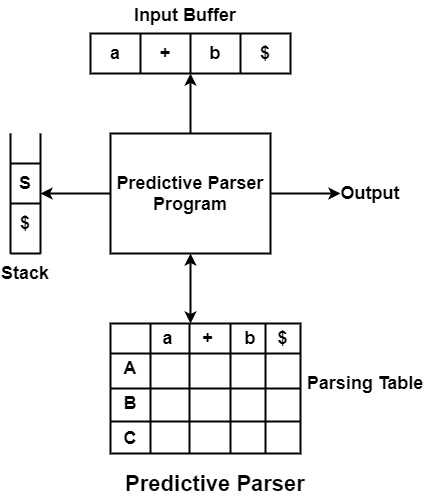
**No Backtracking**: A predictive parser does not need to backtrack because the parsing table contains a unique entry for each combination of non-terminal and input token, ensuring a single, deterministic path.

Key Components:

1. **Input Buffer:** The input buffer includes the string to be parsed followed by an end marker $ to denote the end of the string.



1. **Stack** − It contains a combination of grammar symbols with $ on the bottom of the stack. At the start of Parsing, the stack contains the start symbol of Grammar followed by $.



1. **Parsing Table** − It is a two-dimensional array or Matrix M [A, a] where A is nonterminal and 'a' is a terminal symbol.

**Following are the steps to perform Predictive Parsing**

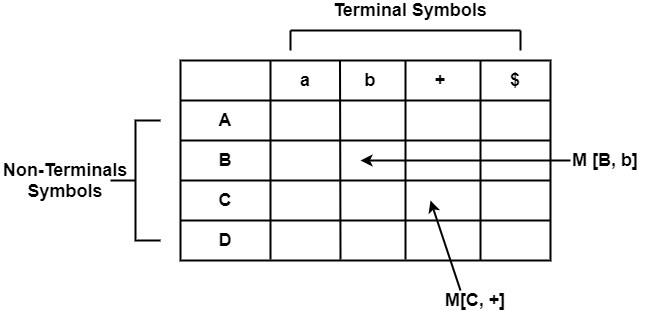
1. Elimination of Left Recursion
2. Left Factoring
3. Computation of FIRST & FOLLOW
4. Construction of Predictive Parsing Table
5. Parse the Input String

**Algorithm to construct Predictive Parsing Table**

**Input** − Context-Free Grammar G

**Output** − Predictive Parsing Table M

**Method** − For the production A → α of Grammar G.

* For each terminal, a in FIRST (?) add A → α to M [A, a].
* If ε is in FIRST (α), and b is in FOLLOW (A), then add A → α to M[A, b].
* If ε is in FIRST (α), and $ is in FOLLOW (A), then add A → α to M[A, $].
* All remaining entries in Table M are errors.

**PROGRAM**

**OUTPUT**

**CONCLUSION**