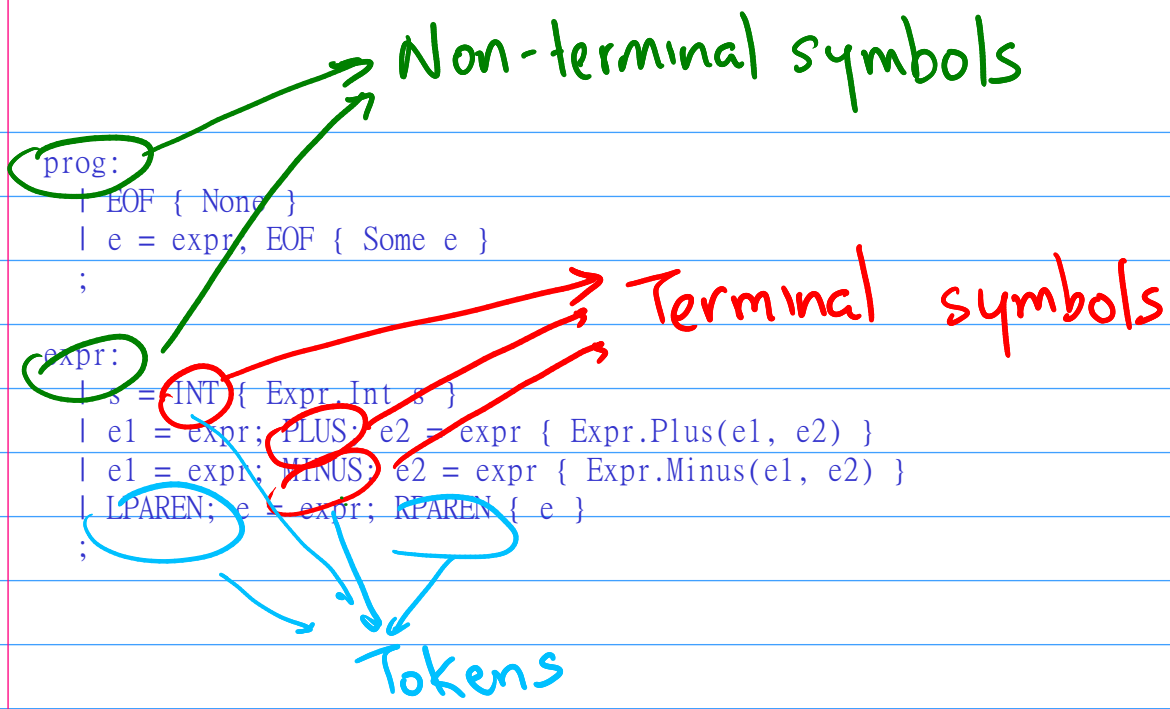
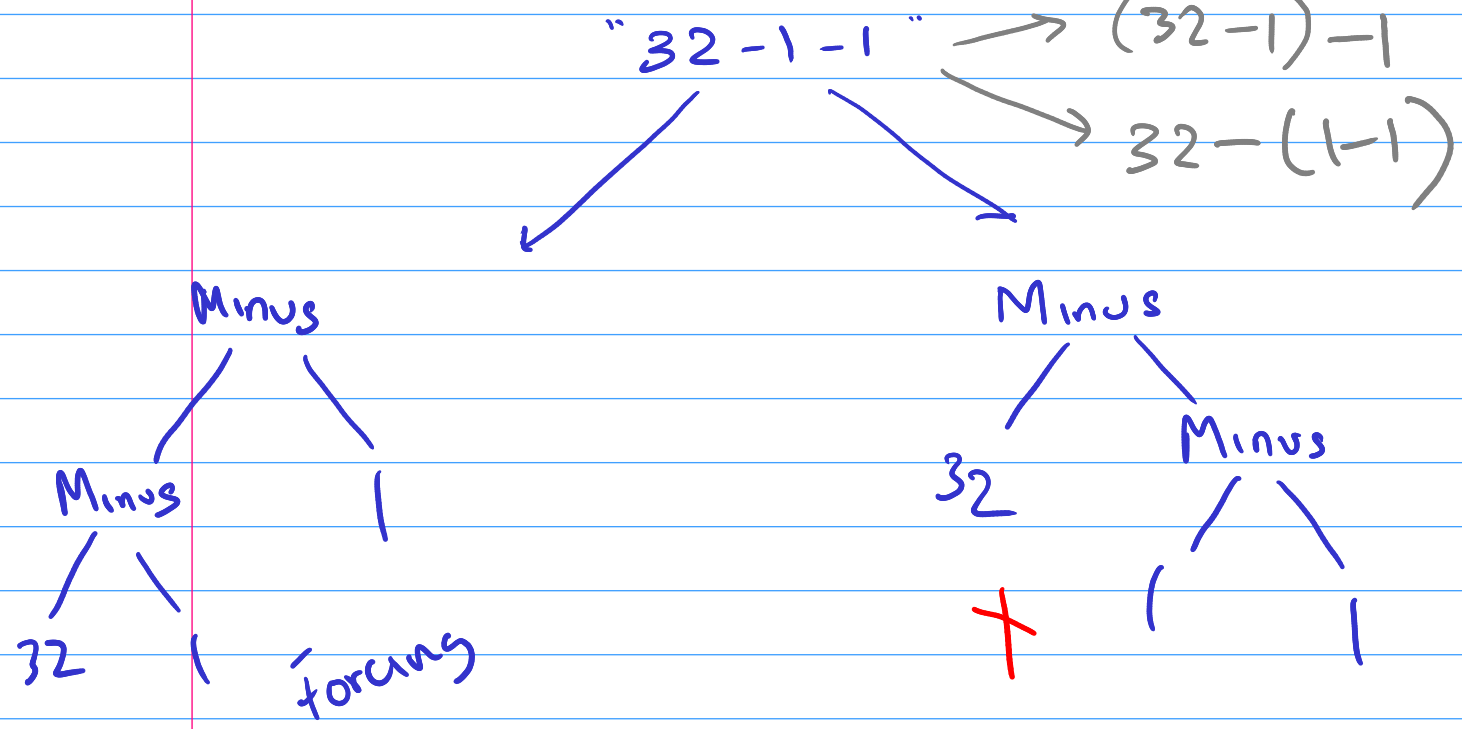



```
prog:
  | EOF { None }
  | e = expr; EOF { Some e }
  ;

expr:
  | s = INT { Expr.Int s }
  | e1 = expr; PLUS; e2 = expr { Expr.Plus(e1, e2) }
  | e1 = expr; MINUS; e2 = expr { Expr.Minus(e1, e2) }
  | LPAREN; e = expr; RPAREN { e }
  ;
```

```
rule read =
  parse
  | white   { read lexbuf }
  | newline { next_line lexbuf; read lexbuf }
  | int     { INT (int_of_string (Lexing.lexeme lexbuf)) }
  | '+'     { PLUS }
  | '-'     { MINUS }
  | '('     { LPAREN }
  | ')'     { RPAREN }
  | _       { raise (SyntaxError ("Unexpected char: " ^ Lexing.lexeme lexbuf)) }
  | eof     { EOF }
```

