Scanner
PArser+SymbolTable
Type checker
\OPTIMIZER - local optimization
- interprocess optimisation
Code generation

syntax - form
semantics = meaning

E -> E+E | E*E | ‘(‘E’)’ | id | num
a + b*c
If String in L(gamma) w/o two trees => gammaa ambiguous.

E - expr
T - term
F - factor

E-> E+T | T -> (T’+’)*T
T-> T*F | F -> (F’*’)* F
F-> ID|num|’(’E’)’

Parsing:
given a string in L(gamma), find the(a) parse tree
- Top-down
- predictive
- Recursive descent
- Table driven LL(1)
() left to right we can only put things on the right subtree after we fill the left subtree

X-> X (same symbol) we call it left recursion

left recursion elimination
E-> E+T |T
E-> TE’
1)E’->\epsilon
2)E’->+TE’

T -> T*F | F
T -> FT’
1)T’->\epsilon
2)T’->*FT’

F-> ID|num|’(’E’)’

New symbbol non-t S’
New rule S’->S$ old start symbol
FOr us S’ -> E$ end of input symbol

FOLLOW(X) (not term != S’) = {all possible derivativesS’=>S$+> …Xa a terminal
For T’, Only +$ we choose epsilon else *FT’

Non-term,FIRST,FOLLOW
E’,(+,epsilon),$
T’….

LL(1) parser
-L->R leftmost derivation #term look ahead

Recursive descent parser
E -> (T'+')* T - T('+'T)*
F -> (F'*')F - F('*'F)*

```
parse_E()
{
        parse_T()
        while (next_token=='+')
        {
                advance()
                parse_T()
        }
}
```