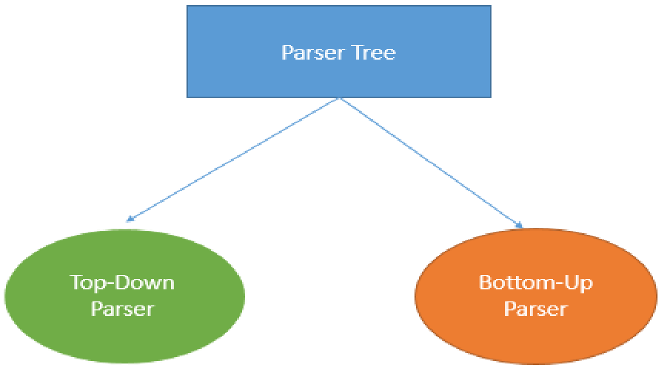
Syntax analyzers follow production rules defined by means of context-free grammar. The way the production rules are implemented (derivation) divides parsing into two types: top-down parsing and bottom-up parsing.



Top-Down Parser

Top-Down Parser starts constructing a parse tree from the root node gradually moving down to the leaf nodes.

Example: Suppose we have a grammar:

$S \rightarrow E$

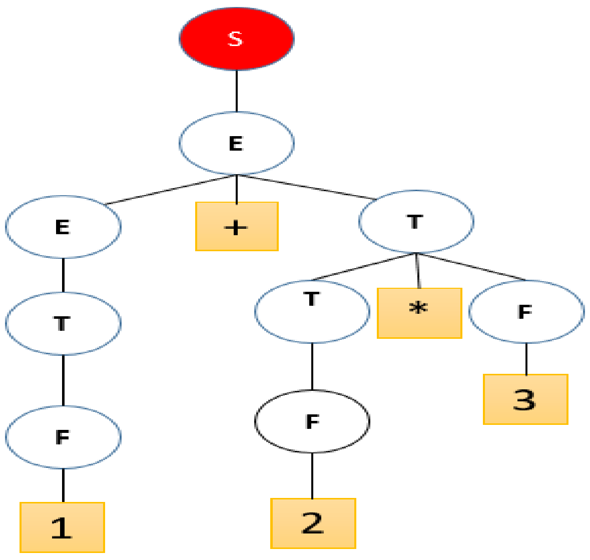
$E \rightarrow T|E + T$

$T \rightarrow F|T*F$

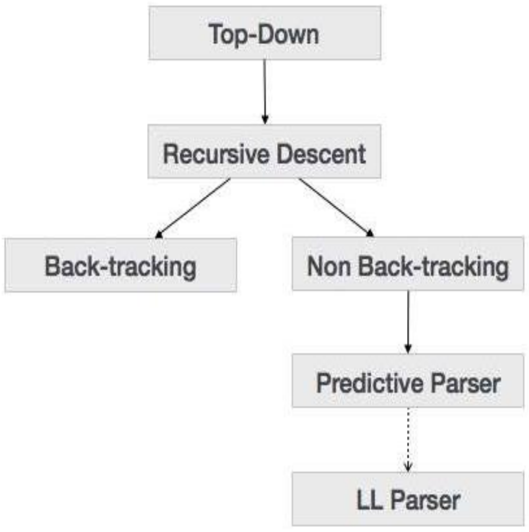
$F \rightarrow 0|1|2|3|4|5|6|7|8|9|\in$

Draw parser tree for derivation the following sentence: 1+2*3

Solution:



The types of top-down parsing are shown in figure bellow:



Recursive Descent Parsing

Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right. It uses procedures for every terminal and non-terminal entity. This parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. But the grammar associated with it (if not left factored) cannot avoid back-tracking. A form of recursive-descent parsing that does not require any back-tracking is known as **predictive parsing**.

Example 1 : Write pseudo-code to parser the following rule or production by recursive descent .

Factor → (exp)|number

Solution:

```
Procedure Factor
Begin
Case token of ( :
match (( );
exp;
match( ));
number;
end case;
end factor;
Procedure match (input)
Begin
```

```
If token =="("|| ")"
accept input
else
reject input
end if
end match
```

Example 2 : Write pseudo-code to parser the following rule or production by recursive descent .

$stmt \rightarrow \textbf{if}\ (exp)statment$

$\quad |\textbf{if}\ (exp)statment\ \textbf{else}\ statment$

Solution:

Procedure stmt
Begin
match(if);
match(();
exp;
match());
statement;
if token=else then
match(else);
match (statement);
end if;
end stmt;
	Procedure match(expected_token)
	Begin
	If token == expected_token;
	Get Token ;
	else
	error;
	end if ;
	end match;

What the Different between Top-Down parser with and without back tracking

In Top-Down Parsing with Backtracking, Parser will attempt multiple rules or production to discover the match for input string by backtracking at every step of derivation.so the different between top-down parser with and without back tracking is shown in table below.

Top-Down Parsing with Backtracking	Top-Down Parsing without Backtracking
The parser can try all alternatives in any order till it successfully parses the string.	The parser has to select correct alternatives at each step.
Backtracking takes a lot of time,	It takes less time.
A Grammar can have left recursion.	Left Recursion will be removed before doing parsing.
t can be difficult to find where actually error has occurred.	It can easy to find the location of the error

Predictive Parsing Method

In many cases, by carefully writing a grammar eliminating left recursion from it, and left factoring the resulting grammar, we can obtain a grammar that can be parsed by a non backtracking predictive parser. We can build a predictive parser by maintaining a stack. The key problem during predictive parser is that of determining the production to be applied for a nonterminal. The non-recursive parser looks up the production to be applied in a parsing table.

