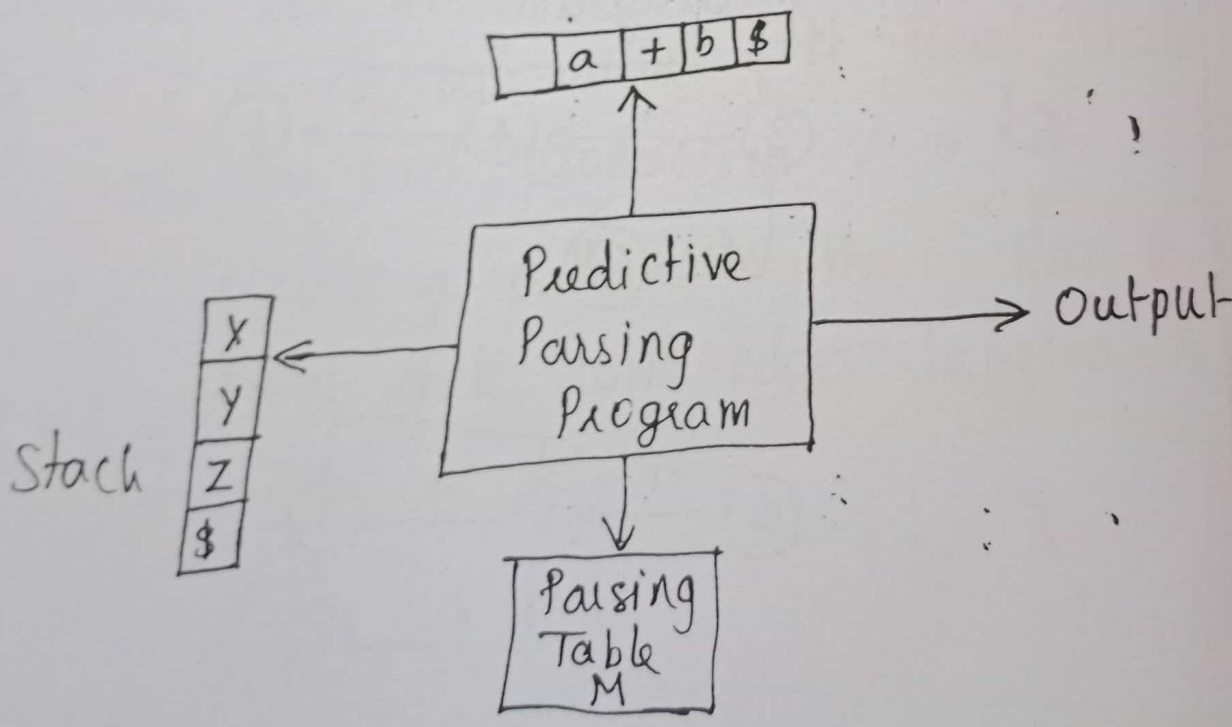
Non-recursive Predictive Parsing:

* The key problem during predictive parsing is that of determining production to be applied for a non-terminal.
* The non-recursive parser looks up the production to be applied in a parsing table.



A table driven predictive parser has

(i) Input buffer
(ii) a stack
(iii) a parsing table
(iv) an output stream

* The input buffer contains the string to be parsed followed by $, a symbol used as a right end marker to indicate the end of the input string.
* The stack contains a sequence of grammar symbols with $ on the bottom, indicating the bottom of the stack.
* Initially stack contains a start symbol of the grammar on top of $.
* The parsing table is a two dimensional array M[A, a], where A is a non-terminal and a is a terminal or the symbol $.
* The predictive parsing program considers X, the symbol on the top of the stack and a, the current input symbol.
* These two symbols determine the action of the parser.

* There are three possibilities.

1. If $X = a = \$$, the parser halts and announces successful completion of parsing.
2. If $X = a \neq \$$, the parser pops X off the stack and advances the input pointer to the next input symbol.
3. If X is a non-terminal, the program consults the entry $M[X, a]$ of the parsing table M. This entry will be either an X-production or an error entry.

   If, for eg) $M[X, a] = \{ X \rightarrow UVW \}$, the parser replaces the X on the top of the stack by WVU (with U on top).

   * As output we shall assume that the parser just prints the production used.

   If $M[X, a]$ = error, the parser calls an error recovery routine.

Algorithm: Non-recursive Predictive parsing

Input : A string w and a parsing table M for grammar G.

Output : If w is in L(G), a left most derivation of w, otherwise an error indication.

Method: Initially the parser is in the configuration
$S is on the stack with S on top.
w$ is in input buffer, ie) the input string to be parsed.

set ip to point to the first symbol of w$
repeat
  let X be the top stack symbol and a the symbol pointed by ip
  if X is a terminal or $ then
    if X = a then
      Pop X from stack and advance ip
    else error()
  else
    if M[X, a] = $X \rightarrow Y_1 Y_2 \cdots Y_K$ then begin
      pop X from the stack
      push $Y_K Y_{K-1} \cdots Y_1$ onto the stack with Y_1 on top
      output the production $X \rightarrow Y_1 Y_2 \cdots Y_K$
    end and
    else
      error()
until X = $

Example:

Consider the Grammar

$E \to TE'$

$E' \to +TE' \mid \epsilon$

$T \to FT'$

$T' \to *FT' \mid \epsilon$

$F \to (E) \mid id.$

* In this predictive parsing table, blanks are error entries and non-blanks indicate a production with which to expand the top non-terminal on the stack.

Non Terminal	Input Symbol					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to$
F	$F \to id$			$F \to (E)$		

* The Parser traces out a left most derivation for this input. i.e) the productions output are those of a left most derivation.

STACK	INPUT	OUTPUT
$E	id+id*id$	
$E'T	id+id*id$	E → TE'
$E'T'F	id+id*id$	T → FT'
$E'T'id	id+id*id$	F → id
$E'T'	+id*id$	
$E'	+id*id$	T' → ∈
$E'T+	+id*id$	E' → +TE'
$E'T	id*id$	
$E'T'F	id*id$	T → FT'
$E'T'id	id*id$	
$E'T'	*id$	
$E'T'F*	*id$	T' → *FT'
$E'T'F	id$	
$E'T'id	id$	F → id
$E'T'	$	T' → ∈
$E'	$	E' → ∈

SYNTAX Tree