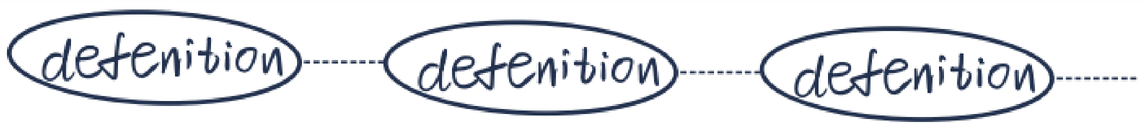
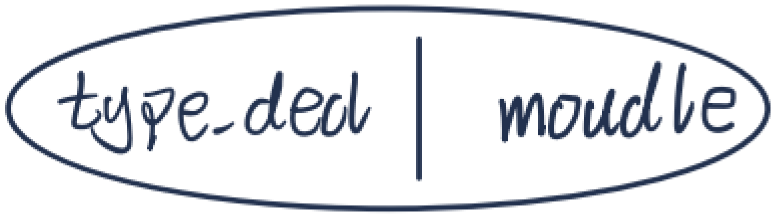
抽象语法树

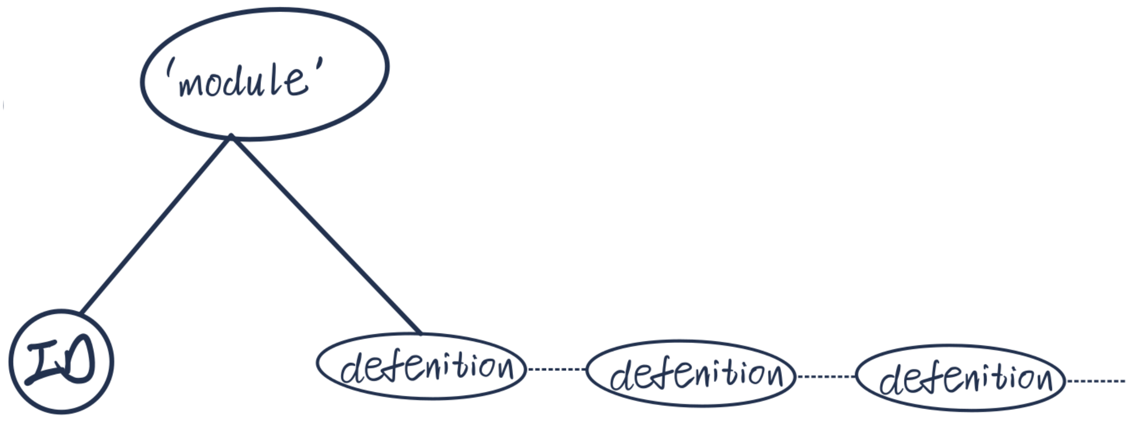
specification -> definition { definition }



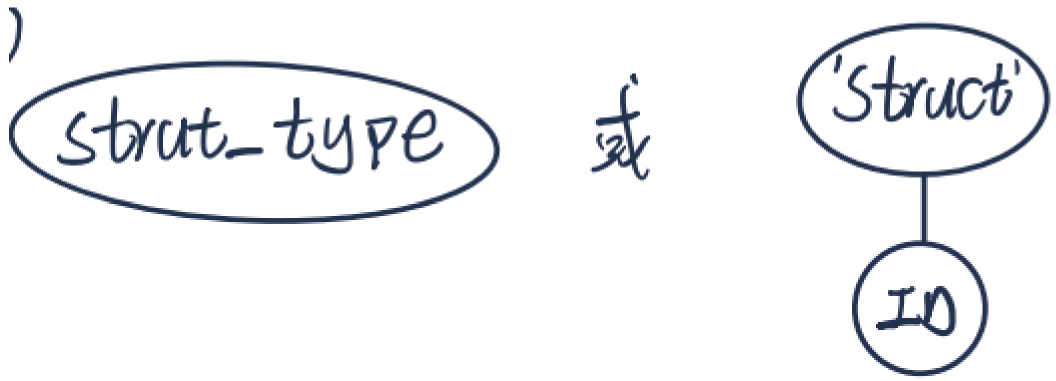
definiton -> type_decl“;”| module “;”



