

LL(1) parsing

****Definition:****

LL(1) parsing is a top-down parsing technique that examines the first token of an input string to determine which parse rule to apply.

****Process:****

1. Read the first token of the input string.
2. Find the parse rule that has the first token of the input string as its left-hand side.
3. Apply the parse rule to the input string.
4. Repeat steps 1-3 until the input string is completely parsed.

****Advantages:****

- * Simple and easy to understand
- * Efficient for small grammars
- * Can be used to parse ambiguous grammars

****Disadvantages:****

- * Not efficient for large grammars
- * May not be able to parse all grammars

****Example:****

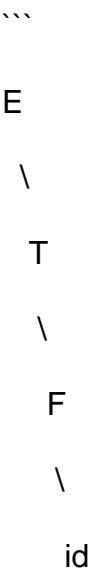
Consider the following grammar:

```
...  
  
E -> T + E | T  
T -> F * T | F  
F -> (E) | id  
...
```

The following input string can be parsed using ll(1) parsing:

```
...  
  
id + id * id  
...
```

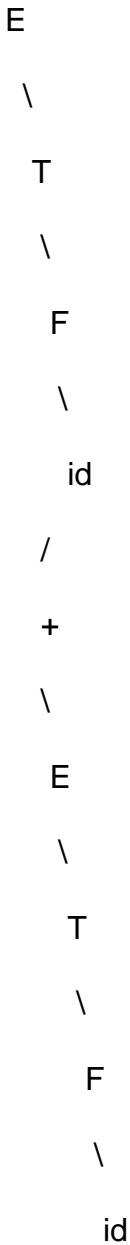
The first token of the input string is "id". The parse rule that has "id" as its left-hand side is "F -> id". This rule is applied to the input string, resulting in the following parse tree:



...

The input string is now "id + id * id". The first token of the input string is "+". The parse rule that has "+" as its left-hand side is "E -> T + E". This rule is applied to the input string, resulting in the following parse tree:

...



...

The input string is now "id * id". The first token of the input string is "*". The parse rule that has "*" as its left-hand side is "T -> F * T". This rule is applied to the input string, resulting in the following parse tree:



\

F

\

id

...

The input string is now "id". The first token of the input string is "id". The parse rule that has "id" as its left-hand side is "F -> id". This rule is applied to the input string, resulting in the following parse tree:

...

E

\

T

\

F

\

id

/

+

\

E

\

T

\

F

\
id
/
*
\
T
\
F
\
id
/
id
...

The input string is now empty. The parse tree is complete.