nanoc grammar

CompUnit	→	{ CompUnit } (Decl | FuncDef | StructDef | ImplDef)
Decl	→	ConstDecl | VarDecl
ConstDecl	→	‘const’ BType ConstDef { ‘,’ ConstDef } ‘;’
BType	→	(‘int’ | ‘float’ | Ident) { ‘*’ } 支持指针类型，如 int*, float*, struct S*
ConstDef	→	Ident { ‘[‘ ConstExp ‘]’ } ‘=’ ConstInitVal
ConstInitVal	→	ConstExp | ‘{‘ [ConstInitVal { ‘,’ ConstInitVal }] ‘}’
VarDecl	→	BType VarDef { ‘,’ VarDef } ‘;’
VarDef	→	Ident { ‘[‘ ConstExp ‘]’ } [‘=’ InitVal]
InitVal	→	Exp | ‘{‘ [InitVal { ‘,’ InitVal }] ‘}’
FuncDef	→	FuncType Ident ‘(‘ [FuncFParams] ‘)’ Block
FuncType	→	‘void’ | ‘int’ | ‘float’
FuncFParams	→	FuncFParam { ‘,’ FuncFParam }
FuncFParam	→	BType Ident [‘[‘ ‘]’ { ‘[‘ Exp ‘]’ }]
Block	→	‘{‘ { BlockItem } ‘}’
BlockItem	→	Decl | Stmt
Stmt	→	LVal ‘=’ Exp ‘;’ | [Exp] ‘;’ | Block | ‘if’ ‘(‘ Exp ‘)’ Stmt [‘else’ Stmt] | ‘while’ ‘(‘ Exp ‘)’ Stmt | ‘break’ ‘;’ | ‘continue’ ‘;’ | ‘return’ [Exp] ‘;’
StructDef	→	‘struct’ Ident ‘{‘ { StructField } ‘}’
StructField	→	BType Ident { ‘[‘ ConstExp ‘]’ } ‘;’ 结构体字段定义
ImplDef	→	‘impl’ Ident ‘{‘ { MethodDef } ‘}’
MethodDef	→	FuncType Ident ‘(‘ [FuncFParams] ‘)’ Block 方法隐式带 this 指针
Exp	→	使用 pratt parse PrimaryExp | Exp BinaryOp Exp | UnaryOp Exp | Exp ‘[‘ Exp ‘]’ | Exp ‘.’ Ident | Exp ‘->’ Ident | Exp ‘(‘ [FuncRParams] ‘)’ -> 为指针成员访问

LVal	⟶	Ident { ‘[‘ Exp ‘]’ | ‘.’ Ident }
PrimaryExp	⟶	‘(‘ Exp ‘)’ | Ident | Number
Number	⟶	IntConst | floatConst
UnaryOp	⟶	‘+’ | ‘-’ | ‘!’ | ‘&’ | ‘*’ & 为取地址，* 为解引用
BinaryOp	⟶	‘+’ | ‘-’ | ‘*’ | ‘/’ | ‘%’ | ‘<’ | ‘>’ | ‘<=’ | ‘>=’ | ‘==’ | ‘!=’ | ‘&&’ | ‘||’