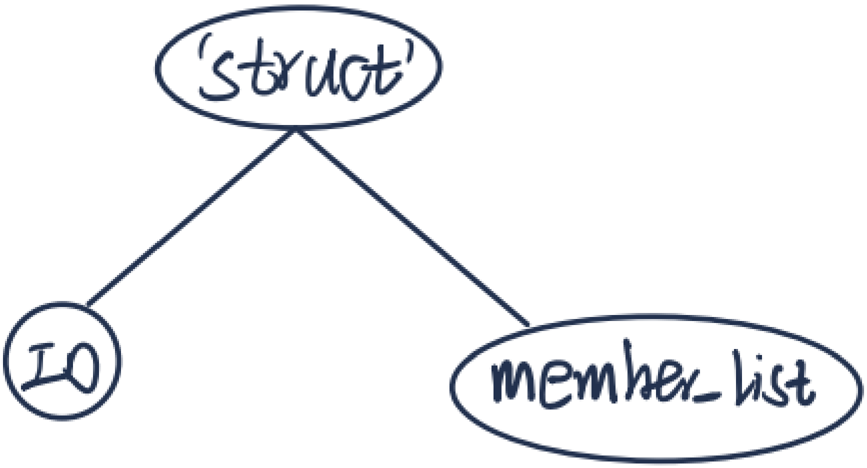
module -> “module”ID “{” definition { definition } “}”



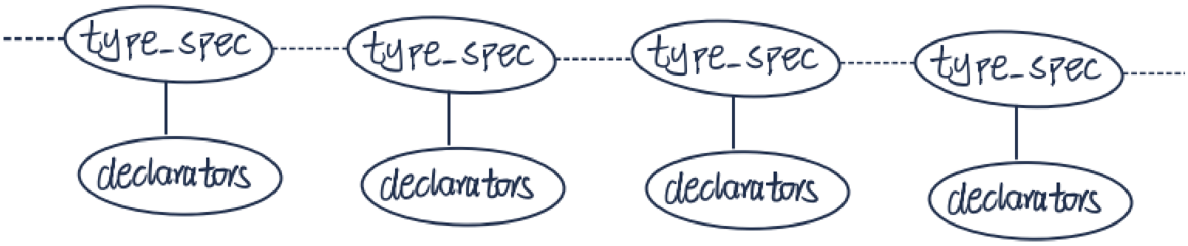
type_decl -> struct_type | “struct” ID



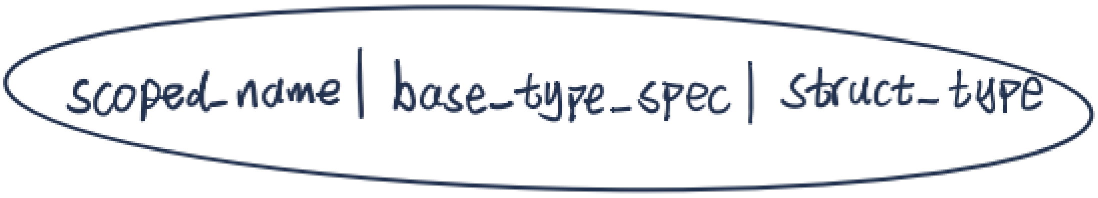
struct_type->“struct” ID “{”   member_list “}”