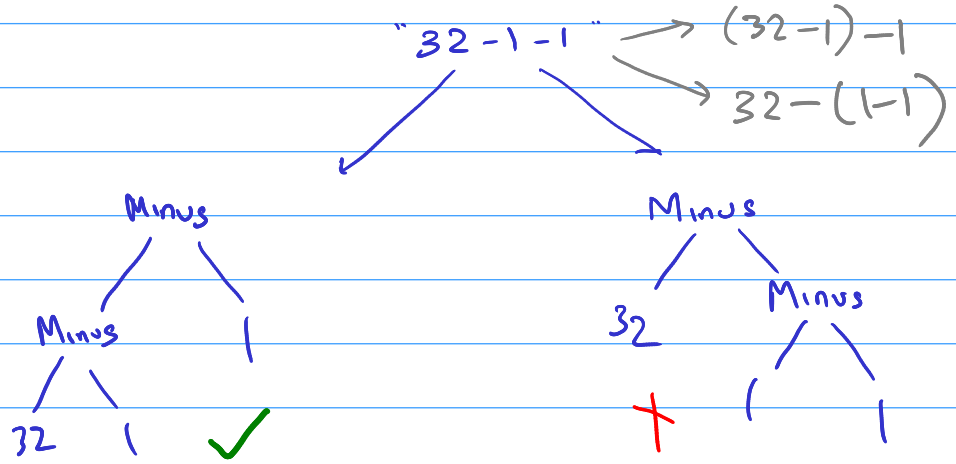


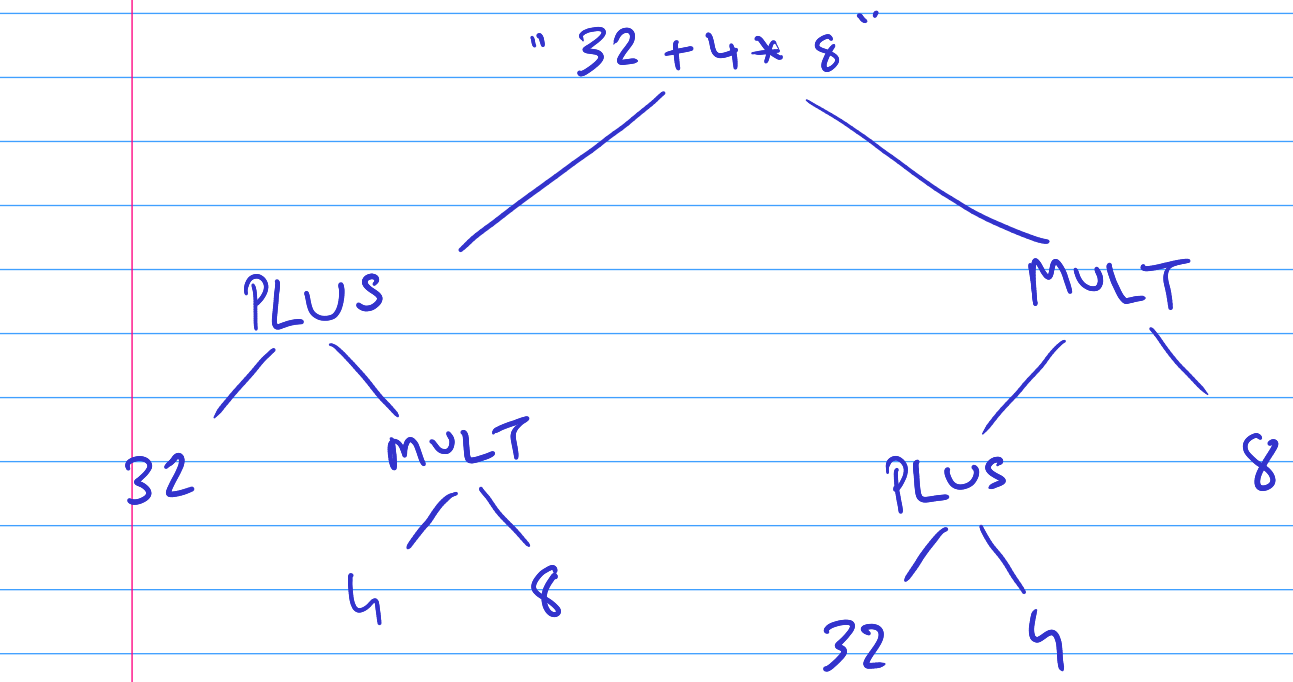
"(32-1)-1" : Right side of the minus is always a raw number

"32-(1-1)" : The right side of the minus is sometimes a complex expression "1-1"

```
expr1:
  | e1 = expr1; PLUS; e2 = expr2 { Expr.Plus(e1, e2) }
  | e1 = expr1; MINUS; e2 = expr2 { Expr.Minus(e1, e2) }
  | e = expr2 { e }
  ;

expr2:
  | s = INT { Expr.Int s }
  | LPAREN; e = expr1; RPAREN { e }
  ;
```






```
expr1:
  | e1 = expr1; PLUS; e2 = expr2 { Expr.Plus(e1, e2) }
  | e = expr2 { e }
  ;

expr2:
  | s = INT { Expr.Int s }
  | e1 = expr2; MULT; e2 = expr2 { Expr.Mult(e1, e2) }
  ;
```

"8*4+32"

"Longest Match Rule" in lexical analyzers