في كثير من الحالات ، من خلال كتابة قواعد نحوية بعناية مع إزالة left recursion منها ، left factoring، يمكننا الحصول على قواعد نحوية يمكن تحليلها بواسطة محلل تنبؤي غير رجعي. يمكننا بناء محلل تنبؤي عن طريق الحفاظ على المكدس. المشكلة الرئيسية أثناء المحلل اللغوي التنبئي هي تحديد production الذي سيتم تطبيقه على a nonterminal.

non-recursive parser يبحث عن الإنتاج ليتم تطبيقه في جدول التحليل.

A table-driven predictive parser has an input buffer, a stack, a parsing table, and an output stream. The input buffer contains the string to be parsed, followed by $, (a symbol used as a right endmarker 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, the stack contains the start symbol of the grammar on the top of $.

Example:

E →E+T|T

T→T*F|F

F→id|(E)

First step

Removing left recursion for grammar

E → T$\bar{E}$

$\bar{E}$ → +T$\bar{E}$|∈

T→ F $\bar{T}$

$\bar{T}$ →$*$ $F\bar{T}$|∈

F→ id|(E)

Second step

Compute First set and Follow set

First set	**Follow set**
(E) ={ **id,(** }	(E) ={ $,)}
($\bar{E}$) ={**+,∈** }	($\bar{E}$) ={ $,)}
(T) ={ **id,(** }	(T) ={ +, $,)}
($\bar{T}$) ={*,∈ }	($\bar{T}$) ={ +, $,)}
(F) ={ **id,(** }	(F) ={ *,+, $,)}

Rules of compute Follow :

1- Put $ in follow set of start symbol which is represents end of file
2- Follow set of Non-terminals (X) is next terminal after it
3- If the next Non-terminals (X) is Non-terminals (Y) then get first this Non-terminals (Y) and put in follow of Non-terminals (X).
4- If the following of the Non-terminals (X) is ∈ then add follow set of current rule left symbol .

Third step

Building parsing or stack table

First set	**Follow set**
(E) ={ **id,(** }	(E) ={ $,)}
($\bar{E}$) ={**+,∈** }	($\bar{E}$) ={ $,)}
(T) ={ **id,(** }	(T) ={ +, $,)}
($\bar{T}$) ={*,∈ }	($\bar{T}$) ={ +, $,)}
(F) ={ **id,(** }	(F) ={ *,+, $,)}

NT \ T	id	(	+	*	)	$
E	E → T$\bar{E}$	E → T$\bar{E}$				
$\bar{E}$			$\bar{E}$ → +T$\bar{E}$		$\bar{E}$ → ∈	$\bar{E}$ → ∈
T	T→ F $\bar{T}$	T→ F $\bar{T}$				
$\bar{T}$			$\bar{T}$ → ∈	$\bar{T}$ →∗ $F\bar{T}$	$\bar{T}$ → ∈	$\bar{T}$ → ∈
F	F→ id	F→ (E)			F→ (E)	

ملاحظات مهمه لمليء الجدول أعلاه:

1- نبدأ ب first
2- ننتقل الى ال follow في حاله ∈
3- عندما تحتوي القاعدة على ∈ **معناها هي $ end of file or**

Fourth step

Derivation based on stack or parser table with expression =(id)

	id	(	+	*	)	$
E	E → T$\bar{E}$	E → T$\bar{E}$				
$\bar{E}$			$\bar{E}$ → +T$\bar{E}$		$\bar{E}$ → ∈	$\bar{E}$ → ∈
T	T→ F $\bar{T}$	T→ F $\bar{T}$				
$\bar{T}$			$\bar{T}$ → ∈	$\bar{T}$ →∗ $F\bar{T}$	$\bar{T}$ → ∈	$\bar{T}$ → ∈
F	F→ id	F→ (E)			F→ (E)	

stack	input	output
$\$E$	$(id)\$$	
$\$\bar{E}T$	$(id)\$$	$E \rightarrow T\bar{E}$
$\$\bar{E}\bar{T}F$	$(id)\$$	$T\rightarrow F\,\bar{T}$
$\$\bar{E}\bar{T})E($	$(id)\$$	$F\rightarrow (E)$
$\$\bar{E}\bar{T})E$	$id)\$$	
$\$\bar{E}\bar{T})\bar{E}T$	$id)\$$	$E \rightarrow T\bar{E}$
$\$\bar{E}\bar{T})\bar{E}\bar{T}F$	$id)\$$	$T\rightarrow F\,\bar{T}$
$\$\bar{E}\bar{T})\bar{E}\bar{T}id$	$id)\$$	$F\rightarrow id$
$\$\bar{E}\bar{T})\bar{E}\bar{T}$	$)\$$	
$\$\bar{E}\bar{T})\bar{E}$	$)\$$	$\bar{T} \rightarrow \in$
$\$\bar{E}\bar{T})$	$)\$$	$\bar{E} \rightarrow \in$
$\$\bar{E}\bar{T}$	$\$$	
$\$\bar{E}$	$\$$	$\bar{T} \rightarrow \in$
$\$$	$\$$	$\bar{E} \rightarrow \in$

بمأنه وصل الاشتقاق الى $\$$ أي نهاية المكدس اذن التعبير او الجملة المدخلة هي مقبولة ضمن هذه القاعدة بالاعتماد على predicative paring methods