member_list-> { type_spec declarators “;” }

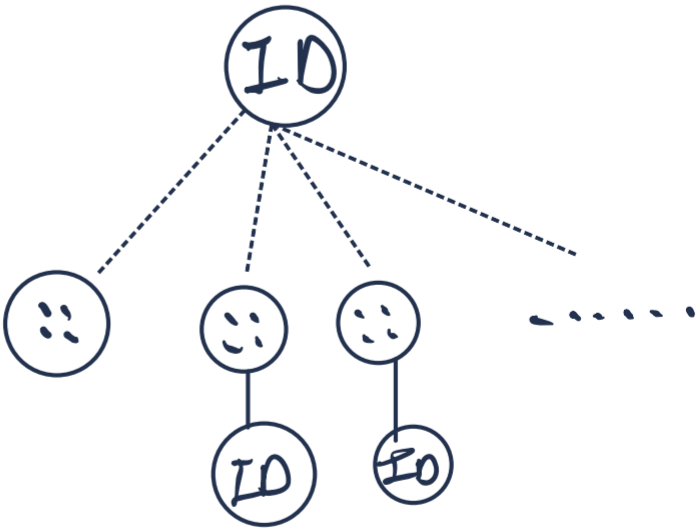


type_spec -> scoped_name | base_type_spec | struct_type



scoped_name -> [“::”] ID {“::” ID }

> **说明：**这里我认为ID更重要一些，所以把第一个ID前面可能存在的“::”作为ID的孩子节点。



base_type_spec->floating_pt_type|integer_type|“char”|“string”|“boolean”

floating_pt_type | integer_type | 'char' | 'string' | 'boolean'

floating_pt_type -> “float” | “double” | “long double”

'float' | 'double' | 'long double'

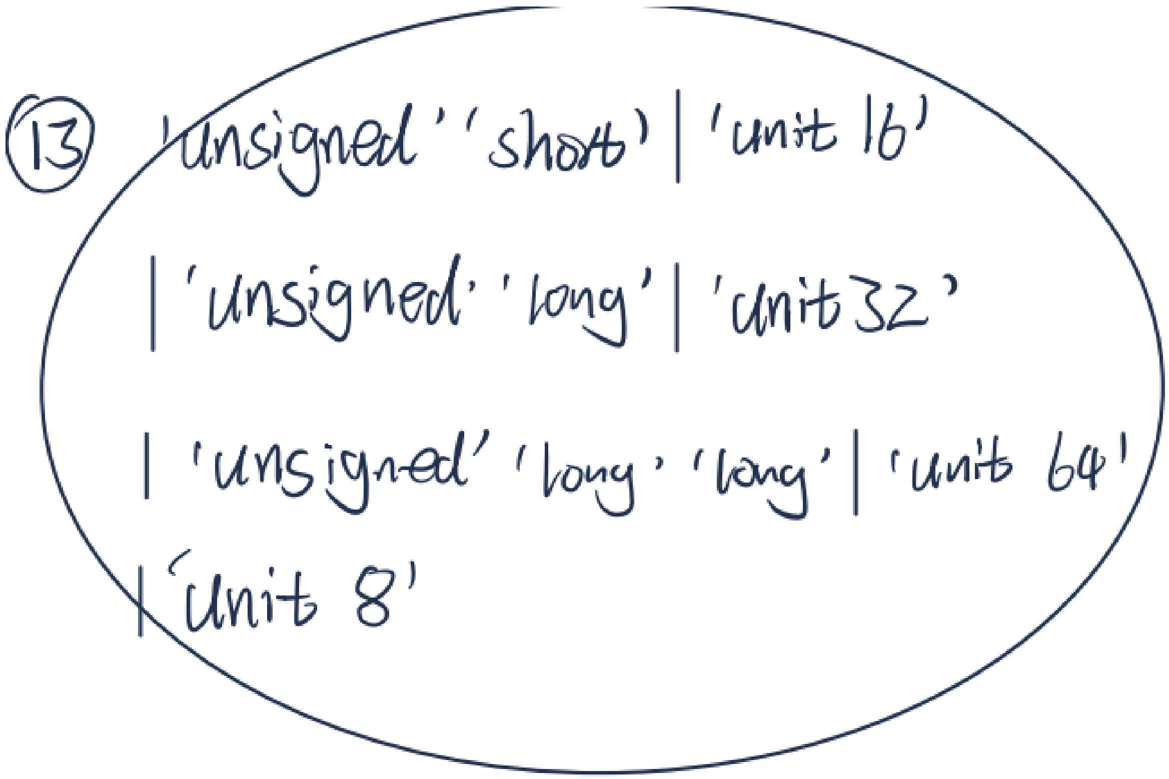
integer_type -> signed_int | unsigned_int

signed_int | unsigned_int

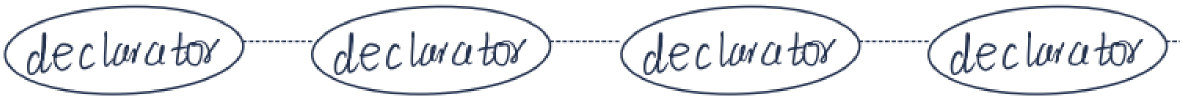
signed_int->(“short”|“int16”)
|(“long”|“int32”)
|(“long” “long”|“int64”)
|“int8”

'short' | 'int 16' | 'long' | 'int32' | 'long' 'long | 'int 64' | 'int8'

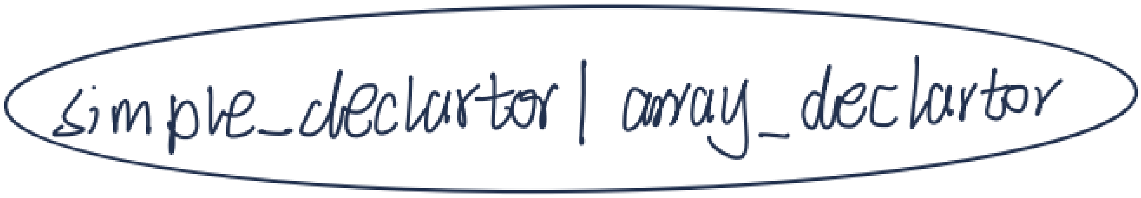
unsigned_int -> (“unsigned”“short”| “uint16”)
| (“unsigned”“long”| “uint32”)
| (“unsigned” “long” “long” | “uint64”)
| “uint8”


13
'unsigned' 'short' | 'unit 16'
| 'unsigned' 'long' | 'unit32'
| 'unsigned' 'long' 'long' | 'unit 64'
| 'unit 8'

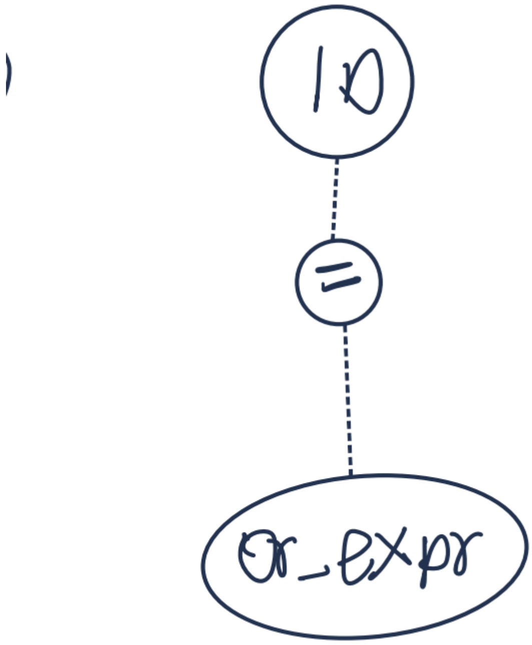
declarators -> declarator {“,” declarator }



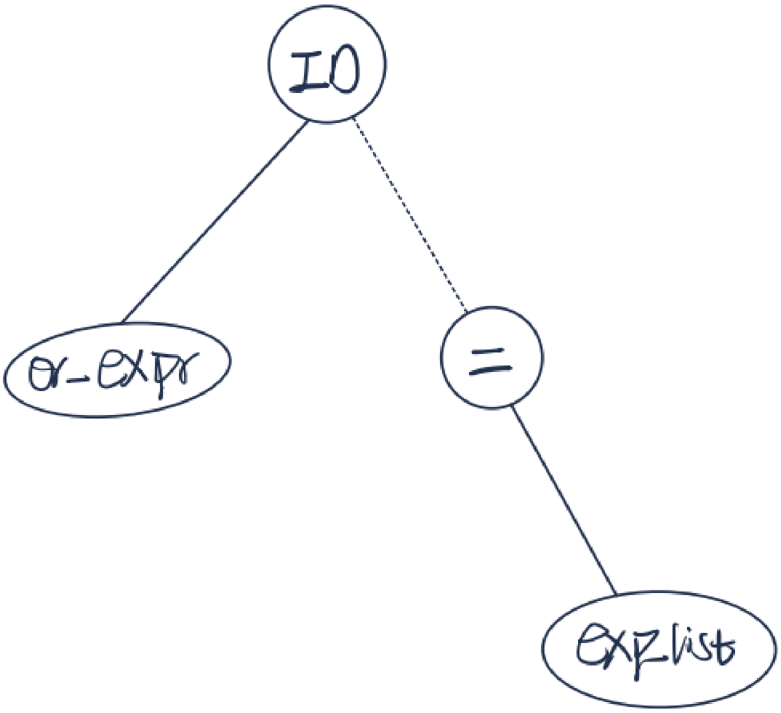
declarator -> simple_declarator | array_declarator



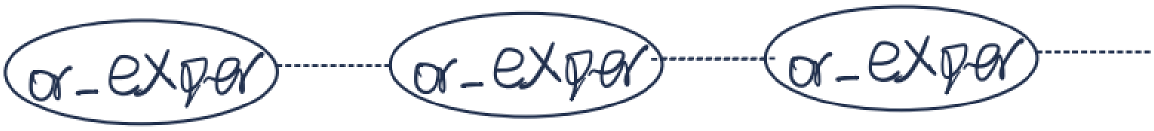
simple_declarator -> ID [“=” or_expr]



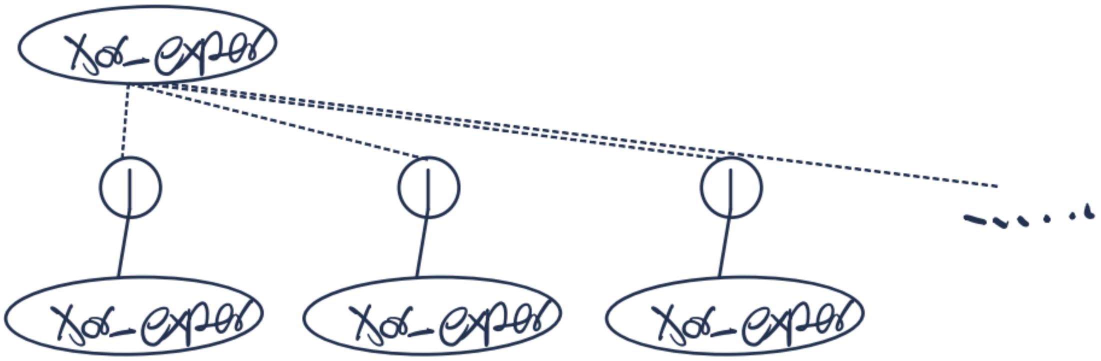
array_declarator -> ID “[” or_expr “]” [“=” exp_list]



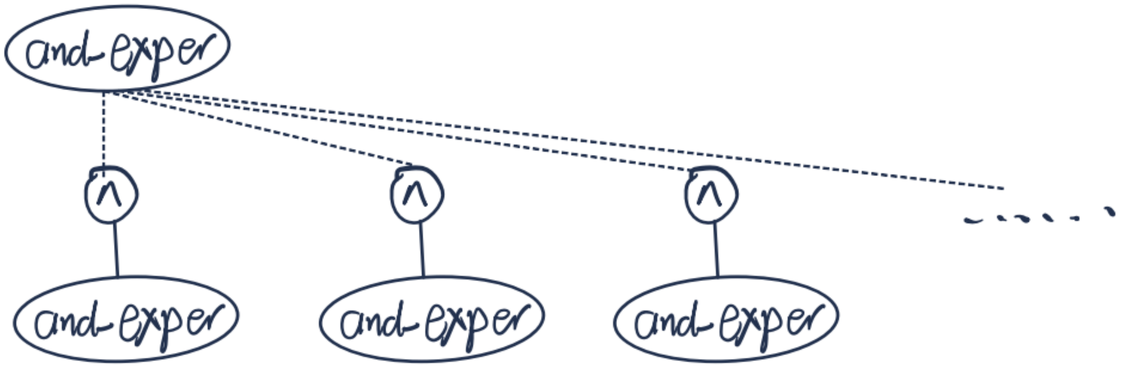
exp_list -> “[” or_expr { “,”or_expr } “]”



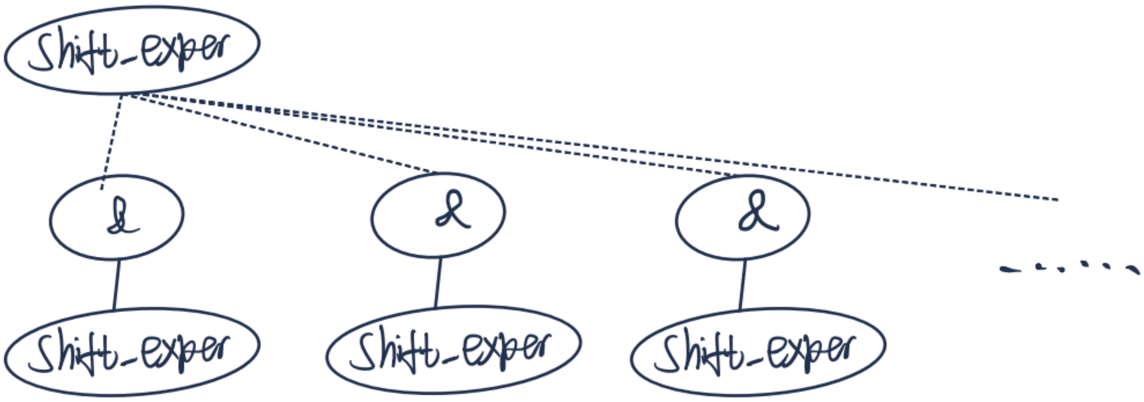
or_expr -> xor_expr {“|” xor_expr }



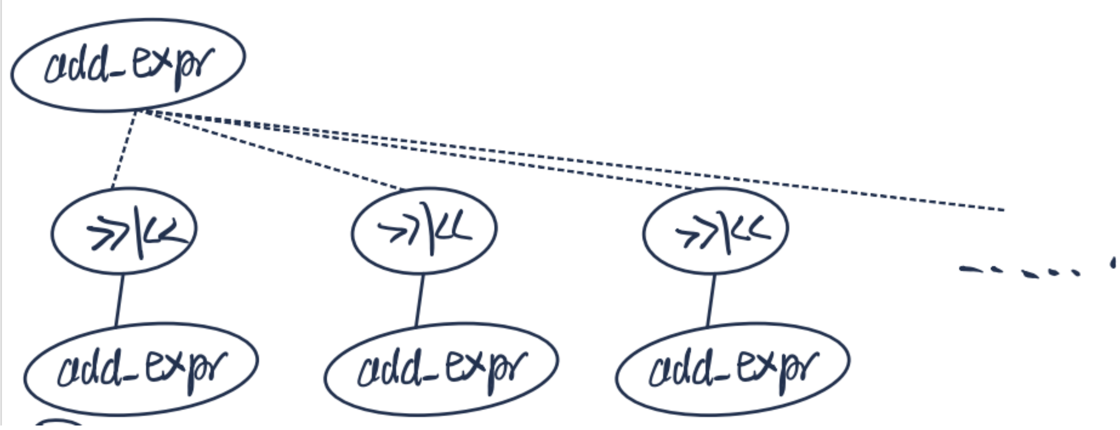
xor_expr -> and_expr {“^” and_expr }



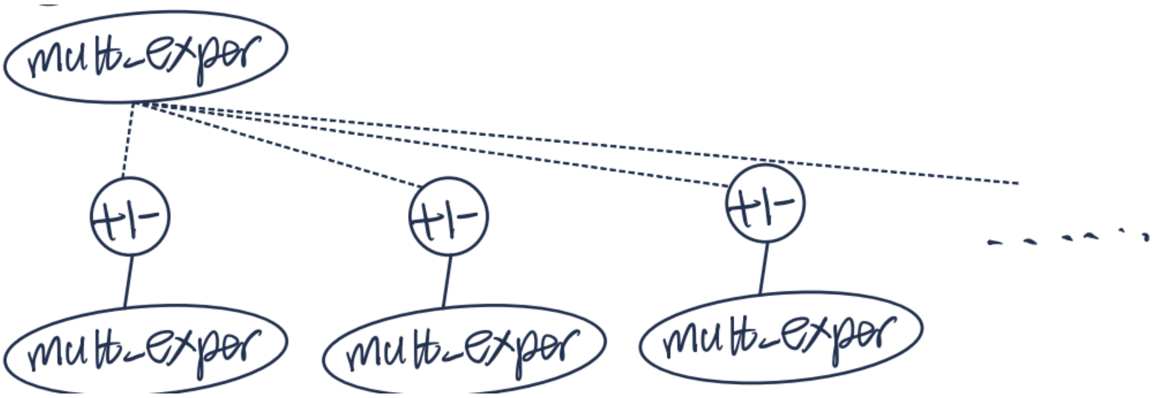
and_expr -> shift_expr {“&”shift_expr }



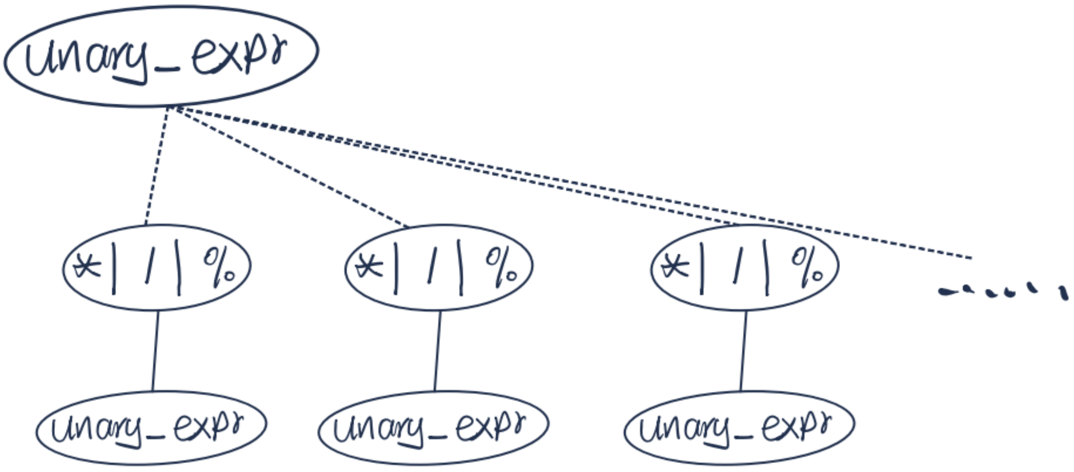
shift_expr -> add_expr { (“>>” | “<<”) add_expr }



add_expr -> mult_expr { (“+” | “-”) mult_expr }

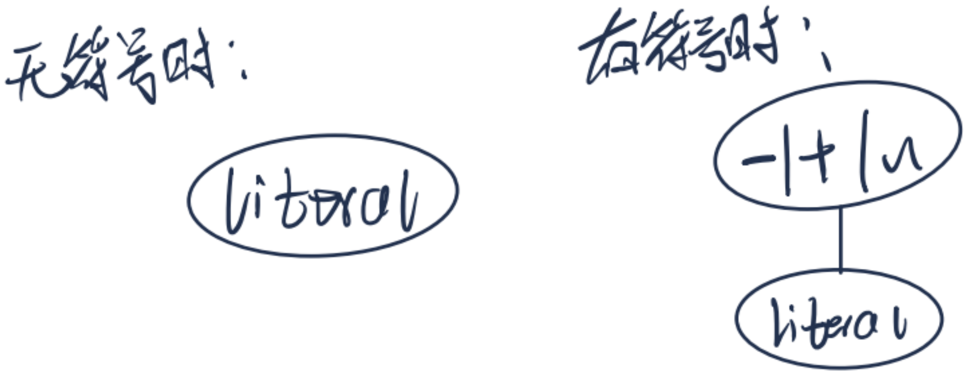


mult_expr -> unary_expr { (“*” |“/”|“%”) unary_expr }



unary_expr -> [“-”| “+” | “~”] literal

> 说明：这里我区分了有符号和无符号两种情况，分别对应两种抽象语法树。



literal -> INTEGER | FLOATING_PT | CHAR | STRING | BOOLEAN

INTEGER| FLOATING_PT|CHAR|STRING | BOOLEAN